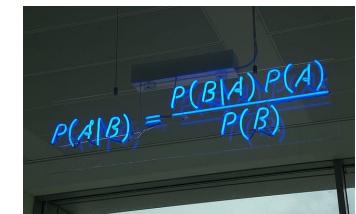


# Aprendizaje Automático

## Clase 4:

Curvas de aprendizaje  
Ensambles

Clasificadores: Modelos Discriminativos y Generativos

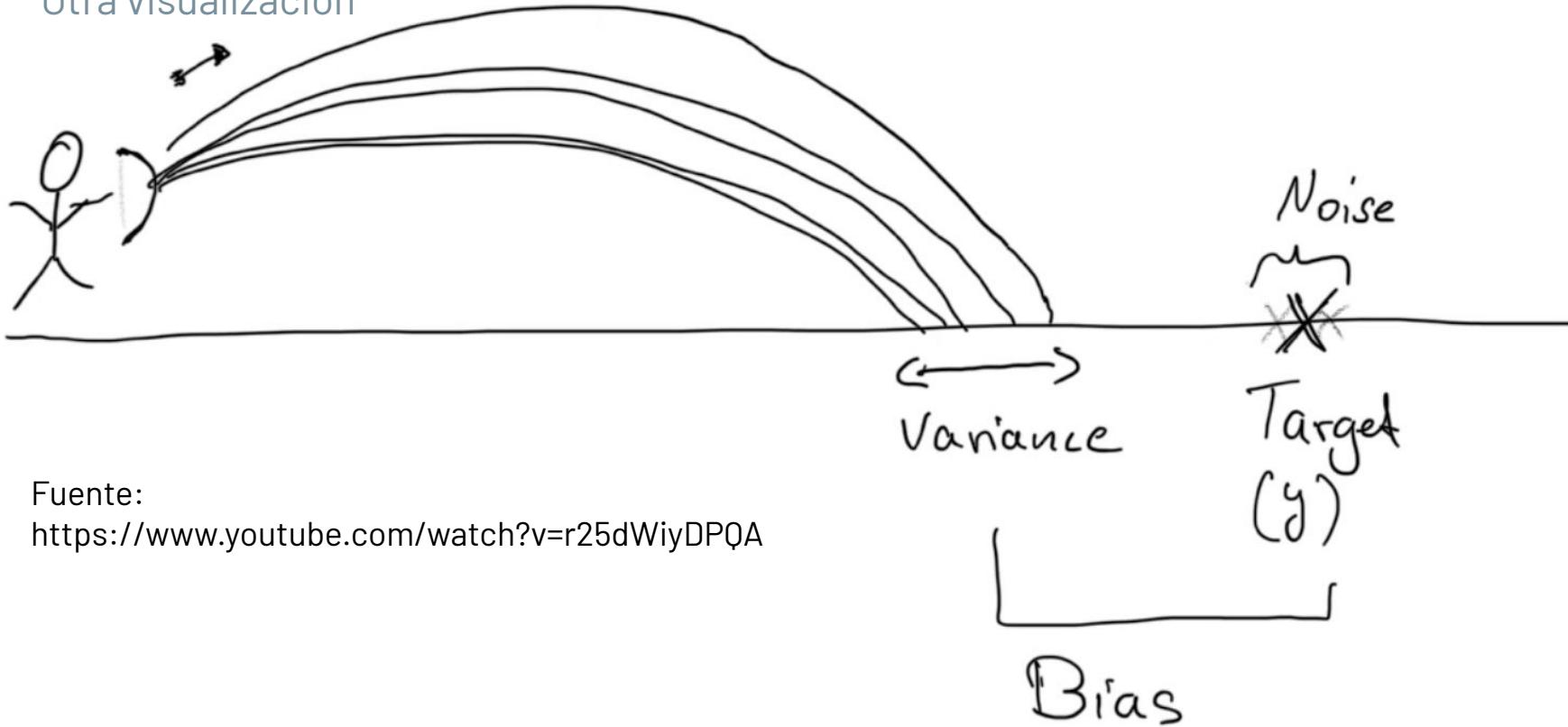


$$\text{Sesgo } [pred^{(i)}] = \mathbb{E} [dif(pred^{(i)}, f(x^{(i)}))]$$

$$\text{Var } [pred^{(i)}] = \mathbb{E}[dif(pred^{(i)}, \overline{pred}^{(i)})^2]$$

# Sesgo y Varianza

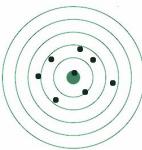
Otra visualización



Fuente:

<https://www.youtube.com/watch?v=r25dWiyDPQA>

# Herramientas de Diagnóstico: Curvas de aprendizaje



# Herramientas de diagnóstico

## Curvas de aprendizaje

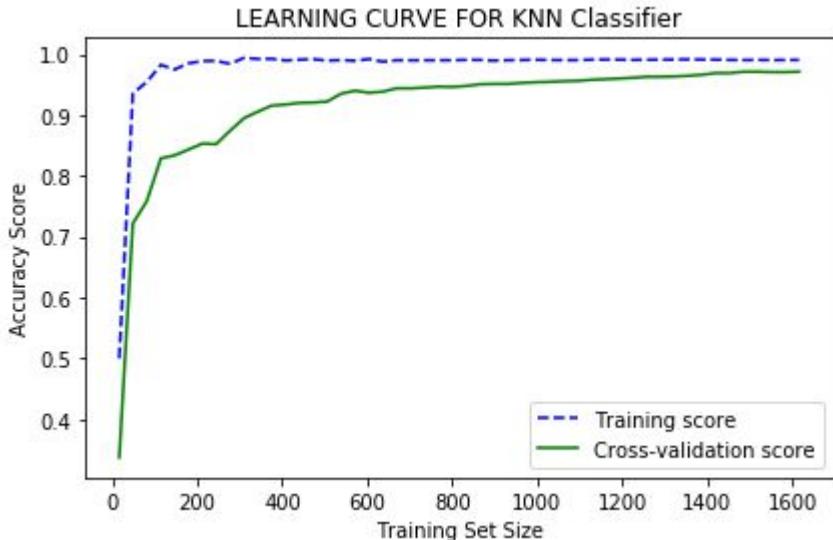


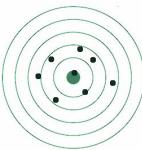
### Procedimiento:

Medir el error de entrenamiento y validación a medida que cambiamos la cantidad de datos de entrenamiento.

**Atención 1:** Manteniendo siempre el mismo conjunto de validación.

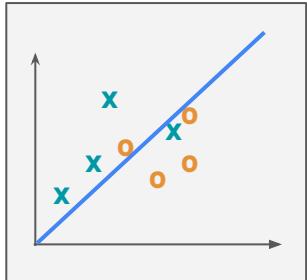
**Atención 2:** Incrementando el train set de manera acumulativa.





# Herramientas de diagnóstico

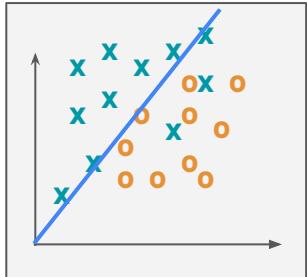
## Curvas de aprendizaje



### Ejemplo 1

LDA por ejemplo (lo veremos hoy)

**Alto sesgo.** No captura patrones complejos.



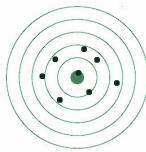
**Baja varianza.** Poca variación aunque cambiemos los datos.

¿Servirá extraer más atributos?

¿Convendrá recolectar más datos?

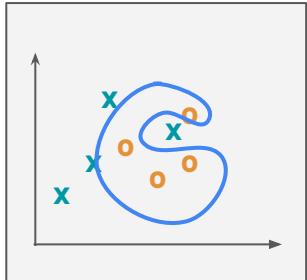
## Curvas de aprendizaje





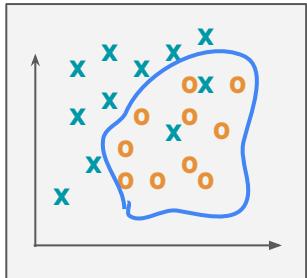
# Herramientas de diagnóstico

## Curvas de aprendizaje



### Ejemplo 2

Algoritmo más complejo  
(ej SVM no lineal, lo  
veremos hoy)



**Bajo sesgo.** En promedio  
encontrará el patrón  
adecuado.

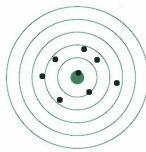
**Alta varianza.** Mucha  
variación ante cambios  
en los datos.

¿Servirá extraer más atributos?

¿Convendrá recolectar más datos?

## Curvas de aprendizaje





# Herramientas de diagnóstico

## Curvas de aprendizaje

### Opciones para disminuir la varianza

- Seleccionar modelos simples
- Reducción dimensional
- Regularización (pruning, lasso, ridge, etc)
- Usar algunos **ensambles**

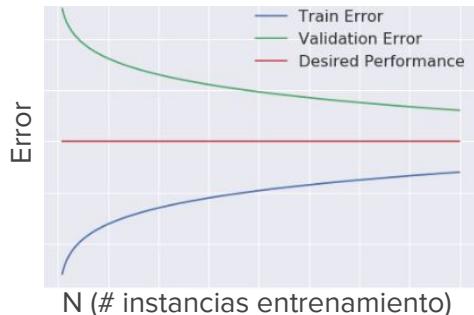
### Opciones para disminuir el bias

- Seleccionar modelos más complejos
- Extraer más features
- Usar otros **ensambles**

### Curvas de aprendizaje



**Alto Sesgo**



**Alta Varianza**

# Ensambles



Bagging

Random Forest

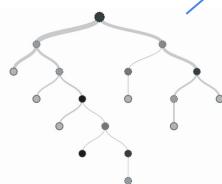
Boosting



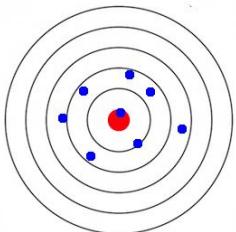
# Sesgo y Varianza

Un método: Ensamblaje

Regla de pulgar:



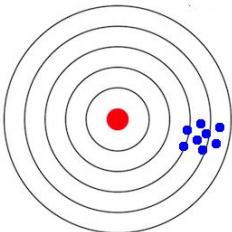
Overfit



“Bagging”  
Methods

Ensamblaje

Underfit



“Boosting”  
Methods



# Bagging

$$\text{Var} [pred^{(i)}] = \text{E}[dif(pred^{(i)}, \text{E}[pred^{(i)}])^2]$$



# Ensambles

## Bagging

**Teo:** Dado un conjunto de  $B$  variables aleatorias i.i.d. (independientes e idénticamente distribuidas)  $Z_1, \dots, Z_B$ , cada una con varianza  $\sigma^2$ : La varianza de la media  $Z$  de las observaciones está dada por

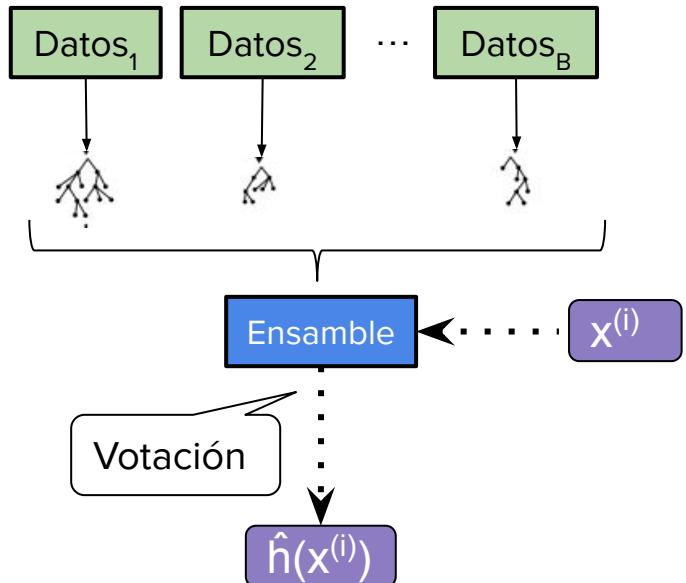
$$\text{Var}(Z) = \sigma^2/B.$$

Surge la idea de **promediar estimadores** para **reducir** su varianza.

