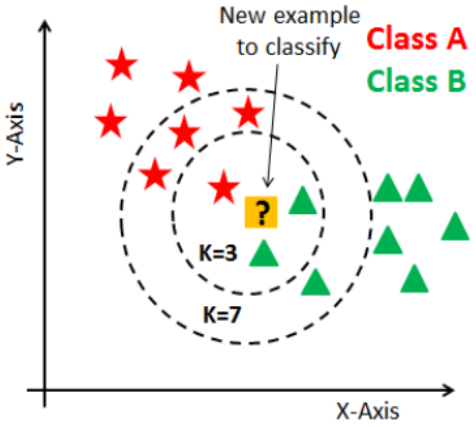
Algoritmos de clasificación supervisada

KNN: k nearest neighbours

¿Cómo funciona?
Elegimos un K y una medida de distancia y cada punto nuevo va a ser clasificado como la clase mayoritaria de los K vecinos. Hay que normalizar, la escala afecta



Devuelve la clase más frecuente entre los k vecinos más cercanos.

Si K es muy grande da la clase mayoritaria, si K es muy chiquito te da, del actual, el dato más cercano.
K = 1 -> overfit
K = n -> underfit

Diagrama de Voronoi

Colorea el área cercana a los puntos para mostrar un área de influencia a cada punto.

¿Cómo mejoramos el modelo?

No todos los puntos tienen la misma importancia, por eso asignamos una importancia:

$$\text{importancia de un } \textbf{vecino} = \frac{1}{\text{distancia}(\textbf{vecino}, \text{ nueva instancia})}$$

Hay distintas distancias que pueden usarse:
- euclidiana
- coseno: el ángulo del coseno de dos vectores.
- etc.

A parte de la clase mayoritaria, podemos devolver las probabilidades:

$$\text{Probabilidad de la Clase} = \frac{\text{Veces que aparece esa clase en los vecinos mas cercanos}}{K}$$

Funciona para cualquier cantidad de clases.

(En el árbol de decisión también se puede devolver la probabilidad a partir de la cantidad de cada clase en las hojas)

Si queremos plotear la probabilidad de que si caigo en algún lado, de qué clase va a ser se grafica la probabilidad de cada clase ($P(y = 1|x,D)$, sería la probabilidad de la clase 1). Ploteamos para que quede entendible cuando tenemos más de una clase.

KNN es muy susceptible a la dimensionalidad

Para solucionar el problema de la alta dimensionalidad, podemos usar una técnica de reducción de dimensionalidad. A veces funciona, a veces no pero se puede intentar.

Ventajas:

- Es una técnica muy simple
- No hay entrenamiento (es casi inexistente). Tenemos que tener el dataset en memoria (lo cual trae un problema para la ram).

Desventajas:

- La consulta es muy lenta, el algoritmo es O(n). Porque compara cada punto.
- El modelo ocupa mucho espacio en disco.
- Es muy susceptible a la dimensionalidad

La distancia se calcula con todos los atributos, entonces…

¿Qué pasa si tengo un feature irrelevante? (un feature que es ruido).
Me puede joder el calculo de los k vecinos. Este feature va a influir en la distancia y no va a tener sentido. Hay que tener cuidado con qué features se le pasa, sino vamos a tener mala performance. Lo mejor es probar variar los features para ver si sacarlos mejora o no la predicción.

¿Qué pasa si están a escalas muy distintas?
Afecta a la distancia si las escalas de cada columna son distintas.

Lo que se puede hacer es pedir que cada columna vaya del cero al uno, o también se utiliza hacer que toda la fila tenga norma uno (se usa este último para la distancia coseno).

Naive Bayes

Definiciones:

- P(A): básicamente se devuelve un número del cero al uno, indicando la probabilidad de que pase.

- P(A|B): probabilidad condicional. Se puede agregar condiciones de que algo pase, se condiciona la probabilidad.

- **P(A|B) = P(B|A).P(A) / P(B)**

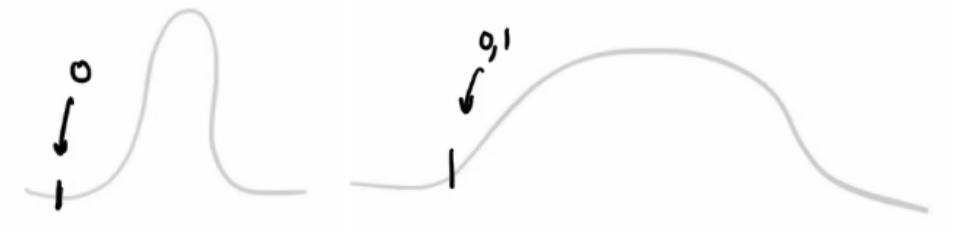
P(CLASE| “algunas”, “palabras”): predigo si es o no spam dado que hay ciertas palabras.

Es simple de implementar.

¿Qué pasa si vemos un caso que no se dio en el entrenamiento?
P(“euro”|SPAM) = 0 porque nunca vio la palabra “euro” en un mail de spam, entonces me va a dar cero la probabilidad de que sea spam. Para hacerlo zafar, se le puede asignar un valor muy chico de probabilidad.

¿Qué pasa si mis variables son continuas?

