Support Vector Machine

Hasta ahora ya hemos visto una gama amplia de algoritmos de aprendizaje distintos. En el aprendizaje supervisado, el desempeño de muchos algoritmos de aprendizaje supervisado será muy similar. Importa poco si utilizamos el algoritmo de aprendizaje “A” o el algoritmo de aprendizaje “B”. Lo que importa más es la cantidad de datos en la que entrenamos estos algoritmos. También cuenta tu habilidad al aplicar estos algoritmos; es decir, la manera en la que elijas las funciones designadas para el algoritmo de aprendizaje y el parámetro de regularización y cosas similares.

Hay otro algoritmo muy efectivo y muy utilizado tanto en la industria como en la academia. Se llama máquinas de soporte vectorial. Comparadas con la regresión logística y las redes neuronales, las máquinas de soporte vectorial o “SMV” por sus siglas en inglés, a veces arrojan funciones de aprendizaje complejas no lineales más limpias y efectivas.   
  
Me gustaría dedicar los siguientes videos a este tema. Más tarde en este curso haré un recuento de la gama de diferentes algoritmos de aprendizaje para describirlos brevemente. Sin embargo, por su popularidad, las máquinas de soporte vectorial será el último algoritmo de aprendizaje supervisado al que le dedique tiempo en este curso.

[**Large Margin Classification**](#_67zvb3h52hpu) **1**

[**Large Margin intuition**](#_gues14o2wtfz) **9**

[**Desarrollo matemático del algoritmo**](#_c3u7h6i65f9c) **16**

[**Kernels**](#_3vy2l8apb0lm) **30**

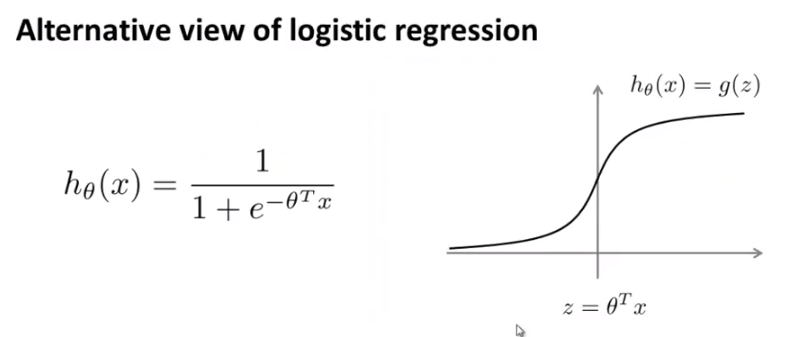
[Kernels II](#_1acsqcovd0g1) 41

[**SVM en la práctica**](#_t7802pgvkrt) **49**

# Large Margin Classification

* 1. **Optimization Objective**

Para describir el SVM, comenzaré con la regresión logística y mostraré cómo podemos modificarla un poco para obtener la máquina de soporte vectorial.   
  
En la regresión logística tenemos nuestra formulación familiar de hipótesis a la izquierda y la función de activación sigmoidal a la derecha.



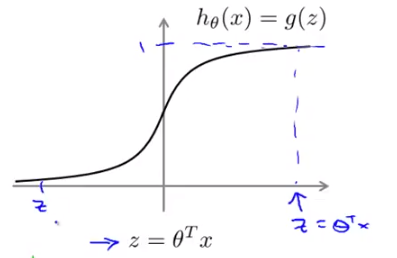
Para explicar la matemática utilizaré “z” para denotar teta transpuesta de “x”.

Ahora, pensemos en qué nos gustaría que hiciera nuestra regresión logística. Si tenemos un ejemplo en el que “y” es igual a 1, con ejemplo me refiero al conjunto de entrenamiento, el de prueba o el de validación cruzada, donde “y” es igual a 1, entonces esperaremos que h(x) resulte cercano a 1. Entonces, esperamos clasificar correctamente ese ejemplo.



Tener h(x) cercano a 1 quiere decir que teta transpuesta de “x” debe ser mucho mayor que 0. Aquí está el signo doble de mayor qué, que significa “mucho mayor que” 0. Esto es porque cuando “z”, es decir, teta transpuesta de “x”, es mucho mayor que 0, el resultado de la regresión logística se encuentra cerca del 1.   
  
Por el contrario, si tenemos un ejemplo donde “y” es igual a 0, estamos esperando que la hipótesis arroje un valor cercano a 0 que corresponde a teta transpuesta de “x” o “z” también cercana al 0, ya que corresponde a la hipótesis con un valor cercano al cero.



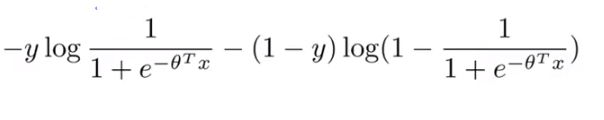


Si observas la función de costos de la regresión logística, lo que encuentras es que cada ejemplo (x, y) contribuye un término como este a la función de costos.



Bueno. En la función de costos usualmente tendremos también una suma de todos los ejemplos de entrenamiento; el término 1 sobre “m”. Esta expresión de arriba es el término que aporta cada ejemplo particular de entrenamiento a la función general objetiva para la regresión logística.

Ahora, si tomo la definición para la formulación de mi hipótesis y la sustituyo donde le corresponde, lo que obtendré será que cada ejemplo de entrenamiento contribuye a este término, sin tomar en cuenta 1 sobre “m”, a mi función de costo general para la regresión logística.



Ahora, consideremos los dos casos, cuando “y” es igual a 1 y cuando “y” es igual a 0. En el primer caso, supongamos que “y” es igual a 1. En este caso, sólo nos importa el primer término del objetivo porque este término, 1 menos “y” será igual a 0 si “y” es igual a 1. Cuando “y” es igual a 1; es decir, cuando en nuestro ejemplo (x,y), “y” es igual a 1, lo que obtendremos será este término:



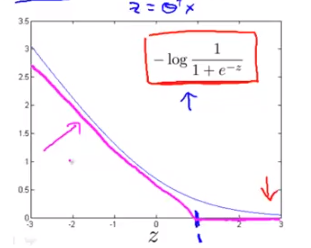
menos log de 1 sobre 1 más “e” elevado a la “z” negativa donde, al igual que en la diapositiva anterior, estoy utilizando “z” para denotar teta transpuesta de “x”. Y, por supuesto, en el costo tenemos este menos “y” pero ya dijimos que si “y” es igual a 1, esto será igual a 1.

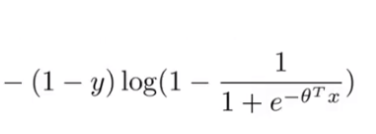
Si trazamos esta variable como una variable de “z”, lo que encontrarás es la curva que se muestra. Entonces, si vemos que cuando “z” es grande; es decir, cuando teta transpuesta de “x” es grande, nos dará un valor de “z” muy bajo o una contribución pequeña a la función de costos. Esto explica porqué cuando la regresión logística trata con un ejemplo positivo, como “y” igual a 1, intenta fijar teta transpuesta de “x” muy alto porque corresponde a un término pequeño en esta función de costos.

Ahora, para construir una máquina de soporte vectorial, haremos lo siguiente. Tomaremos esta función de costos; este menos log de 1 sobre 1 más “e” elevado a “z” negativa

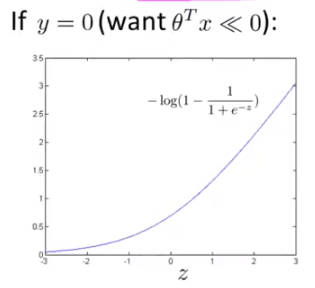


y lo modificaremos un poco.   
  
Tomaré este punto 1 de la recta y trazaré la función de costos que utilizaré. La función de costos será plana de aquí en delante.   
  
Dibujaré algo que sigue una línea recta, igual que en la regresión logística; sin embargo, esta será una línea recta con estas proporciones. La línea curva que dibujé en color magenta o rosa es una aproximación cercana a la función de costos que se utiliza en la regresión logística, excepto porque está hecha de dos segmentos de línea. Tengo esta porción plana de la derecha y esta porción de línea recta de la izquierda.

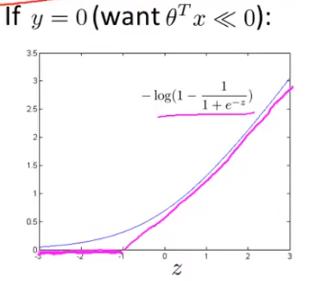


No te preocupes mucho por la pendiente de la porción semivertical. No importa mucho. Esta es la función de costos que utilizaremos, donde “y” es igual a 1. Puedes imaginar que debes hacer algo muy similar a la regresión logística, pero resulta que esto le dará una ventaja computacional a la máquina de soporte vectorial y más adelante nos dará un problema de optimización que sería más sencillo de resolver para el software.   
  
Hasta ahora hablamos de cuando “y” es igual a 1. El otro caso es cuando “y” es igual a 0. En este caso, si vemos el costo, sólo aplicará el segundo término de la función de costes ya que el primer término se elimina porque “y” es igual a 0 Entonces nos quedamos sólo con el segundo término de la expresión de arriba.   


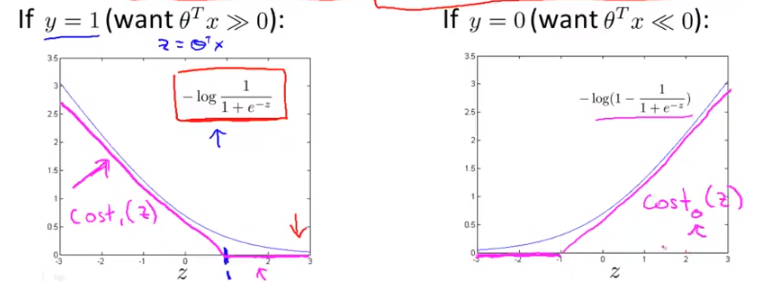
Este término de aquí nos dará el costo de un ejemplo, o la contribución de la función de costos. Si lo trazamos como una variable negativa donde, “z” está en el eje horizontal, obtendremos una curva así.



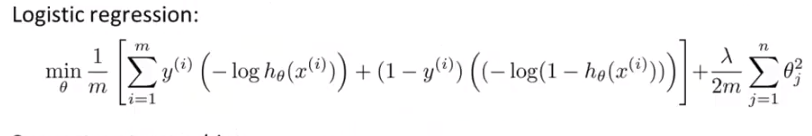
Para la máquina de soporte vectorial, de nuevo, reemplazamos esta línea azul con una similar. Digamos que la reemplazamos con un nuevo costo que es plano al principio y luego sube como una línea recta.



Ahora les pondremos nombre a estas dos variables. A la variable primera, es decir el caso donde y es igual a 1, pasaremos a llamarla costo subíndice 1 de “z” y para el otro caso llamaremos costo subíndice 0 de “z”. El subíndice se refiere al costo correspondiente a “y” igual a 1 o a “y” igual a 0.

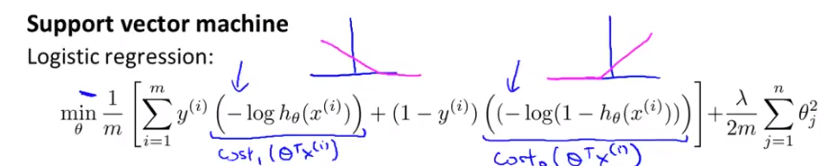


Armados con estas definiciones, ya estamos listos para construir nuestra máquina de soporte vectorial. Aquí está la función de costos de “J” de teta que usamos en la regresión logística.

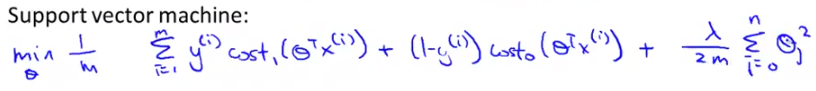


Si esta ecuación te parece extraña es porque antes teníamos un signo de menos afuera. Aquí, lo que hice fue mover los signos de menos hacia adentro de esta expresión. Eso la hace lucir un poco diferente.

Para la máquina de vector de soporte, lo que haremos, esencialmente, es tomar lo que previamente hemos definido como y remplazarlo.

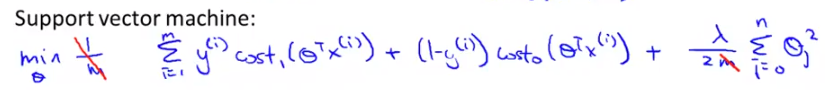


Lo que tenemos para la máquina de soporte vectorial es un problema de minimización de la suma de 1 sobre “m” (sobre los ejemplos de entrenamiento) o de “y(i) por el costo 1 de teta transpuesta de “x(i)” más 1 menos “y(i) por el costo 0 de teta transpuesto de “x(i)”. Más mi parámetro de regularización normal.



Ahora, por costumbre, para la máquina de vectores de soporte, escribimos estas expresiones un poco diferente. Determinamos los parámetros de manera un poco diferente.

Primero, nos olvidaremos de los términos 1 sobre “m”. Esta es una costumbre ligeramente distinta que se utiliza para las máquinas de soporte vectorial en comparación con la regresión logística. Aquí está a lo que me refiero. Lo que haré es deshacerme de los términos 1 sobre “m” para obtener el mismo valor óptimo para teta porque 1 sobre “m” es una constante.   
9:56  
Así que, si resuelvo o no este problema de minimización con 1 sobre “m”, terminaré utilizando el mismo valor óptimo de teta.



Me explicaré mejor dando un ejemplo concreto. Supongamos que tengo un problema de minimización en el que pretendo minimizar un número real “u” de “u” menos 5 al cuadrado más 1. El mínimo de esto es, como sabes, es “u” igual a 5.



Ahora si tomo esta variable objetiva y la multiplico por 10, de manera que mi problema de minimización es mínimo de “u” de 10, “u” menos 5 al cuadrado más 10. El valor de “u” que minimiza esto sigue siendo “u” igual a 5.



Entonces, cuando multiplicas algo que estás minimizando sobre una constante, 10 en este caso, el valor de “u” que minimiza esta variable no cambia.   
Lo que hice, eliminando la “m” es multiplicar mi función objetiva por una constante “m”; por lo tanto el valor de teta que arroja el mínimo no cambia.   
  
El segundo cambio notacional que, una vez más, es la costumbre más generalizada cuando utilizamos SVM en vez de regresión logística, es el siguiente. Para la regresión logística teníamos dos términos en nuestra función objetiva.   
  
El primer término es el costo que obtenemos del conjunto de entrenamiento y el segundo término es el término de regularización. En esta variable controlamos la compensación entre ambos términos minimizando “A más mi parámetro de regularización utilizando «lambda» multiplicado por otro término que denominamos B.



Estoy utilizando A para referirme al primer término y B para el segundo, lo que hicimos es, al ajustar diferentes valores para el parámetro de regularización «lambda» compensamos cuánto queremos ajustar el conjunto aprendizaje, es decir, la minimización de “A”, frente a qué tan bajos queremos mantener los valores de los parámetros. Me refiero a los parámetros de “B”.

Para la máquina de soporte vectorial, por costumbre utilizaremos un parámetro diferente. En vez de utilizar «lambda» para controlar la espera relativa entre el primer y el segundo término, utilizaremos un parámetro diferente al que llaman “C” y minimizaremos “C” por “A” más “B”.