En otros términos, hacer que  $pred^{(i)}$  tienda a  $\text{E}[pred^{(i)}]$

¿Cómo?

- Tomar muchos training sets distintos
- Construir un modelo predictivo distinto por cada set
- Promediar las predicciones resultantes
- $\hat{h}(x) = \text{combinar}(\hat{h}_{D1}(x), \hat{h}_{D2}(x), \dots, \hat{h}_{DB}(x))$
- ¿Es práctico? (¿de dónde sacamos  $B$  datasets?)



$$\text{Var} [pred^{(i)}] = \text{E}[dif(pred^{(i)}, \text{E}[pred^{(i)}])^2]$$

# Ensambles

## Bagging (Bootstrap Aggregating)

Método de Bootstrap (1979).



Dado un dataset, crear otros del mismo tamaño con instancias (filas) elegidas al azar (con reposición)

Data:

$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(6)}, x^{(7)}, x^{(8)}$

Bootstrap<sub>1</sub>:

$x^{(7)}, x^{(6)}, x^{(2)}, x^{(6)}, x^{(5)}, x^{(2)}, x^{(1)}, x^{(1)}$

Bootstrap<sub>2</sub>:

$x^{(2)}, x^{(6)}, x^{(5)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(1)}$

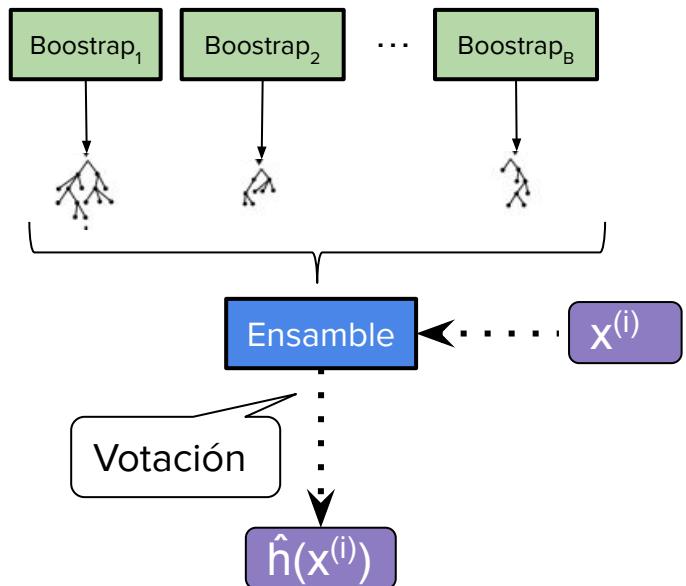
Bootstrap<sub>3</sub>:

$x^{(3)}, x^{(3)}, x^{(1)}, x^{(5)}, x^{(7)}, x^{(6)}, x^{(2)}, x^{(8)}$

**Problema** con Bagging: Árboles (sus predicciones) están muy correlacionadas ¿Por qué? (2 motivos)



## Bagging





# Ensambles

## Bagging (Bootstrap Aggregating)

Método de Bootstrap (1979).



Dado un dataset, crear otros del mismo tamaño con instancias (filas) elegidas al azar (con reposición)

Data:  $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(6)}, x^{(7)}, x^{(8)}$

Bootstrap<sub>1</sub>:  $x^{(7)}, x^{(6)}, x^{(2)}, x^{(6)}, x^{(5)}, x^{(2)}, x^{(1)}, x^{(1)}$

Bootstrap<sub>2</sub>:  $x^{(2)}, x^{(6)}, x^{(5)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(1)}$

Bootstrap<sub>3</sub>:  $x^{(3)}, x^{(3)}, x^{(1)}, x^{(5)}, x^{(7)}, x^{(6)}, x^{(2)}, x^{(8)}$

**Problema** con Bagging: Árboles (sus predicciones) están muy correlacionadas ¿Por qué? (2 motivos)

- Datos de entrenamiento
- Atributos dominantes

¿Qué tan parecidos son entre sí los datos?

P(elegir el mismo elem en dos dataset) =

$$1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632$$

Y cómo afecta la correlación entre predicciones?

**Teo:** Dado un conjunto de  $B$  variables aleatorias i.d. (idénticamente distribuidas, pero no necesariamente independientes)  $Z_1, \dots, Z_n$ , cada una con varianza  $\sigma^2$  con correlación entre pares positiva  $\rho$ :

La varianza de la media  $\bar{Z}$  de las observaciones está dada por:

$$\rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$

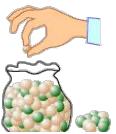
A medida que  $B$  crece, el segundo término desaparece, **pero no el primero**.



# Ensambles

## Random Forest

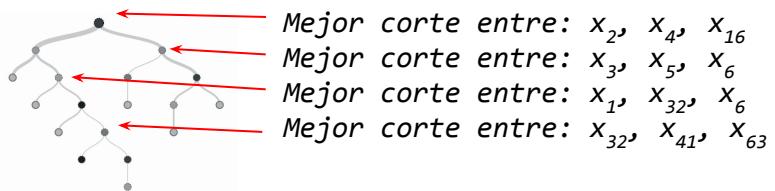
### Random Forest



- Bagging + Features al azar **por nodo**.
- **Para cada nodo**, elegimos al azar  **$m$**  features para considerar en el “atributo que mejor separe”.
- En general funciona  $m \approx \sqrt{p}$  ( $p = \# \text{features}$ ), también  $m \approx \log_2(p)$  aunque se puede ir tan bajo como  $m = 1$

(*Bootstrap*)

( $m=3$ )



“No todos los estimadores pueden mejorarse modificando los datos de esta manera. Parece que los estimadores altamente no lineales, como los árboles, son los que más se benefician.” (Elements of Statistical Learning)

### Algoritmos de la familia “Bagging”

#### Bagging [Breiman 1994]

Subsets basado en filas al azar (con reposición)

#### Random Subspaces (Feature Bagging) [Ho 1998]

Subsets basados en columnas al azar

#### Pasting [Breiman 1999]

Subsets basado en filas al azar. Los votos de los clasificadores se ponderan según su capacidad predictiva en un conjunto de validación.

#### Random Forest [Breiman 2001]

[Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.]

#### Extremely Randomized Trees [Geurts 2006]

Random forest + aleatorización en el mejor corte

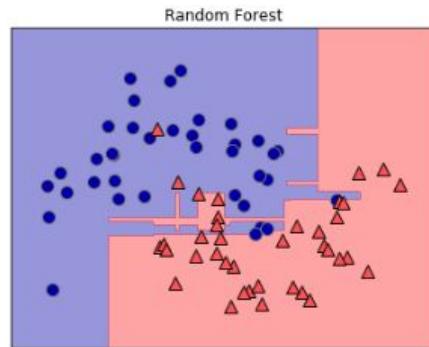
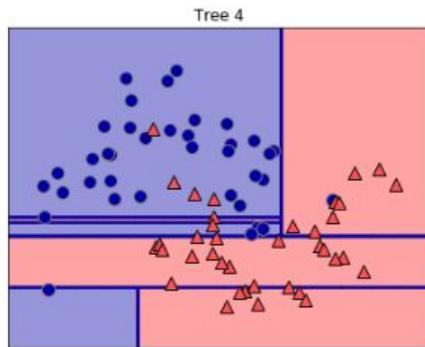
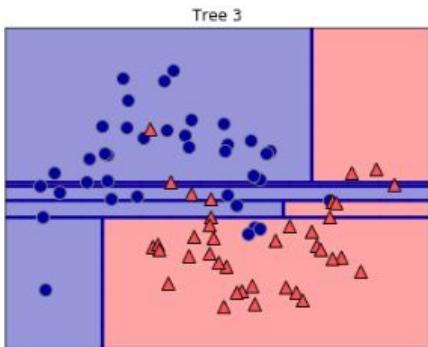
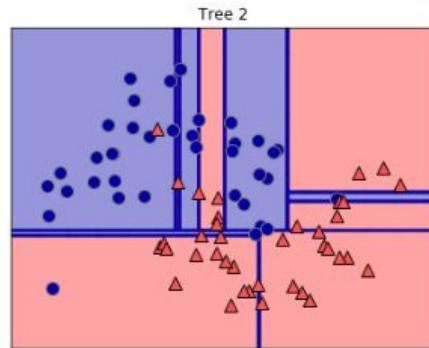
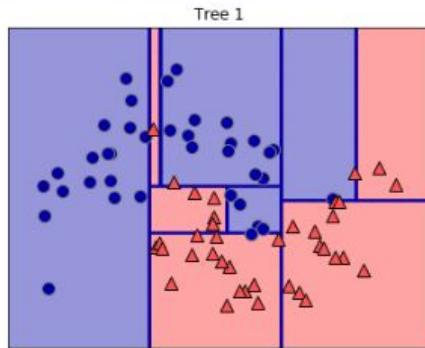
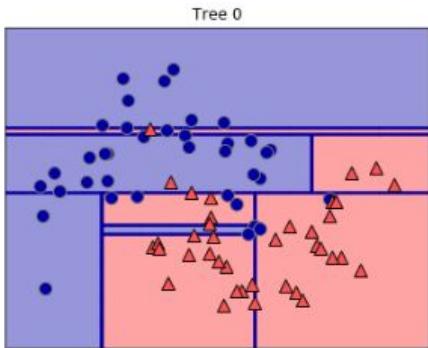
#### Random Patches [Louppe 2012]

Subsets basados en filas y columnas al azar.



# Ensambles

## Random Forest



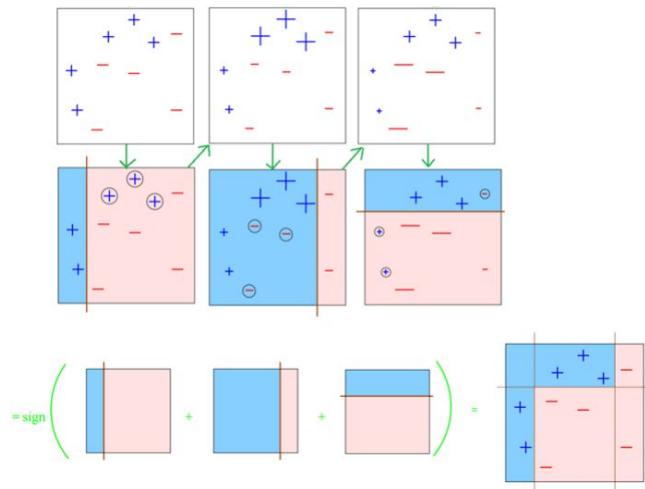
# Boosting

# Ensambles

Idea de Boosting (no lo veremos en detalle)

Estimadores son construidos de manera **secuencial**:

Idea, combinar clasificadores “**débiles**” (**alto sesgo**) de manera que cada clasificador se convierta en un “**experto**” en los errores que cometen los clasificadores anteriores.



