Apendizaje de Modelos Lineales

José Luis Ruiz Reina

Dpto. Ciencias de la Computación e Inteligencia Artificial

Ampliación de Inteligencia Artificial, 2016-2017

Índice

Introducción

Aprendizaje automático

- En este tema se enmarca dentro del campo del *aprendizaje automático*
  - Trataremos sobre el *aprendizaje de modelos* basado en el *análisis de datos*
  - Nos referimos a modelos matemáticos
- Aprendizaje supervisado:
  - A partir de un conjunto de datos (*conjunto de entrenamiento*) que conocemos cómo se comportan respecto a lo que se quiere modelar, se aprende (computacionalmente) un modelo con el que pretendemos predecir el comportamiento sobre instancias nuevas

Ejemplo: predicción de precios de pisos

- Predecir el precio de un piso en función de sus características (número de habitaciones, distancia al centro, renta per cápita del barrio,...)
- Datos:

```
0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30 396.90   4.98  24.00
0.02731   0.00   7.070  0  0.4690  6.4210  78.90  4.9671   2  242.0  17.80 396.90   9.14  21.60
0.02729   0.00   7.070  0  0.4690  7.1850  61.10  4.9671   2  242.0  17.80 392.83   4.03  34.70
0.03237   0.00   2.180  0  0.4580  6.9980  45.80  6.0622   3  222.0  18.70 394.63   2.94  33.40
0.06905   0.00   2.180  0  0.4580  7.1470  54.20  6.0622   3  222.0  18.70 396.90   5.33  36.20
0.02985   0.00   2.180  0  0.4580  6.4300  58.70  6.0622   3  222.0  18.70 394.12   5.21  28.70
0.08829  12.50   7.870  0  0.5240  6.0120  66.60  5.5605   5  311.0  15.20 395.60  12.43  22.90
0.14455  12.50   7.870  0  0.5240  6.1720  96.10  5.9505   5  311.0  15.20 396.90  19.15  27.10
0.21124  12.50   7.870  0  0.5240  5.6310 100.00  6.0821   5  311.0  15.20 386.63  29.93  16.50
...
...
...
...
```

Ejemplo: clasificación de la flor de iris

- Hay tres generos de la planta de iris (*setosa*, *virgínica* y *versicolor*); se pretende poder clasificarla en función de medidas (anchura y longitud) de su sépalo y pétalo
- Datos:

```
5.1 3.5 1.4 0.2 Iris-setosa
4.9 3.0 1.4 0.2 Iris-setosa
4.7 3.2 1.3 0.2 Iris-setosa
...
...
7.0 3.2 4.7 1.4 Iris-versicolor
6.4 3.2 4.5 1.5 Iris-versicolor
6.9 3.1 4.9 1.5 Iris-versicolor
...
...

6.3 3.3 6.0 2.5 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
7.1 3.0 5.9 2.1 Iris-virginica
...
...
```

Modelos lineales

- Los datos que trataremos vendrán dados en forma de vectores numéricos, donde cada componente cuantifica un *atributo* o *característica* del dato
  - Notación para vectores: $(x_1, \ldots, x_n)$, ó $\mathbf{x}$ (negrita)
  - Notación para conjunto de datos: $D = \{(\mathbf{x}^{(j)}, y^{(j)}) : j = 1, \ldots, N\}$
- Modelos lineales:
  - Aquellos que se describen *esencialmente* mediante una combinación lineal de las características: $w_0 + w_1 x_1 + \ldots + w_n x_n$
  - Los coeficientes w_i $(i = 1, \ldots, n)$ se suelen denominar *pesos* (cuantifican la importancia de la característica)
  - w_0 es el *término independiente* (*intercept, bias, threshold*)
  - Notación matricial, siendo $\mathbf{x}$ y $\mathbf{w}$ vectores columna: $\mathbf{w}^\top \mathbf{x} + w_0$ ó $\mathbf{w}^\top \mathbf{x}$ si introducimos $x_0 = 1$

Regresión

- En un problema de regresión, tratamos de aprender cómo depende el valor de una variable en función del valor que toman otras variables o características.
- El problema de predecir el precio de los pisos a partir del conjunto de datos del ejemplo 1 es un problema de regresión

Regresión

- Cuando asumimos que esa relación de dependencia debe expresarse mediante una función lineal, decimos que se trata de una *regresión lineal*
  - Formalmente, el conjunto de entrenamiento tiene la forma $D = \{(\mathbf{x}^{(j)}, y^{(j)}) : j = 1, \ldots, N\}$, donde $\mathbf{x}^{(j)} \in \mathbb{R}^n, y^{(j)} \in \mathbb{R}$
  - Se trata de encontrar una función $h : \mathbb{R}^n \to \mathbb{R}$ de la forma $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ que se *ajuste* al conjunto de entrenamiento y que nos sirva para predecir el valor que toma la variable dependendiente para nuevas instancias
  - Hay que precisar qué entendemos por "ajustarse"

Clasificación

- En un problema de clasificación, en lugar de valores continuos, los datos tienen asignado una valor de k valores posibles $C = \{c_1, \ldots, c_k\}$ (denominados *clases*). Se trata de aprender un modelo que nos permita asignarle una clase (*clasificar*) a datos nuevos
- El problema de predecir el genero de una planta de iris a partir del conjunto de datos del ejemplo 2 es un problema de clasificación

Clasificación

- En este caso:
  - El conjunto de entrenamiento tiene la forma $D = \{(\mathbf{x}^{(j)}, c^{(j)}) : j = 1, \ldots, N\}$, donde $\mathbf{x}^{(j)} \in \mathbb{R}^n, c^{(j)} \in C$
    - Principalmente, hablaremos de clasificación binaria y consideraremos $C = \{0, 1\}$ (a veces $C = \{-1, 1\}$)
  - Se trata de encontrar una función $h : \mathbb{R}^n \to C$ que se *ajuste* al conjunto de entrenamiento y que nos sirva para clasificar nuevas instancias