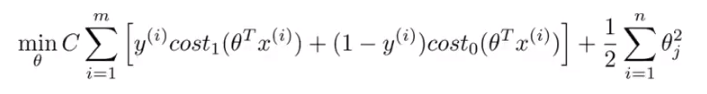


Para la regresión logística si enviamos un valor muy grande de «lambda» obtendremos un valor grande de “B”. Por el contrario, en el SVM, si fijamos a “C” un valor muy pequeño, el valor correspondiente de “B” será mucho más alto que “C” por “A”.   
  
Esta es otra manera de controlar la compensación o una manera distinta de parametrizar cuánto nos interesa optimizar el primer término frente a cuánto nos interesa optimizar el segundo término.   
  
Si quieres, puedes pensar que el parámetro “C” juega un rol muy similar a 1 sobre «lambda». No es que estas dos ecuaciones o expresiones sean iguales si “C” igual a 1 sobre «lambda».

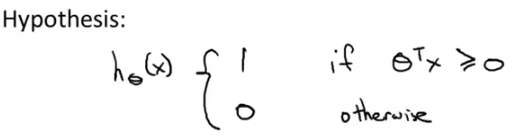


Este no es el caso. Pero si “C” es igual a 1 sobre «lambda», entonces, estos dos objetivos de optimización te darán el mismo valor valor óptimo de teta. Entonces, para actualizarnos voy a borrar esta «lambda» de aquí y escribir la constante “C”.

Esto nos da una función objetiva y general de optimización para la máquina de soporte vectorial donde si minimizas la variable obtienes los parámetros aprendidos por la SVM.



Finalmente, en el ámbito de la regresión logística, la SVM no arroja la probabilidad. En lugar de eso, lo que tenemos es la función de costos, que minimizamos para obtener el parámetro teta. Lo que hace la máquina de soporte vectorial es predecir directamente si “y” es igual a 1 o a 0. Entonces, tendré una hipótesis que predice un valor de 1 si teta transpuesta de “x” es mayor o igual a 0, y sino predice 0, Una vez que aprendimos los parámetros teta.

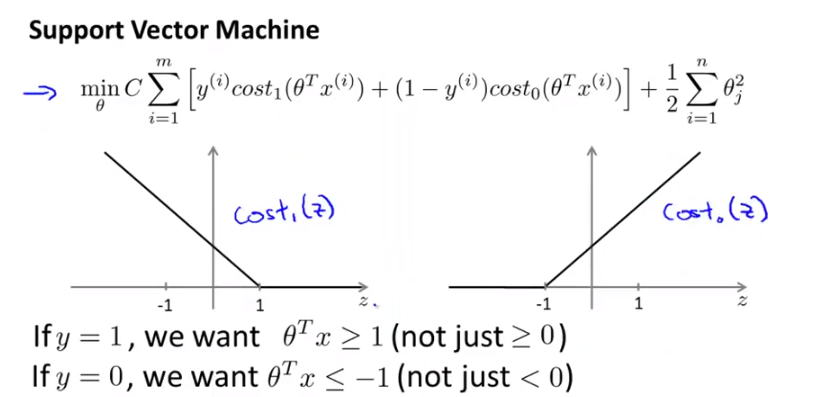


Esta es la forma de la hipótesis para la máquina de vectores de soporte. Esta fue una definición matemática de lo que hace una máquina de soporte vectorial. En los siguientes videos intentaremos ganar un mejor entendimiento de a dónde nos lleva el objetivo de optimización y de cuáles son las fuentes de la hipótesis que aprenderá nuestra SVM. También hablaremos de cómo modificar esto un poco para obtener variables complejas no lineales.

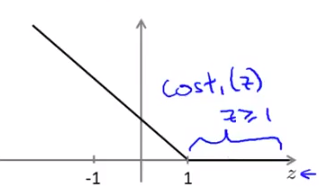
# Large Margin intuition

A veces, la gente habla de las máquinas de soporte vectorial como clasificadores de márgenes amplios. En este video les explicaré qué significa esto y les daré un panorama útil de cómo se ve una hipótesis de una SVM.

Aquí tengo mi función de costos para la máquina de soporte vectorial donde a la izquierda tracé mi función costo 1 de “z” que utilizo en ejemplos positivos, y a la derecha tracé la   
variable 0 de “z”, donde tengo “z” en el eje horizontal.

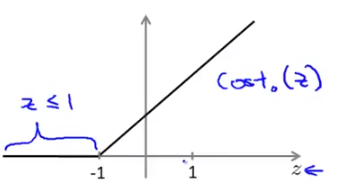


Ahora, pensemos qué hace falta para hacer estas funciones de costo más pequeñas. Si tienes un ejemplo positivo, si “y” es igual a 1, entonces costo 1 de “z” será 0 sólo cuando “z” sea igual o mayor que 1.



En otras palabras, si tienes un ejemplo positivo, queremos que teta transpuesta de “x” sea más grande o igual a 1.

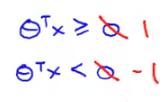
Y a la inversa, si “y” es igual a 0, en la función costo 0 de “z”, entonces sólo en esta región, donde “z” es menor o igual a 1 que tendremos un costo de 0 si “z” es igual a 0.



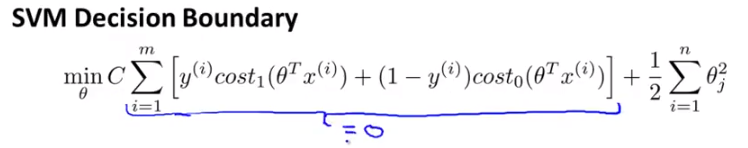
Una propiedad interesante de las máquinas de soporte vectorial es que, si tienes un ejemplo positivo, o sea, si “y” es igual a 1, entonces necesitamos que teta transpuesta de “x” sea mayor que cero. Esto significa que estaríamos realizando una clasificación correcta porque si teta transpuesta de “x” es mayor que 0, nuestra hipótesis hará una predicción de 0.

De manera similar, si tienes un ejemplo negativo, lo que queremos es que teta transpuesta de “x” sea menor que cero. Esto nos asegurará que tenemos el ejemplo correcto.

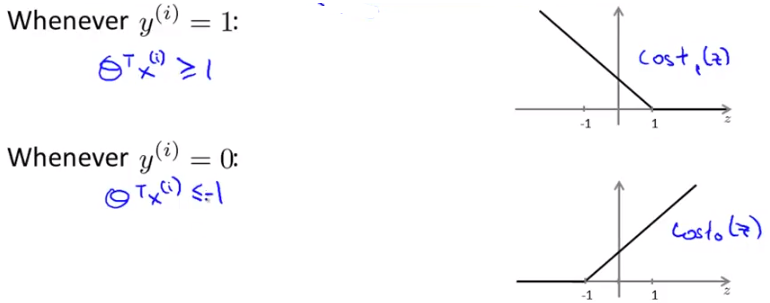
Pero la máquina de soporte vectorial quiere un poco más que esto. No sólo quiere que obtengas el ejemplo correcto, no queremos que sea sólo un poco mayor que cero. Lo que en realidad quiero es que sea mucho mayor a cero; quizá hasta mayor o igual a 1. Y quiero que sea mucho menor a cero. Quizá menor o igual a - 1.



Esto construye un factor de seguridad adicional o un factor de margen de seguridad en la máquina de vector de soporte. La regresión logística hace algo similar, pero veamos qué pasa o cuáles son las consecuencias de esto en el contexto de máquinas de soporte vectorial.   
  
Lo que me gustaría hacer a continuación es considerar un caso en el que determinamos que esta constante “C” es un valor muy grande. Imaginemos que le damos a “C” un valor muy grande; quizá cien mil o un número enorme. Veamos que hace la máquina de soporte vectorial. Si “C” es muy, muy grande, entonces, cuando minimicemos el objetivo de optimización estaremos más inclinados a elegir un valor para que este primer término sea igual a 0.



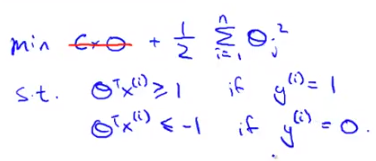
Intentemos entender el problema de optimización en el contexto de qué se necesitaría para hacer que el primer término en el objetivo sea igual a cero porque, quizá “C” tenga una constante enorme. Esto, con suerte, nos dará una intuición adicional de las hipótesis que aprenderá la máquina de vector de soporte. Ya vimos que cuando tenemos un ejemplo de entrenamiento con un valor asignado de “y” igual a 1, si queremos que el primer término sea 0, necesitamos encontrar el valor de teta para que teta transpuesta de “x” sea mayor o igual a 1. De manera similar, cuando tenemos un ejemplo cuyo valor asignado es 0, para asegurarnos de que el costo 0 de “z” sea 0, necesitamos que teta transpuesta de “x(i)” sea menor menor o igual a - 1.



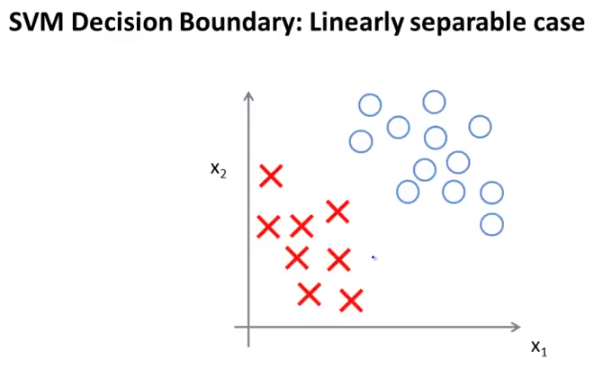
Ahora, si pensamos en nuestro parámetro como ahora, elegir los parámetros nos aseguran que este primer término es igual a cero y nos quedamos con el siguiente problema de optimización. Minimizamos el primer término 0; es decir, “C” por 0, porque elegiremos los parámetros para que sea igual a 0, más un medio. Después, el segundo término.



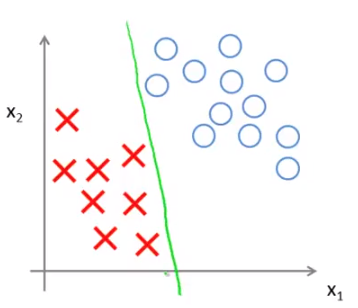
Este primer término es “C” igual a cero así que lo eliminaremos porque sabemos de antemano que será 0. Esto estará sujeto a la condición de que teta transpuesta de “x(i)” es mayor o igual a 1, si “y(i)” es igual a 1 y teta transpuesta de “x(i)” es menor o igual a menos 1 cuando tenemos un ejemplo negativo.



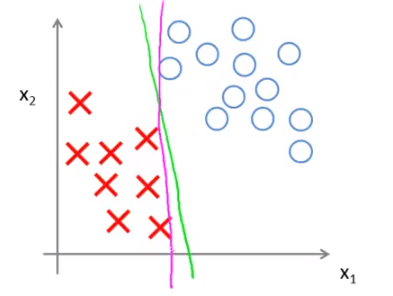
Resulta que cuando resolvemos este problema de optimización, y minimizamos esto como una variable del parámetro teta, obtenemos una barrera de decisión interesante. Si observas un conjunto de datos con ejemplos positivos y negativos como estos,



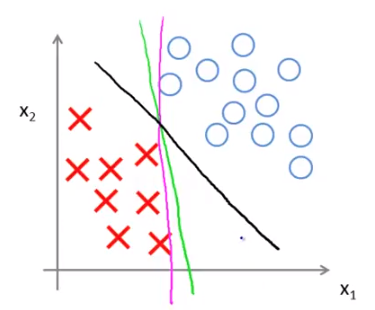
los datos son separables linealmente. Con esto me refiero a que existe una línea recta, o muchas líneas rectas distintas, pueden separar perfectamente los ejemplos negativos y los positivos. Por ejemplo, aquí hay una barrera de decisión que separa los ejemplos positivos y negativos,



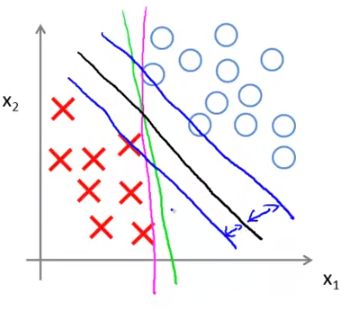
Pero por alguna razón no se ve muy natural, ¿cierto? O podemos trazar otra barrera de decisión que apenas separe los ejemplos positivos y negativos. Pero ninguna de estas líneas parecen opciones muy buenas.



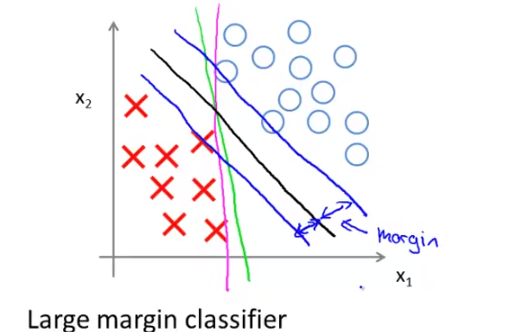
La máquina de vector de soporte elegirá esta barrera de decisión que estoy dibujando en negro.



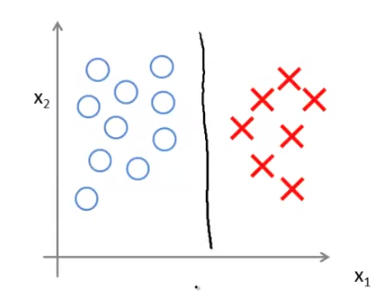
Esta parece ser una mejor barrera de decisión que las que dibujé en magenta o en verde. La línea negra parece ser un separador más sólido. Hace un mejor trabajo separando los ejemplos positivos y los negativos. Matemáticamente, lo que hace esta barrera de decisiones negra es tener una distancia mayor. La distancia es lo que llamamos margen. Si la comparamos con estas dos líneas azules adicionales



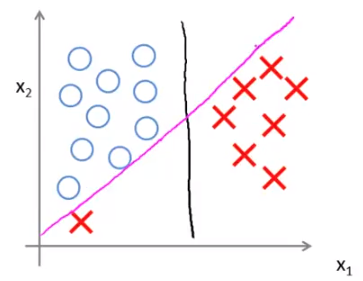
podemos notar que la barrera de decisiones negra tiene una distancia mínima mayor entre los ejemplos de entrenamiento, mientras que las líneas magenta y verde se acercan mucho a los ejemplos de entrenamiento y hacen un trabajo menos efectivo separando los ejemplos positivos y negativos que la línea negra. Por lo tanto, esta distancia se llama margen de la máquina de soporte vectorial y le da a la SVM cierta solidez, porque intenta separar los datos con el mayor margen posible.



La máquina de vector de soporte también se llama clasificador de margen amplio. Esta es una consecuencia del problema de optimización que escribimos en la diapositiva anterior. Sé que se preguntan cómo funciona el problema de optimización que escribí en la diapositiva anterior y cómo nos lleva a este clasificador de margen amplio. Sé que no lo he explicado aún. En el siguiente video presentaré un poco del concepto de porqué el problema de optimización nos da un clasificador de margen amplio. Esta es una variable útil cuando intentas entender los tipos de hipótesis que elegirá la SVM, es decir, cuando intentas separar los ejemplos positivos y negativos con el mayor margen posible.   
  