**Algoritmos de la familia “Boosting”**

**Ada Boost [Freund 1997]**

Cada instancia tiene un peso determinado según si el algoritmo pudo o no predecir bien su valor en árboles anteriores.

**Gradient Boosting [Friedman 1999]**

Generalización del anterior a cualquier función de costo diferenciable

**eXtreme Gradient Boosting (XGBoost) [Chen 2016]**

Implementación eficiente de Gradient Boosting + regularización en los nuevos árboles

**(Algoritmo ganador en competencias de Kaggle)**



# Clasificadores: Modelos Discriminativos y Generativos

# El clasificador óptimo de Bayes

Al construir un clasificador, en general estamos interesados en disminuir el error en "la realidad". Es decir, minimizar

$$Err_{true}(h) = \mathbb{E}_x[\text{error}(h(x), y)]$$

En clasificación, este error puede definirse como  $h(\mathbf{x}) \neq \mathbf{y}$  (se lo llama "riesgo"). Se puede demostrar que el clasificador que minimiza el riesgo es:

El **clasificador óptimo de Bayes**, un modelo probabilístico que devuelve la predicción más probable para un nuevo ejemplo, dado los valores de sus atributos. Es decir,

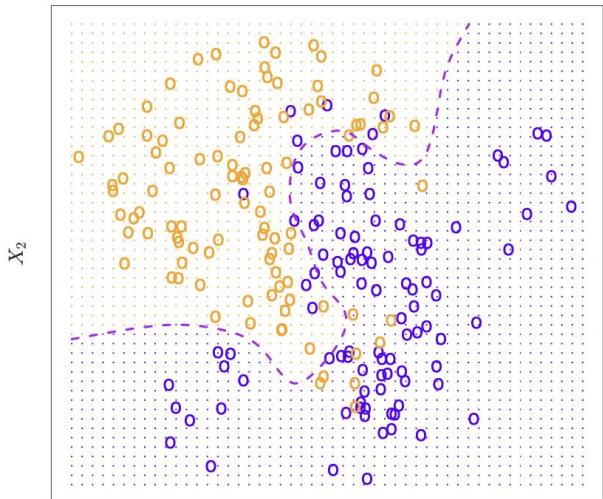
$$Pred(x^{(i)}) = \arg \max_{c \in \text{Clases}} P(Y = c | X = x^{(i)})$$

En teoría, siempre nos gustaría predecir respuestas cualitativas utilizando el clasificador de Bayes. ¿Por qué no hacerlo entonces?

Pero **para datos reales, no conocemos la distribución condicional de Y dada X**, por lo que calcular el clasificador de Bayes es **imposible**.

Por lo tanto, el clasificador de Bayes sirve como un estándar inalcanzable. En casos específicos (datos simulados), permite comparar diferentes métodos.

Fronteras de decisión según un clasificador de tipo "Bayes Optimal Classifier" para datos simulados .



X<sub>1</sub>  
**(notar** que la frontera hubiera sido la misma sin importar que muestra se visualice aquí)

# Modelos Discriminativos

# Enfoques (parte 1)

$$P(Y = c|X = x^{(i)}) \quad \forall c \in Clases$$

Muchos enfoques intentan **estimar** esta probabilidad a partir de un conjunto de **datos de entrenamiento**. Es decir, modelar (aprender):

$$\hat{P}(Y=c|X=x^{(i)}) \quad \forall c \in Clases$$

Luego clasificar una observación dada en la clase con la probabilidad estimada más alta:

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

Estos se conocen como modelos **DISCRIMINATIVOS**

(más adelante en la clase veremos que no es el único enfoque).

(algunos) ejemplos de modelos discriminativos:

- **Árboles de decisión** (ya lo vimos)
- **K-vecinos más cercanos** (next)
- **Support Vector Machines (SVM)** (next)
- **Regresión logística** (proximamente)
- **Random Forest (y otros ensambles)** (ya lo vimos)
- **Maximum-entropy Markov models**
- **Conditional random fields**
- **Redes neuronales (no todas)** (proximamente)

modelos discriminativos

# K Vecinos Más Cercanos

KNN

NOTHING MAKES YOU MORE  
TOLERANT OF A NOISY  
NEIGHBOR'S PARTY THAN  
BEING THERE.

— FRANKLIN P. JONES

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# K vecinos más cercanos (KNN)

## Método discriminativo

Estima la probabilidad condicional para la clase  $c$  dado  $\mathbf{x}$  como: La fracción de puntos en  $N_{D,k}(\mathbf{x})$ .  
 $N_{D,k}(\mathbf{x})$  = los  $k$  vecinos más cercanos a  $\mathbf{x}$  en el conjunto  $D$ , cuyas etiquetas sean iguales a  $c$ .

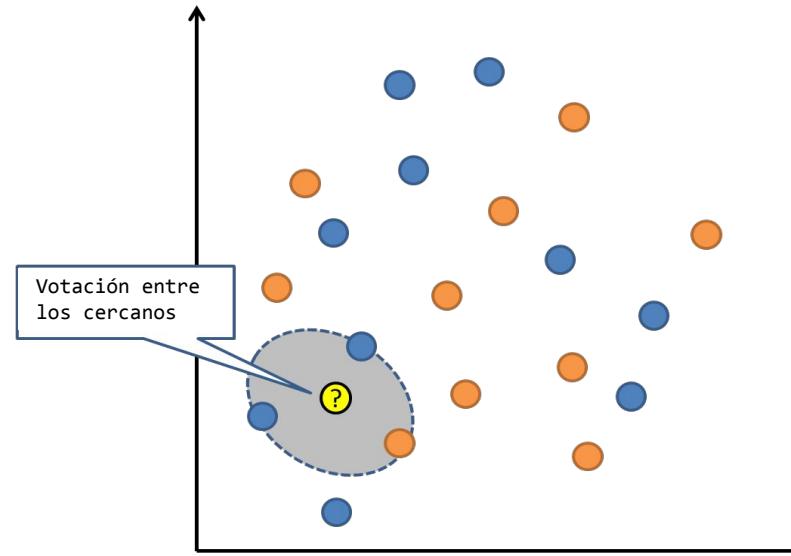
$$Pred(x^{(i)}) = \arg \max_{c \in Clases} P(Y = c|X = x^{(i)})$$

$$\approx \arg \max_{c \in Clases} \sum_{i \in N_{D,k}(x)} I(y^{(i)} = c)$$

+ sesgo inductivo  
+ datos

$$Pred_{proba}(x^{(i)}, c) = P(Y = c|X = x^{(i)})$$

$$\approx \frac{1}{k} \sum_{i \in N_{D,k}(x)} I(y^{(i)} = c)$$



Algunas versiones ponderan la votación según la distancia

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# K vecinos más cercanos (KNN)

## Método discriminativo

¿Cómo se implementa? Algoritmo de asignación a la instancia  $x^{(i)}$ :

1. Computar la distancia  $D(x^{(i)}, x^{(j)})$  para todo punto de entrenamiento  $x^{(j)}$ .
2. Seleccionar las  $k$  instancias más cercanas y sus etiquetas.
3. Devolver la etiqueta más frecuente (o la proporción si es proba).

**Pregunta** (clásico de entrevistas) (**lo dejamos para la práctica**).

Quiero predecir si una casa tendrá techo verde o no.

¿Qué pasa si los atributos son los siguientes y aplico KNN?

$X_1$ : distancia al obelisco (medido en metros),

$X_2$ : cantidad de personas que viven en la cuadra (de 1 a 500).

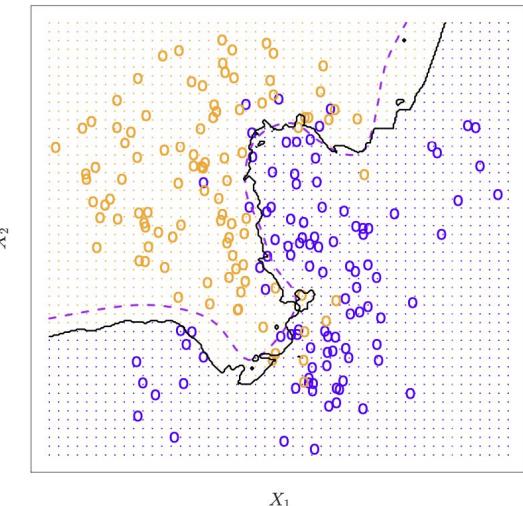
$X_3$ : ¿tiene jardín?: si, no (1 o 0).

¿Qué distancia uso? ¿Euclidiana? ¿Hago algo antes? ¿Por qué no pasaba en árboles?

Fronteras de decisión según un clasificador de tipo **KNN** con **K=10**.

(**¿cómo dibujarían esto?**)

Línea punteada: **Bayes Optimal Classifier**



¿Cómo cambian las fronteras de decisión si cambio el K?

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# K vecinos más cercanos (KNN)

## Método discriminativo

¿Cómo se implementa? Algoritmo de asignación a la instancia  $x^{(i)}$ :

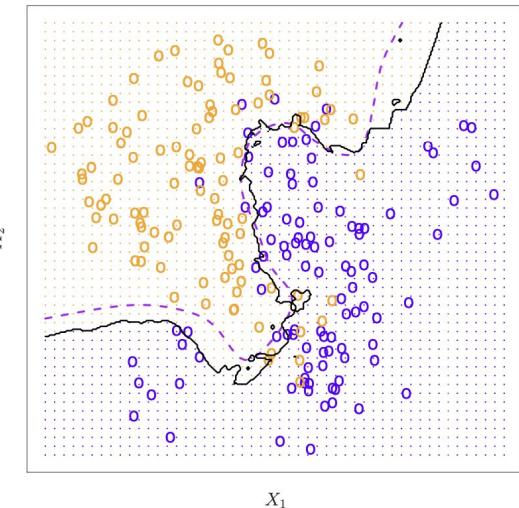
1. Computar la distancia  $D(x^{(i)}, x^{(j)})$  para todo punto de entrenamiento  $x^{(j)}$ .
2. Seleccionar las  $k$  instancias más cercanas y sus etiquetas.
3. Devolver la etiqueta más frecuente (o la proporción si es proba).

- ¿Cuánto ocupa en memoria una vez entrenado el modelo ?
- ¿Modelo?
- ¿Cuál es el algoritmo de entrenamiento?
- ¿Entrenamiento?

Fronteras de decisión según un clasificador de tipo **KNN** con **K=10**.

(**¿cómo dibujarían esto?**)

Línea punteada: **Bayes Optimal Classifier**



modelos discriminativos

# Support Vector Machines

SVM

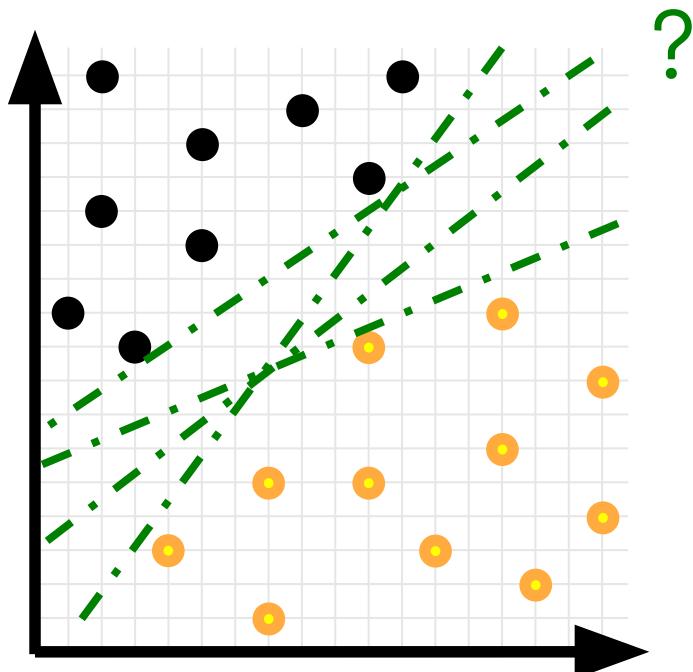
Recomendado [StatQuest - SVM \(youtube\)](#)



$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Margen



**Idea:** buscar una recta que separe las clases lo mejor posible, sin tener que modelar la distribución de los datos de cada clase.

