



# Aprendizaje de Máquina

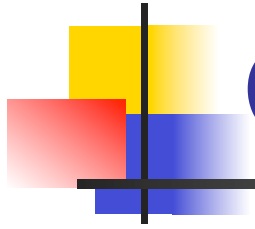
---



# Menú

---

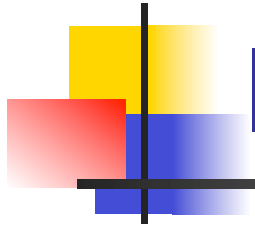
- Máquinas de soporte vectorial “Support Vector Machines” o SVM
  - Intuición
  - Trucos importantes



# Clases anteriores

---

- Hemos visto como la idea básica de regresión lineal
  - Puede utilizarse para clasificación
    - Perceptrón
    - Regresión logística
  - Puede utilizarse para ajustar funciones no lineales
    - Transformado los datos
    - Agregando datos derivados
    - Conectado muchas en una red (función de costo complicada)



# Idea General

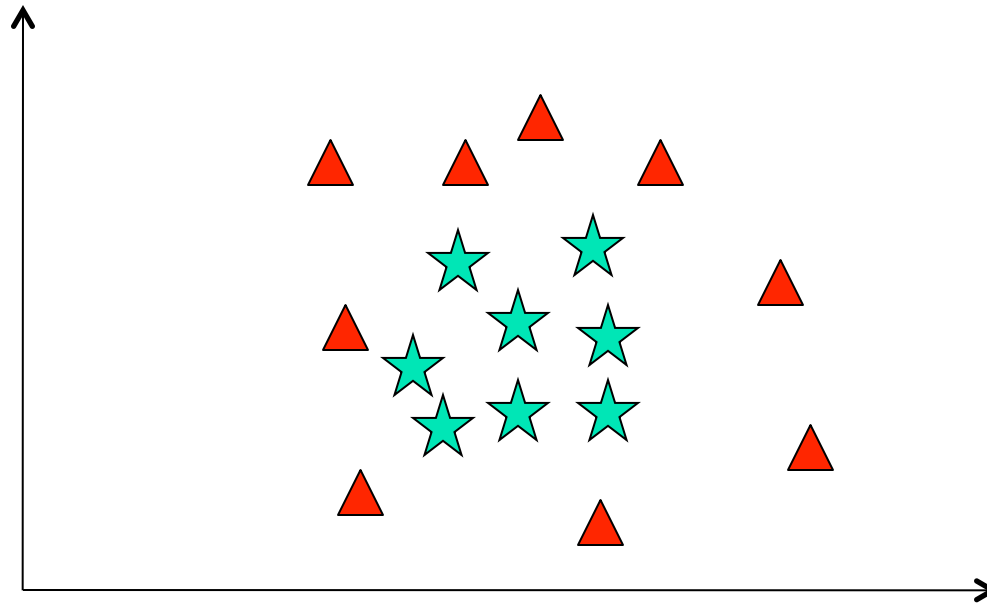
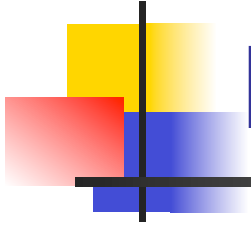
---

- Queremos seguir usando la idea de separar clases con hiperplanos
  - Sabemos resolver esto de manera óptima
  - El problema es que no todos los problemas son linealmente separables
- Idea: Quizá no sean linealmente separables en un espacio pero en otro de más alta dimensión si
  - Crear dimensiones adicionales de manera que las categorías sean linealmente separables

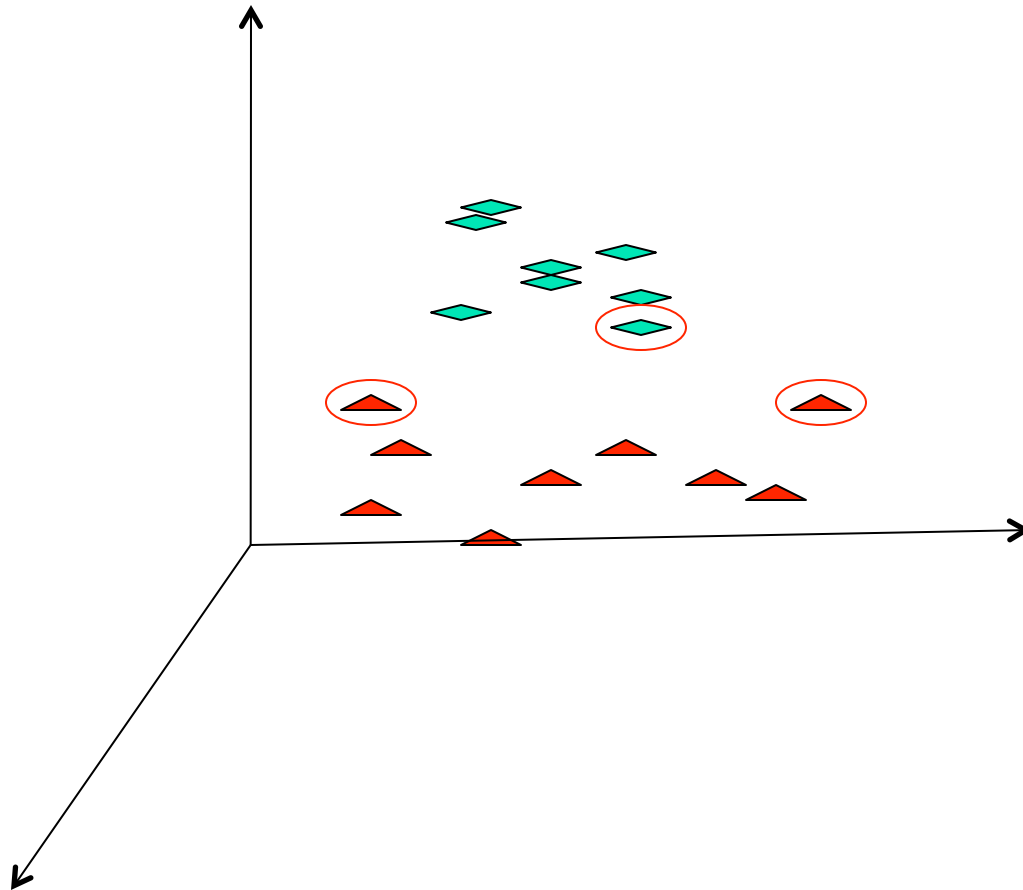
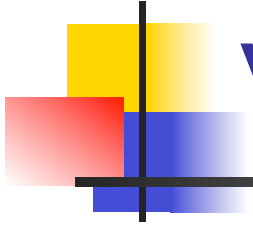
# Resumen