Quiero decir una última cosa acerca de los clasificadores de márgenes amplios en este contexto. Escribimos esta configuración de clasificación de márgenes amplios donde “C”, el concepto de regularización, era muy amplio. Creo que lo determinamos como cien mil o algo similar. Con un conjunto de datos como este, elegiremos la barrera de decisión que separe los ejemplos positivos y negativos por un margen amplio.

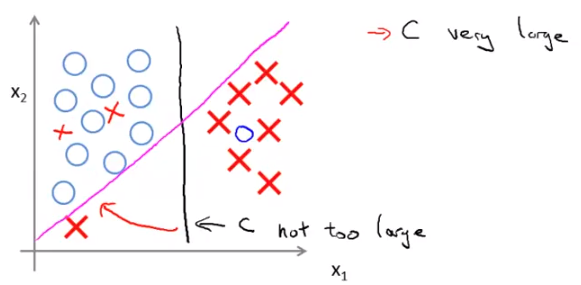


Ahora, el SVM es, de hecho, un poco más sofisticada de lo que sugiere este margen. En particular, si lo que estamos haciendo es utilizar un clasificador de margen amplio, el algoritmo será sensible a valores atípicos. Entonces, añadimos un ejemplo positivo adicional. Si añadimos este ejemplo, parece que para separar los datos con un margen amplio tendremos una barrera de d como esta línea magenta.



No parece ser muy natural cambiar mi barrera de decisiones de la línea negra a la línea magenta con base en ese único ejemplo. De manera que si el parámetro de regularización “C” fuera muy alto, entonces nuestra SVM será adecuada y cambiará la barrera de decisión de la línea negra a la magenta. Pero si “C” fuera razonablemente baja; es decir, si utilizaras “C” con un valor no muy alto entonces conservarías esta barrera de decisión negra.

Por supuesto, si los datos no fueran separables linealmente, la SVM también hará lo correcto.

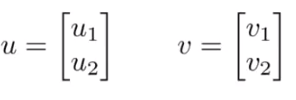


Esta imagen de un clasificador de márgenes amplios es la imagen que nos da el mejor entendimiento para el caso en el que el parámetro de regularización “C” es muy alto. Como recordatorio, “C” juega un papel similar a 1 sobre «lambda», donde «lambda» es el parámetro de regularización que utilizamos anteriormente. Será sólo si 1 sobre «lambda» es mayor o si «lambda» es muy pequeño que obtendremos barreras de decisión como esta magenta. Pero, en la práctica, cuando aplicamos máquinas de vector de soporte y “C” no es muy alta, podemos hacer un mejor trabajo ignorando valores atípicos como estos y aplicando cosas razonables, aún si los datos no son separables linealmente.

Más adelante, cuando hablemos de oscilación y varianza en el contexto de máquinas de soporte vectorial, las compensaciones que involucran el parámetro de regularización quedarán más claras. Espero que esto te de un mejor entendimiento de las funciones de la máquina de vector de soporte como clasificadores de márgenes amplios que intentan separar los datos con el mayor margen posible. Técnicamente, esta imagen sólo es verdadera cuando el parámetro “C” es muy alto y es una manera útil de visualizar una máquina de vector de soporte.   
  
En este video nos falto mencionar cómo nos lleva el problema de optimización que escribimos en estas diapositivas a un clasificador de márgenes amplios. No lo expliqué en este video, pero en el siguiente presentaré rápidamente las matemáticas que están detrás de esto para explicar un razonamiento independiente de cómo nos lleva el problema de optimización que escribimos al clasificador de margen amplio.

# Desarrollo matemático del algoritmo

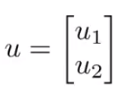
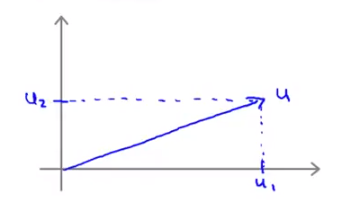
En este video, me gustaría contarles un poco acerca de las matemáticas que están detrás de una clasificación de margen amplio.   
  
  
Para iniciar, permítanme recordarles algunas propiedades de cómo se verían los productos internos de un vector[[1]](#footnote-0).   
Digamos que tenemos dos vectores “U” y “V” que se ven así



y ambos son vectores bidimensionales. Observemos cómo se ve “U” transpuesto de “V”. “U” transpuesto de “V” también es llamado productos internos entre los vectores “U” y “V”.

Utilizaré un vector bidimensional para dibujarlo.

En el eje horizontal, el trazado toma el valor que tenga u1 y en el eje vertical la altura del trazo será el valor que tenga u2, que es el segundo componente del vector “U”.

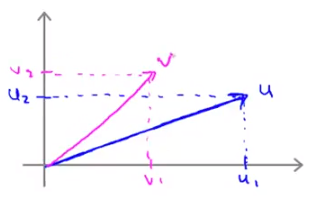
 

Una cantidad que nos sería útil es la norma del vector “U”.

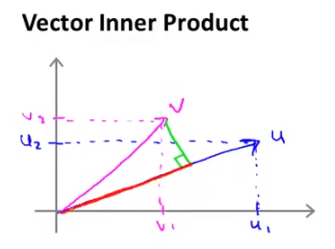
Estas de la derecha y la izquierda son barras dobles que denotan la norma o longitud del vector “U”; es decir, la distancia euclidiana del vector “U”.



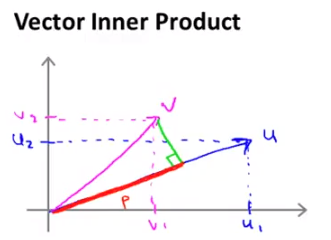
Este teorema de Pitágoras es igual a la raíz cuadrada de “U”1 cuadrada más “U”2 cuadrada. Esta es la longitud del vector “U” expresada como número real. ¿Cuál es la longitud de esta línea o la longitud de el vector? ¿Cuál es la longitud de esta flecha que dibujé para ilustrar la norma de “U”?   
  
Ahora, regresemos un poco y veamos el vector “V” porque queremos calcular el producto interno. “V” es otro vector con los valores “V”1 y “V”2 y “V” se verá así:



Ahora, regresemos para ver cómo calcular el producto interno entre “U” y “V”. A continuación les diré cómo hacerlo. Tomaremos el vector “V” y lo proyectaremos al vector “U”. Tomaré la proyección ortogonal o de 90 grados y la proyectaré en “U” así.



Bajaré una línea aquí. Después, mediremos la longitud de esta línea roja que dibujé aquí. Designaré con una “P” la longitud de esta línea roja. “P”:



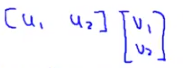
Entonces P, es la longitud o la magnitud de la proyección del vector “V” en el vector “U”. Lo voy a escribir aquí:

*Entonces, “P” es la longitud de la proyección del vector “V” en el vector “U”.*

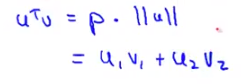
Es posible que el producto interno “U” transpuesta de “V” sea igual a “P” multiplicado por la norma o la longitud del vector “U”.



Esta es una manera de calcular el producto interno. Si desarrollas la geometría y resuelves cuál es “P” y cuál es la norma de “U”, podrás obtener el mismo resultado que con la otra manera de calcular el producto interno que es tomar “U” transpuesta de “V” y trasponer “V” como “U1” y “U2”; es decir, como una matriz de uno por dos, por 1 por V.

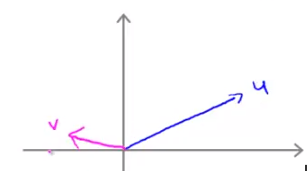


Esto debería darte como resultado “U”1, “V”1 más “U”2, “V”2.

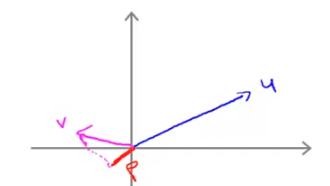


El hecho de que estas dos fórmulas arrojen los mismos resultados es un teorema algebraico. Por cierto “U” transpuesta de “V” es igual a “V” transpuesta de “U” de manera que el proceso es el mismo, pero al revés. En vez de proyectar “V” en “U”, puedes proyectar “U” en “V” y seguir con el mismo proceso pero con las columnas de “U” y “V” invertidas. Deberías poder obtener el mismo número, cuál sea, para ambas.

Aclararé lo que está pasando en esta ecuación: la norma de “U” es un número real al igual que “P”. “U” transpuesta de “V” es la multiplicación regular de dos números reales: la longitud de “P” por la norma de “V”.   
  
Un último detalle: si observas la norma de “P”, “P” está firmado y puede ser positivo o negativo. Explicaré esto. Si “U” y “V” son vectores que se ven así:

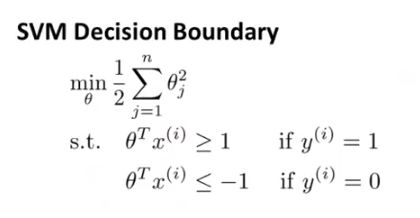


si el ángulo entre “U” y “V” es mayor a 90 grados, entonces, lo que obtendré si proyecto “U” en “V” es una proyección con este aspecto:



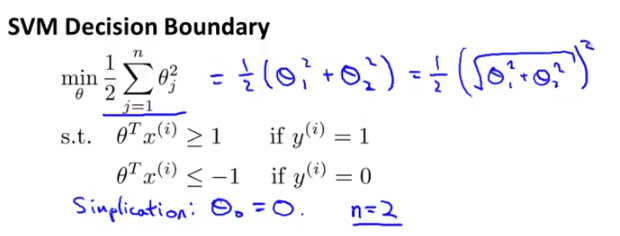
Esta es la longitud de “P”. Y, en este caso, de nuevo tendré que “U” transpuesta de “V” es igual a “P” por la norma de “U”, excepto por que en este ejemplo “P” será negativa.   
  
Así que los productos internos, si el ángulo entre “U” y “V” es menor a 90 grados, entonces “P” es la longitud positiva de esta línea roja, mientras que si el ángulo es mayor a 90 grados, “P” será la longitud negativa de estos segmentos de línea. Por lo tanto, el producto interno entre dos vectores puede ser negativo cuando el ángulo entre ellos es mayor a 90 grados. Así es como funcionan los productos internos de los vectores.

Utilizaremos las propiedades del producto interno de los vectores para intentar entender el objetivo de optimización de la máquina de vector de soporte. Aquí tenemos el objetivo de optimización para la máquina de vector de soporte que utilizamos antes.



En esta diapositiva, haré una simplificación sólo para facilitar el análisis del objetivo. Lo que haré es ignorar los términos de intersección. Ignoraremos «theta» 0 y la igualaremos a 0. Para facilitar el dibujo, pondré “N”, el número de variables, igual a 2. Por lo tanto, tenemos sólo dos variables, “X”1 y “X”2.

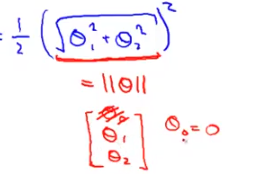
Ahora, veamos la función objetiva o el objetivo de optimización de la SVM. Aquí tenemos sólo dos variables donde “N” es igual a 2. Esto se puede expresar como un medio de «theta» 1 cuadrada más «theta» 2 cuadrada porque sólo tenemos dos parámetros: «theta» 1 y «theta» 2. Lo que haré a continuación es volver a escribir esto. Lo escribiré como un medio por el cuadrado de la raíz cuadrada de «theta» 1 más «theta» 2.



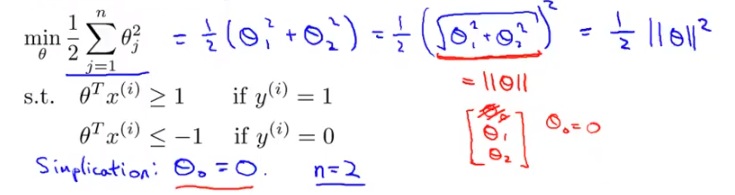
La razón por la cual puedo hacer esto es porque para cualquier número W, el cuadrado de la raíz cuadrada de “W” es igual a “W”. La raíz cuadrada al cuadrado debe darte el mismo resultado.



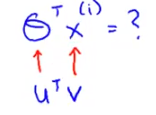
Puedes darte cuenta de que este término de adentro del paréntesis es igual a la norma o la longitud del vector «theta». A lo que me refiero con esto es a que si escribo el vector «theta» como «theta» 1, «theta» 2, entonces el término que subrayé en rojo (“P”) es exactamente la longitud o la norma del vector «theta», relativa a la definición de la norma del vector que teníamos en la diapositiva anterior.   
  
De hecho, esto es igual a la longitud del vector «theta» sin importar si lo escribes como «theta» 0, «theta» 1, «theta», 2, etc; si «theta» 0 igual a 0, como asumimos anteriormente o como la longitud de «theta» 1, «theta» 2. Para esta diapositiva ignoraremos «theta» 0. Entonces, trataremos «theta» o escribiremos la «theta» normal solamente con «theta» 1 y «theta» 2. Los cálculos resultan correctos de cualquier manera ya sea si incluimos «theta» 0 o si no. Esto no importará para el resto de nuestra derivada.



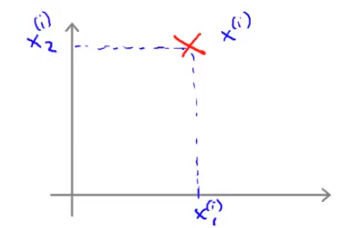
Finalmente, esto indica que mi objetivo de optimización es igual a un medio de la norma de «theta» cuadrada.



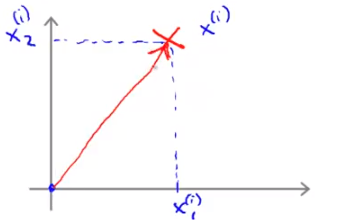
Lo que está haciendo la máquina de vector de soporte en el objetivo de optimización es minimizar la norma cuadrada o la longitud cuadrada del parámetro vector «theta».   
  
Ahora, me gustaría mirar estos términos, «theta» transpuesta de “x” y entender mejor lo que hacen. Con un parámetro vector «theta» dado y un ejemplo “X” dado, ¿cuál es el resultado? En la diapositiva anterior nos dimos cuenta de cómo se ve “U” transpuesta de “V” con diferentes vectores “U” y “V”. Ahora tomaremos esas definiciones en donde «theta» y “X(i)” jugarán el papel de “U” y “V” y veremos cómo se ve nuestro trazo.

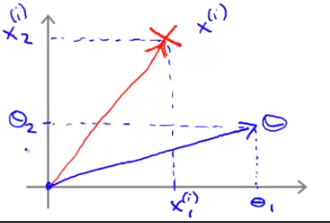


Digamos que sólo tengo en cuenta un ejemplo de entrenamiento o que sólo tengo un ejemplo positivo expresado con esta cruz roja y denominado “X(i)”. Lo que significa esto es que tracé   
en el eje horizontal un valor “X(i)1” y en el eje vertical “X(i)2”. Así es como trazo mis ejemplos de entrenamiento.

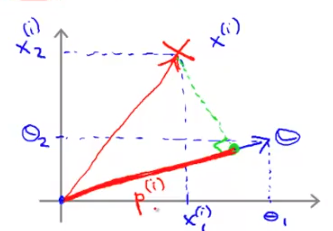


Aunque no hemos estado pensando en esto como un vector, realmente es un vector desde el origen, (0,0) hasta la ubicación de este ejemplo de entrenamiento.

