ACÁMICA

TEMA DEL DÍA

Support Vector Machine

Un modelo avanzado, con un trasfondo matemático interesante. No hay que dejarse intimidar por las cuentas.

Agenda

Daily

Kernels y Extensión a más de dos clases

Break

Hands-on training

Cierre

Daily

Daily

Sincronizando...

Bitácora

¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Challenge

¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Repaso de la bitácora
Support Vector Machine

Representación de los datos

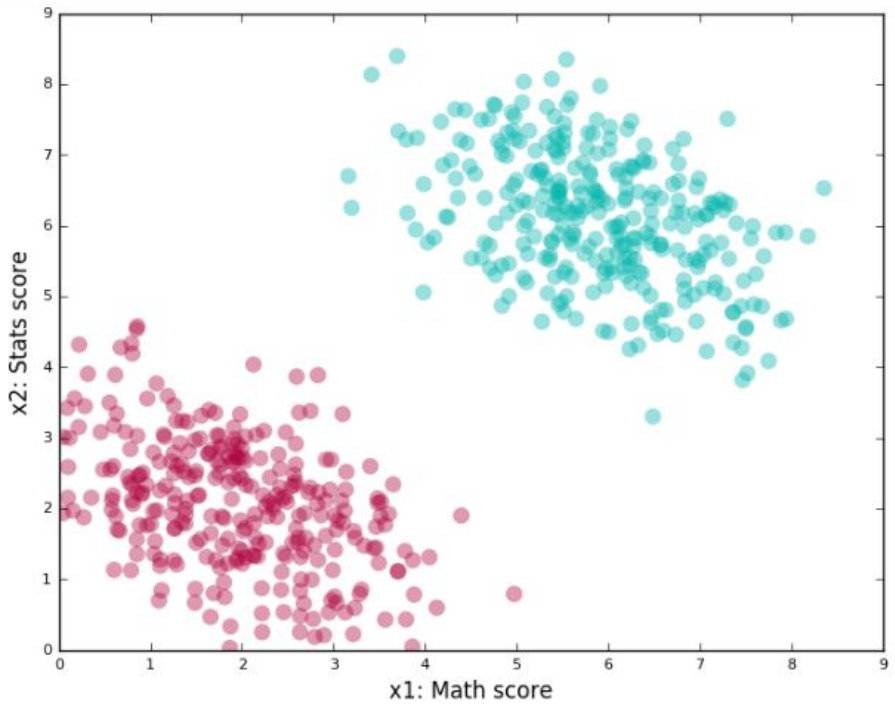
Un estudiante con ciertas puntuaciones se muestra como un punto en el gráfico.
El color del punto (verde o rojo) representa cómo le fue en el curso de ML:
Bueno o **Malo** respectivamente.

Problema:

Separar los puntos verdes de los puntos rojos.

La idea más sencilla:

Utilizar una recta.



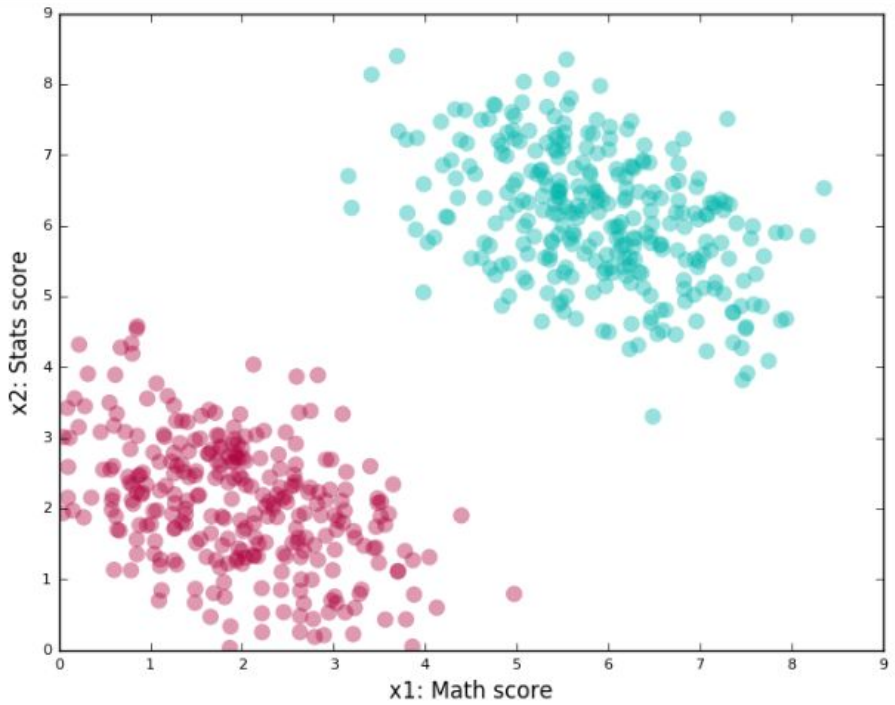
Representación de los datos

Un estudiante con ciertas puntuaciones se muestra como un punto en el gráfico.
El color del punto (verde o rojo) representa cómo le fue en el curso de ML:
Bueno o **Malo** respectivamente.

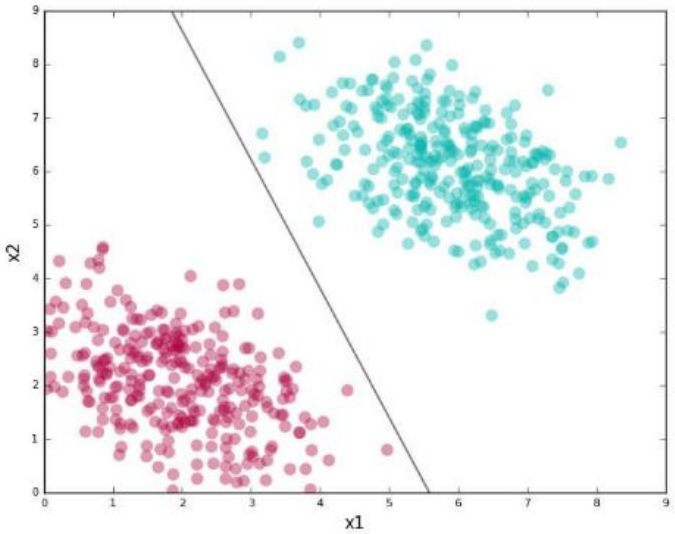
Problema:
Separar los puntos verdes de los puntos rojos.