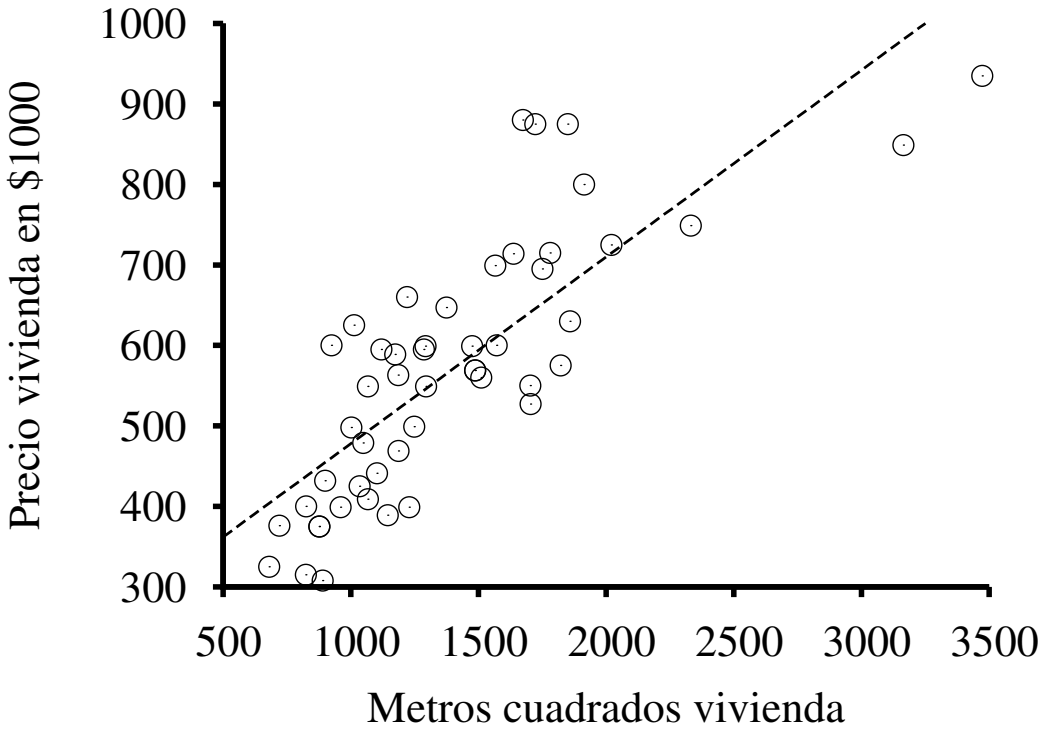
Regresión lineal

Regresión lineal en una variable

- Supongamos que los datos son unidimensionales:
  - El conjunto de datos es de la forma $D = \{(x_j, y_j) : j = 1, \ldots, N\}$, donde $x_j, y_j \in \mathbb{R}$
- Se trata de encontrar la recta $h_{\mathbf{w}}(x) = w_1 x + w_0$ que mejor se ajusta a los datos (donde $\mathbf{w} = (w_0, w_1)$)
- Definimos la *función de coste (o de pérdida)* cuadrática como $L_2(h_{\mathbf{w}}, D) = \sum_j (y_j - (w_1 x_j + w_0))^2$
  - Nótese que la función de coste depende de $\mathbf{w}$
- Mejor ajuste: el $\mathbf{w}$ que minimiza $L_2(h_{\mathbf{w}}, D)$

Recta de mínimos cuadrados

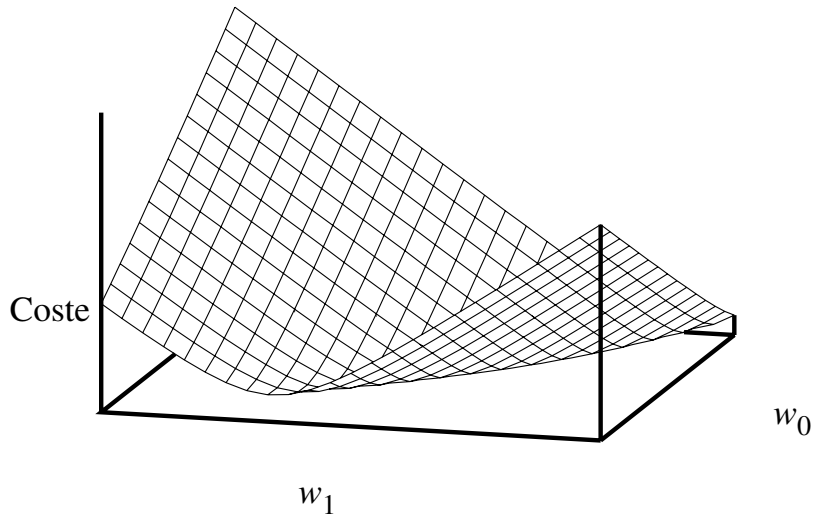
- Es un problema clásico en Estadísticas, resuelto mediante la *recta de mínimos cuadrados*



- De todas las rectas, ¿cuál es la que hace menor las suma de las diferencias (al cuadrado) entre lo que da la recta y el valor real de cada ejemplo?

Recta de mínimos cuadrados

- Es un problema de minimización de una función, cuyos argumentos son los coeficientes de la ecuación de la recta:



- En el caso concreto del coste L_2 para funciones lineales, esta función es convexa, y tiene un único mínimo, que se puede calcular derivando e igualando a cero.

Recta de mínimos cuadrados

- Las derivas parciales son:

$$\frac{\partial L_2}{\partial w_1} = -2\sum_j (y_j - (w_1 x_j - w_0))x_j \quad \frac{\partial L_2}{\partial w_0} = -2\sum_j (y_j - (w_1 x_j - w_0))$$

- Igualando a cero, y despejando:

$$w_1 = \frac{N(\sum_j x_j y_j) - (\sum_j x_j)(\sum_j y_j)}{N(\sum_j x_j^2) - (\sum_j x_j)^2} \quad w_0 = \frac{\sum_j y_j - w_1(\sum_j x_j)}{N}$$

- En este caso, el problema de optimización tiene una solución en forma cerrada, analítica.
  - Pero en otras situaciones (con modelos más complicados) no necesariamentre será así, y habrá que emplear otras técnicas

Regresión lineal multivariable

- Podemos generalizar la regresión lineal al caso en que se tengan más de dos variables o caraterísticas
  - El conjunto de datos es de la forma $D = \{(\mathbf{x}^{(j)}, y^{(j)}) : j = 1, \ldots, N\}$, donde $\mathbf{x}^{(j)} \in \mathbb{R}^n, y^{(j)} \in \mathbb{R}$
- Se trata de encontrar la función lineal $h_{\mathbf{w}}(x) = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x_1 + \ldots + w_n x_n$ que mejor se ajusta a los datos.
- La *función de coste (o de pérdida)* cuadrática se define como antes: $L_2(h_{\mathbf{w}}, D) = \sum_j (y^{(j)} - h_{\mathbf{w}}(\mathbf{x}^{(j)}))^2$

Regresión lineal multivariable

- Como en el caso de una variable, podemos derivar respecto de los pesos, igualar a cero y despejar
- En este caso concreto, el resultado se puede expresar de manera más compacta usando matrices:
  - Si $\mathbf{X}$ es la matrix $N \times (n+1)$ que en cada fila tiene un ejemplo de D (incluyendo $x_0 = 1$), e $\mathbf{y}$ es un vector columna con los valores de cada ejemplo, entonces el mínimo de L_2 se alcanza en:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Descenso (o ascenso) por el gradiente

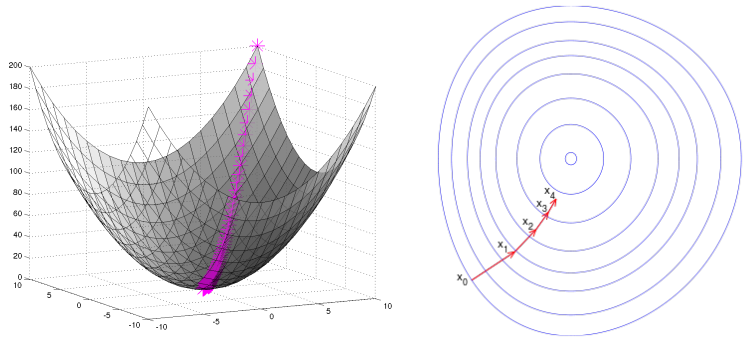
- Como hemos visto, en el caso de regresión lineal, el problema de minimizar la función L_2 tiene una solución cerrada, expresable analíticamente.
- Sin embargo, veremos que en otras situaciones no es posible obtener el mínimo de esta manera. En esos casos *buscamos la solución* mediante un algoritmo iterativo de *búsqueda local*
- La técnica más conocida para ello se denomina *descenso por el gradiente* (o *ascenso*) si se trata de maximizar).
- Recordar: dada una función $f : \mathbb{R}^n \to \mathbb{R}$, su gradiente en $\mathbf{x} \in \mathbb{R}^n$ se define como:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right]$$

Descenso (o ascenso) por el gradiente

- El gradiente es la dirección hacia la que la función produce el máximo asceso
- El descenso por el gradiente consiste en buscar el mínimo de una función iterando pequeños pasos siempre en la dirección opuesta al gradiente:

$$\mathbf{x^{(t+1)}} \longleftarrow \mathbf{x^{(t)}} - \alpha \nabla f(\mathbf{x^{(t)}})$$



- La constante $\alpha > 0$ indica el tamaño de los (pequeños) pasos de descenso
- Análogamente, ascenso por el gradiente para buscar el máximo

Regresión lineal mediante descenso por el gradiente

- Gradiente del coste L_2, en el caso de la regresión lineal:

$$\frac{\partial L_2}{\partial w_i} = -2 \sum_j x_i^{(j)} (y_j - h_{\mathbf{w}}(\mathbf{x}^{(j)}))$$

- Algoritmo de regresión lineal, descenso por el gradiente:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, actualizar los pesos de la siguiente manera:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}^{(j)})) x_i^{(j)}$$

- En este caso concreto, la convergencia al mínimo está asegurada, siempre que se tome un α suficientemente pequeño (es una función convexa).
  - Aunque en general el descenso por el gradiente tiene el problema de los *óptimos locales*

Descenso estocástico por el gradiente

- La versión anterior del descenso por el gradiente se conoce como *batch*, ya que tiene en cuenta *todos los ejemplos* a la hora de calcular el gradiente y realizar cada actualización
- La versión *estocástica* del algoritmo realiza la actualización teniendo en cuenta cada vez el gradiente del coste *en un ejemplo*.