## Datos Originales

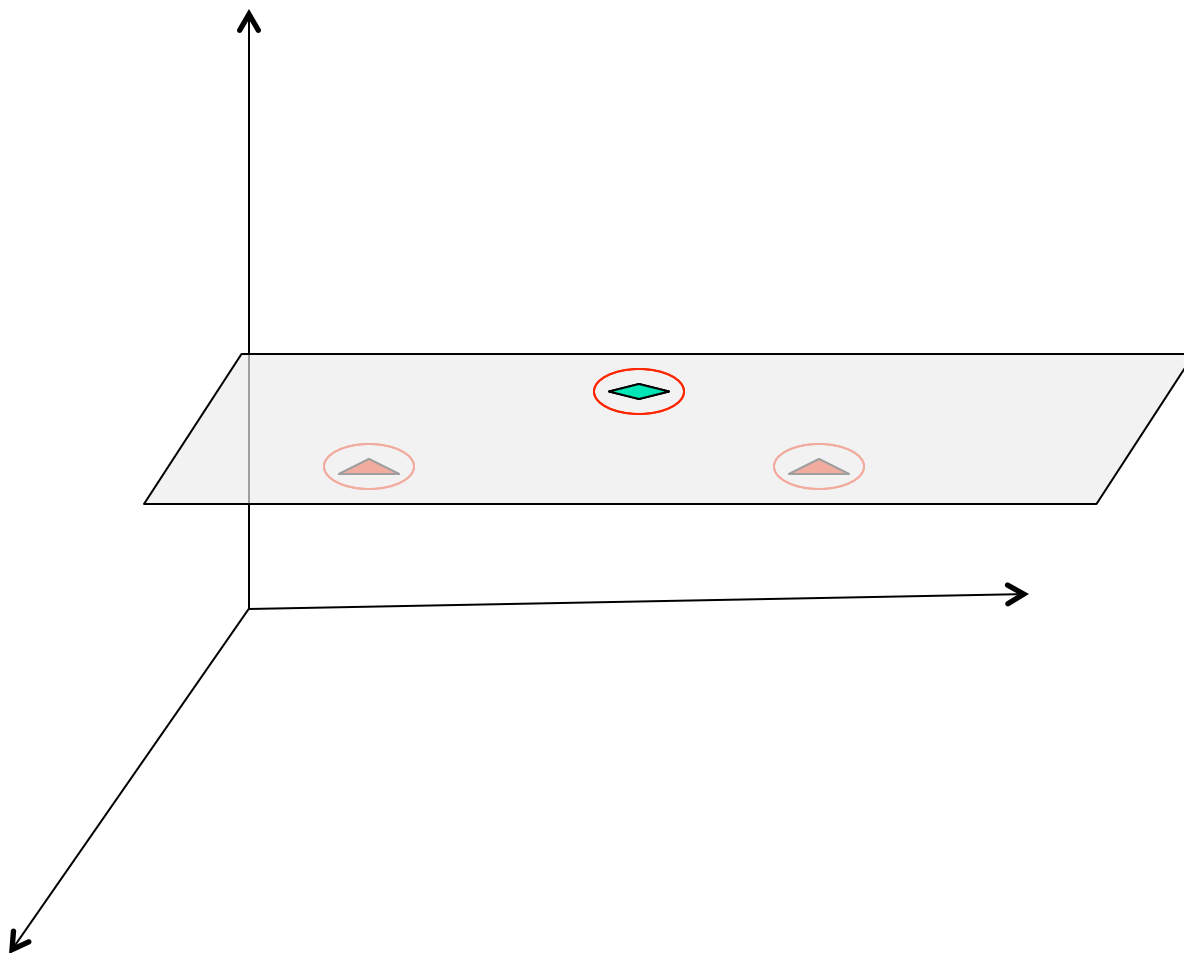
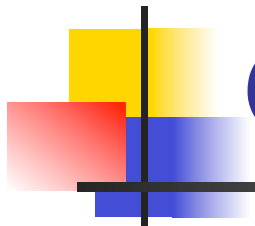
---