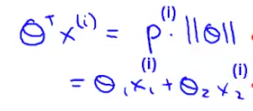


Ahora digamos que tenemos un parámetro vector «theta» y que lo trazaremos también como vector. Si trazo «theta» 1 y «theta» 2   


¿Asi qué cuál es el producto interno de «theta» transpuesta de “X(i)”? Con el método anterior, podemos calcular esto tomando el ejemplo y proyectándolo en mi parámetro vector «theta». Luego veré la longitud de este segmento que estoy marcando con rojo y al que llamaré “P” superíndice “i” para denotar que es la proyección del ejemplo de aprendizaje en el parámetro vector «theta».

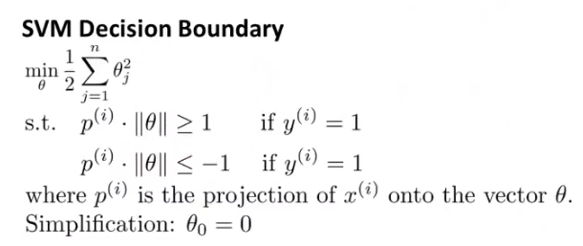


Lo que tenemos que es «theta» transpuesta de “X(i)” es igual a lo que teníamos en la diapositiva anterior; es decir, será igual a por la longitud o la norma del vector «theta». Esto es igual a «theta» 1 “x” 1 más «theta» 2 “x” 2. Cada uno de estos métodos es igualmente válido para calcular el producto interno entre «theta» y “X(i)”.

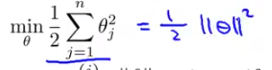


Ok. ¿En dónde nos deja esto? Lo que significa esto es que estas limitaciones; es «theta» transpuesta de “X(i)” sea más grande o igual a 1 o menor que 1, es que puede remplazar el uso de las limitaciones para que “P(i)” por “X” sea mayor o igual a 1. Porque «theta» transpuesta de “X(i)” es igual a “P(i)” por la norma de «theta».

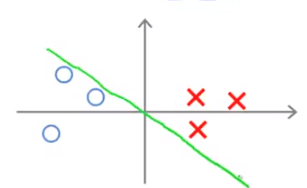
Lo que obtenemos cuando escribimos esto en nuestro objetivo de optimización es, en vez de «theta» transpuesta de “X(i)”, tengo “P(i)” por la norma de «theta».



Como recordatorio, mencionamos antes que este objetivo de optimización puede escribirse como un medio veces la norma de «theta» al cuadrado.

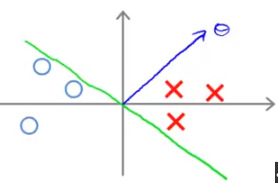


Ahora, consideremos el ejemplo de entrenamiento que está abajo y continuemos utilizando la simplificación de «theta» 0 igual a 0. Veamos qué barrera de decisiones elige la máquina de vector de soporte.   
  
Aquí hay una opción: digamos que la máquina de vector de soporte eligiera esta barrera de decisión:



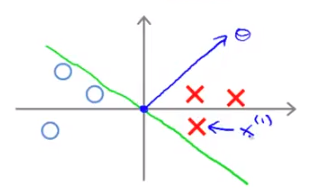
No es una buena decisión porque tiene márgenes muy pequeños; por lo tanto, la barrera de decisión pasa muy cerca de los ejemplos de entrenamiento.   
  
Veamos por qué la máquina de vector de soporte no hace esto.

Para esta elección de parámetros es posible mostrar que el parámetro vector «theta», en realidad, cruza a 90 grados con la barrera de decisión. Entonces, la barrera de decisión verde corresponde al parámetro vector «theta» que apunta en esa dirección.

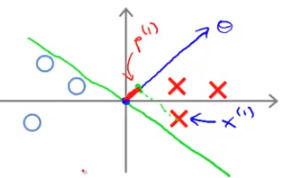


Por cierto, la simplificación de que «theta» 0 es igual a 0 sólo significa que la barrera de decisión debe pasar por el origen (0,0).

Ahora, veamos qué implica esto en el objetivo de optimización. Tomemos este ejemplo y digamos que es mi primer ejemplo “X”1.

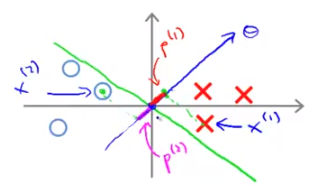


Aquí está la proyección de este ejemplo en el parámetro «theta», Esta es la proyección. Este segmento de línea roja. es igual a “P”1.

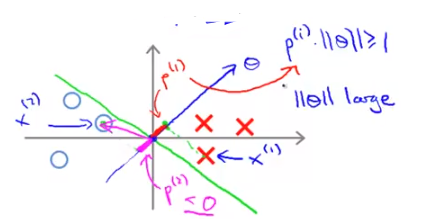


Esto tendrá un resultado pequeño, ¿cierto?

De manera similar, tomaré este ejemplo de aquí es “X”2 o mi segundo ejemplo. Observaré la proyección de este ejemplo en «theta». Entonces, permítanme dibujarlo en magenta.   
Este pequeño segmento de línea magenta será “P”2. Es decir, la proyección del segundo ejemplo en la dirección del parámetro vector «theta» que se extiende así.

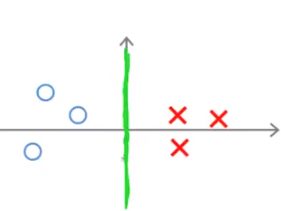


Este pequeño segmento de línea proyectada se vuelve muy pequeño. “P”2, de hecho, será un número negativo porque va en la dirección opuesta. Este vector tiene un ángulo mayor a 90 grados y con el parámetro vector «theta» será menor a 0.   
  
A lo que llegamos es a que estos términos “P(i)” serán números muy pequeños. Si miramos al objetivo de optimización veremos que para los ejemplos positivos necesitamos que “P(i)” por la norma de «theta» sea mayor que cualquiera de las dos, pero si “P(i)” o “P”1 es muy pequeño, significa que necesitamos que la norma de «theta» sea grande ¿cierto?

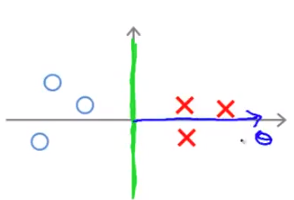


Si “P”1 de «theta» es pequeña, pero queremos que “P”1 por la norma de «theta» sea mayor a 1, la única manera de lograrlo, la única manera de obtener una ganancia alta para estos dos números si “P”1 es pequeña, es ajustar la norma de «theta» con un valor alto.   
  
De manera similar, para nuestros ejemplos negativos necesitamos que “P” 2 por la norma de «theta» sea menor o igual a menos 1. Ya vimos en este ejemplo que “P”2 será un número negativo pequeño. La única manera de obtener este número es si la norma de «theta» es alta. Lo que estamos haciendo en el objetivo de optimización es intentar encontrar una configuración de los parámetros en la que la norma de «theta» sea pequeña. Esto no parece ser una buena dirección para el parámetro vector «theta».

En contraste, veamos una barrera de decisión diferente. Digamos que la SVM elige   
esta barrera de decisión.

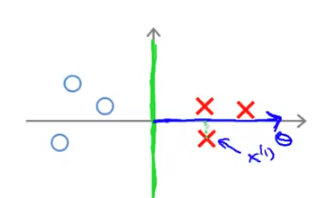


El panorama será muy diferente. Si esta es la barrera de decisión, esta es la dirección correspondiente de «theta».

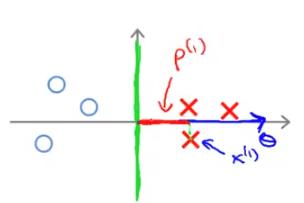


La barrera de decisión es esta línea vertical y es posible mostrar, utilizando álgebra lineal, que la manera de obtener esta barrera de decisión verde es tener el vector «theta» a 90 grados con respecto a ella.

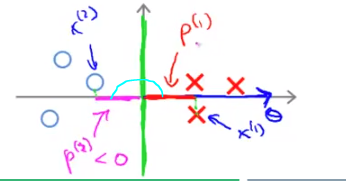
Ahora, veamos la proyección de estos datos en el vector “x”. Digamos que, igual que antes, este ejemplo será “X”1.



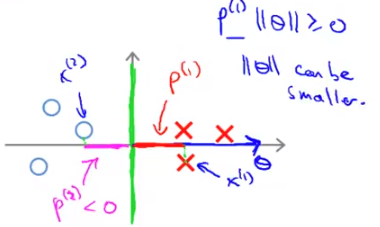
Cuando proyecto esto en x o en «theta», lo que encontraré es “P”1.



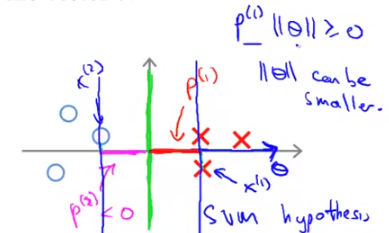
Esta longitud es “P”1.   
  
El otro ejemplo es “X”2. Si hago la misma proyección en “X”2 me encuentro con que esta longitud, “P”2, es menor a 0.



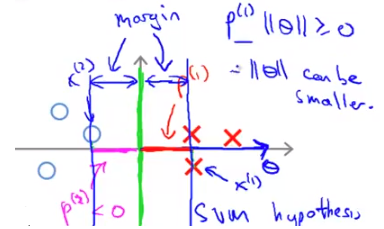
Puedes darte cuenta ahora de que las longitudes proyecciones “P”1 y “P”2 serán mucho mayores. Si aún necesitamos aplicar estas limitaciones, que “P”1 de la norma de «theta» sea igual o mayor que 1, la norma de «theta» podrá ser menor, porque “P”1 es mucho mayor ahora.



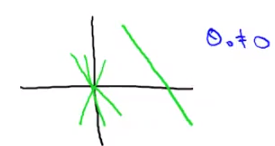
Lo que significa esto es que al elegir la barrera de decisión que se muestra a este segundo ejemplo en vez del anterior, la SVM puede hacer que la norma del parámetro «theta» sea mucho menor. Así que la norma de «theta» será menor y, por lo tanto, la norma cuadrada de «theta» también será menor. Por esto, la SVM elegiría esta hipótesis.   
  
Así es como la SVM origina este efecto de clasificación de márgenes amplios. Tomando como referencia esta línea verde o esta hipótesis verde, donde queremos que la proyección de los ejemplos positivos y negativos en «theta» sea grande. La única manera de lograrlo es si, alrededor de esta línea verde hay un margen amplio o un espacio grande que separa los ejemplos negativos de los positivos.



La magnitud de este margen es exactamente los valores de “P”1, “P”2, “P”3, etc. y al hacer que este margen sea mayor, la SVM puede terminar con un valor más pequeño de «theta» y es justo lo que queremos hacer en el objetivo.



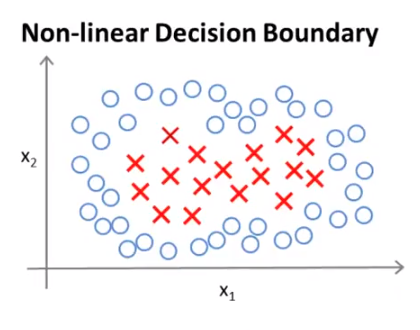
La razón por la cual la máquina termina con clasificadores de márgenes amplios es porque intenta maximizar la norma de estos valores P(i), que son la distancia de los ejemplo de entrenamiento a la barrera de decisión.   
  
Finalmente, hicimos toda esta derivada utilizando la simplificación de «theta» 0 igual a 0. El efecto de esto, como mencioné antes, es que si «theta» 0 es igual a 0, estaremos generando barreras de decisión que pasan por el origen. Las barreras de decisión pasarán, así, por el origen. Si permites que «theta» 0 sea mayor que 0, lo que obtendremos serán barreras de decisión que no pasan por el origen, como esta que acabo de dibujar.



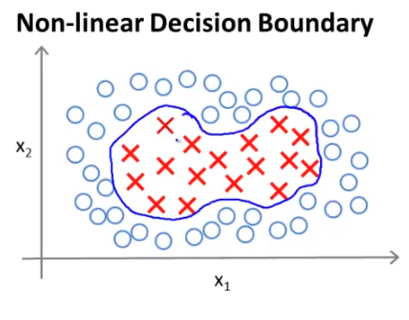
No haré la derivada completa para esto; sin embargo, resulta que la misma prueba de márgenes amplios funciona del mismo modo. Hay una generalización de este argumento que planteamos hace un momento que nos dice que aún cuando «theta» 0 no es 0, lo que intenta hacer la SVM cuando tienes este objetivo de optimización, que corresponde al caso de cuando C es muy grande. Es posible mostrar que cuando «theta» 0 no es igual a 0, esta máquina de soporte vectorial aún intenta encontrar un separador de márgenes amplios entre los ejemplos positivos y negativos. Esto explica la función de esta máquina de vector de soporte como clasificador de márgenes amplios.   
  
En el siguiente video empezaremos a hablar de cómo tomar algunas de estas ideas de SVM y aplicarlas para construir clasificadores no lineales.

# Kernels

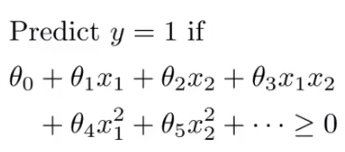
En este video me gustaría empezar a adaptar las máquinas de soporte vectorial para desarrollar clasificadores no lineales complejos.   
  
La técnica principal para hacer esto es una técnica llamada kernels. Veamos qué son estos kernels y cómo podemos utilizarlos.   
  
Si tienes un conjunto de entrenamiento que luce así:



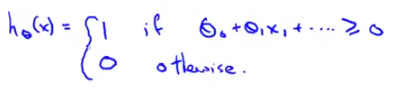
Y quieres encontrar una frontera de decisión no lineal para distinguir entre los ejemplos negativos y los positivos, quizá una frontera parecida a esta:



Una manera de lograrlo es obtener un conjunto de variables polinómicas complejas. Es decir, un conjunto de variables como este:



Entonces tendremos una hipótesisque predice 1 si «theta» 0 más «theta» 1”x”1 más el resto de las variables polinómicas es mayor que 0 y, de lo contrario, predice 0.



Otra manera de escribir esto, con la que introduciré una nueva notación que utilizaré después, es que podemos pensar en una hipótesis que calcula una frontera de decisión utilizando lo siguiente:

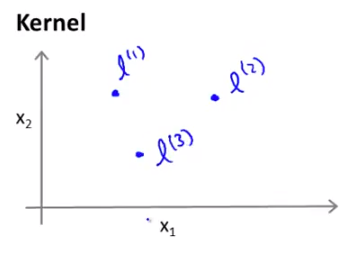


Donde esta nueva notación “f”1, “f”2, “f”3, etc., denota las nuevas variables que estoy calculando. Ahora, “f”1 es igual a “X”1, “f”2 es igual a “X”2 y “f”3 es igual al tercer término “X”1 ”X”2, “f”4 es igual a “X”1 cuadrada y f5 es igual a “X”2 cuadrada, etc.