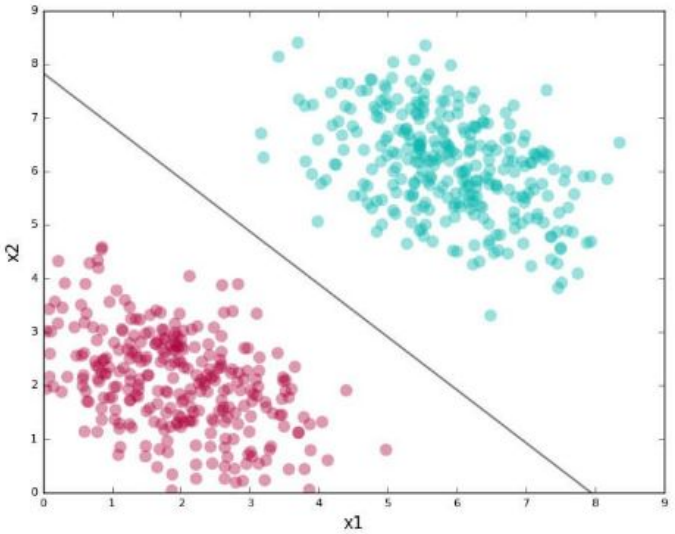
La idea más sencilla:
Utilizar una recta.

¿Y cuál es la recta que “mejor” separa mis datos?




THAT IS AN EXCELLENT QUESTION





Ambas rectas separan los puntos rojos y verdes.

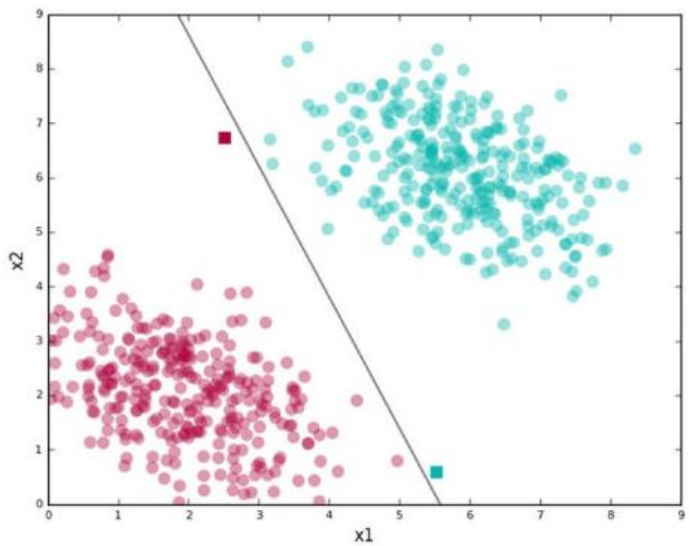
¿Existe una buena razón para elegir una u otra? ¿Son éstas las únicas rectas posibles? ¿Hay vida después de la muerte?

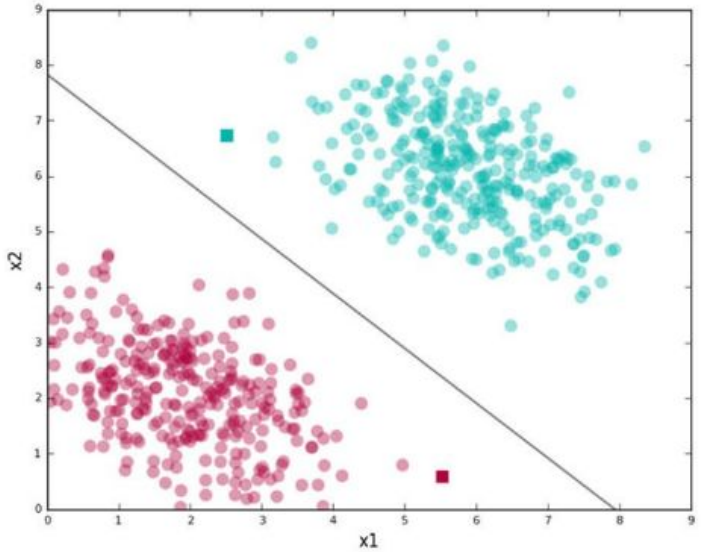
Clasificadores Buenos vs. Malos

El valor de un clasificador no radica en lo bien que separa los datos de entrenamiento. Eventualmente **queremos que clasifique datos nuevos no vistos.**
Dado esto, queremos elegir una línea que capture el patrón general en los datos de entrenamiento, así que hay una buena posibilidad de que le vaya bien en los datos de prueba.