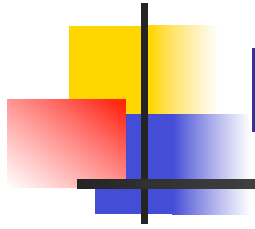


# Aumento de Dimensión y Vectores de Soporte



# Clasificador de Margen Óptimo



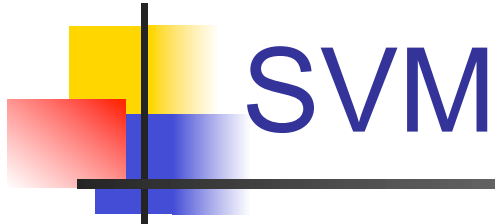


# Idea General

---

- La idea es relativamente simple y razonable pero presenta unas cuantas dificultades
  - ¿Cómo agrego dimensiones? ¿Cuántas?
  - El aumentar dimensiones significa agregar atributos adicionales llamados rasgos al los vectores de datos originales
    - Al tener más dimensiones se corre el riesgo de tener la “maldición de la dimensionalidad”
    - Sensibilidad al ruido
    - Se corre el riesgo también de un excesivo trabajo de cómputo
      - Qué tal si el número de rasgos es muy grande. El simple hecho de calcularlos puede tener un costo prohibitivo





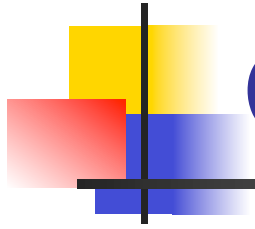
- Las máquinas de soporte vectorial son un propuesta que mitiga o soluciona estas cuestiones
- El desarrollo de las mismas requirió de un par de ideas importantes
  - Veremos un esbozo de su desarrollo

# SVM

## Ideas Principales

---

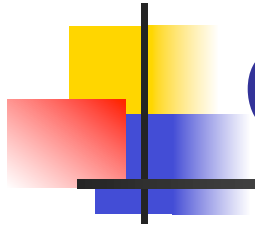
- Clasificador de margen óptimo
  - Reduce la sensibilidad al ruido
  - Acelera el cómputo
  - Mitiga la maldición
- Uso de Kernels
  - Acelera el computo
    - Permite el uso de dimensiones arbitrariamente grandes
- Margen Suave
  - Relajar la restricción de separabilidad lineal
  - Reducir sensibilidad al ruido