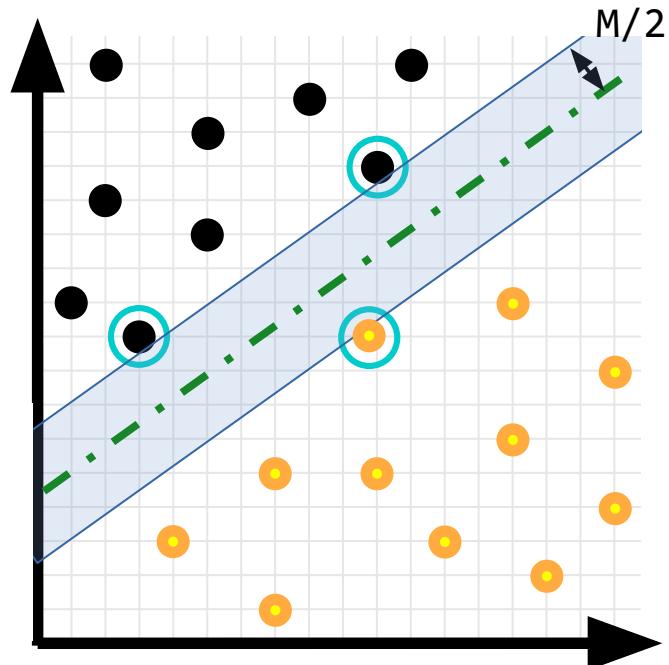
¿Cómo podemos elegir esa recta?  
(supongamos por un rato que existe dicha recta)

Buscaremos el **hiperplano de margen máximo** (también conocido como hiperplano de separación óptimo), que es el hiperplano de separación que está más alejado de las observaciones de entrenamiento.

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Margen



**Hiperplano de margen máximo** (recta verde en este caso). El hiperplano de dimensión ( $d-1$ ) tal que la distancia a las instancias más cercanas es máxima y todas las instancias quedan del lado correcto del plano. Recordar que las instancias viven en el espacio de atributos de dim  $d$ .

### Margen M:

Distancia de las instancias más cercanas a la recta (hiperplano) de decisión ( $x_2$ ).

**“Support Vectors”** (vectores de soporte, en este caso con círculos celestes): Instancias más cercanas al hiperplano de decisión

El algoritmo buscará maximizar M.

- **Problema de optimización** (mundo **prog. cuadrática**).  
**Pinta**: “Maximizar el margen con tal y tal restricción”. Existe una solución eficiente. Ahora lo veremos más en detalle.

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

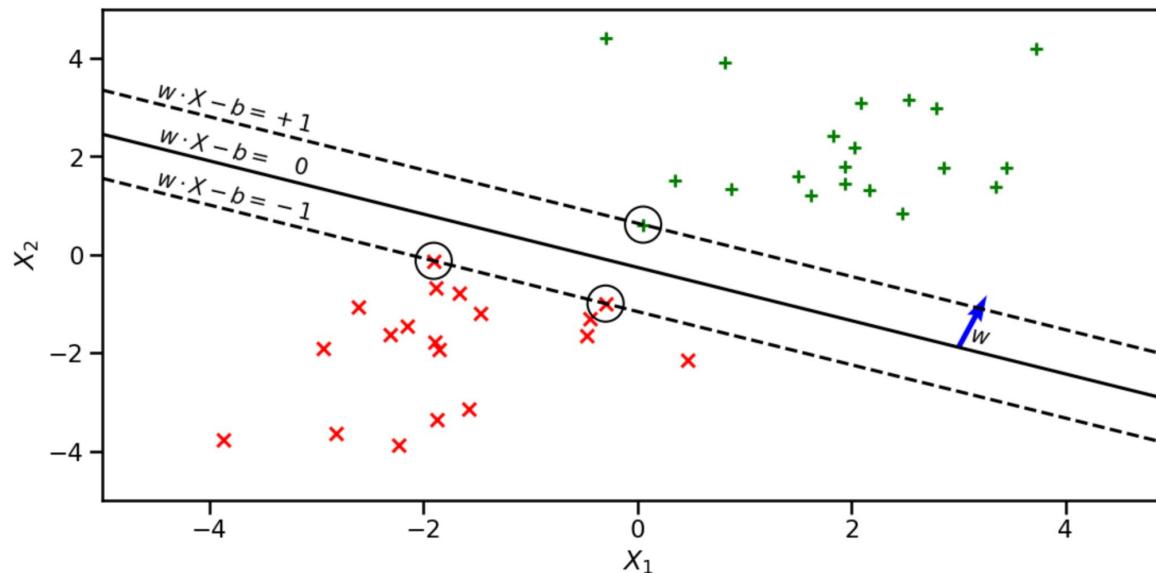
# Support Vector Machine (SVM)

## Definiciones

Consideremos etiquetas  $+1$  y  $-1$  para las clases positiva y negativa, respectivamente.

Recordemos la ecuación de un hiperplano:  $\mathbf{w}^\top \mathbf{X} - \mathbf{b} = 0$ . Esto hará que los vectores de soporte cuplan con  $\mathbf{w}^\top \mathbf{x}^{(i)} - \mathbf{b} = 1$  y el resto  $(\mathbf{w}^\top \mathbf{x}^{(i)} - \mathbf{b}) > 1$  (o equiv. para negativos)

Notar que  $\mathbf{w}$  es un vector normal a la frontera de decisión



El tamaño del margen es  $2/\|\mathbf{w}\|_2$  (demo al final de las slides, para los que quieran)

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Maximización del Margen

El tamaño del margen es  $2/\|\mathbf{w}\|$ : Queremos maximizarlo. O equivalentemente  $\min \|\mathbf{w}\|$

Pero con dos restricciones:

- Ningún punto positivo cae del lado incorrecto:  $\mathbf{w}^\top \mathbf{x}_{(+)} - b \geq 1$  para todo  $\mathbf{x}_{(+)}$
- Ningún punto negativo cae del lado incorrecto:  $\mathbf{w}^\top \mathbf{x}_{(-)} - b \leq -1$  para todo  $\mathbf{x}_{(-)}$

Escrito de otra forma:  $\min_{\mathbf{w}, b} \|\mathbf{w}\|_2$  tal que  $y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} - b) \geq 1 \quad \forall (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in D_{train}$

A saber,  $\mathbf{y}^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} - b) = 1$  son los vectores de soporte

La solución a este problema se puede encontrar utilizando técnicas de **optimización convexa**, (formulación dual de Lagrange). No lo veremos, pero tiene esta pinta:

**Solución** (en donde  $\alpha_i$  y  $b$  se aprenden a partir de los datos)

$$f(\mathbf{x}^{(k)}) = \text{sign} \left( \sum_{i=1}^n \alpha_i \mathbf{y}^{(i)} \mathbf{x}^{(i)\top} \mathbf{x}^{(k)} + b \right)$$

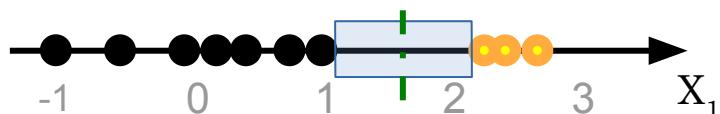
$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

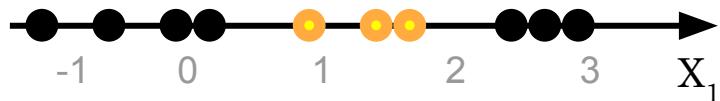
## Separaciones no lineales - Atributos Slack

Supongamos para estos ejemplos  $\mathbf{x}^{(i)} = [x_1]$

En este caso, es sencillo encontrar un punto que separe bien

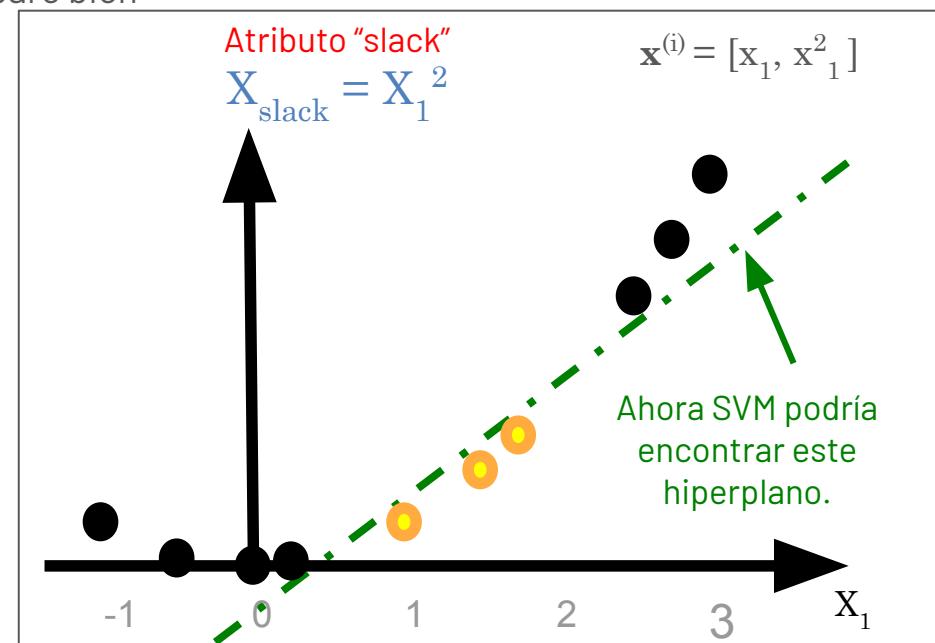


¿Qué sucede en un caso como este? Es un dataset linealmente separable?



**Notación  
(instancia / atributos)**

$$\mathbf{x}^{(i)} = [x_1, x_2, \dots, x_p]$$



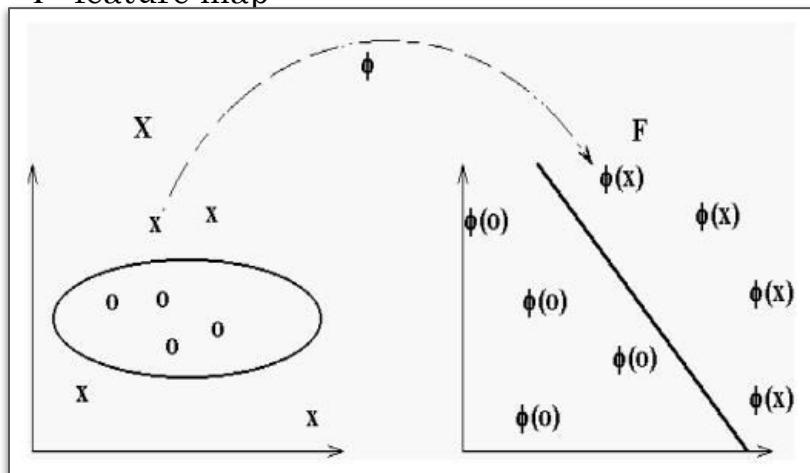
$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Separaciones no lineales - Atributos Slack

Transformación de vectores de atributos.