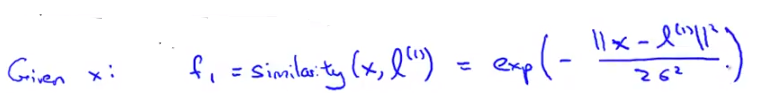


Ya vimos antes que obtener estos polinomios de alto orden es una manera de obtener muchas variables. La cuestión es si existe una elección distinta de variables o una elección de variables mejor que estos polinomios de alto orden, porque no tenemos claro si estos polinomios de alto orden son lo que queremos. Cuando hablamos acerca de la visión computacional hablamos de cuando la imagen de entrada tiene muchos pixeles, también vimos cómo utilizar polinomios de alto orden se vuelve computacionalmente caro por todos estos términos de polinomios de alto orden.   
  
Entonces ¿existe una elección de variables diferente o mejor que podamos utilizar en este tipo de formulación de hipótesis?

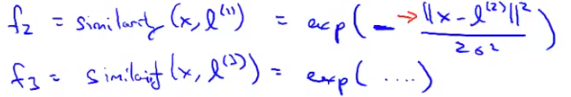
Aquí hay una idea de cómo podemos definir las nuevas variables “f”1, “f”2, “f”3, etc. En esta línea definiré sólo tres nuevas variables, pero en problemas reales podemos definir muchas más. Lo que haré en esta fase de variables tendremos “X”1, y “X”2 y dejaremos “X”0 fuera de la ecuación, y elegiré algunos puntos manualmente. A este puntos les llamaré y luego elegiré otro punto y lo llamaré y luego un tercer punto y lo llamaré .



Por ahora digamos que sólo elegiré estos tres punto manualmente. Llamaré estos tres puntos “puntos de referencia”. Tengo el punto de referencia uno, dos y tres. Lo que haré es definir mis variables como sigue: con un ejemplo “X” definiré mi primera variable, “f”1, como la medida de la similaridad entre el ejemplo de entrenamiento “X” y el primer punto de referencia. La fórmula específica que utilizaré para medir la similaridad es “E” elevada a menos la longitud de “X” menos “L”1 cuadrada, dividida entre 2 sigma al cuadrado.   
3Dependiendo de si viste el video opcional anterior, entenderás, o no, que esta notación es la longitud del vector “W”. Esta “X” menos “L”1 es, de hecho, la distancia euclidiana cuadrada;   
es decir, la distancia euclidiana entre el punto “X” y el punto de referencia . Hablaremos de esto más adelante.



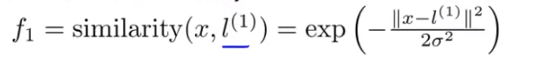
Esta fue mi primera variable. Mi segunda variable, “f”2 será una variable de similaridad que mide qué tan similares son “X” y “L”2. Definiremos esto con la siguiente variable: “E” elevado a menos la raíz cuadrada de la distancia euclidiana entre “X” y el segundo punto de referencia que se expresa en este numerador, dividido entre 2 sigma cuadrada. De igual manera, “f”3 es la similaridad entre “X” y “L”3 que es igual a una fórmula similar.



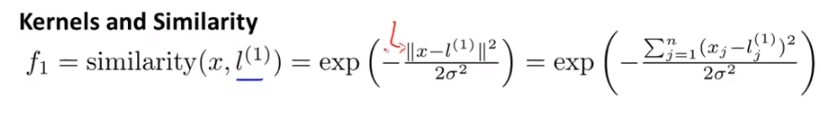
El término matemático para esta variable de similaridad será función de kernel. El kernel específico que utilizaré aquí se llama kernel Gaussiano.   
  
Esta fórmula o esta elección particular de la variable de similaridad se llama kernel Gaussiano. A nivel abstracto, estas variables de similaridad se llaman kernels y podemos tener diferentes variables de similaridad. El kernel específico que estoy explicando ahora es el kernel Gaussiano. Veremos otros ejemplos de otros kernels pero, por ahora, pensaremos en estos como variables de similaridad.   
  
Entonces, en vez de escribir la similaridad entre “X” y “L”, a veces también escribiremos esto como kernel, que denotaremos con una “k” minúscula, sobre “x” y uno de mis puntos de referencia “L”.



Veamos qué hacen los kernels en realidad y por qué este tipo de variables de similaridad o este tipo de expresiones tienen sentido. Tomaré mi primer punto de referencia; el punto de referencia “L”1 que es uno de los puntos que elegí en de mi figura hace un momento.   
  
La similaridad del kernel entre “X” y “L”1 se obtiene con esta expresión.

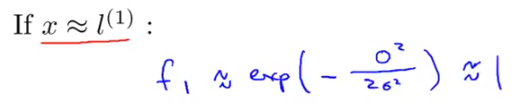


Para asegurarme de que estamos en el mismo canal acerca de cuál es término del numerador, les recuerdo que también se puede escribir como la sumatoria de “J” igual a 1 a la “n”.

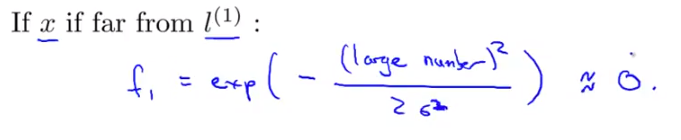


Esta es la distancia de los componentes entre el vector “X” y el vector “L”. En esta diapositiva, les reitero que ignoraré “X”0. Estoy ignorando el término “X”0 porque siempre es igual a 1.

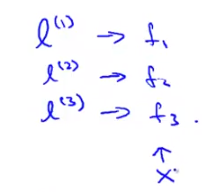
Porque, como sabes, esto es se calcula el kernel con la similaridad entre “X” y el punto de referencia. Ahora veamos qué hace esta variable. Supongamos que “X” es casi igual a uno de los puntos de referencia. La fórmula euclidiana para la distancia del numerador será casi igual a 0; es decir, la distancia entre “X” y “L” será casi igual a 0 y “f”1 será aproximadamente “e” a la menos 0 cuadrada en el numerador sobre 2 sigma cuadrada. Ahora, “e” a la menos 0 será casi igual a 1.



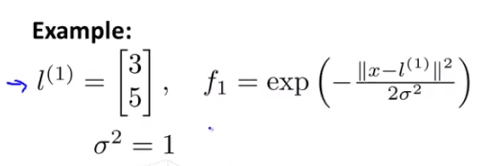
Pongo el signo de aproximación aquí porque la distancia puede variar del cero, pero si “X” es casi igual al punto de referencia “L”, entonces este término será casi igual a 0 y, por lo tanto, “f”1 será casi igual a 1.   
  
Por el contrario, si “X” está lejos de “L1”, esta primera variable “f1” será “e” a la menos un valor alto al cuadrado dividido entre dos «sigma» cuadrada y “e” a la menos un número alto será casi igual a 0.



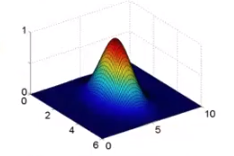
Lo que hacen estas variables es medir qué tan similar es “X” a uno de los puntos de referencia. La variable “f” será casi igual a 1 cuando “X” sea casi igual al punto de referencia y 0 o casi igual a 0 cuando “X” esté lejos del punto de referencia. Cada uno de estos puntos de referencia que escribí en la diapositiva anterior, “L1”, “L2” y “L3” cada una, define una nueva variable “f1”, “f2” y “f3”. Con el ejemplo de entrenamiento “X”, podemos calcular tres nuevas variable: “f1”, “f2” y “f3” con los tres puntos de referencia que acabo de escribir.



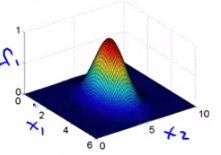
Primero, veamos esta función exponencial o esta variable de similaridad y tracemos algunas figuras para entender mejor cómo se representan.   
Para este ejemplo, digamos que tengo dos variables “x1 y “x2” y digamos que mi primer punto de referencia, “L1” está en la ubicación 3,5.



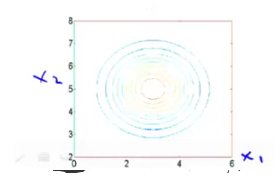
Ahora, supongamos que fijo sigma cuadrada igual a 1, por el momento. Si trazo esto, lo que obtendré será esta figura:



En el eje vertical, la altura de la superficie es el valor “f1” y abajo, en el eje horizontal están los ejemplos de entrenamiento como “x1” y “x2”.

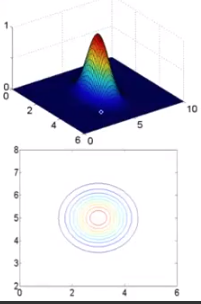


Con un ejemplo de entrenamiento dado, como este ejemplo de entrenamiento que muestra los valores de “x1 y “x2”, la altura sobre la superficie muestra el valor correspondiente de “f1”. Abajo tenemos la figura que había mostrado antes llamada gráfica de superficie donde “x1” está en el eje vertical y “x2” está en el eje horizontal. Esta figura de abajo es sólo la gráfica de superficie de la superficie 3D.

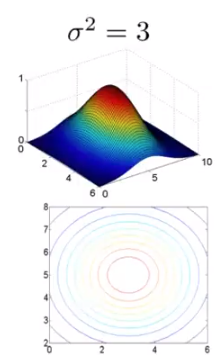


Puedes ver que cuando “x” es igual a [3 5] exactamente, “f1” toma el valor de 1, porque es su máximo y a medida que se aleja “x” esta variables asume valores cercanos al 0.   
  
Concluimos que esta variable “f1” mide qué tan cercana es “x” al primer punto de referencia y varía entre 0 y 1 dependiendo de qué tan cerca esté “x” al punto de referencia “L1”.   
  
Lo siguiente que me gustaría hacer en esta diapositiva es mostrar los efectos que se provocan al variar el parámetro sigma cuadrada. Sigma cuadrada es el parámetro del kernel Gaussiano y, a medida que varía, arroja resultados ligeramente distintos.   
  
Voy a fijar sigma cuadrada como 0.5 y ver qué obtengo. Cuando fijamos sigma cuadrada en 0.5 lo que encontraremos es que el kernel se ve similar, excepto por que el ancho del cono se hace más delgado.

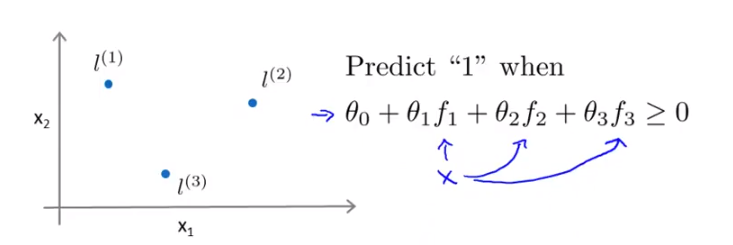




Los contornos también se hacen más pequeños. Iniciamos con sigma cuadrada igual a 0.5 y si inicias con “x” igual a 3, 5 pero, a medida que nos alejamos, la variable “f1” cae a 0 mucho más rápido. Por el contrario, si aumentamos sigma cuadrada y la fijamos como sigma cuadrada igual a 3, conforme me alejo de “L”, la medida de similitud ya no es tan sensible.



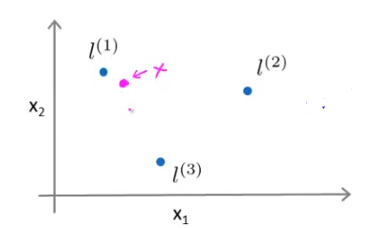
Si sigma cuadrada es grande, entonces te alejas de "L"1 el valor de la variable sigma al cuadrado caerá mucho más lentamente.   
  
Ahora, dada esta definición de variables, veamos qué fuentes de hipótesis podemos aprender.   
Con el ejemplo de entrenamiento “X”, calcularemos estas variables “f1”, “f2” y “f3” y nuestra hipótesis predecirá 1 cuando «theta» 0 más «theta»1, “f1” más «theta» 2, “f2” etc. es mayor que o igual a 0.



Para este ejemplo en particular, digamos que ya apliqué el algoritmo de aprendizaje y terminé con estos valores para los parámetros: «theta» 0 es igual a menos 0.5, «theta» 1 es igual a 1, «theta» 2 s igual a 1 y «theta» 3 es igual a 0.

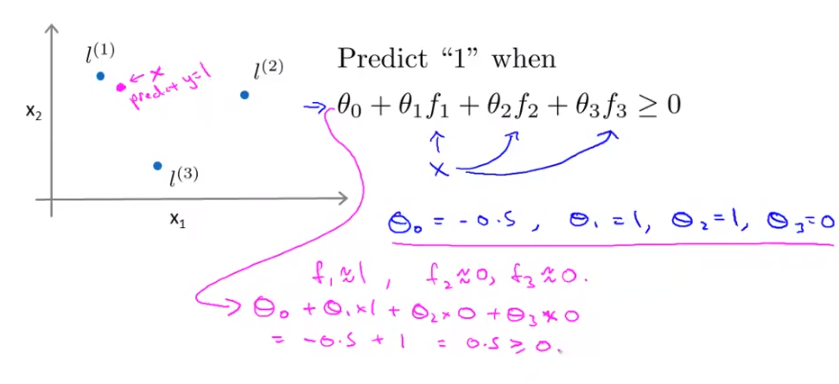


Me gustaría considerar qué sucede cuando tengo un ejemplo de entrenamiento que   
tiene su ubicación en este punto magenta, justo aquí arriba.

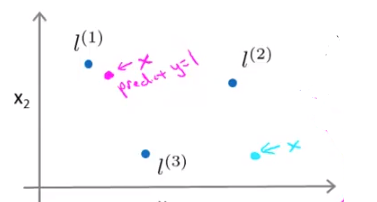


Supongamos que tenemos el ejemplo de entrenamiento “x”, ¿qué predecirá mi hipótesis?

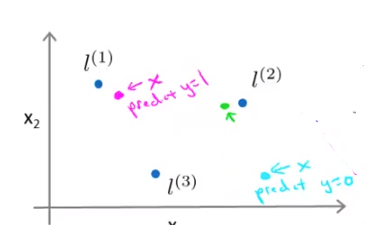
Analicemos esta fórmula: Si mi ejemplo de entrenamiento “x” es casi igual a “L1”, “f1” también será casi igual a 1 y si mi ejemplo de entrenamiento “x” está lejos de “L2” y “L3” “f2” será casi igual a 0 y “f3” también será igual a 0. Así que, si miro esa fórmula, tengo «theta» 0 más «theta» 1 por 1 más «theta» 2 por algún valor que quizá no sea exactamente 0, pero que está cerca, más «theta» 3 por algún valor cercano a 0. Y esto nos da menos 0.5 más 1 por 1 que es 1, etc., que es igual a 0.5, que es mayor o igual a 0. En este punto predeciré que “y” es igual a 1, porque esto es mayor que o igual a 0.



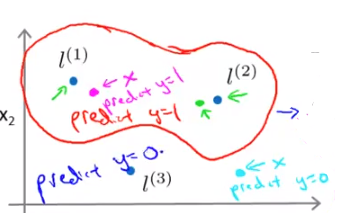
Ahora, tomemos un punto diferente. Digamos que tomo este punto y lo dibujo en un color diferente. De color cian. Si este fuera mi ejemplo de entrenamiento “x”, e hiciera un cálculo similar, encontraré que “f1”, “f2 y “f3” estarán cerca de 0.



Tenemos «theta» 0 más «theta» 1, “f1” más todo lo de más, que será igual a menos 0.5, porque «theta» 0 es menos 0.5 y “f1, “f2” y f3” serán cero. Esto será menos 0.5 y esto menor que 0. En este punto, predeciremos que “y” es igual a cero. Y si hace esto, esto tú mismo para un rango de puntos, asegúrate de entender que si tienes un ejemplo de entrenamiento casi igual a “L2”. A este punto también prediremos que “y” es igual a 1.