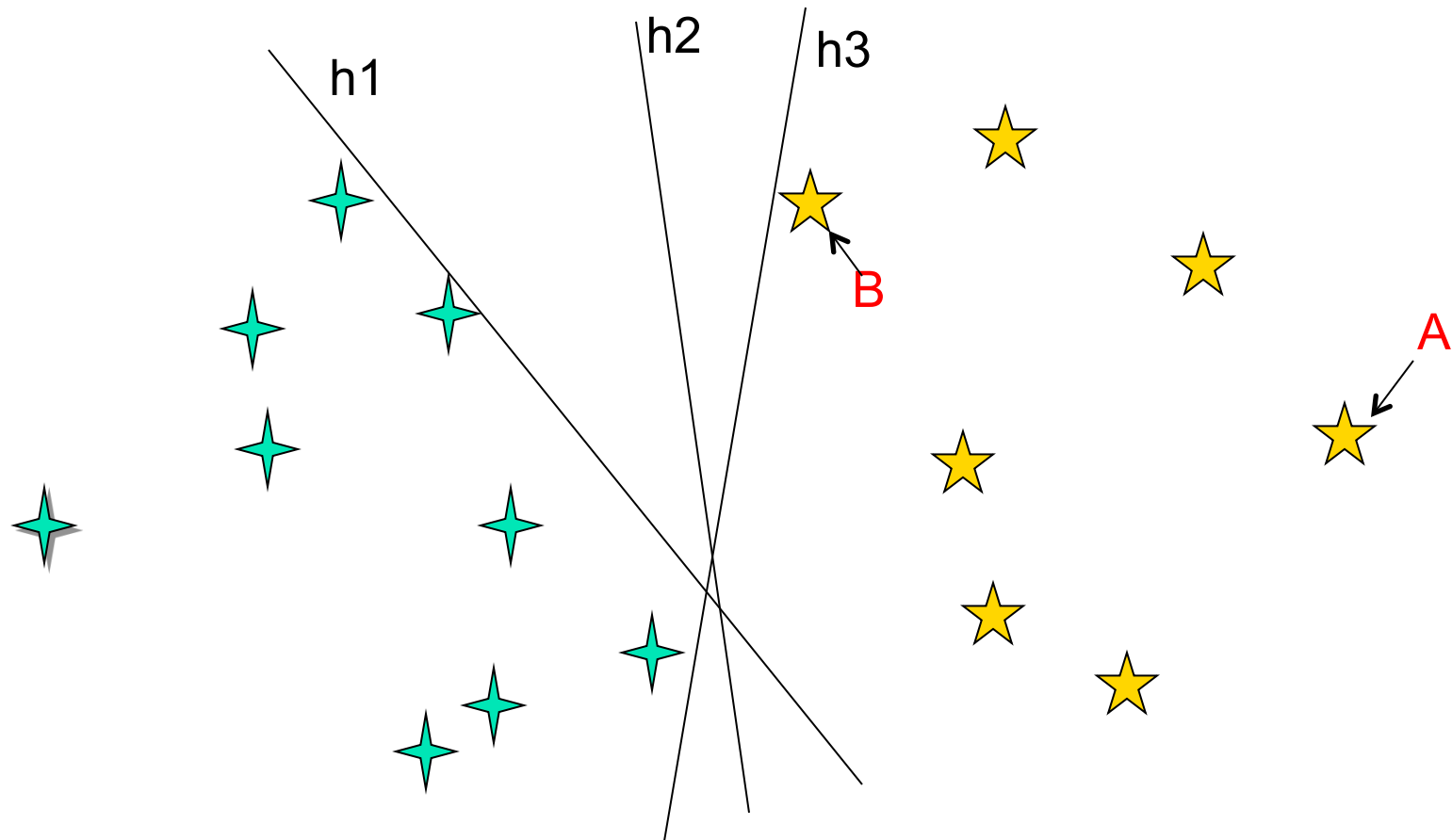
# Clasificador de Margen Óptimo

---

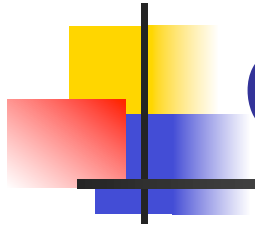
- Recuerdan nuestro clasificador lineal?
  - Hay muchos posibles hiperplanos para separar las dos clases
  - Suponemos para todo lo que sigue que las clases de las que hablamos son linealmente separables
    - Este supuesto se relaja en cierta medida para el SVM de margen suave



# Clasificación Binaria



¿Cuál separación es mejor,  $h1$   $h2$  o  $h3$ ?



# Clasificador de Margen Óptimo

---

- De alguna manera podemos pensar que nuestra predicción debe ser más “confiable” mientras más lejos se encuentre el dato del hiperplano
  - Estamos más seguros que **A** pertenece a la clase Estrella que **B**.
  - Un pequeño cambio en el  $h_3$  provocaría un cambio de clase para **B**
- $h_2$  generaliza mejor



# Clasificador de Margen Óptimo

## Intuición

---

- Si pensamos en nuestro clasificador logístico, mientras más lejos de la barrera de decisión más cerca de 1 (o cero dependiendo de la clase) será  $g(w^T x + w_0)$
- Si pensamos en el perceptrón este sólo tiene dos valores de salida y da el mismo resultado para todos los ejemplos bien clasificados
- La idea del margen óptimo combina estas dos ideas
  - Queremos que los ejemplos que esten dentro de una banda cumplan lo primero
  - Queremos que los ejemplos  $x$  que estan fuera no contribuyan al error y que el valor de  $w^T x + w_0$  pueda cambiar sin impacto
  - De esta manera tratamos de fijarnos únicamente en los puntos de las diferentes clases que estén proximos entre si



# El Problema de Optimización: el clasificador de margen óptimo

---

- La función “costo” que se utiliza pretende maximizar esta banda
- Pensando en la función de transferencia de escalón tenemos que
  - $V^{\wedge}(x) = 1$  si  $w^T x \geq 0$  y  $-1$  de otra forma
- Ahora queremos
  - $V^{\wedge}(x) = 1$  si  $w^T x \geq 1$  y
  - $V^{\wedge}(x) = -1$  si  $w^T x \leq -1$
  - Y que siga alguna función suave entre  $-1$  y  $1$