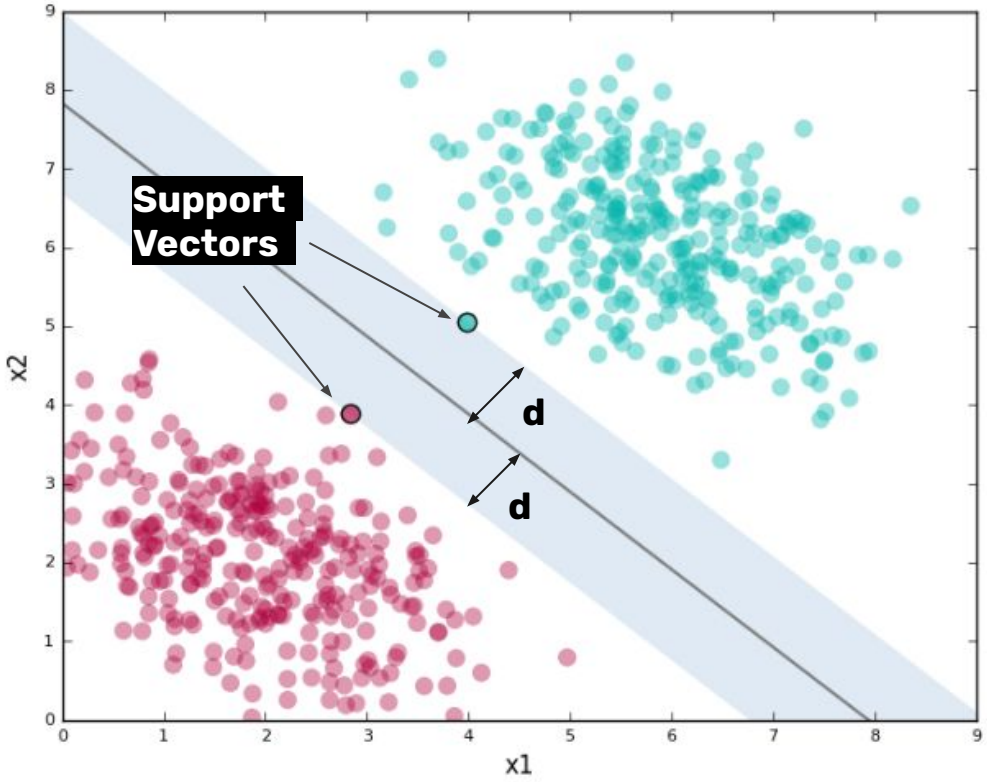
Clasificadores Buenos vs. Malos

¿Y entonces, qué nos dicen estas rectas?





Clasificadores Buenos vs. Malos • SVMs



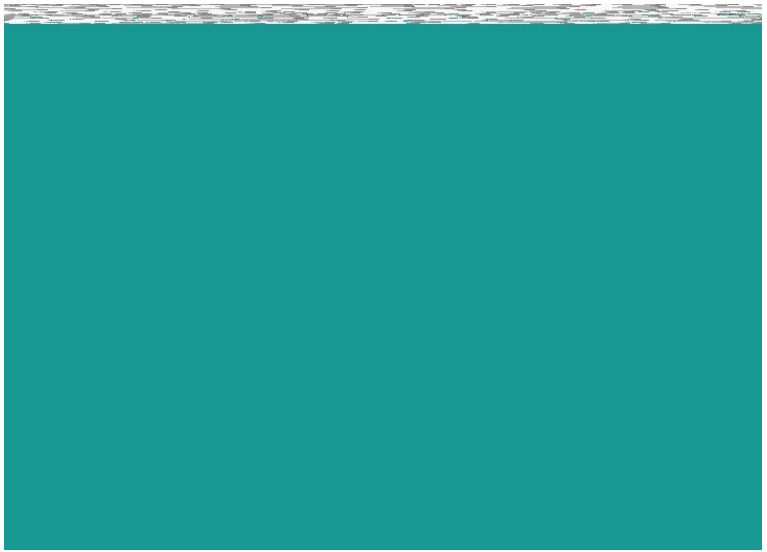
Necesitamos entonces:

- Buscar rectas que clasifiquen correctamente los datos de entrenamiento.
- Entre todas estas rectas, elegir la que tenga la mayor distancia **d**, a los puntos más cercanos a ella.

Los puntos más cercanos que identifican esta recta se conocen como vectores de apoyo (**Support Vectors**). Y la región que definen alrededor de la línea se conoce como el **Margen**.

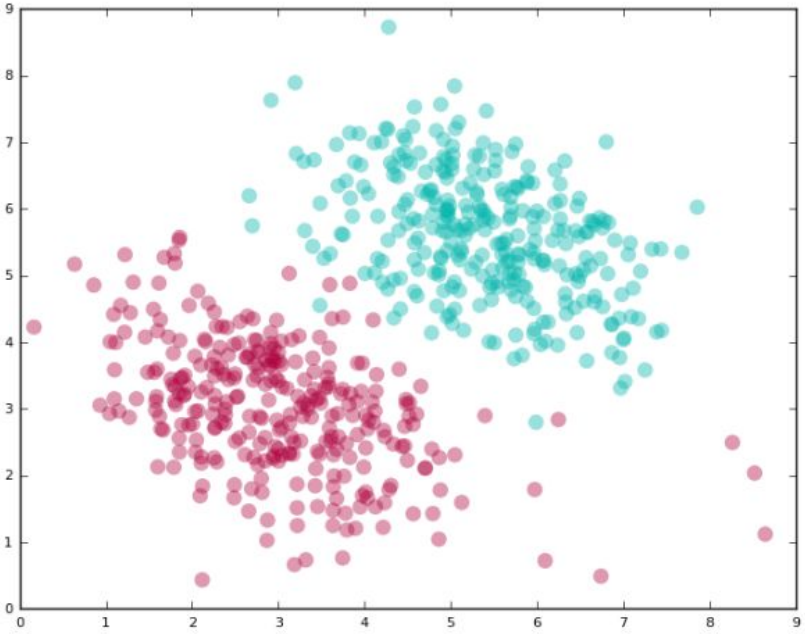
¿Solo funciona en 2D?

¿Solo funciona para 2D?



IMPORTANTE: Los datos que pueden ser separados por una recta (o en general, un hiperplano) se conocen como **datos linealmente separables**. El hiperplano actúa como un clasificador lineal.

Permitiendo errores (Soft Margin)



- Los datos del mundo real son casi siempre muy complicados.
- Si utilizamos un clasificador lineal, pocas veces podremos separar perfectamente las etiquetas.
- Tampoco queremos descartar por completo el clasificador lineal, ya que parece adecuado para el problema, excepto por algunos puntos erróneos.