Descenso estocástico por el gradiente: algoritmo

- Algoritmo de regresión lineal, descenso estocástico por el gradiente:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, tomar un ejemplo $(\mathbf{x}, y) \in D$ y actualizar los pesos de la siguiente manera:

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$$

- Usualmente, se dan varias pasadas por el conjunto de entrenamiento
- Aprendizaje *dirigido por el error*:
  - La actualización de los pesos depende de la diferencia entre el valor esperado y el valor calculado.

Batch vs Estocástico

- La versión estocástica se aproxima mucho más rápido a valores cercanos al mínimo
- Sin embargo, no tiene garantizada la convergencia, si α permanece fijo (puede ser necesario su actualización a medida que se acerca al mínimo).
- Podría oscilar alrededor del mínimo, pero en la práctica las aproximaciones al mínimo suelen ser razonablemente buenas
- Uusalmente se prefiere la versión estocástica a la versión *batch*, especialmente en aprendizaje *online* o cuando el conjunto de entrenamiento es grande

Regularización para combatir el sobreajuste

- Sobreajuste: el modelo que se aprende se ajusta tanto al conjunto de entrenamiento que es excesivamente complejo y no explica bien el caso general.
- Un caso típico de *sobreajuste* en regresión es que alguna de las variables o características parezca relevante para la predicción, cuando en realidad no lo es
- Para ello, hay que dar prioridad a modelos más simples frente a modelos más complejos
- La *regularización* consiste en introducir una penalización por la complejidad del modelo en la función de coste. Por ejemplo:

$$Coste(h) = Error(h) + \lambda Complejidad(h)$$

  - Donde λ es una constante que regula la importancia de la complejidad en el coste total

Regularización en regresión lineal

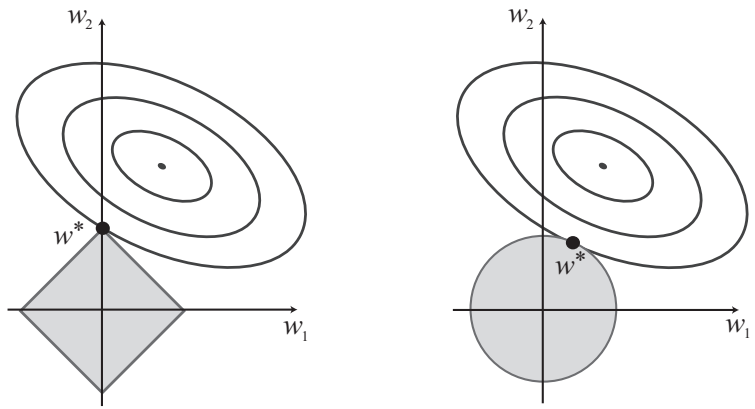
- En regresión lineal, el sobreajuste se combate buscando minimizar la magnitud de los pesos
- Usualmente la complejidad del modelo se mide de la siguiente manera:

$$Complejidad(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

  - Si $q = 1$, tenemos la penalización L_1, también llamada *lasso* o *dispersa*
  - Si $q = 2$, es la penalización L_2, también llamada *ridge*
- Con la regularización aumentamos el *sesgo*, pero reducimos la *varianza*

Regularización L_1 vs L_2

- Ventaja de la regularización L_1: tiende a *modelos dispersos*
  - El valor final en muchos pesos es cercano a cero, de manera que explicita qué características son realmente irrelevantes
  - Intuición geométrica:



- L_2 es derivable, pero L_1 no lo es siempre
  - Con regularización L_2 es posible hacer descenso por el gradiente
  - Los algoritmos de minimización cuando usamos regularización L_1 son más complicados (descenso por coordenadas)
- *Elastic nets*: regresión combinando regularización L_1 y L_2

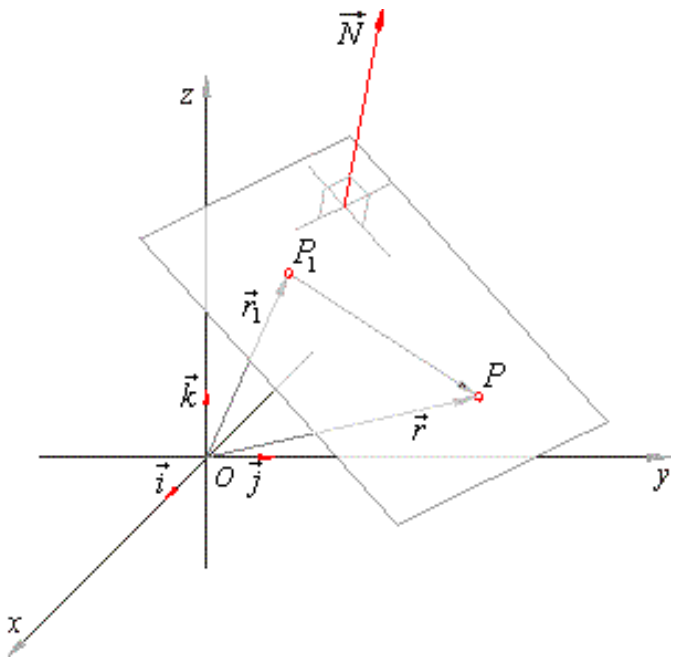
El preceptrón como clasificador lineal

Clasificación lineal binaria: perceptrones

- Además de para hacer regresión, podemos usar las funciones lineales para problemas de clasificación binaria (dos clases c_1 y c_2)
- En el modelo más sencillo de clasificación lineal, dados unos coefientes $(w_1, \ldots, w_n)$, y un *umbral* b, clasificaremos un vector $(x_1, \ldots, x_n)$ como que pertenece a una clase c_1 si $\sum_i w_i x_i >= b$; en caso contrario lo clasificamos como c_2.
- Usualmente, notamos $w_0 = b$, suponemos $x_0 = 1$, $c_1 = 1$ y $c_2 = 0$, por lo que el clasificador lineal se identifica con la función $h_{\mathbf{w}}(\mathbf{x}) = \mathit{umbral}(\mathbf{w}^\top \mathbf{x})$
  - Donde $\mathit{umbral}(x) = 1$ si $x >= 0$, $\mathit{umbral}(x) = 0$ en caso contrario.
- Este clasificador se conoce como *perceptrón*