# El Problema de Optimización: el clasificador de margen óptimo

---

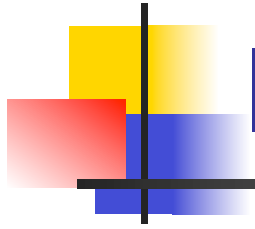
$$\text{Min} \frac{1}{2} w^T w$$

*tal que*

$$y_i(w^T x_i + w_0) \geq 1 \quad \text{para} \quad i = 1, \dots, M$$

Con  $y_i$  1 o -1

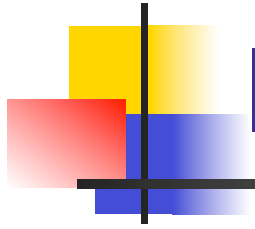




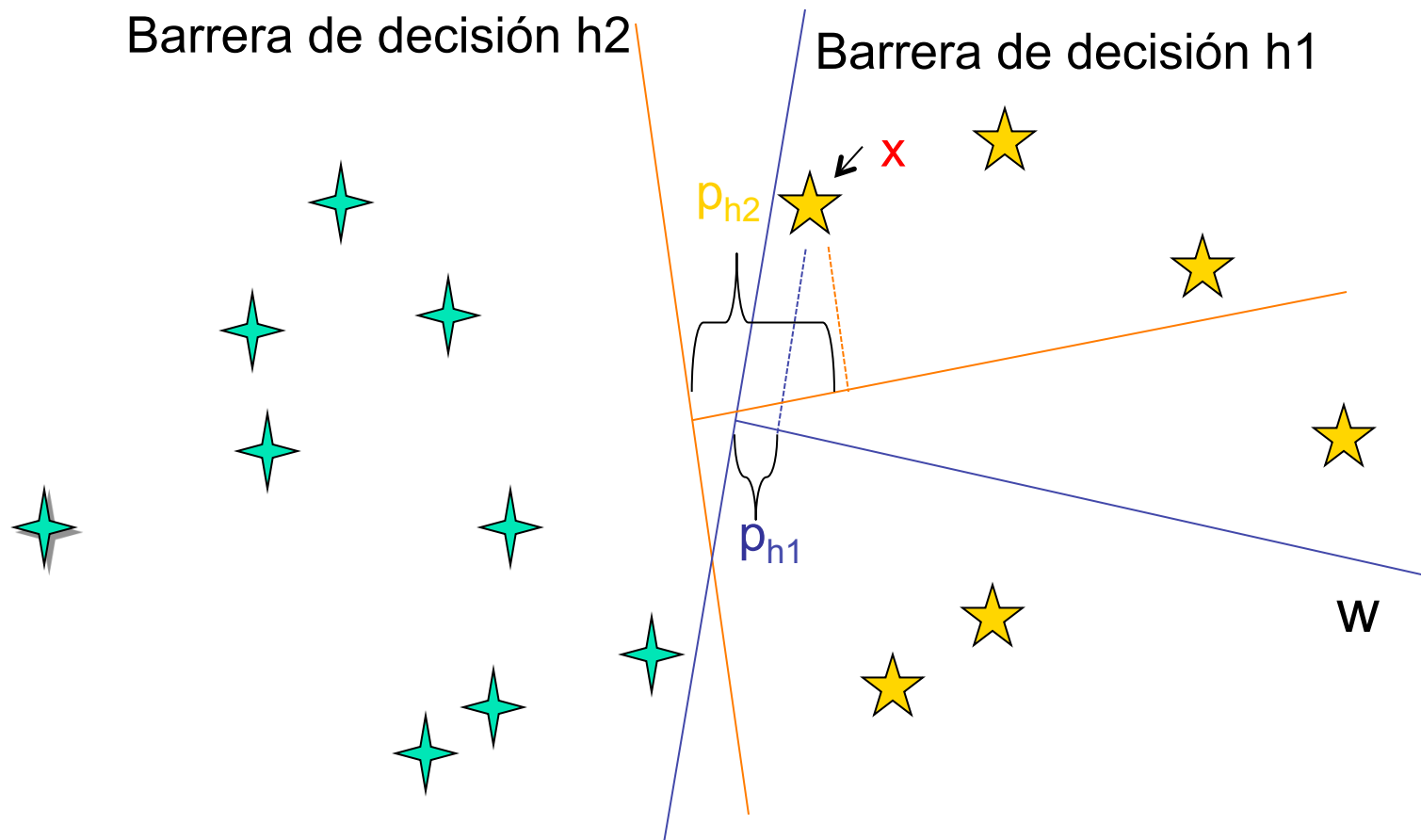
# Intuición Geométrica

---

- Porque el minimizar las  $w$  agranda el margen?
  - La barrera de decisión es  $w^T x^* = 0$  y es ortogonal al vector de pesos  $w$  ( $x^*$  son los valores de  $x$  que satisfacen la ecuación)
  - Requerimos  $w^T x^{(i)} \geq 1$  cuando  $y=1$  y  $w^T x^{(i)} \leq -1$  con  $y=-1$ 
    - Recordando que  $w^T x^{(i)} = p||w|| = p \sqrt{\sum w_j^2}$  donde  $p$  es la proyección del vector  $x^{(i)}$  sobre el vector  $w$
  - Requerimos que  $||w||^2$  sea chica
    - El problema obliga a hacer grande la magnitud de la proyección  $p$  (o muy negativa o muy positiva) para satisfacer las restricciones y por tanto alejar la barrera de decisión a lo largo de  $w$  con respecto a cada  $x^{(i)}$
    - Por simplicidad suponemos  $w_0$  inexistente por lo que  $||w||^2 = \sum w_j^2$



