

# Aprendizaje supervisado

Mtro. José Gustavo Fuentes Cabrera



Facultad de Estudios Superiores

# Acatlán

Apuntes de Análisis Multivariado

Licenciatura en Actuaría

# Índice

<b>1. Introducción</b>	<b>4</b>
<b>2. El problema de clasificación como un problema de aprendizaje</b>	<b>4</b>
2.1. Vectores de características . . . . .	4
2.2. Formalización del problema de aprendizaje . . . . .	5
<b>3. Clasificadores lineales</b>	<b>8</b>
3.1. El conjunto de clasificadores lineales . . . . .	8
3.2. Consideraciones sobre algoritmos para aprendizaje para clasificadores lineales . . . . .	10
3.3. Error de entrenamiento . . . . .	11
3.4. El perceptrón . . . . .	11
3.4.1. Deducción del algoritmo . . . . .	11
3.4.2. Teorema de convergencia del perceptrón . . . . .	14
3.4.3. Perceptron medio . . . . .	17
3.5. Funciones de pérdida . . . . .	18
3.6. Algoritmo Pasivo-Agresivo . . . . .	20
3.7. Máquinas de soporte vectorial . . . . .	22
<b>4. Regresión Lineal</b>	<b>26</b>
4.1. Mínimos Cuadrados . . . . .	27
4.2. Regularización . . . . .	30
<b>5. Kernels</b>	<b>31</b>
5.1. Perceptrón de kernel . . . . .	32
5.2. Regresión lineal de kernel . . . . .	35
5.3. Funciones kernel . . . . .	37
<b>6. Análisis Discriminante</b>	<b>39</b>
<b>7. Regresión Logística</b>	<b>42</b>
7.1. Métricas de desempeño . . . . .	42
7.2. Especificación del modelo . . . . .	44
7.3. Estimación de parámetros . . . . .	45
7.4. Método de Newton . . . . .	47
7.5. Transformación WoE . . . . .	48

<b>8. Redes Neurales Artificiales</b>	<b>49</b>
8.1. Redes neurales alimentadas hacia adelante . . . . .	50
8.2. Propagación hacia atrás . . . . .	52

## 1. Introducción

En esta unidad, revisaremos lo correspondiente a modelos de soporte supervisados, es decir aquellos que se entrenan a partir de datos etiquetados, se revisaran distintos tipos de clasificadores y regresores, el soporte matemático detrás de estos y los correspondientes algoritmos de implementación.

## 2. El problema de clasificación como un problema de aprendizaje

Considérese el problema mínimo de formular una expresión que produzca una salida binaria con dos posibles categorías, las cuales nombraremos por conveniencia “-” y “+”. Pudiésemos pensar en primera instancia en un método que a través de condicionales nos permita obtener la salida deseada, sin embargo, encontrar de manera exhaustiva todas las posibles reglas y parámetros que nos lleven a predecir correctamente la categoría no es una estrategia razonable. En este orden de ideas, es mejor abordar el problema desde una perspectiva inversa, es decir, a partir de la salida (vectores resultantes etiquetados con las categorías “+” y “-”) diseñamos un algoritmo que “aprenda” una función que nos permita predecir dicha categoría (clasificador) a partir de los vectores (mediciones) de entrada. El conjunto que contiene los vectores de mediciones y las etiquetas de clasificación es llamado *conjunto de entrenamiento*. Por tanto, si contamos únicamente con dicho conjunto de entrenamiento, podemos generalizar este resultado para aplicarlo a nuevas mediciones que carecen de etiqueta y poder predecir el resultado de manera simple.

### 2.1. Vectores de características

En la sección pasada se mencionó lo correspondiente a las mediciones presentes en el conjunto de entrenamiento. Dichas mediciones son conocidas como *vectores de características* y su finalidad es la representación matemática de algún elemento de la realidad. Los vectores de características son fundamentales, ya que su elaboración con detenimiento será clave para la potencia de nuestro clasificador. Esto debido a que fundamentalmente se relacionará la variabilidad estadística de

los vectores con respecto a las etiquetas para la clasificación. Es muy importante la correcta elección de la unidad muestral para el problema en cuestión, ya que todas las agregaciones y cruces de datos se harán en correspondencia con ella y por ende, serán determinante para la precisión en las predicciones así como en la utilidad práctica de nuestro modelo. El proceso de generar los vectores de características se conoce a menudo como *ingeniería de datos* y es crucial para lograr construir un buen modelo, debido a que la forma en la que resumiremos la información en las coordenadas de los vectores dictará en gran medida la correcta abstracción de las relaciones y particularidades del fenómeno de estudio. En la práctica, el proceso de creación de los vectores de características suele ser el más laborioso y propenso a fallos o híper-simplificaciones.