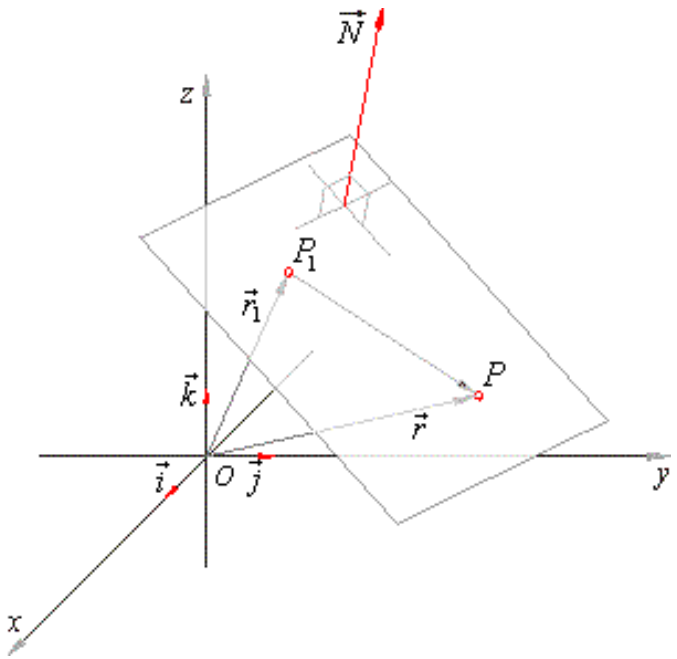
Un poco de geometría

- La ecuación $\mathbf{w}^\top \mathbf{x} = 0$ (es decir, $w_0 + w_1 x_1 + \ldots + w_n x_n = 0$) representa un hiperplano en $\mathbb{R}^n$
  - Una recta si $n = 2$, un plano si $n = 3$
  - El vector $(w_1, \ldots, w_n)$ es perpendicular al hiperplano
  - w_0 es proporcional a la distancia del hiperplano al origen



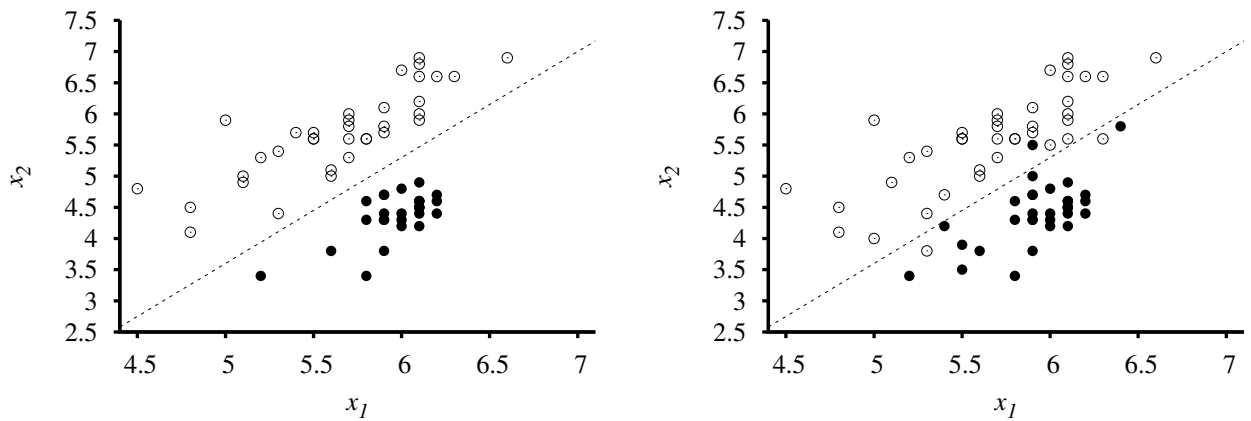
Un poco de geometría

- Para cualquier punto $\mathbf{x} \in \mathbb{R}^n$, el signo de $\mathbf{w}^\top \mathbf{x}$ nos da la posición del punto respecto al hiperplano
  - Si $\mathbf{w}^\top \mathbf{x} = 0$ está en el hiperplano
  - Si $\mathbf{w}^\top \mathbf{x} > 0$ está a un lado del hiperplano
  - Si $\mathbf{w}^\top \mathbf{x} < 0$ está al otro lado del hiperplano
- Además, $\mathbf{w}^\top \mathbf{x}$ es proporcional a la distancia del punto al hiperplano.



Conjuntos linealmente separables

- Por tanto, un perceptrón clasifica un punto según el lado en el que queda el punto, respecto del hiperplano que representa
- Establece una *frontera de decisión lineal*
- Si en el conjunto de entrenamiento D las dos clases no pueden ser separadas por un hiperplano, entonces ningún perceptrón podrá clasificar correctamente todas las instancias de D
- D se dice *linealmente separable* si las instancias de cada clase pueden ser separadas por un hiperplano



Algoritmo de entrenamiento del Perceptrón

- Entrada: Un conjunto de entrenamiento D (con ejemplos de la forma $(\mathbf{x}, y)$, con $\mathbf{x} \in \mathbb{R}^n$ e $y \in \{0, 1\}$), y una *tasa de aprendizaje* η
- Algoritmo de entrenamiento del perceptrón:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, tomar un ejemplo $(\mathbf{x}, y) \in D$, calcular $o = \mathit{umbral}(\mathbf{w}^\top \mathbf{x})$ y actualizar los pesos de la siguiente manera:

$$w_i \leftarrow w_i + \eta(y - o)x_i$$

Intuición sobre las actualizaciones

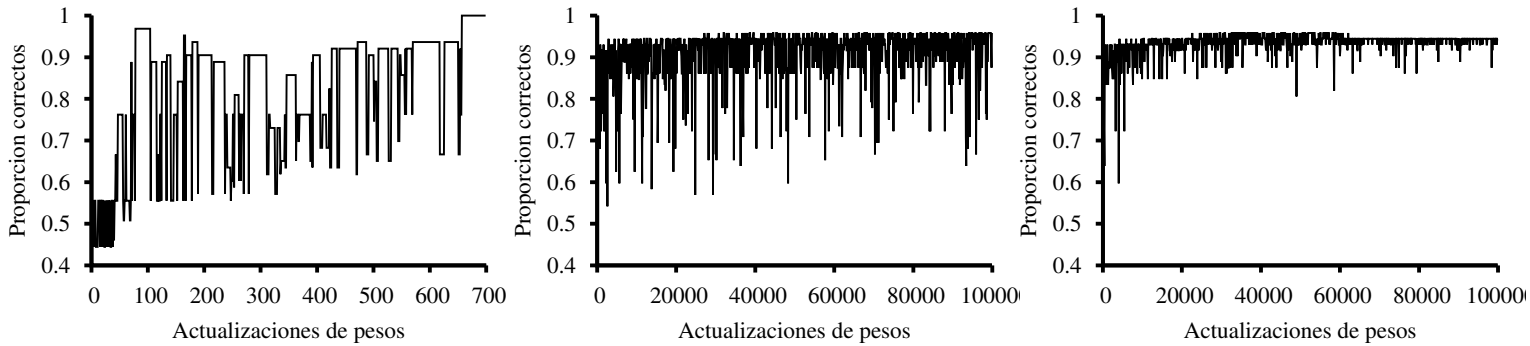
- Aprendizaje dirigido por el error:
  - En cada iteración, si $y = 1$ y $o = 0$, entonces $y - o = 1 > 0$, y por tanto los w_i correspondientes a x_i positivos aumentarán (y disminuirán los correspondientes a x_i negativos), lo que aproximará o (salida real) a y (salida esperada)
  - Análogamente ocurre si es $o = 1$ e $y = 0$
  - Cuando $y = o$, los w_i no se modifican
- Usualmente, los ejemplos se toman de manera aleatoria

Convergencia en el aprendizaje del perceptrón

- Teorema: El algoritmo anterior *converge* en un *número finito de pasos* a un vector de pesos $\vec{w}$ que clasifica correctamente todos los ejemplos de entrenamiento, siempre que éstos sean *linealmente separables* y η *suficientemente pequeño* (Minsky and Papert, 1969)
- Por tanto, en el caso de conjuntos de entrenamiento linealmente separables, la condición de terminación puede ser que se clasifiquen correctamente todos los ejemplos
- En el caso de que el conjunto no sea linealmente separable, ni siquiera está garantizado que se converja al error mínimo
  - El algoritmo no es un descenso por el gradiente (la función umbral no es diferenciable)
  - Sin embargo, se puede demostrar que si η va disminuyendo con las iteraciones, a razón $O(1/t)$, entonces se converge a unos pesos de error mínimo (aunque tras muchas iteraciones)

Curvas de entrenamiento en el perceptrón

- Curvas de entrenamiento: reflejan el número de instancias bien clasificadas del conjunto de entrenamiento, en función del número de actualizaciones realizadas