$\Phi$  “feature map”



Ejemplo de una transformación polinómica de grado 2:

$$\Phi_{poly}(\mathbf{x}) = \Phi_{poly}([x_1, x_2]) = (x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

$$\dim(\Phi_{poly}(\mathbf{x})) \in O(p^d) \quad (\text{p dim x, d grado polinomio})$$

$$\min \|\mathbf{w}\|$$

Ya mencionamos que (sin feature maps), la solución tiene la siguiente forma:

$$f(\mathbf{x}^{(k)}) = \text{sign} \left( \sum_{i=1}^n \alpha_i \mathbf{y}^{(i)} \mathbf{x}^{(i)\top} \mathbf{x}^{(k)} + b \right)$$

Al utilizar feature maps, la solución tiene la pinta:

$$f(\mathbf{x}^{(k)}) = \text{sign} \left( \sum_{i=1}^n \alpha_i \mathbf{y}^{(i)} \Phi(\mathbf{x}^{(i)})^\top \Phi(\mathbf{x}^{(k)}) + b \right)$$

Expandir las transformaciones  $\Phi(..)$  es muy costoso (o a veces imposible).

**Clave:** la solución sólo usa “ $\Phi(\mathbf{x}^{(i)})^\top \Phi(\mathbf{x}^{(k)})$ ” (el producto interno,  $\langle \Phi(\mathbf{x}^{(i)}), \Phi(\mathbf{x}^{(k)}) \rangle$ ) donde  $\mathbf{x}^{(i)}, \mathbf{x}^{(k)}$  son dos instancias distintas.

¿Podremos calcular  $\langle \Phi(\mathbf{x}^{(i)}), \Phi(\mathbf{x}^{(k)}) \rangle$  eficientemente?

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Separaciones no lineales - Kernel Trick

**Kernel:** Generalización del producto interno que nos permite operar con nuevos atributos en forma implícita.

$K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \langle \Phi(\mathbf{x}^{(1)}), \Phi(\mathbf{x}^{(2)}) \rangle$  donde  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$  son dos instancias.

Si un algoritmo (ej. SVM) puede expresarse en términos de productos internos entre instancias, reemplazamos las apariciones de  $\langle \Phi(\mathbf{x}^{(1)}), \Phi(\mathbf{x}^{(2)}) \rangle$  por  $K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$ . Sin nunca tener que computar  $\Phi(\mathbf{x})$ .

Así, ejecutamos SVM **implícitamente** en dimensiones superiores (nunca computamos  $\Phi(\cdot)$ )

Sugerencia,

- ver: <https://www.youtube.com/watch?v=07vT0-5VII>
- **O para mucho más detalle:** <https://www.youtube.com/watch?v=8NYoQiRANpg>

## Ejemplos

### Kernel Lineal

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)\top} \cdot \mathbf{x}^{(j)}$$

### Kernel Polinómico (dim d)

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)\top} \cdot \mathbf{x}^{(j)} + c)^d$$
 $\dim(\Phi(\mathbf{x})) \in O(\mathbb{R}^{(p^d)})$

### Gaussian Kernel

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = e^{-\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}$$
 $\Phi(\mathbf{x}) \in \mathbb{R}^\infty$

### Kernel Sigmoideo

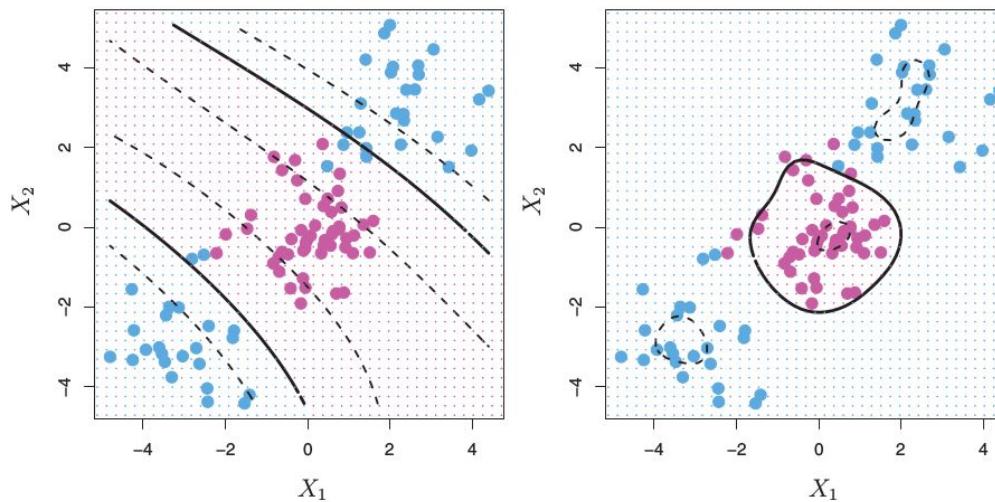
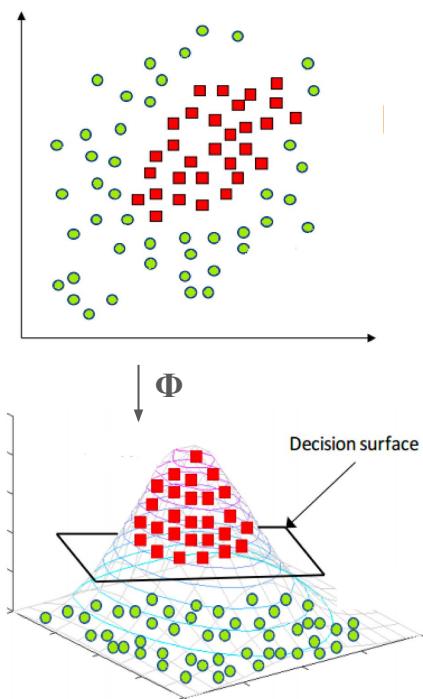
$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\gamma(\mathbf{x}^{(i)\top} \cdot \mathbf{x}^{(j)}) + r)$$

[scikit-learn.org/stable/modules/svm.html#svm-kernels](http://scikit-learn.org/stable/modules/svm.html#svm-kernels)

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

## Separaciones no lineales - Kernel Trick



**Izquierda:** Se aplica SVM con un kernel polinómico de grado 3.

**Derecha:** Se aplica SVM con un kernel radial.

En este ejemplo, ambos kernels son capaces de capturar la frontera de decisión. [ISLR, cap9, FIGURA 9.9]

**La visualización se hace sobre el espacio de atributos original.** Son proyecciones del hiperplano y de los márgenes en dimensiones más altas

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# Support Vector Machine (SVM)

Falto ver:

- Ver cómo permitir instancias “incorrectas” dentro de los márgenes. Soft Margin Classifiers. Sugiero [youtube.com/watch?v=ljSfa708nqs](https://youtube.com/watch?v=ljSfa708nqs) y busquen para qué sirve el hiperparámetro **C**
- Busquen cómo se puede utilizar SVM para **problemas multiclas**.
- Busquen cómo se calcula el **score** de una predicción.
- Ejemplo de uso para anomaly detection:  
[medium.com/@roshmitadey/anomaly-detection-using-support-vectors-2c1b842213ed](https://medium.com/@roshmitadey/anomaly-detection-using-support-vectors-2c1b842213ed)

**¿De todo esto qué tengo que saber?**

- Entender que es un Hiperplano de margen máximo
- Entender qué son los vectores de soporte.
- Entender para qué sirve  $\Phi$  (y saber que no se computa)
- Entender el Kernel Trick

**Costo computacional:**

**De entrenamiento.**  $O(n^3 * d)$  (pero hay trucos/consideraciones que lo hacen práctico)

Fuente:

[scikit-learn.org/stable/modules/svm.html#complexity](https://scikit-learn.org/stable/modules/svm.html#complexity)

2 Answers

Sorted by: Highest score (default) 



8



In order of increasing on-line complexity:

1. Linear SVM has prediction complexity  $O(d)$  with  $d$  the number of input dimensions since it is just a single inner product.
2. NN complexity is related to the architecture, but will surely be above that of linear SVM.
3. Prediction complexity of kernel SVM depends on the choice of kernel and is typically proportional to the number of support vectors. For most kernels, including polynomial and RBF, this is  $O(n_{SV} d)$  where  $n_{SV}$  is the number of support vectors. [An approximation exists for SVMs with an RBF kernel that reduces the complexity to  \$O\(d^2\)\$](#) . For computer vision applications, [additive kernels](#) are often used because they yield very fast prediction speed (independent of the number of SVs).

# Modelos Generativos

# Enfoques (parte 2)

Vimos hasta ahora: Modelos **DISCRIMINATIVOS** clase con la posterior  $Pred(x^{(i)}) = \arg \max_{c \in Clases} P(Y = c | X = x^{(i)})$

Otros métodos surgen de aplicar el teorema de bayes:

$$\begin{aligned} Pred(x^{(i)}) &= \arg \max_{c \in Clases} P(Y = c | X = x^{(i)}) \\ &= \arg \max_{c \in Clases} \frac{P(Y = c)P(X = x^{(i)} | Y = c)}{P(X = x^{(i)})} \\ &= \arg \max_{c \in Clases} P(Y = c) \boxed{P(X = x^{(i)} | Y = c)} \\ &= \arg \max_{c \in Clases} P(X = x^{(i)}, Y = c) \end{aligned}$$

Esto es lo que interesa aproximar ahora

Estos se conocen como modelos **GENERATIVOS**.

¡¡Conocer  $P(X = x^{(i)} | Y = c)$  permite generar muestras!!!

Para pensar: ¿cómo obtener la probabilidad de una predicción en generativos?  $\text{Pred}_{\text{proba}}(x^{(i)}, c)$

(algunos) ejemplos de modelos discriminativos:

- **Árboles de decisión** (ya lo vimos)
- **K-vecinos más cercanos** (hoy)
- **Support Vector Machines (SVM)** (hoy)
- **Regresión logística** (prox)
- **Random Forest (y otros ensambles)** (prox)
- **Maximum-entropy Markov models**
- **Conditional random fields**

Teorema de Bayes

$$P(Y = c | X = x^{(i)}) = \frac{\underbrace{P(X = x^{(i)} | Y = c)}_{\text{posterior}} \cdot \underbrace{P(Y = c)}_{\text{prior}}}{\underbrace{P(X = x^{(i)})}_{\text{marginal likelihood}}} \quad \begin{matrix} \text{likelihood (verosimilitud)} \\ \text{prior} \end{matrix}$$

(algunos) ejemplos de modelos generativos:

- **linear discriminant analysis (LDA)** (hoy)
- **gaussian/naive bayes** (hoy)
- **Gaussian mixture model (GMMs)** (prox)
- **Hidden Markov Models (HMMs)**
- **Generative adversarial networks (GANs)**,
- **Autoregressive models** (por ejemplo **GPT-3**)
- **Diffusion models**

modelos generativos

# Linear y Quadratic Discriminant Analysis

LDA  
QDA



# Linear / Quadratic Discriminant Analysis

## Método generativo

$$\begin{aligned} Pred(x^{(i)}) &= \arg \max_{c \in Clases} P(Y=c|X=x^{(i)}) \\ &= \arg \max_{c \in Clases} \frac{P(Y=c)P(X=x^{(i)}|Y=c)}{P(X=x^{(i)})} \\ &= \arg \max_{c \in Clases} P(Y=c) \boxed{P(X=x^{(i)}|Y=c)} \\ &= \arg \max_{c \in Clases} P(Y=c) pdf_c(x^{(i)}) \end{aligned}$$