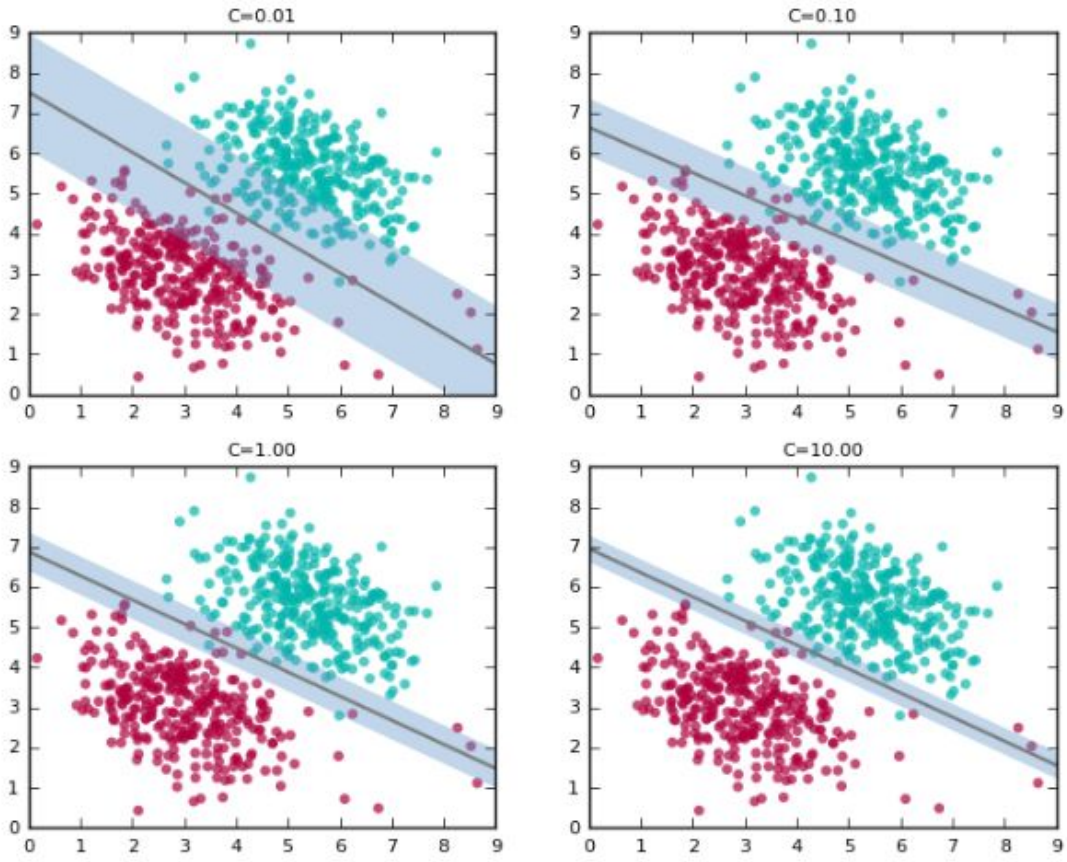
Permitiendo errores (Soft Margin)

¿Cómo se enfrentan los SVMs a esto?

Nos permiten especificar cuántos errores estamos dispuestos a aceptar mediante un parámetro llamado **C**, lo que permite dictaminar la relación entre:

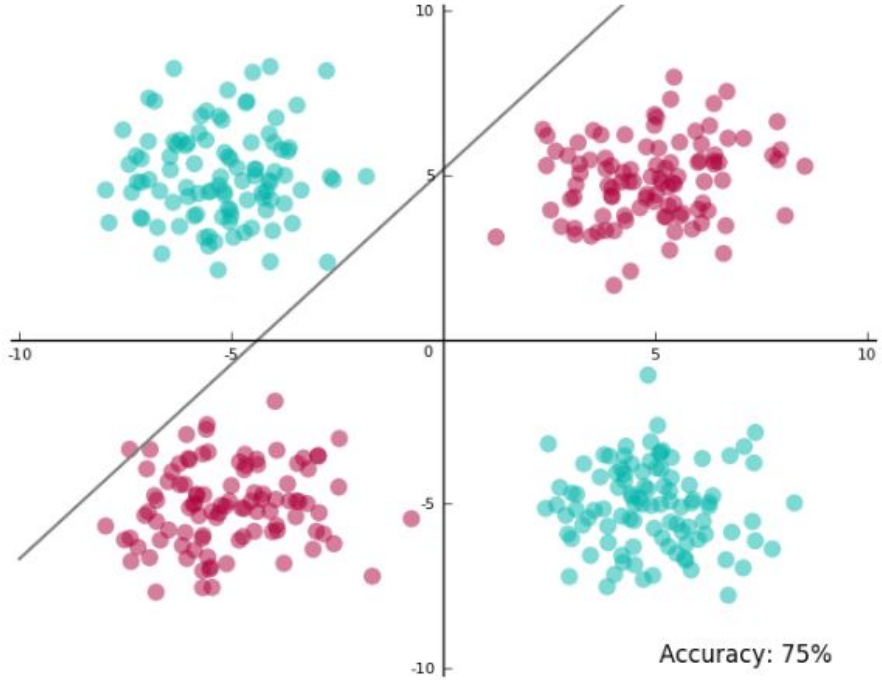
- Tener un amplio margen.
- Clasificar correctamente la mayor cantidad de puntos de entrenamiento (un valor más alto de C implica que queremos menos errores en los datos de entrenamiento).

Las siguientes gráficas muestran cómo varían el clasificador y el margen a medida que aumentamos el valor de C (vectores de apoyo no mostrados).



¿Y si no podemos “separar” nuestros datos con alguna recta, plano o hiperplano?