1. Aprendizaje en conjunto de entrenamiento linealmente separable
2. Aprendizaje en conjunto de entrenamiento no linealmente separable
3. Aprendizaje en conjunto no separable con $\eta(t) = \frac{1000}{1000+t}$

Regresión logística

Modelo lineal probabilístico

- Hasta ahora hemos visto aprendizaje de dos tipos de hipótesis:
  - Regresión: $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$
  - Clasificador perceptrón: $h_{\mathbf{w}}(\mathbf{x}) = \mathit{umbral}(\mathbf{w}^\top \mathbf{x})$
- El clasificador perceptrón tiene dos problemas principales:
  - La función umbral no es diferenciable (no podemos hacer descenso por el gradiente)
  - La clasificación es categórica, no hay *graduación* en la predicción
- ¿Podemos interpretar $\mathbf{w}^\top \mathbf{x}$ como una probabilidad de pertenecer a la clase $c = 1$?
  - En principio no, ya que es un número entre $-\infty$ y $+\infty$

Modelo lineal probabilístico

- Idea: considerar que $\mathbf{w}^\top \mathbf{x}$ nos devuelve la razón de proporcionalidad entre las probabilidades de pertenecer a la clase $c = 1$ y pertenecer a $c = 0$ (lo que en estadística se conoce como *odds ratio*): $\frac{P(c=1|\mathbf{x})}{P(c=0|\mathbf{x})}$
  - Tampoco vale, ya que es una cantidad entre 0 y $+\infty$
  - Pero si tomamos el logaritmo (neperiano), ya se convierte en una cantidad entre $-\infty$ y $+\infty$
- Un modelo de *regresión logística* es un modelo lineal que estima el logaritmo del *odds ratio*:

$$\mathbf{w}^\top \mathbf{x} = log \frac{P(c = 1|\mathbf{x})}{P(c = 0|\mathbf{x})}$$

Regresión logística como clasificador probabilístico

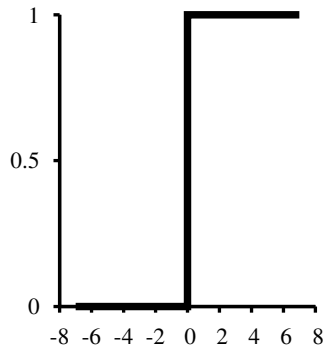
- Equivalentemente, podemos considerar que en este caso queremos aprender una hipótesis $h_{\mathbf{w}}(\mathbf{x})$ que estima a probabilidad de que $\mathbf{x}$ pertenezca a la clase $c = 1$:
- Si en la fórmula anterior despejamos $P(c = 1|\mathbf{x})$ (y teniendo en cuenta que $P(c = 0|\mathbf{x}) = 1 - P(c = 1|\mathbf{x})$):

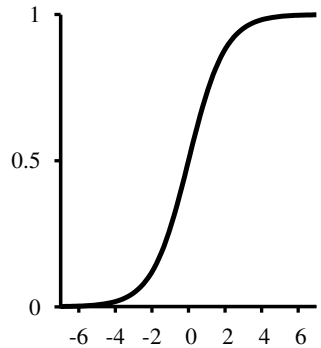
$$h_{\mathbf{w}}(\mathbf{x}) = P(c = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

- La función $\sigma(z) = \frac{1}{1+e^{-z}}$ se denomina *sigmoide* o *función logística*
- Por tanto, en regresión logística, $h_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$, y lo que la hipótesis va a estimar es *la probabilidad de pertenecer a una de las clases*.
    - Es un clasificador, no un modelo de regresión

Perceptrón vs Regresión logística

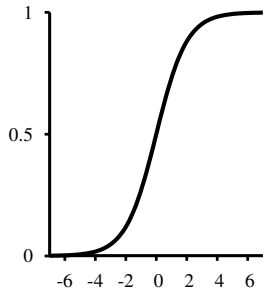
- Función umbral vs Sigmoide





- A diferencia de la función umbral (o la función signo), el sigmoide es derivable en todo $\mathbb{R}$, y su derivada es $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Los valores están *suavizados*, y han de ser interpretados como una probabilidad

Intuición geométrica de la clasificación



- Recordar que $\mathbf{w}^\top \mathbf{x}$ es proporcional a la distancia al hiperplano definido por $\mathbf{w}$. Por tanto:
  - Valores grandes y positivos para $\mathbf{w}^\top \mathbf{x}$: entonces $h_{\mathbf{w}}(\mathbf{x})$ será cercana a 1 (seguridad en la predicción)
  - Valores "grandes" y negativos para $\mathbf{w}^\top \mathbf{x}$: entonces $h_{\mathbf{w}}(\mathbf{x})$ cercana a 0 (seguridad en la predicción).
  - Valores pequeños de $h_{\mathbf{w}}(\mathbf{x})$, tanto negativos como positivos, significan que la instancia está cerca del hiperplano de separación y por tanto, $h_{\mathbf{w}}(\mathbf{x})$ es cercano a 0.5 (poca seguridad en la predicción).

Aprendizaje en regresión logística

- Igual que con el perceptrón, un conjunto de entrenamiento D con ejemplos de la forma $(\mathbf{x}, y)$, donde $\mathbf{x} \in \mathbb{R}^n$ e y indica una de las dos clases.
- Aprender el clasificador es encontrar unos pesos $\mathbf{w}$ que optimizan una función
- ¿Qué función debemos optimizar? Dos opciones más comunes:
  - Minimizar la función de coste del error cuadrático, L_2
  - Maximizar la verosimilitud (*maximum likelihood*)

Minimización L_2 en regresión logística

- Para dar la definición de L_2 en este caso, conviene considerar que las etiquetas de las clases son 1 ó 0
  - Es decir, los ejemplos del conjunto de entrenamiento D son de la forma $(\mathbf{x}^{(j)}, y^{(j)})$, con $\mathbf{x}^{(j)} \in \mathbb{R}^n$ e $y^{(j)} \in \{0, 1\}$).
- Si notamos $o^{(j)} = \sigma(\mathbf{w}^\top \mathbf{x}^{(j)})$, entonces el error cuadrático es:

$$L_2(h_{\mathbf{w}}, D) = \sum_j (y^{(j)} - o^{(j)})^2$$

- Y por tanto las componentes del vector gradiente son:

$$\frac{\partial L_2}{\partial w_i} = -2 \sum_j (y^{(j)} - o^{(j)}) x_i^{(j)} o^{(j)} (1 - o^{(j)})$$

La regla delta

- Se puede hacer descenso por el gradiente anterior (versión *batch*), pero es más común hacer descenso por el gradiente de cada ejemplo (versión *estocástica*)
  - Este último algoritmo se conoce como *regla delta*, *Adaline*, o *Widrow-Hoff*
- Algoritmo regla delta:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, tomar un ejemplo $(\mathbf{x}, y) \in D$, calcular $o = sigma(\mathbf{w}^\top \mathbf{x})$ y actualizar los pesos de la siguiente manera:

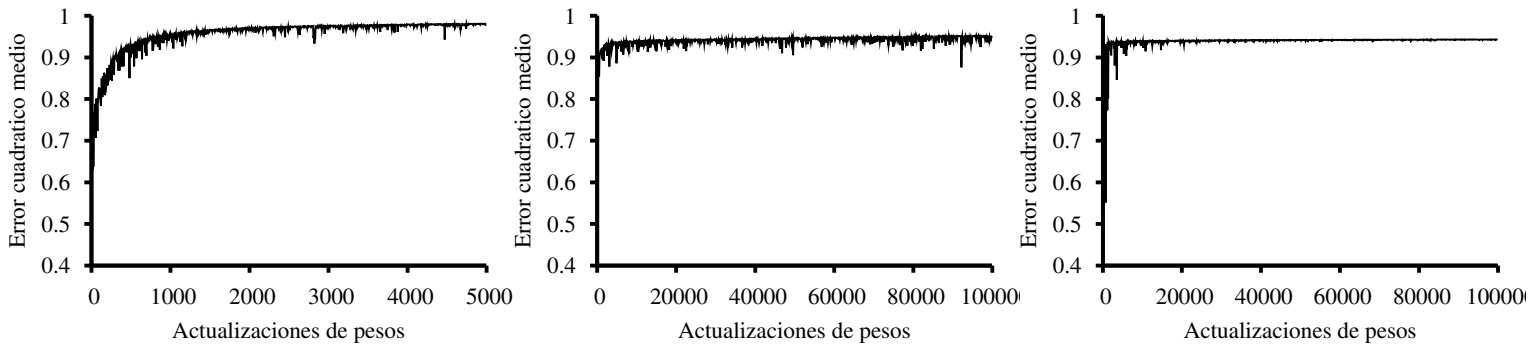
$$w_i \leftarrow w_i + \eta(y - o)x_i o(1 - o)$$

Convergencia de la regla delta

- La versión batch del algoritmo de descenso por el gradiente para regresión logística converge hacia un mínimo local del coste L_2, siempre que se tome η suficientemente pequeño
- Se puede demostrar que la Regla Delta se puede aproximar arbitrariamente al método de descenso por el gradiente, siempre que se tome η suficientemente pequeño
- En la Regla Delta la actualización de pesos es más simple, aunque necesita valores de η más pequeños. Además, a veces escapa más fácilmente de los mínimos locales

Curvas de entrenamiento con la regla delta

- Comparar estas curvas de entrenamiento con las del aprendizaje del perceptrón (diapositiva 36)



1. Aprendizaje en conjunto de entrenamiento linealmente separable
2. Aprendizaje en conjunto de entrenamiento no linealmente separable
3. Aprendizaje en conjunto no separable con $\eta(t) = \frac{1000}{1000+t}$

Maximizando la verosimilitud

- Tomando un enfoque probabilístico, podemos plantear una alternativa a la minimización del error cuadrático
- Recordar: dado un $\mathbf{w}$ fijo, $h_{\mathbf{w}}(\mathbf{x})$ nos da la probabilidad de que $\mathbf{x}$ pertenezca a cada clase:

$$P(c = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} \qquad P(c = 0|\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}}}$$

- La idea es buscar el $\mathbf{w}$ que maximiza la probabilidad de que cada ejemplo de D pertenezca a la clase que tiene asignada

Maximizando la verosimilitud

- Notación: D^+ son los ejemplos $(\mathbf{x}, y)$ de D con $y = 1$; D^- son aquellos con $y = 0$
- Dado D y vector de pesos $\mathbf{w}$, la *verosimilitud* (*likelihood*) de la hipótesis $h_{\mathbf{w}}$ es la probabilidad que esa hipótesis le asigna a la información que nos proporciona D. En este caso:

$$\mathcal{L}(\mathbf{w}) = \prod_{\mathbf{x}^{(j)} \in D^+} P(c = 1|\mathbf{x}^{(j)}) \prod_{\mathbf{x}^{(j)} \in D^-} P(c = 1|\mathbf{x}^{(j)}) =$$

$$= \prod_{\mathbf{x}^{(j)} \in D^+} \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} \prod_{\mathbf{x}^{(j)} \in D^-} \frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}}}$$

- Buscamos un $\mathbf{w}$ que maximiza $\mathcal{L}(\mathbf{w})$

Maximizando la log-verosimilitud

- En lugar de $\mathcal{L}(\mathbf{w})$, es equivalente y más conveniente utilizar su logaritmo (*log-verosimilitud*):

$$\mathcal{LL}(\mathbf{w}) = -\sum_{\mathbf{x}^{(j)} \in D^+} log(1 + e^{-\mathbf{w}^\top \mathbf{x}}) - \sum_{\mathbf{x}^{(j)} \in D^-} log(1 + e^{\mathbf{w}^\top \mathbf{x}})$$

- Esta función es diferenciable y *cóncava*.
  - Buena noticia: podemos hacer *ascenso* por el gradiente y además puesto que no tiene máximos locales, converge hacia el máximo global.
- Componentes del vector gradiente:

$$\frac{\partial \mathcal{LL}}{\partial w_i} = -\sum_{\mathbf{x}^{(j)} \in D^+} \frac{-x_i^{(j)} e^{-\mathbf{w}^\top \mathbf{x}}}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} - \sum_{\mathbf{x}^{(j)} \in D^-} \frac{x_i^{(j)} e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}} =$$

$$= \sum_{\mathbf{x}^{(j)} \in D^+} P(c = 0 | \mathbf{x}^{(j)}) x_i^{(j)} - \sum_{\mathbf{x}^{(j)} \in D^-} P(c = 1 | \mathbf{x}^{(j)}) x_i^{(j)}$$

Ascenso por el gradiente para maximizar verosimilitud

- Algoritmo de ascenso por el gradiente:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, actualizar los pesos de la siguiente manera:

$$w_i \leftarrow w_i + \eta\Big(\sum_{\mathbf{x}^{(j)} \in D^+} P(c=0|\mathbf{x}^{(j)}) x_i^{(j)} - \sum_{\mathbf{x}^{(j)} \in D^-} P(c=1|\mathbf{x}^{(j)}) x_i^{(j)}\Big)$$

- Interesante: como los anteriores algoritmos, puede verse como actualizaciones *dirigidas por el error* (¿por qué?)
- La regla de actualización anterior puede escribirse de manera más compacta:

$$w_i \leftarrow w_i + \eta \sum_j [(y^{(j)} - \sigma(\mathbf{w}^\top \mathbf{x}^{(j)})) x_i^{(j)}]$$

Regresión logística, versión estocástica

- La anterior es la versión *batch*, podríamos también hacer una versión estocástica que actualice por cada ejemplo
- Algoritmo de ascenso estocástico por el gradiente:
  - Inicializar los pesos $(w_0, \ldots, w_n)$ aleatoriamente
  - En cada iteración, y hasta la condición de parada, tomar un ejemplo $(\mathbf{x}, y) \in D$ y actualizar los pesos de la siguiente manera:

$$w_i \leftarrow w_i + \eta(y - \sigma(\mathbf{w}^\top \mathbf{x}))x_i$$

Regularización en regresión logística

- Como en el caso de la regresión lineal, en regresión logística se puede combatir el sobreajuste mediante regularización
  - Tanto si se minimiza el coste como si se maximiza la verosimilitud
- Por ejemplo, maximizando $\mathcal{LL}(\mathbf{w}) - \lambda \sum_i |w_i|$ hacemos regularización L_1 (lasso)
- Maximizando $\mathcal{LL}(\mathbf{w}) - \lambda \sum_i w_i^2$, hacemos regularización L_2 (ridge)

Regularización en regresión logística

- Se aplican las mismas ventajas e inconvenientes:
  - L_1 premia que algunos pesos se anulen, aunque los algoritmos de optimización son más complejos
  - L_2 es diferenciable, y por tanto permite optimizar usando ascenso por el gradiente (por ejemplo, en las actualizaciones anteriores, bastaría con introducir $-\lambda w_i$)

Probabilidad multiclase: *softmax*

- Supongamos que tenemos ahora un problema de clasificación de más de dos clases
  - $D = \{(\mathbf{x}^{(j)}, y^{(j)}) : j = 1, \ldots, N\}$, donde $x^{(j)} \in \mathbb{R}^n, y^{(j)} \in \{1, \ldots, K\}$
- Función *softmax*, generaliza el sigmoide:

$$softmax(z_1, \ldots, z_n) = \frac{1}{\sum_k e^{z_k}}(e^{z_1}, \ldots, e^{z_n})$$

  - *softmax* se comporta de manera similar a *max*, pero suavizado y diferenciable