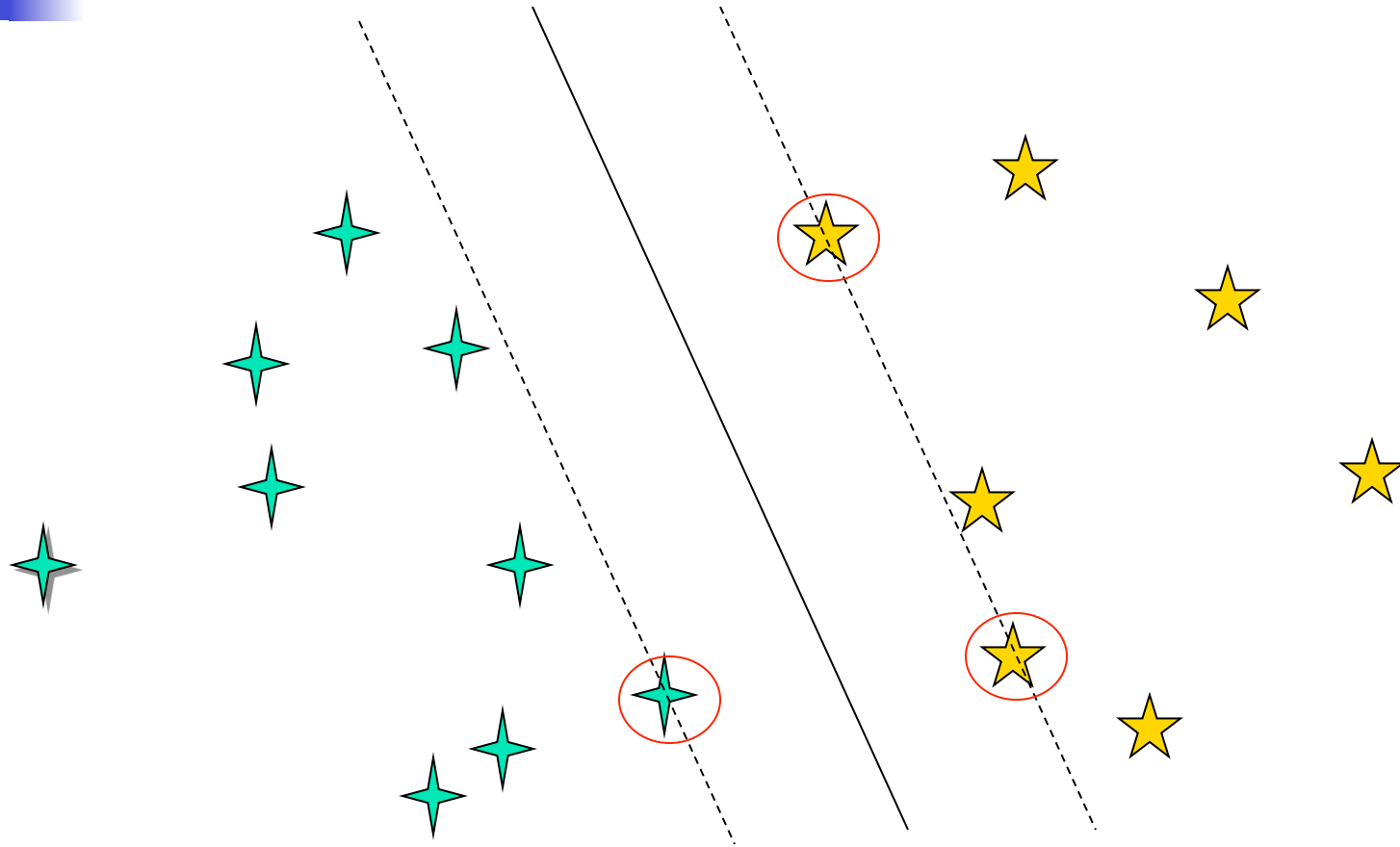
# Intuición Geométrica



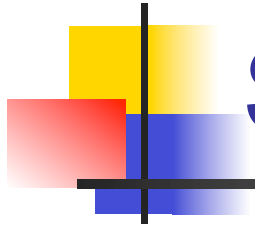
¿Cuál separación es mejor,  $h1$  o  $h2$  para  $x$ ?



# Clasificador de Margen Óptimo



Los tres vectores señalados son los vectores de soporte



# Siguiente Pasos

---

- Lo que deseamos ahora es
  - Encontrar los puntos que caracterizan al margen óptimo
    - Esto nos ayudará que el algoritmo sea eficiente
    - A mejorar la generalización
  - Elevar la dimensionalidad del problema de manera que los datos sean linealmente separables (o casi)
- Para esto tenemos que transformar el problema



# El Problema Dual

---

- El problema dual
  - Transforma el problema en otro de manera que la solución de uno es la solución del otro bajo ciertas condiciones
- El objetivo de esto es tener un problema que es más fácil, o más eficiente para resolver
  - El problema de optimización incluye una desigualdad en las restricciones (que  $|w^T x| \geq 1$ ) lo que lo dificulta
  - Nos permite conservar sólo ciertos puntos (los vectores de soporte), los que caracterizan al margen óptimo
  - Nos permite usar kernels para elevar la dimensión del problema de manera eficiente



# Vectores de Soporte

---

- Expresamos el problema usando multiplicadores de Lagrange

$$L(w, w_0, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)} (w^T x^{(i)} + w_0) - 1]$$

- Note que las restricciones estan como resta en la fórmula
- Para contruir el problema de optimización minimizamos con respecto a  $w$  y  $w_0$  e incluimos las restricciones (el problema dual maximiza con respecto a las alfas)
- Existe un conjunto de condiciones (KKT) bajo las cuales las soliciones del dual son iguales a las del original para restricciones de desigualdad
- Esas condiciones dan origen a los vectores de soporte



# Condiciones Karush-Kuhn-Tucker

---

1.  $\frac{\partial}{\partial w_i} L(w, w_0, \alpha) = 0$
2.  $\frac{\partial}{\partial w_0} L(w, w_0, \alpha) = 0$
3.  $\alpha(y(w^T x + w_0) - 1) = 0$
4.  $y(w^T x + w_0) - 1 \geq 0$
5.  $\alpha \geq 0$



# El Problema Dual de Optimización

---

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

tal que  $\alpha_i \geq 0, i = 1, \dots, m$

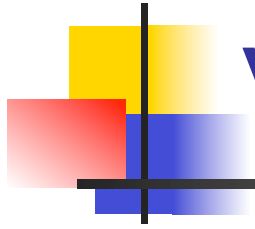
$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- Si resolvemos el problema dual entonces podemos calcular las  $w$ 's (salvo  $w_0$ ) para nuestro problema usando

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}$$

- Esto proviene de derivar e igualar a cero (condición 1)
- La  $w_0$  proviene, de igual forma de la condición 2





# Vectores de Soporte

---

- Note que seguimos suponiendo que los datos son linealmente separables
- Note que la restricción del problema nos obliga a maximizar  $\alpha$  y a que sea mayor o igual a cero.
- Note además que la condición 3 obliga a que muchas  $\alpha$  se vuelvan 0. De ahí que solo unos vectores importen, de ahí que sean el soporte del hiperplano

$$\alpha(y(w^T x + w_0) - 1) = 0$$

- Esto es cero  $y(w^T x + w_0) - 1$  sólo para los puntos  $x$  que definen el margen, para los demás, alfa debe ser cero