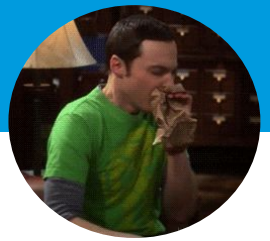
Datos linealmente no separables

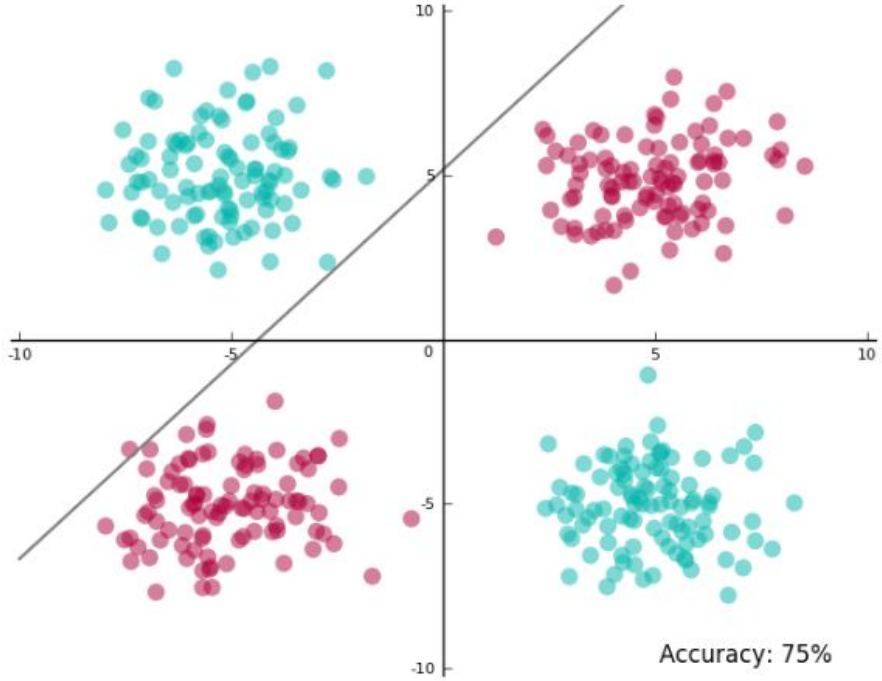


En esos casos es posible **aumentar la dimensionalidad de nuestros datos,** agregando nuevas dimensiones que permitan aplicar clasificadores lineales.

¿Algún problema con esto que acabamos de plantear?

Datos no separables linealmente



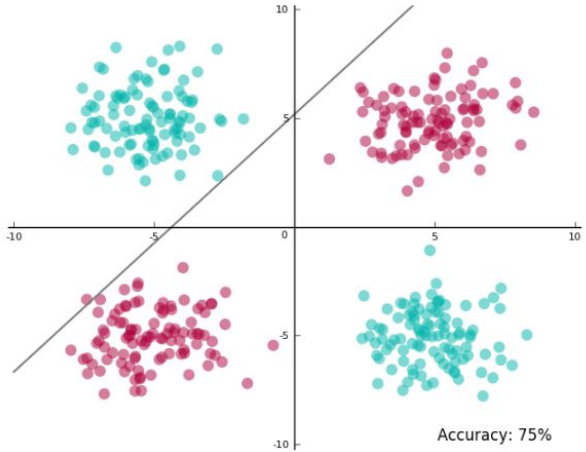


¡Sólo tenemos un 75% de precisión en los datos de entrenamiento, lo mejor posible con una recta!

¿Qué hacemos?

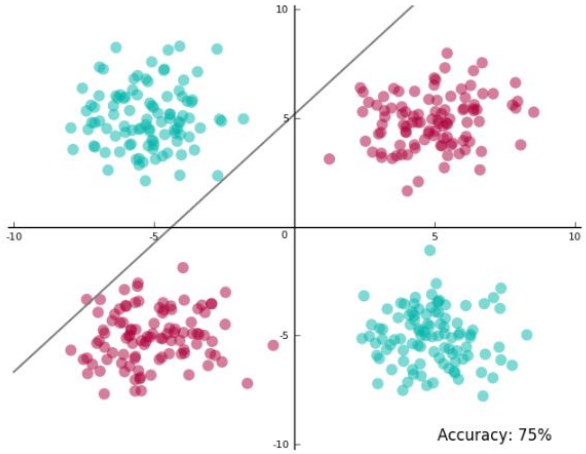
Comenzamos con el conjunto de datos de la figura anterior y lo proyectamos a un espacio tridimensional con las siguientes nuevas coordenadas:

$$X_1 = x_1^2$$
$$X_2 = x_2^2$$
$$X_3 = \sqrt{2}x_1x_2$$

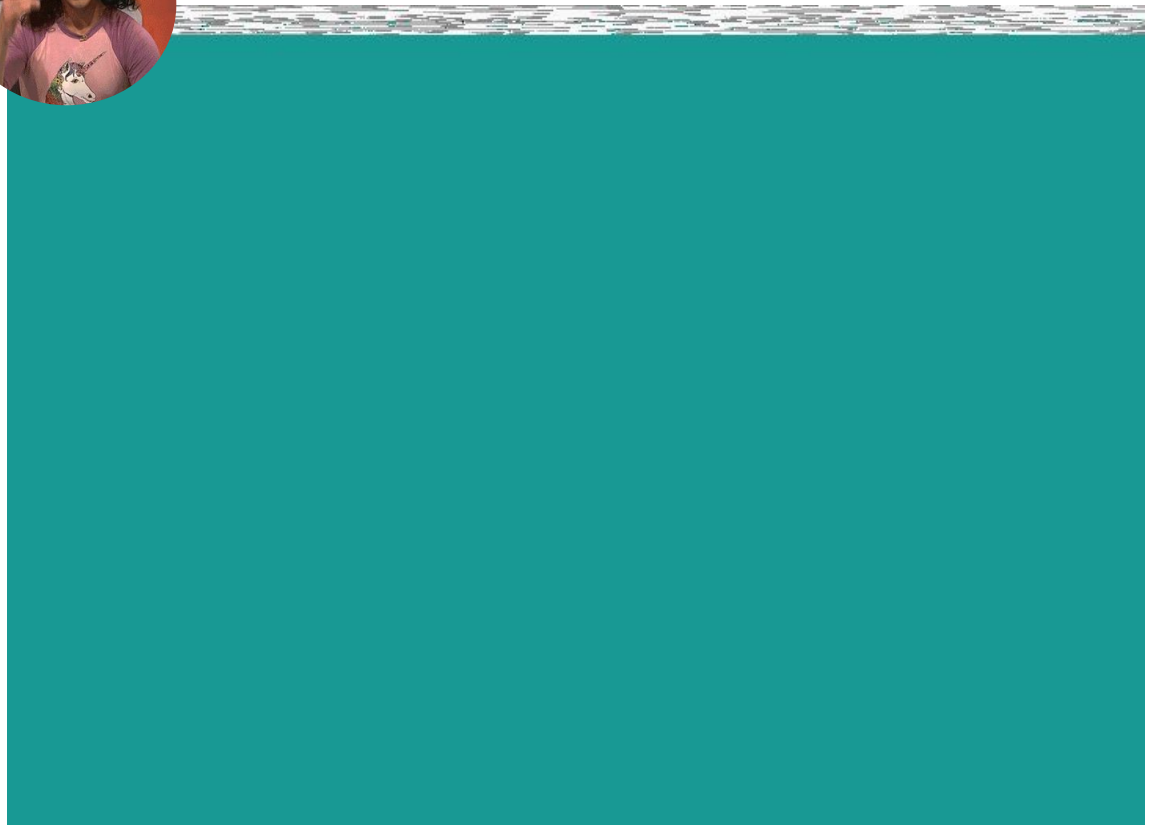


Comenzamos con el conjunto de datos de la figura anterior y lo proyectamos a un espacio tridimensional con las siguientes nuevas coordenadas:

$$X_1 = x_1^2$$
$$X_2 = x_2^2$$
$$X_3 = \sqrt{2}x_1x_2$$



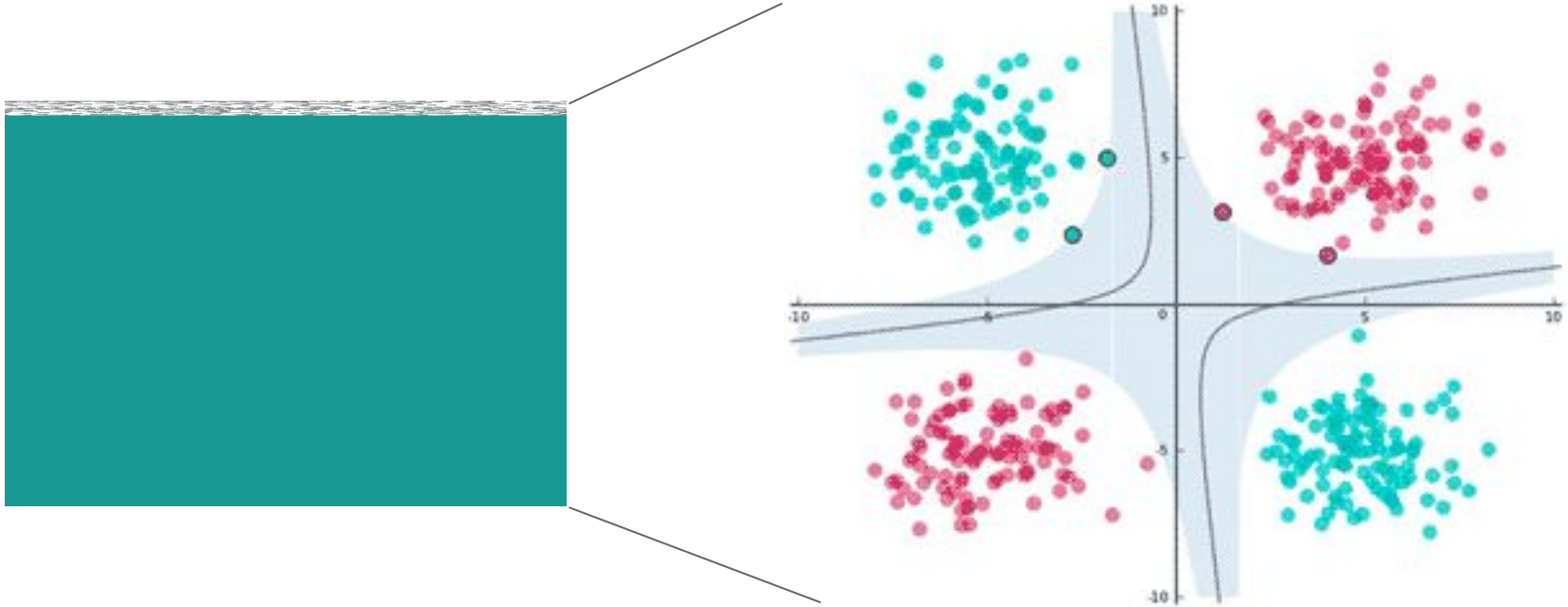
Así es como se ven los datos proyectados. Ahora SÍ podemos separar los datos con un plano!



¡Bingo!
Tenemos una perfecta separación de etiquetas!

Proyectemos el plano de vuelta al espacio bidimensional original y veamos cómo se ve el límite de separación

- La forma del límite de separación en el espacio original depende de la proyección. **En el espacio proyectado, esto es siempre un hiperplano.**
- Cuando se mapea de vuelta al espacio original, el límite de separación ya no es una recta. Esto también es cierto para los vectores de margen y de soporte. En cuanto a nuestra intuición visual, tienen sentido en el espacio proyectado.



¿Y cómo sabemos a dónde proyectar nuestros datos?

Kernels

Kernels: La salsa secreta de las SVMs

Hagamos un resumen de lo que hemos visto hasta ahora:

- Para datos linealmente separables, las SVMs trabajan increíblemente bien.
- Para los datos que son casi linealmente separables, las SVMs pueden funcionar bastante bien usando el valor correcto de C.
- Para los **datos que no son linealmente separables, podemos proyectar los datos** a un espacio en el que sean perfectamente/casi linealmente separables, lo que reduce el problema a 1 ó 2.

Kernels: La salsa secreta de las SVMs

Parece que una gran parte de lo que hace que las SVMs sean universalmente aplicables es proyectarlos a dimensiones superiores. Y aquí es donde entran los **kernels**.

¿Qué es un kernel?