Regresión multinomial logística

- La idea es que el modelo al clasificar dé una probabilidad para cada clase, y decidir en función de eso
- Podemos generalizar la regresión logística, considerando K vectores de pesos $\mathbf{w}^{(m)}, m = 1, \ldots, K$
  - Cada $\mathbf{w}^{(m)}$ modela el grado de pertencia a la clase m, que luego se transforma en una probabilidad, usando *softmax*
- Regresión logística multinomial:

$$h(\mathbf{x}) = softmax(\mathbf{w^{(1)}}^\top \mathbf{x}, \ldots, \mathbf{w^{(K)}}^\top \mathbf{x})$$

- Como en el caso de dos variables, podemos aprender los pesos $\mathbf{w^{(m)}}$ tratando de maximizar la verosimilitud del conjunto de entrenamiento

Aprendizaje en regresión multinomial logística

- Regla de actualización de pesos (*batch*), para cada clase m

$$w_i^{(m)} \leftarrow w_i^{(m)} + \eta \sum_j [(c_m(y^{(j)}) - \frac{e^{\mathbf{w}^{(m)\top}\mathbf{x}^{(j)}}}{\sum_k e^{\mathbf{w}^{(k)\top}\mathbf{x}^{(j)}}})x_i^{(j)}]$$

- Regla de actualización de pesos (*estocástica*), para cada clase m

$$w_i^{(m)} \leftarrow w_i^{(m)} + \eta(c_m(y) - \frac{e^{\mathbf{w}^{(m)\top}\mathbf{x}}}{\sum_k e^{\mathbf{w}^{(k)\top}\mathbf{x}}})x_i$$

- Notación: $c_m(y) = 1$ si $y = m$, en caso contrario $c_m(y) = 0$

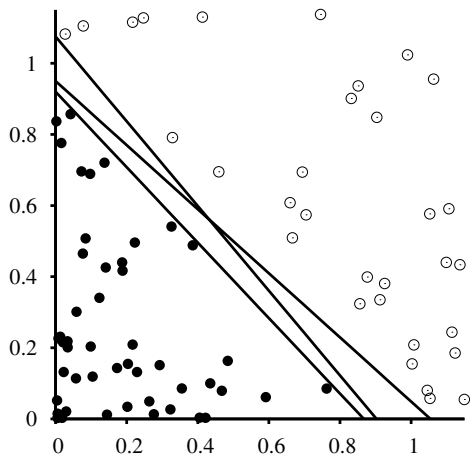
Máquinas de vectores de soporte

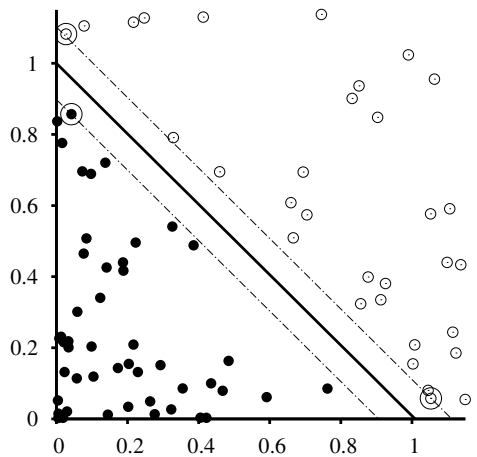
Clasificador basado en vectores soporte (SVC,SVM)

- Volvemos ahora a clasificadores tipo perceptrón:
  - Dado un vector de pesos $\mathbf{w}$ y una instancia $\mathbf{x}$, clasificar en una clase sólo si $\mathbf{w}^\top \mathbf{x} > 0$
- Se trata por tanto de encontrar una *frontera de decisión* definida por un hiperplano
- No intentaremos minimizar el error, sino maximizar el *margen de separación* entre clases (la distancia del hiperplano a las instancias más cercanas de cada clases)

Maximizar el margen de separación

- Consideremos en primer lugar *conjuntos de datos linealmente separables*





- Margen de un hiperplano: suma de las distancias del hiperplano a los puntos más cercanos de cada clase (*vectores soporte*)
- Idea: con una frontera con mayor margen de separación, generalizamos mejor en la clasificación de nuevas instancias
  - En las figuras, todos las rectas que aparecen suponen error cuadrático 0, pero la de la derecha es la que mejor generaliza. Los vectores soporte son los puntos marcados con un círculo.

Maquinas de vectores soporte: notación

- Antes de formular el problema de maximización, convienen fijar algunas notaciones
- En el conjunto de entrenamiento, las etiquetas de clase serán 1 y -1
  - Es decir, tenemos $D = \{(\mathbf{x}^{(j)}, y^{(j)}) : j = 1, \ldots, N\}$, donde $\mathbf{x}^{(j)} \in \mathbb{R}^n, y^{(j)} \in \{-1, 1\}$
- No integraremos el término independiente con los otros pesos, sino que lo explicitaremos:
  - Es decir, el hiperplano de separación estará definido por $\mathbf{w}^\top \mathbf{x} + b = 0$, donde $\mathbf{w} \in \mathbb{R}^n$ y $b \in \mathbb{R}$ (no usamos ni x_0 ni w_0)
- Por tanto, no usamos ahora la función umbral sino la función signo: $h_{\mathbf{w},b}(\mathbf{x}) = \mathit{signo}(\mathbf{w}^\top \mathbf{x} + b)$
- Se trata de buscar $\mathbf{w}$ y b que *maximicen el margen de separación*

SVC para conjuntos linealmente separables

- Una opción podría ser buscar mediante descenso por el gradiente el $\mathbf{w}$ y el b que maximiza el margen y separa los datos. Pero existe otra manera de atacar el problema: *relajación Lagrangiana*
- Se puede demostrar que el problema se reduce a encontrar coeficientes $\lambda_j >= 0, j = 1, \ldots, N$, cumpliendo $\sum_j \lambda_j y_j = 0$, y que maximicen el siguiente *dual Lagrangiano*:

$$L_D = \sum_j \lambda_j - \frac{1}{2} \sum_{j,k} \lambda_j \lambda_k y^{(j)} y^{(k)} (\mathbf{x}^{(j)\top} \mathbf{x}^{(k)})$$

- Este problema de maximización se denomina *problema dual*

El problema dual en SVC

- Una vez resuelto el problema de encontrar los λ_j que maximicen la expresión anterior (con las restricciones), entonces se pueden obtener los correspondientes $\mathbf{w}$ y b que maximizan el problema original:
  - $\mathbf{w} = \sum_j \lambda_j y^{(j)} \mathbf{x}^{(j)}$, y b despejando de la ecuación $y(\mathbf{w}^\top \mathbf{x} + b) = 1$, donde $(\mathbf{x}, y)$ es cualquiera de los vectores soporte (aquellos con $\lambda_j > 0$)
- Pero en realidad no es necesario recurrir al problema original, ya que podemos clasificar nuevos ejemplos usando sólo los λ_j:

$$h_{\mathbf{w},b}(\mathbf{x}) = signo(\sum_j \lambda_j y^{(j)} (\mathbf{x}^\top \mathbf{x}^{(j)}) + b)$$

Algunos comentarios interesantes sobre el problema dual

- Es un problema de optimización en programación cuadrática
- Es una expresión *convexa*, y por tanto tiene un único máximo global que se puede encontrar de manera eficiente
- Los datos intervienen en la expresión *sólo a través de los productos escalares entre ellos*
- Incluso para clasificar nuevas instancias, sólo intervienen a través del producto escalar de los datos con la nueva instancia
- La mayoría de los λ_j son cero, *excepto los correspondientes a los vectores soporte*, que son muchos menos

Ventajas de los SVC

- Los SVC son un modelo no paramétrico (necesitan eventualmente guardar todos los ejemplos para clasificar nuevas instancias), pero en la práctica sólo necesitan una pequeña parte de ellos (los vectores soporte)
- La estrategia de buscar el margen de separación máxima les otorga buena capacidad de generalización
- Combinan ventajas de métodos paramétricos y no paramétricos
  - Resistencia a sobrejuste
  - Flexibilidad para representar funciones complejas
- Se pueden ajustar a fronteras de decisión no lineales, haciendo una inmersión en espacios de dimensión superior (*kernel trick*)