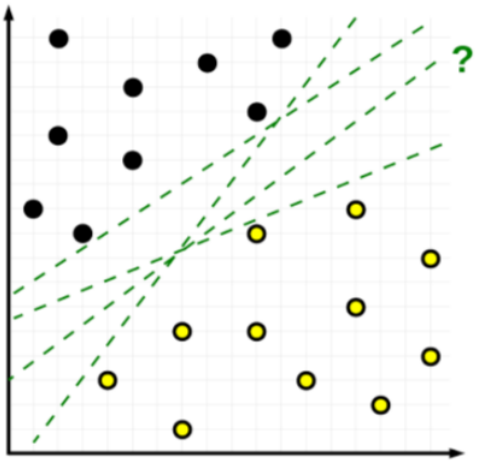
Por cada feature continuo se calcula la varianza y la media (para la **campana** de gauss). Queremos una campana ancha para que no haya valores con 0 que cancelen.



Naive: **ingenuo** porque asume que estadísticamente pasan cosas que en la vida real no son así.

SVM: support vector machines

La idea es buscar un hiperplano que separe las clases lo mejor posible, sin tener que modelar la distribución de los datos de la clase.

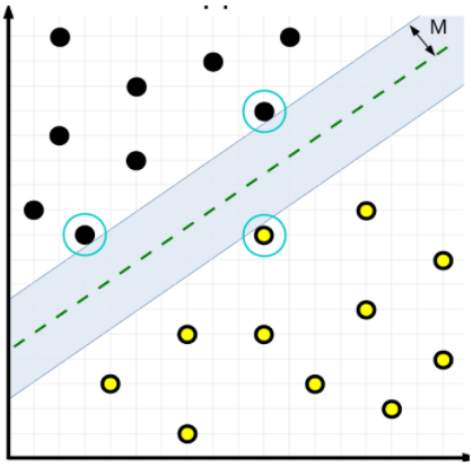


Quiero dividir los datos con un hiperplano, para que me queden los datos de una clase de un lado y los de la otra separado también.

Al modelo le interesa encontrar el **W**:

- **W**.X + b > 0: para la clase A.
- **W**.X + b < 0: para la clase B.

Este W logra dividir todos los puntos. Busca maximizar la distancia entre las clases



Margen M: Distancia de las instancias más cercanas a la recta (hiperplano) de decisión.

Los puntos que se chocan con el margen son los support vectors.

Fijarse como la recta está justo en el medio

La recta divide en 2 todos los puntos.

Este modelo es un problema de optimización. **Es buscar un W que me optimice el M.**

Si no hay una recta que separe los puntos, el problema no es linealmente separable.

Si las instancias no son linealmente separables, se puede modificar la dimensión y ver si en esa nueva dimensión se puede hallar un hiperplano pero cuesta mucho computacionalmente, entonces se hace lo siguiente:

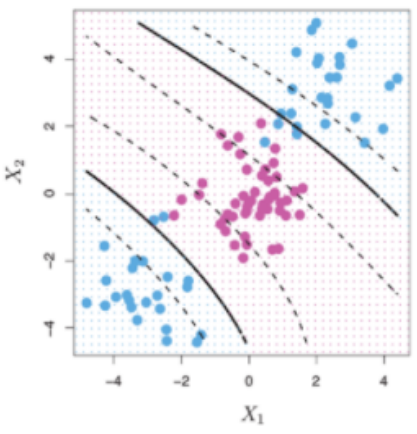
<u>Kernel trick</u>

Lo que se hace es engañar a los puntos, al algoritmo: **cambia la función del producto interno** entre dos vectores, por distintas definiciones de productos. Engañamos a los puntos diciendoles que están en otra dimensión.

Ejemplos de kernels (cada kernel tiene sus hiperparametros):

- Polinomial:

  K(a, b) = (a . b + r) ^ d
  con r y d parametros elegibles



- Radial kernel:

  $$K(a, b) = e^{-\alpha (a-b)^2}$$

- Sigmoid:

  $$K(a, b) = \tanh(\alpha.a.b + c)$$

Márgenes duros y márgenes suaves

Modificando el paramétro **C** podemos pasar de pedirle al SVM una recta que divida perfectamente los puntos a un hiperplano a que me divida bastante bien los puntos y que zafe en algún punto de ruido.

Hard Margin (C → infinito): SVM pide una recta que separe los puntos **perfectamente**. Estoo no funciona con outliers y ruido.

Soft Margin (C → 0): permite cierto nivel de ruido/outliers permitiendo que SVM tenga **miss-classifications**.

Clases múltiples

SVM divide entre dos, y lo que se hace para zafar de esa limitación es un one versus all:

Esto es, entrenamos un SVM para entrenar la clase A, y el resto, después otro para la clase B y el resto. Así para todas las clases. Después lo que hacemos es juntar todas y hacer una votación. Cuando llega una nueva instancia, se corren los n SVM y se retorna la clase con mayor margen o mayor confianza.

<u>Comentarios</u>

- SVM es **iterativo**.
- Como está bien planteado matemáticamente, lleva a la mejor solución matemáticamente planteada.
- Parámetros de stop:
  - Cantidad de iteraciones.
  - Delta de mejora entre iteraciones, si deja de mejorar, corto.