Para X continua

$$\begin{aligned} &\approx \begin{cases} \arg \max_{c \in Clases} \hat{P}(Y=c) f_{norm}(x^{(i)}; \hat{\mu}_c, \hat{\sigma}_c) & \text{para } x^{(i)} \in R \\ \arg \max_{c \in Clases} \hat{P}(Y=c) f_{norm}^d(x^{(i)}; \hat{\boldsymbol{\mu}}_c, \hat{\boldsymbol{\Sigma}}_c) & \text{para } x^{(i)} \in R^d \end{cases} & X|Y=c \sim \mathcal{N}(\mu, \sigma) \\ && X|Y=c \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \end{aligned}$$

$$f_{norm}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$f_{norm}^d(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2} (\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$

$$\begin{aligned}
 Pred(x^{(i)}) &= \arg \max_{c \in Clases} P(Y=c|X=x^{(i)}) \\
 &= \arg \max_{c \in Clases} \frac{P(Y=c)P(X=x^{(i)}|Y=c)}{P(X=x^{(i)})} \\
 &= \arg \max_{c \in Clases} P(Y=c)P(X=x^{(i)}|Y=c) \\
 &= \arg \max_{c \in Clases} P(Y=c)pdf_c(x^{(i)})
 \end{aligned}$$

# Linear / Quadratic Discriminant Analysis

## Método generativo

(de la diapo anterior, caso general, multidimensional)

$$Pred(x^{(i)}) = \boxed{\arg \max_{c \in Clases} \hat{P}(Y=c)f_{norm}^d(x^{(i)}; \hat{\mu}_c, \hat{\Sigma}_c)} \quad \text{para } x^{(i)} \in R^d$$

- La suposición de estos métodos es que  $\mathbf{X} | \mathbf{Y} = c$  sigue una distribución normal  $\mathbf{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ . Es decir, los puntos en cada clase fueron generados por distribuciones normales distintas.
- Con esta suposición, para modelar la distribución de instancias de cada clase,  $P(X=x | Y=c)$ , alcanza con estimar  $\boldsymbol{\mu}_c$ ,  $\boldsymbol{\Sigma}_c$ ,  $P(\mathbf{Y}=c)$  usando los datos de entrenamiento.
- Encontrar la clase  $c$  con probabilidad máxima a posteriori sale directo (usando las fórmulas de la diapo anterior).

Si la matriz de covarianza (o los desvíos en univariado) se suponen iguales para toda clase, este método se llama:  
**Linear Discriminant Analysis (LDA)**. Si no: **Quadratic Discriminant Analysis (QDA)**

- $\boldsymbol{\mu}_c$  y  $\boldsymbol{\Sigma}_c$  pueden ser estimados mediante el método de máxima verosimilitud (con la corrección de Bessel):

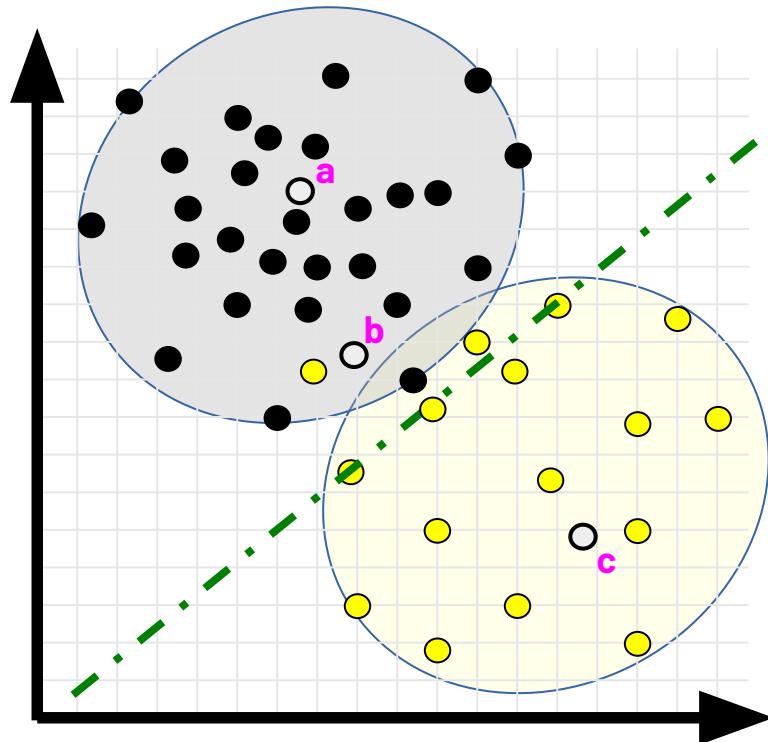
$$\hat{\mu}_c = \frac{1}{n_c} \sum_{x^{(i)} \in \{D | y^{(i)}=c\}} x^{(i)}$$

$$\hat{\Sigma}_c = \frac{1}{n_c - 1} \sum_{x^{(i)} \in \{D | y^{(i)}=c\}} (\mathbf{x}^{(i)} - \hat{\mu}_c)(\mathbf{x}^{(i)} - \hat{\mu}_c)^T$$

$$\hat{P}(Y=c) = \frac{n_c}{n}$$

# Linear / Quadratic Discriminant Analysis

## Método generativo



Esquematización (LDA)

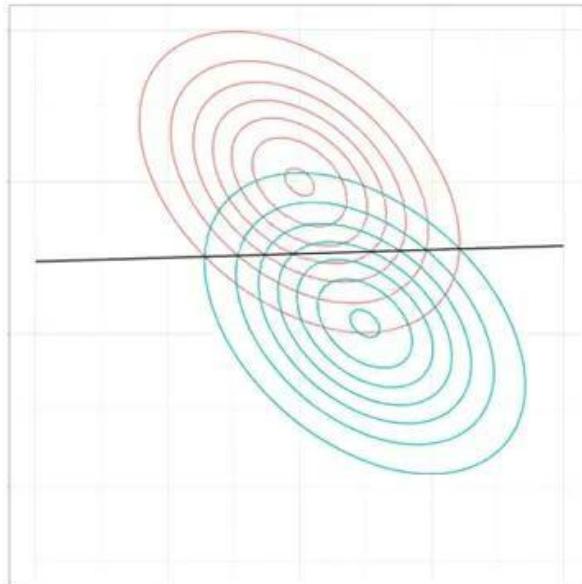
¿De qué clase serán **a**, **b** y **c**?

- 1) Modelar la distribución (normal) de puntos de cada clase;
- 2) Asignar nuevas instancias a la clase que tiene probabilidad máxima a posteriori (MAP).
- 3) La recta verde es la frontera de decisión entre las 2 clases (en este caso, es una recta, es decir LDA).  
**¿Por qué no está justo al medio?**

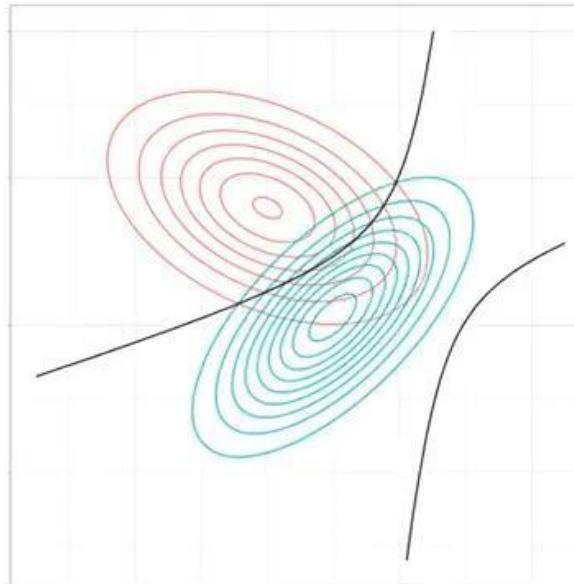
# Linear / Quadratic Discriminant Analysis

Método generativo

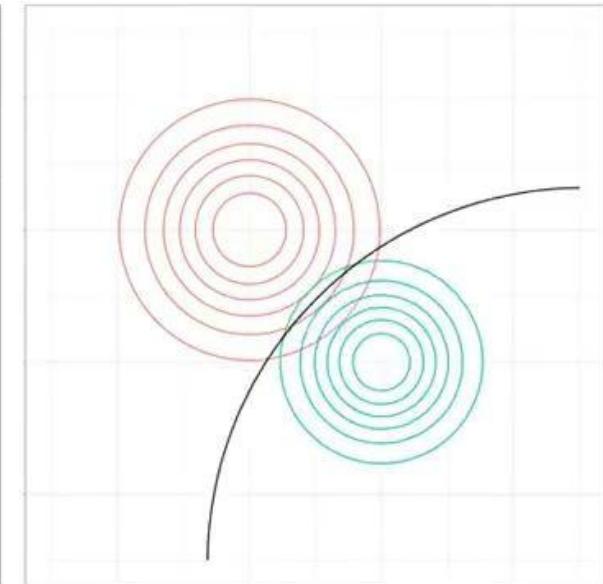
LDA variando la media de una clase



QDA variando la media de una clase



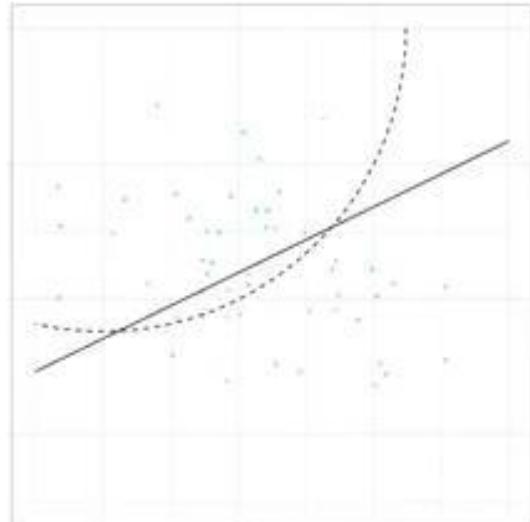
QDA variando la varianza de una clase



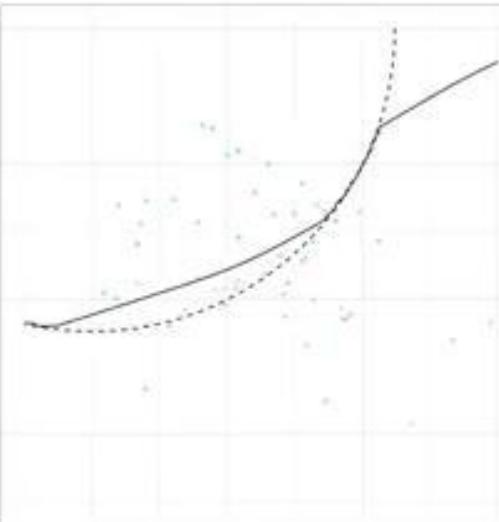
# Linear / Quadratic Discriminant Analysis

Método generativo

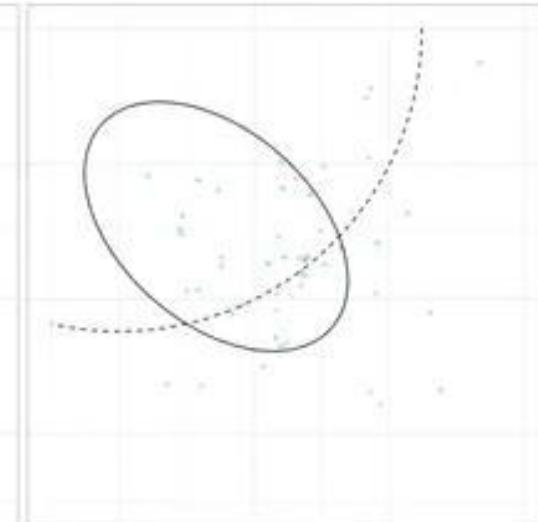
LDA



QDA (con mat. covarianza diagonal)



QDA general



La línea punteada representa la frontera de decisión óptima (Bayes Optimal Classifier)

Fuente: <https://mathformachines.com/posts/discriminant-analysis/>  
VER hilo de twitter (para más animaciones como esta): [Animated-machine-learning-classifiers](#)

modelos generativos

# Naive Bayes



# Naive Bayes (versión Gaussiana)

## Método generativo

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} P(Y=c|X=x^{(i)})$$

$$= \arg \max_{c \in Clases} \frac{P(Y=c)P(X=x^{(i)}|Y=c)}{P(X=x^{(i)})}$$

$$= \arg \max_{c \in Clases} P(Y=c)P(X=x^{(i)}|Y=c)$$

$$= \arg \max_{c \in Clases} P(Y=c)P(X_1 = x_1^{(i)} \wedge \dots \wedge X_p = x_p^{(i)}|Y=c)$$

+ sesgo inductivo  
**(suposición "Naive")**  
¿Cuál es?