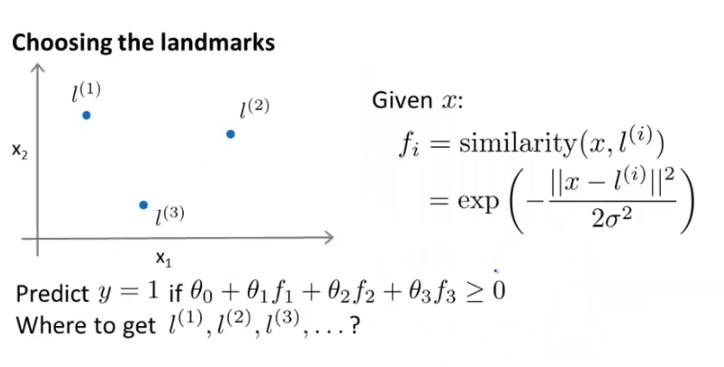
De hecho, lo que terminarás haciendo es, al ver esta frontera, este espacio, encontraremos que para los puntos cercanos a “L1” y “L2” tendremos una predicción positiva y para los puntos que están lejos de los puntos de referencia “L1” y “L2”, terminaremos prediciendo que la clase es igual a 0. Al final la frontera de decisión de esta hipótesis terminará siendo algo así:



En donde dentro de la frontera de decisión predeciremos que “y” es igual a 1 y fuera de ella predeciremos que “y” es igual a 0. Y entonces esto es de los puntos de referencia y de la variable de kernel podemos aprender fronteras de decisiones no lineares muy complejas, como las que acabo de trazar aquí, con las que predecimos un resultado positivo cuando estamos cerca de alguno de los puntos de referencia y predecimos un resultado negativo cuando estamos lejos de los puntos de referencia. Esto es parte del concepto de kernels y de cómo podemos utilizarlos con la máquina de soporte vectorial para definir las variables adicionales utilizando puntos de referencia y variables de similaridad para aprender clasificadores no lineales más complejos.   
  
Espero que esto te dé un sentido o una idea de los kernels y de cómo los podemos utilizar para definir nuevas variables para la máquina de soporte vectorial.   
  
Pero todavía hay algunas preguntas que no hemos respondido. Una es ¿cómo obtenemos estos puntos de referencia? ¿cómo los elegimos? Otra es, ¿qué otras variables de similaridad, si las hay, pueden utilizarse además del kernel Gaussiano, que es de la que hablamos? En el video siguiente responderemos estas preguntas y pondremos todo junto para mostrar cómo pueden ser eficientes las máquinas de soporte vectorial con los kernels para aprender variables complejas no lineales.

## Kernels II

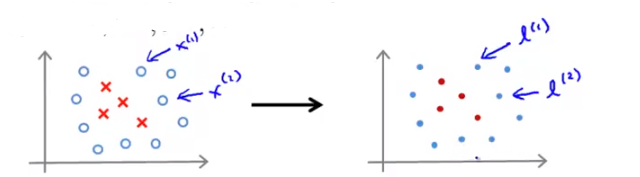
En el video anterior hablamos de los kernels y de cómo los podemos utilizar para definir nuevas variables para la máquina de soporte vectorial. En este video me gustaría explicar algunos detalles faltantes y decir unas palabras acerca de cómo utilizar estas ideas en la práctica, tales como lo concerniente a la compensación del oscilación y la varianza en la máquina de soporte vectorial.   
  
En el video anterior también hablé acerca el proceso de elegir puntos de referencia como “L1”, “L2” y “L3” y de cómo esto nos ayudarían a definir la variable de similaridad también llamada kernel. En este ejemplo en particular, esta variable de similaridad es un kernel Gaussiano



Lo anterior nos ayudó a construir esta formulación de una variable de hipótesis. Pero, ¿de dónde sacamos estos puntos de referencia? ¿De dónde sacamos “L1”, “L2” y “L3”? Parece que para los problemas de aprendizaje complejos necesitaremos más puntos de referencia que estos tres que consideramos por elección manual.   
  
En la práctica los puntos de referencia se eligen de la siguiente manera: En un problema de aprendizaje automático dado, tenemos un conjunto de datos con ejemplos positivos y negativos. Cuando tomo estos ejemplos, por cada ejemplo de entrenamiento que tenga, pondré un punto de referencia en la misma posición o la misma ubicación que los ejemplos de entrenamiento.



Si tengo un ejemplo de entrenamiento “x1”, entonces elegiré mi primer punto de referencia en la misma ubicación que mi primer ejemplo de entrenamiento. Y si tengo otro ejemplo de entrenamiento “x2”, pondré el segundo punto de referencia en la ubicación de mi segundo ejemplo de entrenamiento:

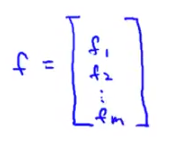


En la figura de la derecha utilicé puntos rojos y azules como representación. El color de los puntos en la figura de la derecha no es importante. Entonces, utilizando este método terminaré con un número “m” de puntos de referencia “L1”, “L2”, etc., hasta “L(m)”, si tengo “m” ejemplos de entrenamiento con un punto de referencia por ubicación de cada uno de mis ejemplos de entrenamiento. Esto es bueno porque indica que mis variables medirán qué tan cerca está un ejemplo de uno de los puntos que dibujé en mi conjunto de entrenamiento.

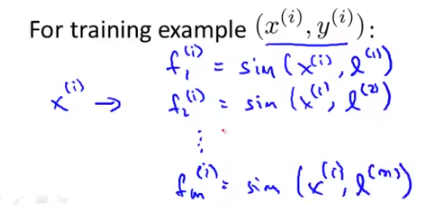
Ahora, escribiré este esbozo un poco mejor tomando mis ejemplos de entrenamiento como referencia y eligiendo la ubicación de mis puntos de referencia para que sea exactamente cercana a mis “m” ejemplos de entrenamiento.   
Cuando tienes un ejemplo “x”, (que en este caso puede ser algo del conjunto de entrenamiento, en el conjunto de validación cruzada o en el de prueba) con este ejemplo “x” calcularemos las variables “f1”, “f2”, etc., donde “L1” es igual a “x1”, etc.,



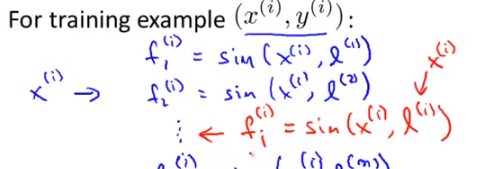
y estas me darán un vector de variables que denotaré con una “f”. Tomaré estas “f1”, “f2”, etc., y las agruparé en un vector de variables que incluya todas estas, hasta “fm”.



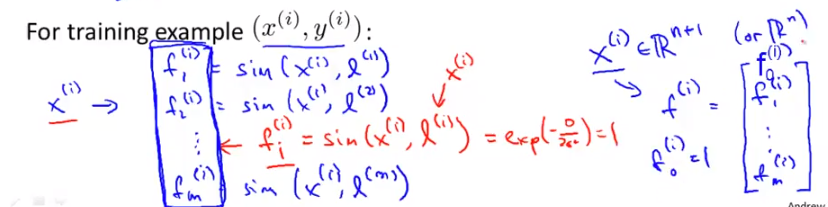
Sólo por costumbre, si queremos podemos añadir una variables adicional “f0” que siempre es igual a 1. Esta juega un rol muy similar al que teníamos anteriormente para “x0”, que era nuestro interceptor.   
  
Entonces, por ejemplo, si tenemos un ejemplo de entrenamiento “x(i), y(i)”, la variable que calcularíamos para este ejemplo de entrenamiento serán las que siguen: mapearemos “x(i)” a “f1(i)”, la similaridad de “f2(i)” es igual a la similaridad entre “x(i)” y “L2”, etc. hasta “fm(i)” igual a la similaridad de “x(i)” y “L(m)”.



En algún lugar a lo largo de esta lista; es decir, en el componente i, tendré un componente característico “f” subíndice “i(i)” que será la similaridad entre “x(i)” y “L(i)” en la que “L(i)” es igual a “x(i)” . “fi(i)” será la similaridad entre x y “fi(i)” misma.

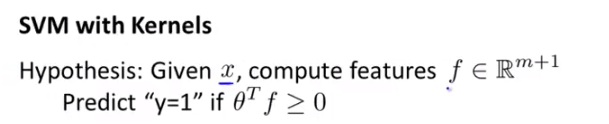


Si estás utilizando un kernel Gaussiano, esto es “e” a la menos 0 sobre 2 «sigma» cuadrada, por lo que será igual a 1, y es correcto. Una de mis variables para este ejemplo de entrenamiento será igual a 1. De manera similar a lo que tenemos arriba, puedo tomar todas estas variables “m” y agruparlas en un vector de variables. En vez de representar mi ejemplo utilizando “x(i)”, que es este vector R(n) más R(n)1 vector dimensional (dpendiendo de cómo hayas llamado tu interceptor, será R(n) o R(n) más 1). Ahora, podemos representar el conjunto de entrenamiento utilizando, en cambio, el vector de variables “f”. Escribiré “f” superíndice “i” que tomará todas estas cosas y las agrupará en un vector. “f1(i)” hasta ·fm(i)”



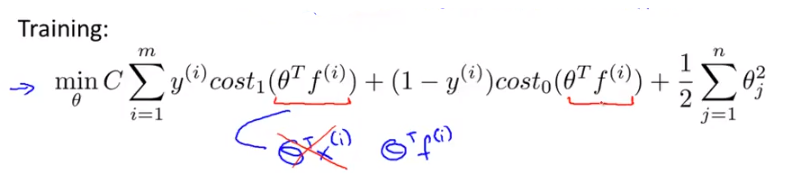
y si quieres añadir “f0(i)” donde “f0(i)” es igual a 1. Este vector de aquí me dará mi nuevo vector de variables con el que representaré mi ejemplo de entrenamiento.

A continuación explicaré cómo utilizar una máquina de soporte vectorial con estos kernels y variables de similaridad. Si ya tienes un conjunto de parámetros teta aprendidos o si ya le has dado un valor a “x” y quieres hacer una predicción, lo que harás es calcular las variables “f”, que son ahora un vector “R(m)” más 1 vector dimensional.



Tenemos “m” aquí porque tenemos “m” ejemplos de entrenamiento y, por lo tanto “m” puntos de referencia. Lo que haremos será predecir 1 si teta transpuesta de “f” es mayor que o igual a 0. Entonces teta traspuesta de “f” es igual a teta 0, “f0” más teta 1, “f1” más todo esto más teta “m”, “f(m)”. Ahora, mi parámetro vector teta también será un vector “m” más 1 vector dimensional.



Tenemos “m” aquí porque el número de puntos de referencia es igual al tamaño del conjunto de entrenamiento. De manera que “m” es el tamaño del conjunto de entrenamiento y ahora, el parámetro vectorial teta será “m” más 1 dimensional.   
  
Así es como se hace una predicción si ya tienes un valor fijo para el parámetro teta. ¿Cómo obtienes el parámetro teta? Lo puedes hacer utilizando el algoritmo de aprendizaje de las SVM. Específicamente, lo que puedes hacer es resolver este problema de minimización minimizando el parámetro teta de C por esta función de costo que teníamos antes. Ahora, en vez de hacer predicciones utilizando teta transpuesta de “x(i)” utilizando nuestras variables originales, “x(i)” tomamos las variables “x(i)” y las remplazamos por nuevas variables para utilizar teta transpuesta de “f(i)” para hacer una predicción en los ejemplos de entrenamiento “i” y vemos que en ambos casos podemos obtener los parámetros de la máquina de soporte vectorial resolviendo este problema de minimización.   


Un último detalle es que para este problema de optimización tenemos “n” igual a “m” variables; es decir el número efectivo de variables que tenemos, que es la dimensión de “f”. Entonces, “n” será igual a “m”. Si gustas, puedes pensar en esto como una suma. Realmente es una suma de “J” igual a 1 a “m”. Una manera de pensar en “n” es verla como igual a “m”, porque si “f” no es una variable nueva, entonces tendremos “m” más 1 variables donde más 1 viene del interceptor. Aquí, seguimos haciendo la suma de “J” igual a 1 a “n”, porque, al igual que en nuestros videos anteriores de la regularización, aún no hemos regularizado el parámetro teta cero, que es la razón de la suma de “J” igual a 1 a “m” en vez de “j” igual a cero a “m”.



Esto que expliqué fue la el algoritmo de aprendizaje de la máquina de soporte vectorial. Hay un detalle matemático adicional que debería mencionar: La manera en la que se implementa una máquina de soporte vectorial este último término se desarrolla de manera distinta. No necesitas saber este último detalle para utilizar máquinas de soporte vectorial. De hecho, las ecuaciones que están escritas aquí deberían darte todo el conocimiento que necesitas. Pero en la manera en la que se implementa una máquina de soporte vectorial, otra manera de escribir el término, la suma de “j” de teta “j” cuadrada es como teta transpuesta de teta si ignoramos el parámetro teta 0. Así que tenemos teta 1 hasta teta “m” ignorando teta 0.

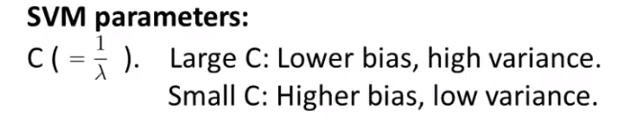


La suma de “J” de teta “j” cuadrada también se puede escribir como teta transpuesta de teta.   
  
Lo que hacen la mayoría de las implementaciones de máquinas de soporte vectorial es reemplazar esta teta transpuesta de teta con teta transpuesta por una matriz, que depende del número de kernel, por teta.

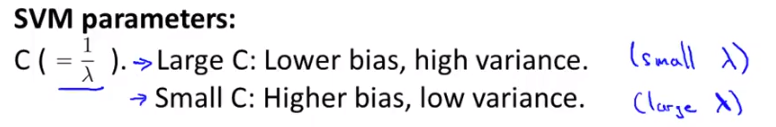


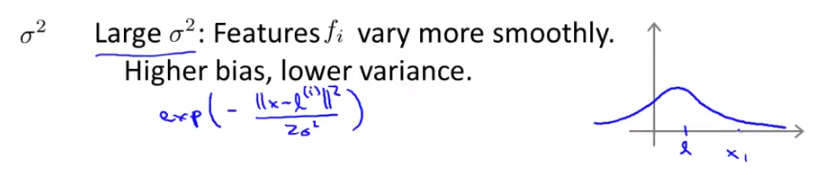
Esto resulta en una medición ligeramente distinta. Esto nos da una métrica de la distancia un poco distinta. En vez de minimizar la norma de teta cuadrada o minimizamos algo muy similar, que es como la versión a escala del parámetro vector teta que depende del kernel. Este es un detalle matemático que le permite a la máquina de soporte vectorial ejecutarse con más eficiencia.   
  