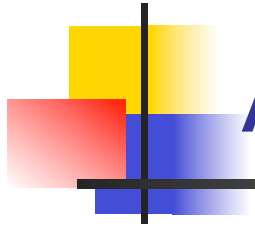
# El Problema Dual

---

- Para hacer una predicción hay que calcular  $w^T x + w_0$ . Sustituimos en la ecuación anterior

$$w^T x + w_0 = \left( \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^T x + w_0 = \sum_{i=1}^m \alpha_i y^{(i)} (x^{(i)})^T x + w_0$$

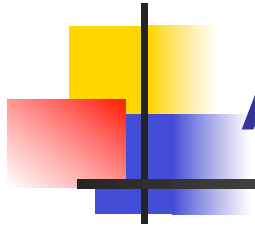
- Este es nuestro modelo
- Entonces, si tenemos las alfas, lo único (costoso) que necesitamos para hacer una predicción es calcular el producto interno de los vectores de soporte con el dato de entrada



# Aumento de Dimensión

---

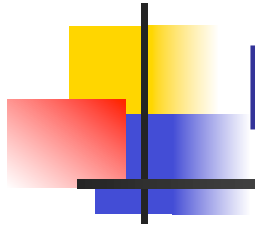
- Esto es relevante pues existe una manera de calcular el producto interno de vectores sin tener que manipular explícitamente los vectores
- Esto en conjunto con que sólo se usan unos cuantos vectores implica que podemos aumentar arbitrariamente la dimensionalidad de nuestros datos de entrada a bajo costo



# Aumento de Dimensión

---

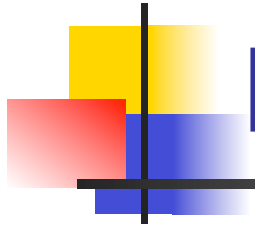
- Recuerdan de la clase de métodos lineales que incluimos el rasgo  $x^2$  a nuestro modelo (teníamos sólo el atributo  $x$ )?
  - Hablamos que podíamos incluir  $x^3$ ,  $\sin(x)$  o lo que se nos ocurriera
  - El problema era que sobre entrenaríamos
    - Esto se mitiga aquí pues sólo usamos unos cuantos puntos (los vectores de soporte)



# Kernels

---

- Lo que se necesita es una manera de aumentar la dimensión de los datos de manera que se pueda calcular el producto interno de manera eficiente
  - Si lo logramos podremos usar dimensiones MUY grandes, incluso infinitas
- Para esto vamos a usar Kernels que son justamente el truco para hacer esto



# Kernels

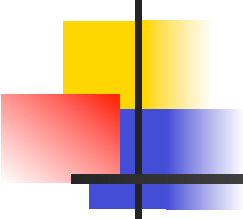
---

- Definimos el Kernel entre dos vectores como:

$$K(x, z) = \phi(x)^T \phi(z)$$

Donde  $\phi$  es el mapeo de los atributos del vector a los rasgos. Por ejemplo, para un vector  $x$  en  $R$

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$



# Kernels

## Ejemplo

---

- Supongamos que tenemos nuestros vectores  $x$  y  $z$  están en  $R^3$
- Supongamos que nuestro mapeo para cada vector consiste en multiplicar todos sus atributos entre si

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$



# Kernels

## Ejemplo

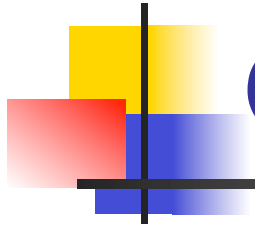
---

- El kernel

$$\begin{aligned} K(x, z) &= \phi(x)^T \phi(z) = \sum_{i=1}^n \sum_{j=1}^n x_i x_j z_i z_j \\ &= \left( \sum_{i=1}^n x_i z_i \right) \left( \sum_{j=1}^n x_j z_j \right) = (x^T z)^2 \end{aligned}$$

- Calcular el kernel toma tiempo proporcional al número de atributos, mientras que el número de rasgos usados es proporcional al cuadrado de los atributos





## Otros Kernels

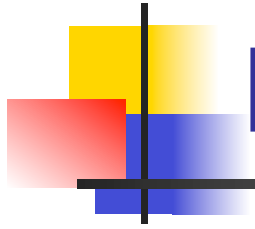
---

- No todos los mapeos tienen esta propiedad (el teorema de Mercer establece cuando un Kerner el válido).
- Estos son algunos Kernels comunes

$$K(x, z) = (x^T z + c)^d$$

$$K(x, z) = \exp\left(\frac{-\|x - z\|^2}{2\sigma^2}\right)$$

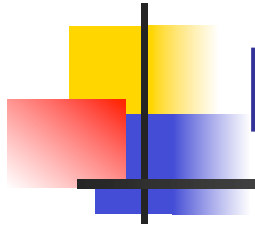
$$K(x, z) = \tanh(\kappa x^T z - \partial)$$



# Kernels

---

- Escoger un Kernel y sus parámetros es un poco un arte
  - Poca teoría
  - Podemos diseñar uno propio. El teorema de Mercer nos ayuda a asegurarnos que es válido
- Una intuición útil para guiar la elección de Kernel es pensar en  $K(x,z)$  como una medida de que tanto se parecen  $\phi(x)$  y  $\phi(z)$ 
  - El valor de  $K(x,z)$  será grande cuando se parezcan mucho y chico de otra forma