Un Kernel es una forma de calcular el producto punto de dos vectores x y y en algún espacio de características (posiblemente de mayor dimensionalidad), por lo que las funciones del kernel a veces se denominan "producto punto generalizado".

Supongamos que tenemos un mapeo $\Phi : \mathbb{R}_n \rightarrow \mathbb{R}_m$ que trae nuestros vectores en $\mathbb{R}_n$ a algún espacio de características $\mathbb{R}_m$ Entonces el producto punto de x y y en este espacio es $\Phi(\mathrm{x})^T \Phi(\mathrm{y})$. Un kernel es una función K que corresponde a este producto punto, es decir, $K(\mathrm{x}, \mathrm{y}) = \Phi(\mathrm{x})^T \Phi(\mathrm{y})$.

Ejemplo: Definimos un mapeo polinómico a una espacio 3D

$$\Phi(\mathbf{X}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix}$$

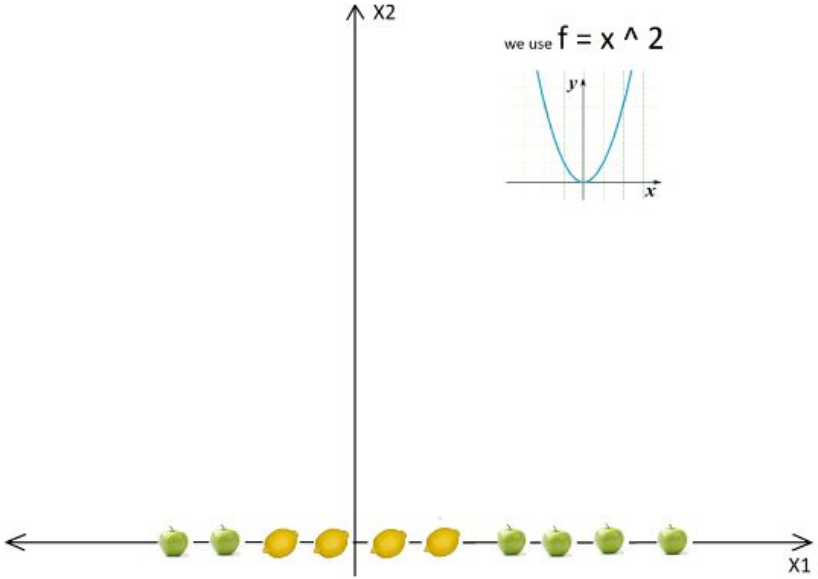
Entonces, la función Kernel asociada es

$$K(\mathrm{x}, \mathrm{y}) = \langle \Phi(\mathrm{x}), \Phi(\mathrm{y}) \rangle = \Phi(\mathrm{x})^T \Phi(\mathrm{y}) = x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 y_1 x_2 y_2$$

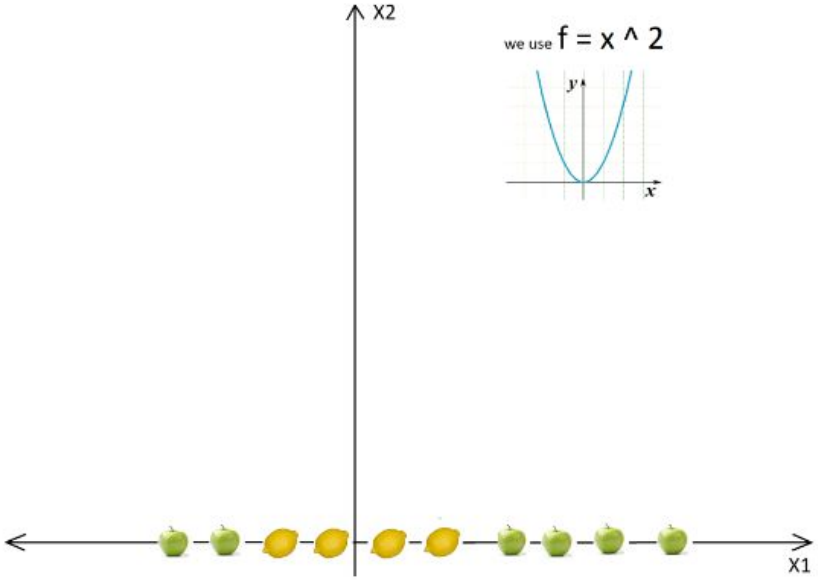
La función Kernel asociada es

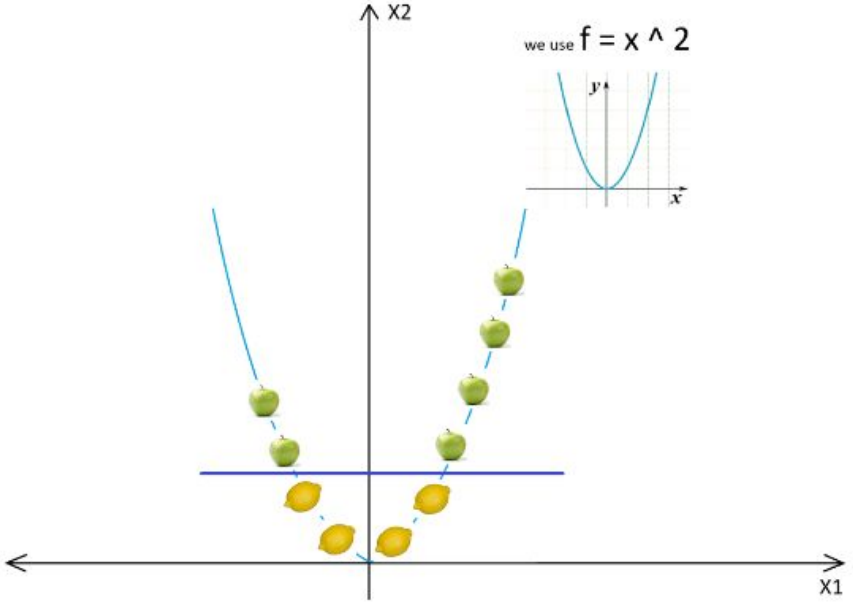
$$K(\mathrm{x}, \mathrm{y}) = \langle \Phi(\mathrm{x}), \Phi(\mathrm{y}) \rangle = \Phi(\mathrm{x})^T \Phi(\mathrm{y}) = x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 y_1 x_2 y_2$$


Dr.



we use $f = x \wedge 2$


X2
we use f = x ^ 2
y
x
X1


X2
we use f = x ^ 2
y
x
X1

Kernels típicos

Normalmente no definimos una proyección específica para nuestros datos. En su lugar, seleccionamos entre kernels disponibles, ajustándolos en algunos casos, para encontrar el que mejor se adapte a nuestros datos.
Por supuesto, nada nos impide definir nuestros propios kernels, o realizar la proyección nosotros mismos, pero en muchos casos no es necesario.