La razón por la que la máquina de soporte vectorial hace esto es porque es esta modificación le permite escalarla a conjuntos de entrenamiento más grandes. Por ejemplo, si tienes un conjunto de entrenamiento con 10,000 ejemplos de entrenamiento, entonces, deberemos terminar con 10,000 puntos de referencia, por lo que teta se vuelve 10,000 dimensional. Tal vez funcione. Pero cuando “m” se hace muy muy grande para resolver todos estos parámetros, quizá 50,000 o 100,000, resolverlos puede resultar caro para el software de optimización de la máquina de vector de soporte que, por lo tanto, resolverá el problema que dibujé aquí abajo. Como detalle matemático, que realmente no es necesario que conozcas, de hecho modifica un poco el último término para optimizar algo ligeramente distinto que sólo minimizar la norma cuadrada de teta cuadrada de teta. Pero si gustas, puedes pensar esto como un detalle de implementación que cambia el objetivo un poco pero se realiza por eficiencia computacional. Usualmente no tienes que preocuparte por esto.   
  
Por cierto, en caso de que te estés preguntando por qué no aplicamos la idea del kernel a otras regresiones logísticas resulta que de hecho, si quieres, puedes aplicar el kernel y definir la fuente de las variables utilizando puntos de referencia para la regresión logística. Pero los trucos computacionales que aplican para la máquina de soporte vectorial no se generalizarán bien en otros algoritmos como la regresión logística. Utilizar kernels con la regresión logística puede ser lento pero debido a los trucos computacionales que modifican este término y los detalles de cómo se implementa el software de la máquina de soporte vectorial, estas máquinas y los kernels combinan muy bien juntos, mientras que la regresión logística y los kernels no son tan compatibles, se ejecutarían muy lentamente y no serán capaces de tomar las ventajas de las técnicas de optimización avanzadas que se han descubierto para el caso particular de ejecutar una máquina de soporte vectorial con un kernel. Todo esto se refiere a cómo puedes implementar software para minimizar la función de costos. Hablaré más al respecto en el siguiente video aunque realmente no necesitas tener conocimiento de cómo construir software para minimizar esta función de costos, porque puedes encontrar software ya desarrollado que resulta excelente para esto. Yo no recomendaría escribir código para invertir la matriz o para calcular la raíz cuadrara ni para minimizar la función de costo por ti mismo. En vez de ello, recomendaría utilizar paquetes de software ya desarrollados por otros que incluyen estos trucos de optimización numérica para que tú no te preocupes por ello.

Otra cosa que vale la pena saber cuando aplicas una máquina de soporte vectorial es cómo elegir los parámetros para la máquina de soporte vectorial. Lo último que quiero hacer en este video es decir unas palabras acerca de la compensación entre el sesgo y la varianza cuando utilizamos una máquina de soporte vectorial. Cuando utilizamos una SVM, una de las cosas que debes elegir es el parámetro “C”, que teníamos en el objetivo de optimización. Recordemos que “C” jugaba un papel similar a 1 sobre «lambda», donde «lambda» era el parámetro de regularización para la regresión logística.



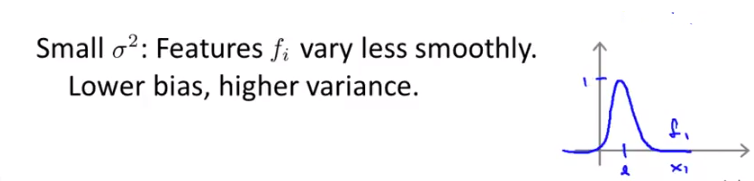
Si tienes un gran valor de “C”, corresponde a lo que teníamos en la regresión logística: un valor pequeño de «lambda» quiere decir que no se está usando mucha regularización. Haciendo esto tenderemos a obtener una hipótesis con un sesgo bajo y una varianza más alta, mientras que si utilizamos un valor de “C” más bajo y estamos utilizando la regresión logística con un valor alto de «lambda», obtendremos una hipótesis con un sesgo más alto y una varianza más baja. La hipótesis con la “C” mayor tendrá una varianza más alta y será más propensa al sobreajuste, mientras que la hipótesis con la “C” menor tendrá un sesgo mayor y será más propensa al subajuste.



Este parámetro “C” es uno de los parámetros que debemos elegir. El otro parámetro es «sigma» cuadrada que apareció en el kernel Gaussiano. Si en el kernel Gaussiano tenemos un valor alto para «sigma» cuadrada, en la variable de similaridad, que es “e” a la menos “x” menos punto de referencia cuadrado sobre 2 «sigma» cuadrada, y si tengo sólo una variable “x1” y un punto de referencia en esa ubicación y si el valor de «sigma» cuadrado es alto, el kernel Gaussiano tenderá a decrecer relativamente lento   


Esta de aquí sería mi variable “f(i)”, y representa una variable más suave, que varía más suavemente y que nos dará hipótesis con altas oscilaciones y varianzas bajas. Debido a que el kernel Gaussiano decrece suavemente, tiende a generar hipótesis que varían lentamente o suavemente a medida que cambiamos el valor de entrada “x”.

En contraste, si «sigma» cuadrada tenía un valor bajo y si mi punto de referencia dado o mi variable “f1” mi kernel Gaussiano, o mi variable de similaridad tendrán una variación más abrupta. En ambos casos elegiría uno. Si «sigma» cuadrada es pequeña, entonces mis variables variarán menos suavemente así que tendremos pendientes más altas o derivadas más altas. Utilizando esto terminarás por ajustar una hipótesis de sesgo bajo y podrás tener una varianza más alta.



Si observas esta curva de esta semana, podrás jugar con algunas de estas ideas y ver sus efectos por ti mismo.

Eso es todo acerca de la máquina de soporte vectorial con algoritmos de kernel. Espero que esta discusión del sesgo y la varianza te hayan dado un mejor sentido de cómo puedes esperar que se comporte un algoritmo.

# SVM en la práctica

Hasta el momento hemos hablado de las SVM a un nivel abstracto. En este video me gustaría necesitas hacer para ejecutar o utilizar una SVM.   
  
El algoritmo de la máquina de soporte vectorial presenta un problema de optimización peculiar. Como comenté en el video anterior, no recomiendo que escribas tu propio software para resolver por ti mismo los parámetros teta.   
  
Hoy en día, muy pocos de nosotros, quizá ninguno, pensaría en escribir código nosotros mismos para invertir una matriz o sustraer la raíz cuadrada de un número, etc., simplemente utilizamos una función de biblioteca para hacerlo. De la misma manera, el software para resolver el problema de optimización de las SVM es muy complejo. Algunos investigadores han realizado investigaciones de optimización numérica esencial por muchos años para desarrollar excelentes bibliotecas de software y paquetes de software para llevar esto a cabo. Yo recomiendo ampliamente utilizar una de las bibliotecas de software altamente optimizado en vez de intentar implementar algo tú mismo. Existen bibliotecas de software excelentes. Las dos que utilizo más seguido son la SVM lineares pero hay muchas bibliotecas de software excelentes para hacer esto que puedes adaptar a muchos de los lenguajes de programación más importantes que quizá estés utilizando para codificar un algoritmo de aprendizaje. Aunque no deberías escribir tu propio software de optimización de SVM, hay algunas cosas sí de debes hacer.

1. Primero, debes elegir el parámetro “C”. Hablamos de las propiedades de oscilación y varianza en videos anteriores.
2. Segundo, necesitas elegir el kernel o la función de similaridad que deseas utilizar. Una elección pudiera ser si decidimos no utilizar ningún kernel.

La idea de que no haya un kernel también se llama kernel lineal. Si alguien dice “utilizo una SVM con un kernel lineal” se refiere a que utilizan una SVM sin un kernel y a que era un versión de SVM que sólo utiliza teta transpuesta de “x”. Esto predice que teta 0 más teta 1, “x1” más el resto más teta “n”, “Xn” es mayor que o igual a 0.

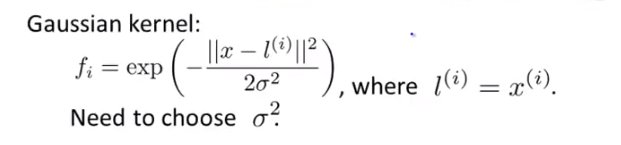


El término de kernel lineal lo puedes utilizar como la versión de la SVM que resulta en un clasificador estándar lineal.   
  
Esta sería una elección razonable para algunos problemas y, como mencioné, hay muchas bibliotecas de software, como SVM linear, que fue mi ejemplo, con las que se puede entrenar una SVM sin un kernel. A esta SVM se le llama kernel lineal. Ahora ¿por qué queremos hacer esto? Si tienes un gran número de variables, si “n” es grande y “m”, el número de ejemplos de entrenamiento, es pequeño, tendrás un número enorme de variables “x”-

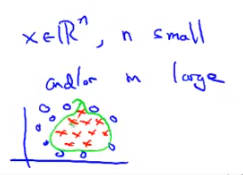


Así que, tenemos un número enorme de variables con un conjunto de entrenamiento pequeño. Quizá sólo quieres ajustar una barrera de decisión lineal y no ajustar una función no lineal complicada porque quizá no tengas suficientes datos. Y si estás intentando ajustar una variable muy complicada en un espacio con una dimensión de variables alta, corres el riesgo de sobreajustar. Por el contrario, si la muestra de tu conjunto de entrenamiento es pequeña, esta sería una condición razonable en la que puedes decidir no utilizar un kernel o algo equivalente. A esto se le llama kernel lineal.

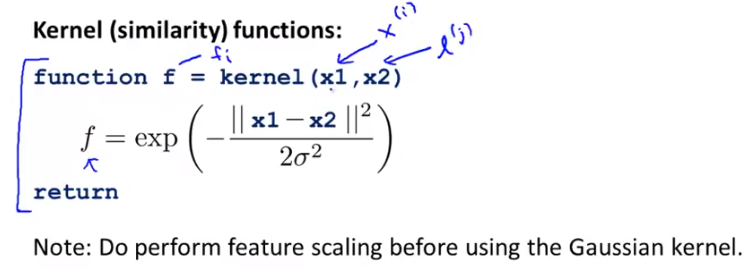
Una segunda elección para el kernel que puedes generar es el kernel Gaussiano, que utilizamos previamente.



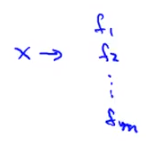
Si utilizas este kernel, la otra elección que debes tomar es fijar el parámetro «sigma» cuadrada. Cuando hablamos de las compensaciones entre oscilación y varianza hablamos de que cuando «sigma» cuadrada es alta, tiende a generar un clasificador de oscilación alto y varianza baja; por el contrario, si «sigma» cuadrada es pequeña, tendremos un clasificador de varianza alta y oscilación baja.   
  
Entonces ¿cuándo podemos elegir el kernel Gaussiano? En tus variables originales “x”, quiero decir “Rn”, “n” si es “n” es bajo y si “m” es alto, tendremos un conjunto de entrenamiento bidimensional como en el ejemplo que ilustre anteriormente. Así que “n” es igual a 2. Aquí tenemos un conjunto de entrenamiento muy grande, así que dibujé una gran cantidad de ejemplos de entrenamiento. Quizá después quieras utilizar un kernel para ajustar una barrera de decisión no lineal más compleja. Una buena opción sería utilizar el kernel Gaussiano.



Al final de este video hablaré un poco más acerca de cuando elegimos un kernel lineal, como el Gaussiano, u otros.   
  
De manera concreta, si decides utilizar un kernel Gaussiano, aquí está lo que debes hacer:   
Dependiendo del paquete de software de máquina de soporte vectorial que utilizarás, te preguntará si quieres implementar una función de kernel o de similaridad. Si utilizarás una implementación octave o MATLAB de una SVM, quizá te pida que proveas una función para calcular una variable específica en el kernel.



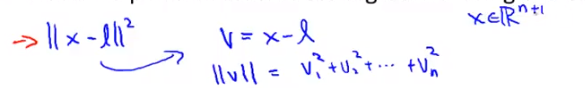
Esto será calcular “f” subíndice “i” para un valor particular de “i”, donde esta “f” es sólo un número real simple. Quizá debería denotar esto como “f(i)”. Lo que debes hacer es escribir una función de kernel que toma una entrada; es decir, un ejemplo de entrenamiento o un ejemplo de prueba, lo que sea, en un vector “x” y tomar la entrada de alguno de los puntos de referencia. Aquí sólo escribí “x1” y “x2”, pero los puntos de referencia también son ejemplos de entrenamiento. Lo que necesitas hacer, ahora, es escribir el software que tome esa entrada, “x1” y “x2”, y calcule este tipo de función de similaridad entre ellos y arroje un número real.   
  
Lo que hacen algunos paquetes de máquina de soporte vectorial es esperar que proveas la función de kernel que toma entrada “x1” y “x2”, y arrojará un número real.También, de estos datos, generará automáticamente todas las variables; es decir, tomara “x” para mapearlo en “f1” y “f2” hasta “f(m)” utilizando la función que escribas.



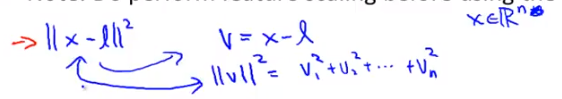
También generará todas las variables y entrenará a la máquina de soporte vectorial a partir de ello. Algunas veces tú necesitas proveer la función. Otras implementaciones de SVM también incluirán el kernel Gaussiano, y otros kernels adicionales, ya que probablemente el kernel Gaussiano es el kernel más común.   
  
Los kernels Gaussianos y no lineales son, por mucho, los dos kernels más populares. Añadiré una nota de implementación: Si tienes variables de escalas muy distintas, es importante modificar la escala antes de utilizar el kernel Gaussiano. Explicaré por qué: Imagina el cálculo de la norma entre “x” y “L”.



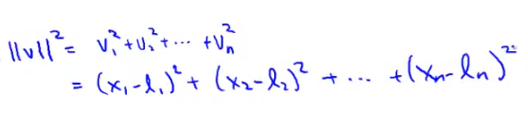
Lo que hacemos con esto es calcular el vector “V” que es igual a “x” menos “L” y luego calcularemos la norma del vector “V”, que es la diferencia de “x-L”. La norma de “V” es igual a “V1” cuadrada más “V2” cuadrada más el resto, más “Vn” cuadrada, porque “X” es “Rn” o “Rn” más 1, pero ignoraré “x0”



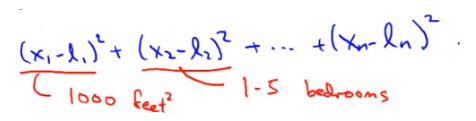
y fingiremos que “x” es “Rn”. El cuadrado en el lado izquierdo hace esta ecuación correcta. Entonces, esto es igual a esto de acá



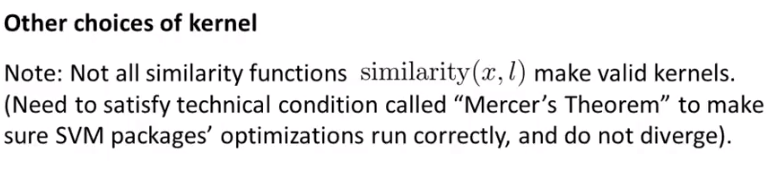
Esto, escrito de otra manera será

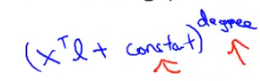


Ahora veremos qué pasa cuando las variables tienen un amplio rango de valores. Tomaremos el ejemplo de la predicción de la vivienda donde nuestros datos son acerca de casas. Si “x” está en el rango de miles de pies cuadrados para la primera variable “x1”, pero para la segunda variable “x2” se refiere al número de habitaciones y su valor variará de entre 1 y 5 habitaciones, entonces “x1” menos “L1” será enorme; resultaría como en mil pies cuadrados mientras que “x2” menos “L2” será mucho más pequeño. Si ese es el caso, entonces, estas distancias estarán determinadas esencialmente por el tamaño de las casas y el número de cuartos se ignoraría.