Resolución del problema dual

- Problema dual: maximizar

$$L_D = \sum_j \lambda_j - \frac{1}{2} \sum_{j,k} \lambda_j \lambda_k y^{(j)} y^{(k)} (\mathbf{x}^{(j)\top} \mathbf{x}^{(k)})$$

  sujeto a $\lambda_j >= 0$ y $\sum_j \lambda_j y_j = 0$

- Se puede resolver mediante ascenso por el gradiente. Los componentes del gradiente de L_D son:

$$\frac{\partial L_D}{\partial \lambda_j} = 1 - \frac{1}{2} y^{(j)} \sum_k y^{(k)} \lambda_k (\mathbf{x}^{(j)\top} \mathbf{x}^{(k)})$$

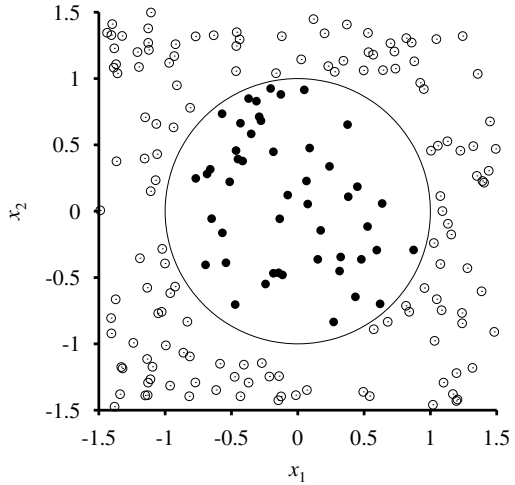
Resolución del problema dual

- Pero ahora es un problema de maximización sujeto a las restricciones $\lambda_j \geq 0$ y $\sum_j \lambda_j y_j = 0$
- Inicialmente, se toman $\lambda_j = 0$.
- Las actualizaciones no se hacen en la dirección de ∇L_D, sino en la direccción que resulta al proyectar ortogonalmente ∇L_D sobre el hiperplano $\sum_j \lambda_j y_j = 0$
  - Si $\hat{y} = \frac{1}{\sqrt{n}}(y^{(1)}, \ldots, y^{(n)})$, entonces esa proyección es $G = \nabla L_D - (\hat{y}^\top \nabla L_D)\hat{y}$
- Y si salen λ_j negativos, se hacen cero

SVC con datos no linealmente separables (*soft margin*)

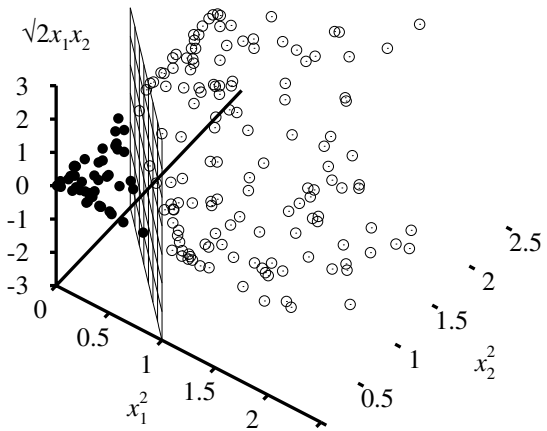
- En la práctica es infrecuente que los datos sean linealmente separables
  - Aunque sí que pudieran serlo de manera aproximada, con algunos *datos ruidosos*
- En ese caso, se introduce una penalización por cada dato que no queda dentro de su correspondiente región del margen. Esta penalización se parametriza por una constante C
  - Cuanto mayor C, menos tolerancia a romper las restricciones de margen
- El problema dual en este caso (*soft margin*), es el mismo que en el caso linealmente separable, pero se añaden las restricciones $\lambda_j < C$

Fronteras de decisión no lineales



- A veces los datos no se ajustan en ningún modo a una frontera de decisión lineal
- En la figura de la izquierda vemos un conjunto de datos que no es linealmente separable, pero separable por una circunferencia
- Sin embargo, los datos pueden ser linealmente separables si aplicamos una transformación que pasa los datos a tres dimensiones

Aumentando el número de dimensiones



- La figura muestra los mismos datos tras aplicarles la transformación $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ definida por $\Phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
- Los mismos datos son ahora linealmente separables en $\mathbb{R}^3$

Aumentando el número de dimensiones

- En general, se puede demostrar que toda frontera de decisión polinomial se puede trasformar en una lineal en un espacio con más dimensiones
  - Y además los polinomios pueden aproximar muchas otras funciones no lineales
- Pero en general esto incrementa sustancialmente la complejidad computacional, y el número de parámetros que hay que aprender
- Sin embargo, los SVC nos permiten aprender fronteras de decisión arbitrarias, *sin tener que manejar explicitamente el espacio con las dimensiones aumentadas*
- Es el llamado *kernel trick*

Kernel trick

- Como ya vimos, los SVC se pueden resolver completamente a partir de los productos escalares (simllitudes) entre los datos
  - No es necesario manejar directamente los valores de las características de cada dato
- $K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)})$ es una *función kernel* si existe una transformación $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ (con $m > n$) tal que $K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = \Phi(\mathbf{x}^{(j)})^\top \Phi(\mathbf{x}^{(k)})$
- Es decir, K devuelve *el producto escalar en el espacio con dimensión aumentada*
  - Pero sin manejar explícitamente dicho espacio

Kernel trick

- Podemos aplicar SVC para aprender una frontera de decisión no linear, usando una función kernel.
- Problema dual: maximizar

$$L_D = \sum_j \lambda_j - \frac{1}{2} \sum_{j,k} \lambda_j \lambda_k y^{(j)} y^{(k)} K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)})$$

  sujeto a $\lambda_j >= 0, j = 1, \ldots, N$ y $\sum_j \lambda_j y_j = 0$
- Clasificación de nuevas instancias:

$$h(\mathbf{x}) = signo(\sum_j \lambda_j y^{(j)} K(\mathbf{x}, \mathbf{x}^{(j)}) + b)$$

Funciones kernel y teorema de Mercer

- Algunas funciones kernel comunmente usadas:
  - Función de base radial gaussiana: $K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = e^{-\frac{|\mathbf{X}^{(j)} - \mathbf{X}^{(k)}|^2}{2\sigma^2}}$
  - Polinómico: $K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = ({\mathbf{x}^{(j)}}^\top \mathbf{x}^{(k)} + c)^h$
  - Sigmoide: $K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = tanh(k{\mathbf{x}^{(j)}}^\top \mathbf{x}^{(k)} - d)$
- Estos kernels tienen parámetros, su ajuste es importante
- Se pueden usar muchos otros kernels, siempre que se cumpla que $[K(\mathbf{x}^{(j)}, \mathbf{x}^{(k)})]$ es una matrix positiva semidefinida (teorema de Mercer)

Otros usos de funciones kernels

- Intuitivamente, un kernel expresa una función de similitud entre los elementos del espacio
- Se puede aplicar en cualquier algoritmo de aprendizaje que use el producto escalar entre datos
  - Clustering. k-medias
  - Análisis de componentes principales
  - Aprendizaje en grafos
  - Kernels en cadenas de símbolos

Bibliografía

- Russell, S. y Norvig, P. *Artificial Intelligence (A modern approach)* (Third edition) (Prentice Hall, 2003) *[Figuras tomadas de este libro]*
  - Cap. 18: "Learning from examples"
- Aggarwal, C.C.. *Data Mining: The Textbook* (Springer, 2015)
  - Cap. 10: "data classification"
- Alpaydim, E. *Introduction to Machine Learning* (Third edition) (MIT Press, 2014)
  - Cap. 10: "Linear Discrimination"
- Abu-Mostafa Y.S. et al. *Learning from Data* (AMLbook, 2012)
  - Cap. 3: "The linear model"