Kernels típicos

- **Lineal**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

- **Polinómico** (parámetros p y c)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T \mathbf{x}_j + c\right)^p$$

- **Gaussiano** (parámetro σ^2)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Podemos crear nuevos kernel mediante transformaciones
  1. $k_1(\mathbf{x}, \mathbf{y}) + k_2(\mathbf{x}, \mathbf{y})$
  2. $k_1(\mathbf{x}, \mathbf{y}) k_2(\mathbf{x}, \mathbf{y})$
  3. $\exp(k_1(\mathbf{x}, \mathbf{y}))$

SVMs y Python

sklearn.svm.SVC

class sklearn.svm. **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

Examples

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.svm import SVC
>>> clf = SVC(gamma='auto')
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

¡A leer la documentación!

REPASO

CHALLENGE BITÁCORA

¡Muéstranos qué hiciste!

¿Qué cosas te costaron más del ejercicio? ¿Cómo las resolviste?

¿Cuál el principal aprendizaje que te llevas?

Si tuvieras que hacerle alguna recomendación a alguien que va a hacer el ejercicio por primera vez, ¿qué le dirías?

REPASO

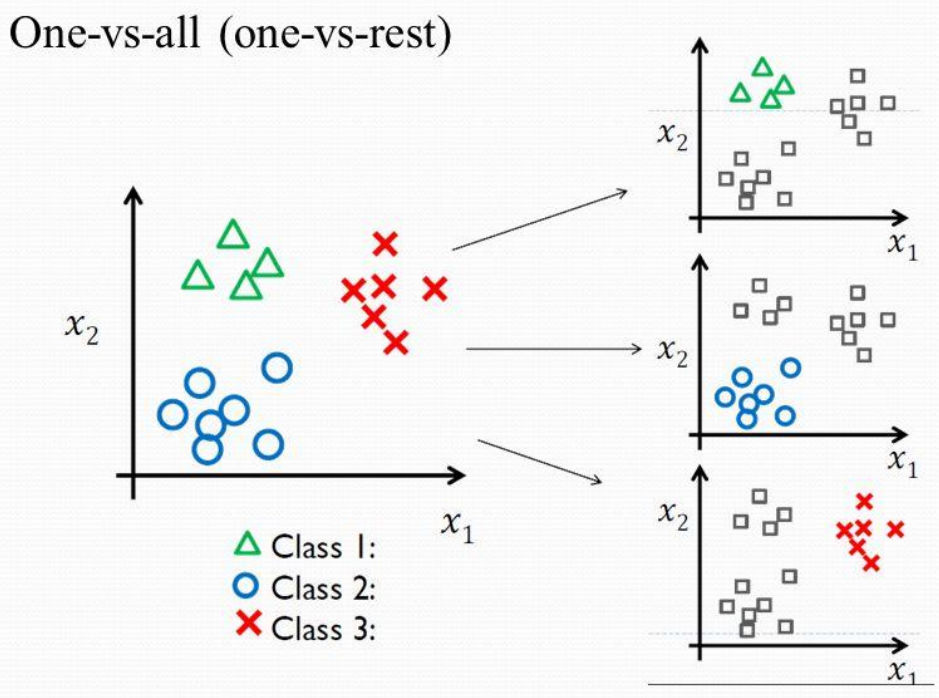
CHALLENGE BITÁCORA

¿Alguien hizo algo diferente que quiera mostrar?

SVM para múltiples clases

SVM para problemas multiclase: One vs All

- En un problema con K clases, resolvemos K problemas binarios.
- Cada SVM está entrenada para separar una clase del resto de los patrones.
- Para una nueva instancia **x**, se corren los K clasificadores y se retorna la clase que tenga una función de decisión con el valor más alto (la clase con mayor confianza).



SVM • Resumen

SVM es un algoritmo de aprendizaje supervisado que se propone encontrar el hiperplano que mejor separe los datos, tal que se maximice el margen.

Hiperparámetros: C y Kernels.

Ventajas:

- Eficaz en espacios de alta dimensión (incluso cuando son más que el número de instancias!).
- Eficiente en memoria (sólo los vectores de soporte definen el hiperplano frontera).
- Los kernels lo hacen súper versátil.

Desventajas:

- Si usas kernels, hay que tener mucho cuidado de no *overfittear*.
- Funciona bien BIEN sólo para clasificación.

¡BREAK!

Ph. Credit: Drew Coffmann

Hands-on training

Hands-on training

DS_Bitácora_25_SVM.ipynb

Recursos

Recursos

Si te quedaste con ganas de más...

- Support Vector Machines: Introducción SVM (nivel básico) algo de código al final.
- SVM (Support Vector Machine) — Theory: Introducción SVM y Kernels (nivel intermedio).
- Kernel Functions: Introducción (nivel básico) con ejemplo

Para la próxima

- Termina el notebook de hoy.
- Lee la bitácora 26 y carga las dudas que tengas al Trello.

En el encuentro que viene uno/a de ustedes será seleccionado/a para mostrar cómo resolvió el challenge de la bitácora. De esta manera, ¡aprendemos todos/as de (y con) todas/as, así que vengan preparados/as.

ACÁMICA