Para X continua

+ sesgo inductivo  
**Gaussian Naive Bayes**

$$\approx \arg \max_{c \in Clases} P(Y=c) \prod_{j=1}^p P(X_i = x_j^{(i)}|Y=c)$$

$$= \arg \max_{c \in Clases} P(Y=c) \prod_{j=1}^p pdf_c(x_j^{(i)})$$

$$\approx \arg \max_{c \in Clases} \hat{P}(Y=c) \prod_{j=1}^p f_{norm}(x_j^{(i)}; \hat{\mu}_{c,j}, \hat{\sigma}_j))$$

Una media por clase, por atributo.

Un desvío por atributo (en general compartido entre clases)

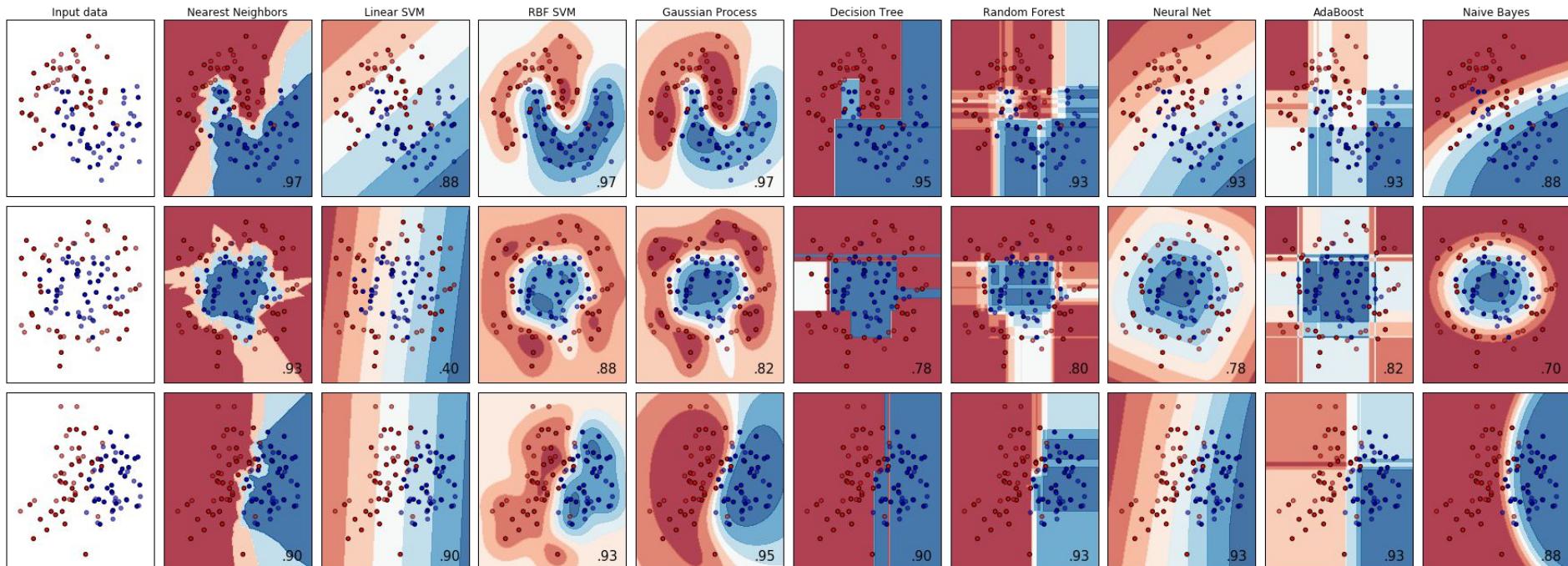
# Nota: Modelos paramétricos vs no paramétricos

(Murphy pp.16)

Tiene el modelo una **cantidad fija de parámetros (modelos paramétricos)** o esta cantidad **crece con la cantidad de datos** de entrenamiento (**modelos no paramétricos**)?

¿Cuáles suelen tener “más” sesgo inductivo?

# Siempre recordar...



Fuente: [https://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html](https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)

# Resumen

- **Ensambles**
- **Bayes Optimal Classifier**
- Modelos **discriminativos vs generativos**
- **K Vecinos más Cercanos (KNN)**
- **Support Vector Machines (SVM)**
  - (vimos muy poco, es todo un mundo este tema)
- **Linear Discriminant Analysis (LDA) / Quadratic Discriminant Analysis (QDA)**
- **Naive Bayes Classifier / Gaussian Naive Bayes**

Próximo tema: Métricas

(algunos) ejemplos de modelos discriminativos:

- **Árboles de decisión** (ya lo vimos)
- **K-vecinos más cercanos** (hoy)
- **Support Vector Machines (SVM)** (hoy)
- **Regresión logística** (prox)
- **Random Forest (y otros ensambles)** (hoy)
- **Maximum-entropy Markov models**
- **Conditional random fields**

(algunos) ejemplos de modelos generativos:

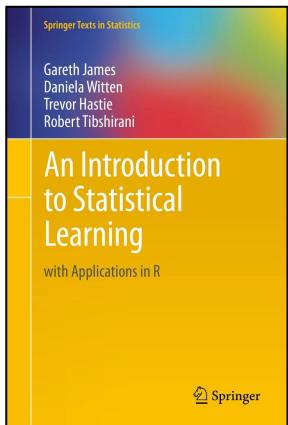
- **linear discriminant analysis (LDA)** (hoy)
- **gaussian/naive bayes** (hoy)
- **Gaussian mixture model (GMMs)** (prox)
- **Hidden Markov Models (HMMs)**
- **Generative adversarial networks (GANs)**,
- **Autoregressive models** (por ejemplo **GPT-3**)
- **Diffusion models**



# TAREA

- Leer **sección 8.2.1 Bagging y 8.2.2 Random Forest** del [ISLR](#)
- Obligatorio: Leer la **Sec. 4.4 Generative Models for Classification** del ISLR.
- Obligatorio: **Cap 9 hasta 9.4. Support Vector Machines** del ISLR.
- Opcional
  - Recomendado: Leer la **Sec. 4.5 A Comparison of Classification Methods** del ISLR.
  - **Capítulo 15 Random Forests** del [ESLI](#)
  - **Capítulo 10 Boosting and Additive Trees** del [ESLI](#)
  - Breiman, L. (2001). *Random forests*. [IPDF](http://www.springer.com) [springer.com](http://www.springer.com)

## ISLR



Gareth James



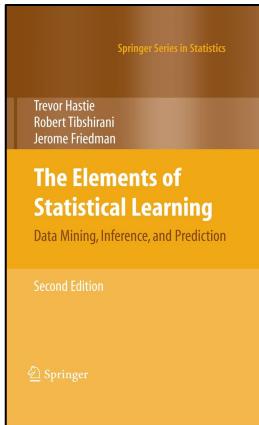
Trevor Hastie

Daniela Witten



Robert Tibshirani Jerome Friedman

## ESLI



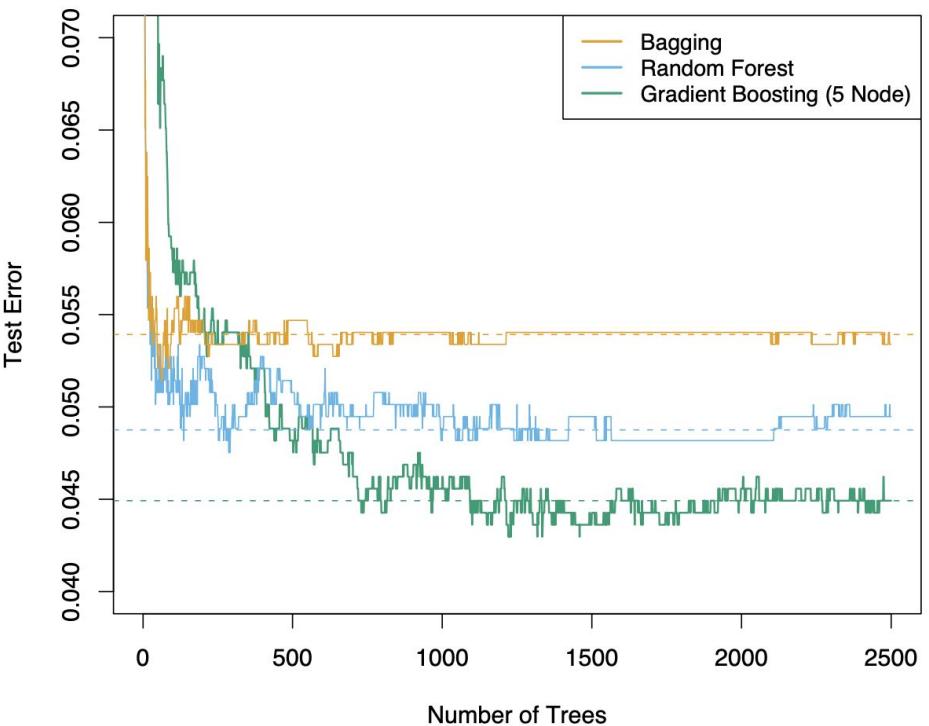
Las siguientes son material complementario  
que les puede servir o interesar  
(no dadas en clase)

# Random Forest - OOB estimation

## Spam Data

# Ensambles

## Random Forest



[Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). **The elements of statistical learning: data mining, inference, and prediction** (Vol. 2, pp. 1-758). New York: Springer.]

**FIGURE 15.1.** Bagging, random forest, and gradient boosting, applied to the spam data. For boosting, 5-node trees were used, and the number of trees were chosen by 10-fold cross-validation (2500 trees). Each “step” in the figure corresponds to a change in a single misclassification (in a test set of 1536).



# Ensambles

## OOB Error

¿Podremos aprovechar las características de **Bagging** para obtener estimaciones realistas de la performance sin hacer cross-validation?