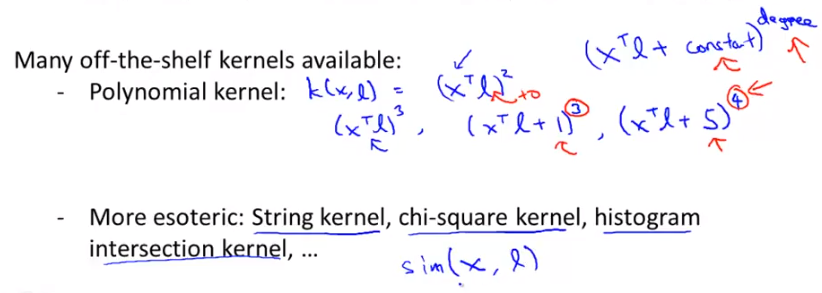


Para evitar esto y para lograr que funcione nuestra máquina, debemos hacer una modificación de escalas. Esto nos asegura que la SVM le pondrá una atención comparable a todas las variables y no sólo al tamaño de las casas, como en este ejemplo, ignorando todas las otras variables.   
  
Cuando probamos una máquina de soporte vectorial, las probabilidades de que utilices los dos kernels más comunes, es decir, el kernel lineal, o sin kernel, y el kernel Gaussiano del que ya hablamos, son altísimas. Agregaré una advertencia: No todas las funciones de similaridad que te encuentres serán kernels válidos. El kernel Gaussiano, el kernel lineal y otros kernels que se utilizan, necesitan satisfacer una condición técnica llamada teorema de Mercer. La razón por la cual se necesita esto es porque un algoritmo o implementación de una máquina de soporte vectorial tiene muchos trucos de optimización numérica para despejar los parámetros teta de manera eficiente. En el diseño original de las SVM se tomó la decisión de restringir nuestra atención solamente a los kernels para satisfacer esta condición técnica llamada teorema de Mercer. Lo que hace esto es asegurarnos de que todos estos paquetes de SVM y de software de SVM puedan utilizar una gran clase de optimizaciones y puedan obtener el parámetro teta rápidamente.

  
Lo que muchos terminan haciendo es utilizar ya sea el kernel lineal o el Gaussiano, pero hay otros kernels que también satisfacen el teorema de Mercer y que puedes encontrar en el trabajo de otros. Yo, personalmente, utilizo otros kernels muy rara vez, si acaso. Mencionaré algunos de los kernels con los que te puedes topar:   
  
Uno es el kernel de polinomio. Para este kernel, la similaridad entre “x” y “L” se define, entre muchas opciones, como “x” transpuesta de “L” cuadrada. Esta es una medida de lo similar que resultan “x” y “L”. Si “x” y “L” están muy cerca una de la otra, el producto interno tenderá a ser grande. Bueno, como ya sabes, este es un kernel un tanto inusual. No se utiliza seguido, pero puedes encontrar que algunas personas lo usan. Esta es otra versión de un kernel de polinomio. Otra se expresa “x” transpuesta de “L” al cubo. Todos estos son ejemplos de kernels de polinomios. “x” transpuesta de “L” más 1 al cubo. “x” transpuesta de “L” más un número diferente de 1, como 5, a la cuarta potencia. El kernel de polinomio tiene dos parámetros: Uno es el número que se debe añadir aquí. Puede ser 0. Este de aquí es, en realidad, más 0. El otro es el grado del polinomio. El grado o la potencia y estos números. La formulación más general del kernel de polinomio es “x” transpuesta de “L” más una constante elevada a un grado en X1.

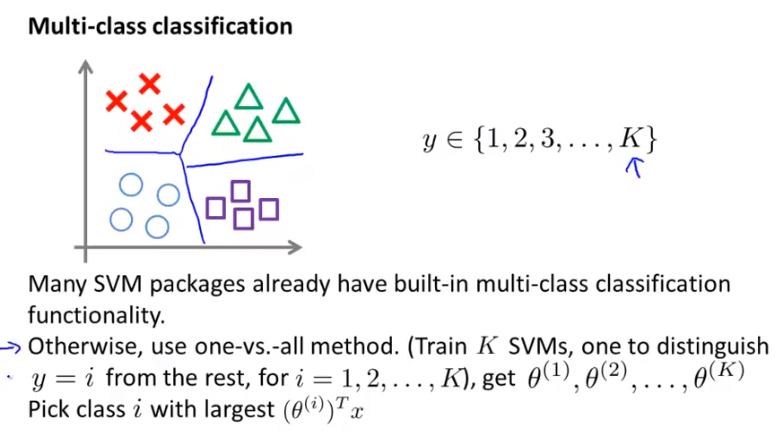


Estos dos son parámetros para el kernel de polinomio. Los kernels de polinomio casi siempre tienen un menor desempeño que el kernel Gaussiano y no es un kernel que se utilice mucho pero quizá en algún momento te lo topes. Se utiliza generalmente sólo para datos donde “x” y “L” son estrictamente no negativos. Esto nos asegura que los productos internos tampoco sean negativos.   
  
Esto captura la intuición de que “x” y “L” son muy similares uno al otro, por lo que el producto interno entre ellos será grande. Tienen otras propiedades, pero no se utilizan mucho.   
Después, dependiendo de lo que eses haciendo, hay otros kernels más esotéricos que te puedes encontrar. Está el kernel de secuencia que se utiliza a veces si tus datos de entrada son cadenas de variables u otras cadenas. Hay otros como el kernel chi-square, el kernel de intersección de histograma, etc. Hay otros kernels más raros que podemos utilizar para medir la similaridad entre objetos distintos. Por ejemplo, si intentas hacer algún tipo de problema de clasificación de texto, donde la entrada “x” es una cadena, entonces, quizá quieras encontrar la similaridad entre dos cadenas utilizando el kernel de secuencia, pero personalmente sé que el resultado será raro, si es que lo obtenemos, utilizando kernels extraños.



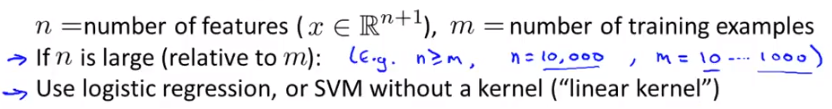
Creo que he utilizado el kernel chi-square sólo una vez en mi vida y el kernel de histogramas una o dos veces. Nunca he utilizado el kernel de secuencias. Pero, en caso de que te los encuentres en otras aplicaciones, puedes hacer una búsqueda rápida en Google o en Bing y encontrar las definiciones de estos kernels también.

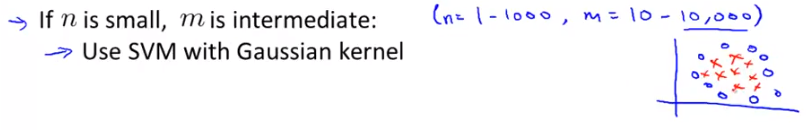
Añadiré sólo dos detalles de los que quiero hablar en este video. Uno es la clasificación de clases múltiples. Digamos que tienes cuatro clases o, de manera más general, “k” clases ¿cómo tomas la decisión más adecuada entre tus clases múltiples? La mayoría, o muchos de los paquetes de SVM ya tienen una funcionalidad de clasificación de clases múltiples integrada. Entonces, si utilizas un paquete como este, sólo debes usar la funcionalidad y con eso debe ser suficiente. De otra manera, una manera de hacer esto es utilizar el método uno contra todos del que hablamos cuando desarrollamos la regresión logística. Entonces, intercambias kSVM si tienes "k" clases para distinguirlas del resto. Esto nos arrojará “k” vectores parámetro; es decir teta 1, que intenta distinguir la clase “y” igual a 1 de todas las otras clases.

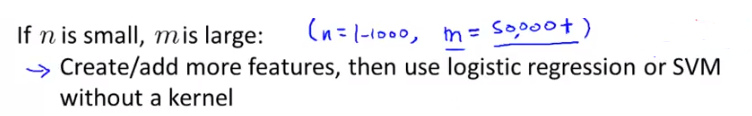


Después tenemos un segundo parámetro vector teta 2, que es lo que obtenemos cuando tenemos “y” igual a 2 como clase positiva, y todas las demás como negativas, etc., hasta el parámetro vector teta “k”, que es el parámetro vector para distinguir la clase final de todo lo de más. Por último, este es exactamente igual el método uno contra todo que vimos en la regresión logística con el que predijimos la clase (i) con la teta transpuesta de “x” más alta. Esta es la clasificación de clase múltiple. Pero el caso más común es que cualquier paquete de software que utilices tendrá ya integrado una funcionalidad de clasificación de clases múltiples. Si no la tiene, no te preocupes por este resultado.

Finalmente, desarrollamos máquinas de soporte vectorial iniciando con la regresión logística y luego modificando un poco la función de costos. Lo último que quiero hacer en este video es hablar un poco acerca de cuando utilizas uno de estos dos algoritmos. Digamos que “n” es el número de variables y “m” es el número de ejemplos de entrenamiento.   
  
¿Cuándo debemos utilizar un algoritmo en vez del otro?   
Aquí “n” es mayor en relación con el tamaño del conjunto de aprendizaje. Podemos pensar en “n” como un número de variables mayor que “m”. Esto puede presentarse cuando tenemos un problema de clasificación, donde la dimensión del vector de variables es, quizá, 10 mil. El tamaño del conjunto de aprendizaje es de 10 hasta hasta 1000. Imagina un problema de spam donde tenemos 10,000 variables de correos spam que corresponden a 10,000 palabras, pero sólo tienes 10 ejemplos de entrenamiento o hasta 1,000 ejemplos. Entonces, “n” es grande con relación a “m”. Lo que yo haría usualmente es utilizar la regresión logística o utilizarla como “m” sin un kernel o utilizar un kernel lineal, porque si tienes tantas variables con un conjunto de entrenamiento pequeño, una función lineal funcionará porque probablemente no tengas datos suficientes para ajustar una función no lineal complicada.



Ahora, “n” es pequeña y “m” es intermedia; a lo que me refiero con esto es que “n” está entre 1 y 1000 (1 sería muy pequeño, pero podemos fijarlo en 1000 variables), y el número de ejemplos de entrenamiento está entre 10 y 10,000 ejemplos, o quizá hasta 50,000 ejemplos; es decir, “m” es muy grande (alrededor de 10,000), pero no tanto como un millón. Si “m” es de un tamaño intermedio como esta, entonces una SVM con un kernel lineal funcionarán bien. Ya hablamos de esto anteriormente con un ejemplo concreto en el que teníamos un conjunto de entrenamiento bidimensional. Entonces, si “n” es igual a 2 y has dibujado un gran número de ejemplos de entrenamiento, el kernel Gaussiano hará un buen trabajo separando las clases negativas y las positivas.   


Una tercera opción de interés es si “n” es pequeño pero “m” es grande. Si “n” es, de nuevo, entre 1 a 1000, o mayor pero “m” es de entre 50,000 o mayor, hasta millones (50,000 a 100,000 o un millón o un billón). Puedes tener un conjunto de entrenamiento muy muy grande ¿cierto? Si este es el caso, una SVM con kernel Gaussiano será lenta. Los paquetes de SVM actuales, cuando usamos el kernel Gaussiano, tienen algunas dificultades. Si tienes unos 50,000 ejemplos está bien, pero si tienes un millón de ejemplos de entrenamiento, o incluso 100,000 con un valor masivo de “m”. Los paquetes de SVM actuales son muy buenos, pero todavía pueden encontrar dificultades cuando tienes un conjunto de entrenamiento masivo y utilizas el kernel Gaussiano. En este caso, lo que haría es tratar de crear manualmente más variables y luego utilizar la regresión logística o una SVM sin el kernel.   


Si ves esta diapositiva encontrarás subrayado “regresión logística o SVM sin un kernel” en dos ocasiones. Los puse juntos por una razón en específico. La regresión logística y la SVM sin el kernel son algoritmos muy similares porque, ya sea la regresión logística o la SVM sin un kernel, harán cosas similares y arrojarán un desempeño similar. Dependiendo de tus detalles de implementación, una pudiera resultar más eficiente que la otra. Cuando uno de estos algoritmos funciona, ya sea la regresión logística o la SVM sin kernel, lo más seguro es que el otro también funcione bien. Buena parte de la eficacia de la SVM surge cuando utilizas kernels diferentes para aprender funciones complejas no lineales. El régimen de cuando tienes hasta 10,000 ejemplos o hasta 50,000 ejemplos, y el número de variables es razonablemente grande, es un régimen muy común en el que una máquina de soporte vectorial con kernel fallaría y haría cosas mucho más difíciles que la regresión logística.

Finalmente, ¿dónde encajan las redes neuronales? Para todos estos problemas o para todos estos regímenes diferentes, una red neuronal bien diseñada funcionará bien también.   
  
La única desventaja o la única razón por la que a veces evitamos las redes neuronales es que, para algunos problemas, la red neuronal puede ser de entrenamiento lento. Pero con un paquete de implementación de SVM muy bueno que pueda ejecutarse más rápidamente que una red neuronal podrás resolver esto.   
  
Aunque no mostramos esto anteriormente, resulta que el problema de optimización que presenta la SVM es un problema de optimización convexo y, por lo tanto, los mejores paquetes de software de optimización de SVM siempre encontrarán el mínimo global o un valor cercano. Al utilizar las SVM no necesitas preocuparte por las óptimas locales.   
En la práctica, las óptimas locales o son un gran problema para las redes neuronales porque todas resuelven. Esta es una preocupación menos cuando utilizas una SVM. Dependiendo de tu problema, la red neuronal podrá ser más lenta, especialmente en este tipo de regímenes que con la SVM. En caso de que los lineamientos que expliqué aquí parezcan vagos, y si ves tus problemas y los lineamientos no parecen claros o no te queda claro qué algoritmo debes utilizar, está bien. Cuando me enfrento con un problema de aprendizaje automático, a veces no me queda claro cuál es el mejor algoritmo pero, como viste en los videos anteriores, el algoritmo es de importancia, aunque a veces importan más otras cosas como la cantidad de datos que tenemos y qué tan hábil seas realizando análisis de errores, limpiando algoritmos de aprendizaje y entendiendo cómo diseñar nuevas variables y qué otras variables debes darle a tu algoritmo de aprendizaje, etc. Algunas veces estas cosas importarán más que tu decisión de utilizar la regresión logística o la SVM. Una vez dicho esto, la SVM sigue siendo percibida como uno de los algoritmos de aprendizaje más eficaces y existe un régimen que provee una manera efectiva de aprender funciones complejas no lineales. Así que en realidad, creo que utilizando la regresión logística, las redes neuronales y las SVMs estás bien colocado para construir sistemas de aprendizaje automático de última generación para una gama amplia de aplicaciones. Esta es excelente arma en tu arsenal. Un arma que se utiliza por todos lados en Silicon Valley, en la industria y en la academia para construir sistemas de aprendizaje automático de alto rendimiento.

1. https://es.wikipedia.org/wiki/Producto\_escalar [↑](#footnote-ref-0)