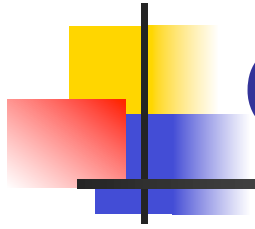


# El Nuevo Modelo

---

$$\text{Prediccion}(x) = \sum_{i=1}^m \alpha_i y^{(i)} K(x^{(i)} x) + w_0$$

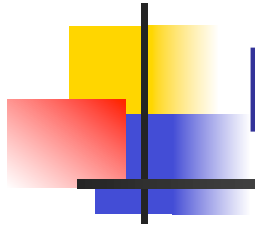
- Note que el modelo final hace uso sólo de los vectores de soporte y de la función  $K$  y no se calculan explícitamente atributos adicionales



# Otros detalles

---

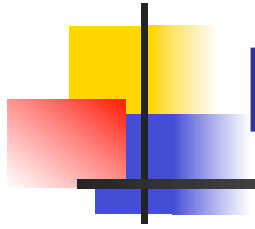
- Todo lo anterior lo hicimos suponiendo que las dos clases son linealmente separables (en algún espacio)
  - Esta suposición se puede relajar de manera que las clases sean “casi” linealmente separables y se tolere cierta cantidad de ruido. Se usa el margen suave
- Existen unos cuantos algoritmos eficientes para resolver el problema dual de optimización
  - El más usado se llama “sequential minimal optimization” (SMO)



# Margen Suave

---

- Los datos puede que no sean linealmente separables por dos razones
  - Por “outliers” como ruido
  - Porque los datos son intrínsecamente no linealmente separables
- Para el segundo caso utilizamos Kernels para elevar la dimensionalidad
- Para el primer caso utilizamos el Margen Suave



# Margen Suave

---

- La idea del margen suave es permitir que algunos elementos estén mal clasificados y aun así mantener una buena barrera de decisión
- Para esto introducimos una variable de tolerancia en nuestro modelo

- Teníamos

$$y(w^T x + w_0) \geq 1$$

- Ahora

$$y(w^T x + w_0) \geq 1 - \xi$$

- Dejamos que los datos estén un poquito mal clasificados, que su distancia a la barrera del lado equivocado sea a lo más psi



# El Problema de Optimización: el clasificador de margen suave

---

$$\text{Min} \frac{1}{2} w^T w + C \sum_{i=1}^M \xi_i$$

*tal que*

$$y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad \text{para } i = 1, \dots, M$$

$$\text{y } \xi_i \geq 0 \quad \text{para } i = 1, \dots, M$$



# El Problema de Optimización Dual

---

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

tal que  $0 \leq \alpha_i \leq C, i = 1, \dots, m$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- Quedo igual!
- La diferencia es que las alfas deben ser menores o iguales a una constante
- Los vectores de soporte que regrese el optimizador que sean iguales a C (o -C) corresponden a los datos que no respetan el margen

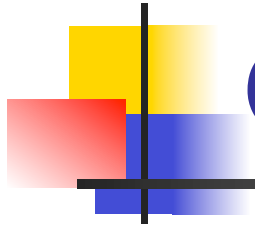




# Consejos de Uso de Andrew Ng

---

- Si se tienen más atributos que datos
  - Usar regresión logística regularizada o SVMs con kernel lineal
- Si se tienen pocos atributos y muchos ejemplos de entrenamiento (menos de 10k)
  - Usar SVM con kernel gaussiano
- Si se tienen pocos atributos y muchos ejemplos de entrenamiento (más de 10k)
  - Agregar atributos derivados y usar regresión logística o SVM con kernel lineal



# Otros Comentarios

---

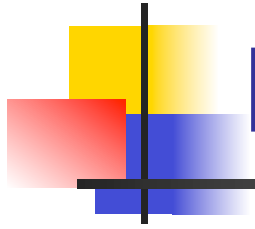
- Se puede pensar en la complejidad del modelo como el número de vectores de soporte que resultan
- Un buen estimado del error de generalización es el cociente de los vectores de soporte (esperados) sobre el número de datos de entrenamiento
  - No pagamos un precio alto por aumentar la dimensión!!!!
- Si el cociente es alto no se va a generalizar bien



# Paquetes

---

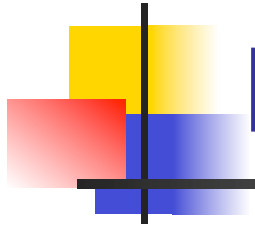
- Implementar SVMs es un poco tedioso. Existe una librería muy probada que utilizan muchas aplicaciones. Se llama libSVM



# Ejercicio

---

- Utilice el archivo andSVM.csv
- Entrene un perceptrón y grafique la barrera de decisión
- Entrene una SVM usando:
  - `from sklearn.svm import SVC`
  - Use un kernel lineal
- Grafique los datos y la barrera de decisión
- Grafique el margen, las rectas que pasan por los vectores de soporte
  - Utilice diferentes valores de  $C$  (1 y 100)



# Ejercicio

---

- Cree un conjunto de datos en dos dimensiones  $x_1$  y  $x_2$  de manera que los puntos que se encuentren dentro de un círculo (centro en 0,0) sean de la clase 1 y los que estén fuera de la clase 0
  - $x_1^2 + x_2^2 = r^2$
- Entrene una Red Neuronal para este problema
  - Visualice los datos bien y mal clasificados
- Repita el ejercicio para un SVM y compare