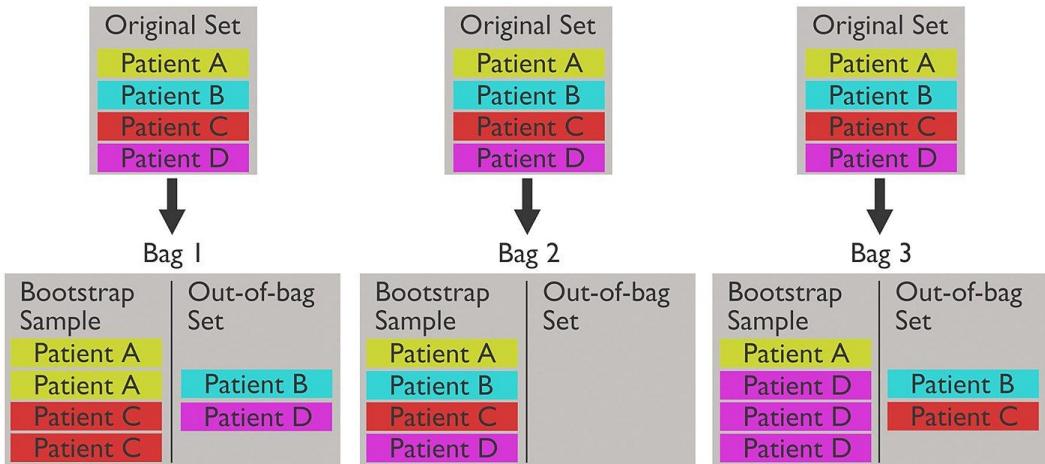
### Out-of-Bag Error:

Cómo técnica para obtener estimaciones de qué tan bien generalizan los modelos. Idea: para cada instancia, utilizar los árboles que no contienen a  $x^{(i)}$  en su conjunto de entrenamiento.

La ventaja del método OOB es que requiere menos cómputo y permite probar el modelo a medida que se entrena.

### No reemplaza a Cross Validation:

[Janitza, S., & Hornung, R. (2018). **On the overestimation of random forest's out-of-bag error.** PloS one, 13(8), e0201904.

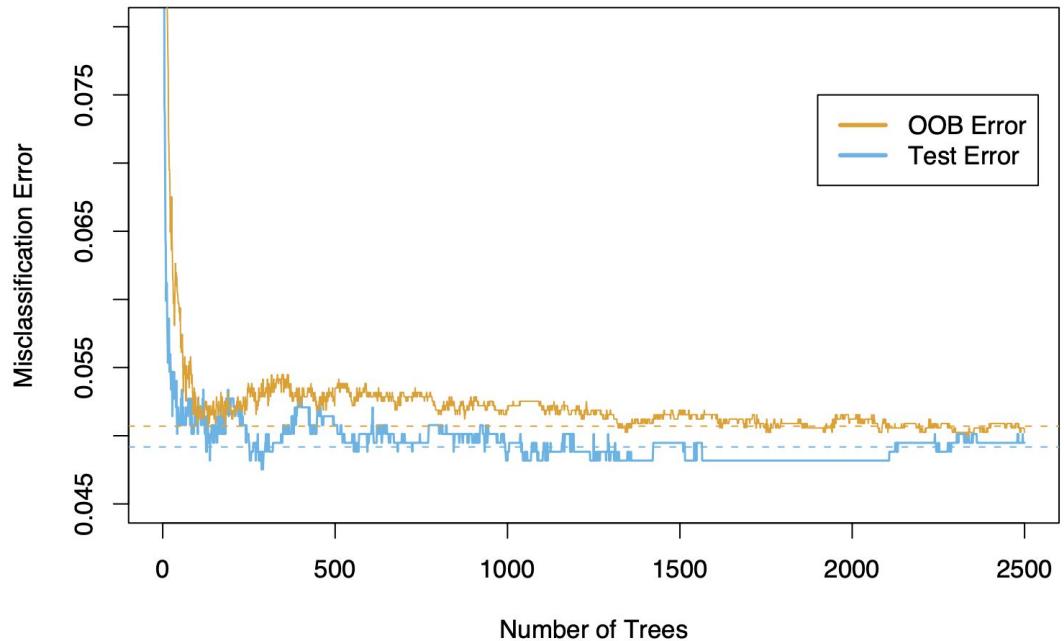


[https://en.wikipedia.org/wiki/Out-of-bag\\_error](https://en.wikipedia.org/wiki/Out-of-bag_error)



# Ensambles

## OOB Error



[Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). **The elements of statistical learning: data mining, inference, and prediction** (Vol. 2, pp. 1-758). New York: Springer.]

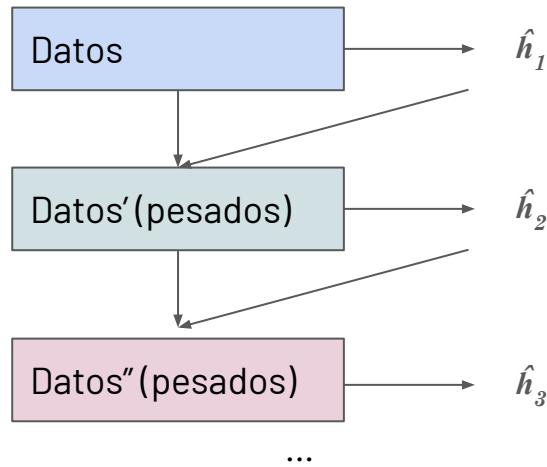
**FIGURE 15.4.** OOB error computed on the spam training data, compared to the test error computed on the test set.

# AdaBoost



# Ensambles

## Boosting: AdaBoost



$$\hat{h}(x) = \text{combinación pesada } \hat{h}_t$$

### AdaBoost [Freund 1997]

Predictores en secuencia (malo para la parallelización), de tal manera que el segundo ajuste bien lo que el primero no ajustó, que el tercero ajuste un poco mejor lo que el segundo no pudo ajustar y así sucesivamente.

[Yoav 1997] Freund, Yoav; Schapire, Robert E (1997). "A decision-theoretic generalization of on-line learning and an application to boosting". Journal of Computer and System Sciences.

# Ensambles

## Boosting: AdaBoost

### Algorithm 1: Algoritmo AdaBoost

**Data:**  $(x^{(1)}; y^{(1)}), \dots, (x^{(n)}; y^{(n)})$ ,  $x^{(i)} \in X, y^{(i)} \in Y = \{-1, 1\}$

**Iniciar:**  $D_1[i] = \frac{1}{n}$  para  $i = 0..n$

**for**  $t = 1$  to  $T$  **do**

1) Entrenar un clasificador  $h_t : X \rightarrow \{-1, 1\}$  débil tomando en cuenta los pesos  $D_t$ ;

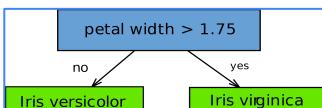
2) Computar el error ponderado  $\epsilon_t = \sum_{i:h_t(x^{(i)}) \neq y^{(i)}} D_t[i]$ ;

3) Elegir el ponderador  $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ ;

4) Actualizar  $D_{t+1}[i] = \frac{D_t[i]}{Z_t} e^{(-\alpha_t y^{(i)} h_t(x^{(i)}))}$  para todo  $i$ , donde  $Z_t$  es una constante de normalización que logra que  $\text{sum}(D_{t+1}) = 1$ ;

**Output:**  $h(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$

Notar que este algoritmo es una meta heurística. Los  $h$  pueden ser árboles, LDA, SVM, etc). Se suelen utilizar árboles "decision stump"



Vector de pesos para cada observación

Sumamos sólo los pesos de las instancias mal clasificadas

Si el error es grande,  $\alpha$  chico.

El nuevo peso será grande si el predictor anterior fue muy bueno en general pero erró para esa instancia



# Más sobre SVM

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

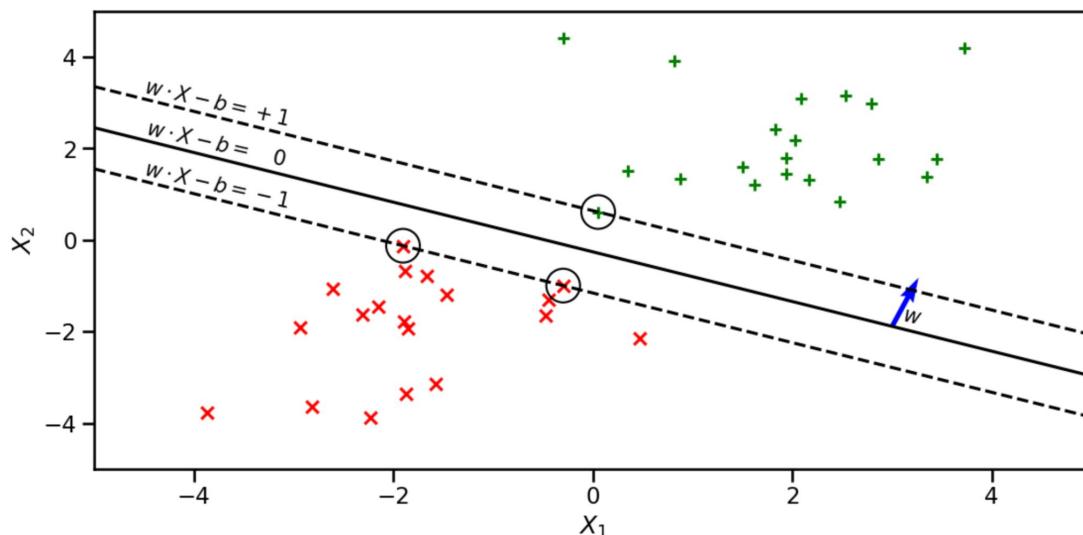
# Support Vector Machine (SVM)

Demo: el tamaño del margen es  $2 / \|\mathbf{w}\|_2$

Vector unidad en dirección de  $\mathbf{w}$

Sea  $\mathbf{p}$  un vector sobre la frontera de decisión.  $\mathbf{p}$  tiene que cumplir  $\mathbf{w}^* \mathbf{p} - b = 0$ .

Busquemos  $\mathbf{k}$ , cuánto mover en la dirección de  $\mathbf{w}$  para llegar hasta el límite del margen?



$$\begin{aligned}
 & \mathbf{w} \cdot \left( \mathbf{p} + k \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) - b = 1 \\
 \iff & \mathbf{w} \cdot \mathbf{p} + \mathbf{w} \cdot k \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} - b = 1 \\
 \iff & \mathbf{w} \cdot k \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = 1 \\
 \iff & k \cdot \frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = 1 \\
 \iff & k = \frac{1}{\|\mathbf{w}\|}
 \end{aligned}$$

El margen es  $2k = 2 / \|\mathbf{w}\|$

# Más sobre KNN

$$Pred(x^{(i)}) = \arg \max_{c \in Clases} \hat{P}(Y=c|X=x^{(i)})$$

# K vecinos más cercanos (KNN)

## Método discriminativo

¿Cómo se implementa? Algoritmo de asignación a la instancia  $x^{(i)}$ :

1. Computar la distancia  $D(x^{(i)}, x^{(j)})$  para todo punto de entrenamiento  $x^{(j)}$ .
2. Seleccionar las  $k$  instancias más cercanas y sus etiquetas.
3. Devolver la etiqueta más frecuente (o la proporción si es proba).

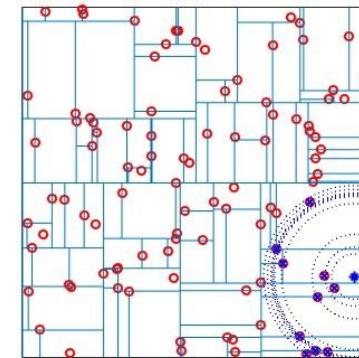
Costo computacional:  $O(n^*d + k^*n)$  tiempo       $O(n)$  memoria

(hay otras opciones dependiendo de decisiones de implementación).

Ver <https://stats.stackexchange.com/questions/219655/k-nn-computational-complexity>

¿Se puede mejorar el costo por query?

Ver estructuras de datos eficientes: “**BallTree**” y “**KDTree**”.



Fuente: <https://www.vlfeat.org/overview/kdtree.html>