## 2.2. Formalización del problema de aprendizaje

Abordemos ahora el problema de clasificación desde una perspectiva más formal. Sea  $x \in \mathbb{R}^d$  un vector de características de dimensión  $d$  y consideremos  $n$  ejemplos en el conjunto de entrenamiento, entonces podremos denotar como  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$  cada uno de los ejemplos en el conjunto de entrenamiento y de manera similar  $y^{(1)}, y^{(2)}, \dots, y^{(n)}$  sus correspondientes etiquetas de clasificación. Por tanto, el conjunto de pares ordenados  $S_n = \{(x^{(i)}, y^{(i)})\}, i = 1, \dots, n\}$ , será todo lo que el método de clasificación conocerá sobre el problema. Un clasificador  $h$  es un mapeo entre los vectores de características y las etiquetas:  $h : \mathbb{R}^d \rightarrow \{+, -\}$ . Cuando aplicamos el clasificador a un ejemplo en particular  $x$  escribimos  $h(x)$  como la etiqueta predicha. Así, cualquier clasificador divide el hiperespacio  $\mathbb{R}^d$  en regiones positivas y negativas dependiendo de la salida predicha. Un algoritmo de aprendizaje consiste entonces en un conjunto de clasificadores  $H$  (conjunto de hipótesis sobre las reglas que determinan la relación entre los ejemplos y las etiquetas) y la correspondiente selección de un clasificador  $\hat{h} \in H$  basado en el conjunto de entrenamiento  $S_n$ . La meta del algoritmo será encontrar a  $\hat{h} \in H$  que tenga el mejor desempeño en clasificar observaciones fuera del conjunto de entrenamiento, esto se conoce como *generalización*. Lo anterior significa que un clasificador que generaliza bien, será aquel que se desempeña de manera similar en nuevos ejemplos tal y como lo hace en el conjunto de entrenamiento. Diseñar clasificadores que generalicen correctamente estará ligado a diversos factores que de manera enunciativa mas no limitativa listamos a continuación:

- La elección de los vectores de características

- El tamaño del conjunto de entrenamiento ( $n$ )
- La elección del conjunto de clasificadores  $H$
- La estrategia de selección de  $\hat{h} \in H$
- La representatividad de la muestra de entrenamiento

Para evaluar con mayor precisión estos factores, definamos una métrica específica. Sea  $\mathcal{E}_n(h)$  la proporción de errores que comete el clasificador, formalmente:

$$\mathcal{E}_n(h) = \frac{1}{n} \sum_{i=1}^n \llbracket h(x^{(i)}) \neq y^{(i)} \rrbracket$$

donde  $\llbracket \text{verdadero} \rrbracket = 1$  (cuando  $h$  comete error) y  $\llbracket \text{falso} \rrbracket = 0$  (cuando  $h$  acierta). Podemos entonces, evaluar el error de entrenamiento  $\mathcal{E}_n(h)$  para cualquier clasificador  $h$  debido a que tenemos los ejemplos de entrenamiento, sin embargo, no debemos enfocar nuestros esfuerzos en dicho error, ya que existe la posibilidad de que ciertos clasificadores tengan errores cercanos a cero o incluso cero para un conjunto de entrenamiento específico  $S_n$ . Es por ello que nos concentraremos en minimizar el error de validación (generalización), es decir, desempeño en ejemplos futuros:

$$\mathcal{E}(h) = \frac{1}{n'} \sum_{i=n+1}^{n+n'} \llbracket h(x^{(i)}) \neq y^{(i)} \rrbracket$$

Que no es más que el error de entrenamiento evaluado sobre los siguientes  $n'$  ejemplos. El problema de esta aproximación es que se ve limitada por la ausencia de ejemplos futuros al momento en el que se elige a  $\hat{h}$ . Para ello, recurriremos a un artificio, asumiremos que los ejemplos de validación son en muchas formas similares a los ejemplos de entrenamiento (lo cual es completamente razonable). En específico, establecemos que los ejemplos de entrenamiento y validación junto con sus etiquetas de clasificación son extraídos de forma aleatoria de una distribución conjunta. Así que será una estrategia adecuada tener más ejemplos en el conjunto de entrenamiento tal que el error de entrenamiento (el que podemos medir) se reflejará de mejor forma en el error de validación (el que queremos minimizar). Existen dos partes clave en el diseño de un clasificador tal que generalice correctamente, en primera instancia la elección del conjunto de clasificadores  $H$  (selección del modelo) y en segunda como  $\hat{h} \in H$  es seleccionado (ajuste de parámetros). En

la práctica, es conveniente restringir  $H$  a un conjunto pequeño de clasificadores debido a que nuestro objetivo es la predicción y no el minimizar el error de entrenamiento, en el segundo caso, discutiremos acerca de *algoritmos de aprendizaje y algoritmos de entrenamiento* cuyo objetivo es encontrar algún  $\hat{h} \in H$  para el conjunto  $H$  elegido (preferentemente pequeño) que minimiza el error de entrenamiento al mismo tiempo que busca la mayor generalización posible. Supóngase que  $H$  contiene un único clasificador, en otras palabras, no tenemos que hacer ningún entrenamiento y por consiguiente, la única tarea disponible es la de evaluar que tan bien se desempeña el clasificador en el conjunto de entrenamiento. Desde esta perspectiva, los ejemplos de entrenamiento jugarán el rol de los nuevos ejemplos (ya que no fueron usados para entrenar a  $h$ ) de tal forma que el clasificador generalizará bien en coincidencia con el hecho de que su desempeño en el conjunto de entrenamiento será muy cercano al desempeño de validación. En este ejemplo, incluso si el clasificador es pobre o ni siquiera es adecuado para la tarea, el hacer uso de los ejemplos de entrenamiento para seleccionar  $\hat{h}$  solamente hará crecer la brecha entre  $\mathcal{E}_n(\hat{h})$  y  $\mathcal{E}(\hat{h})$ . Tomemos ahora el otro extremo, supongamos que  $H$  contiene todas las funciones binarias, es decir, podemos seleccionar el clasificador que nos plazca, por tanto,  $H$  será muy grande. Por ejemplo, dado el conjunto de entrenamiento  $S_n = \{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}$ , podemos seleccionar:

$$\hat{h}(x) = \begin{cases} +, & \text{si hay una etiqueta positiva asignada al vector } x^{(i)} \text{ tal que } \|x - x^{(i)}\| \leq \epsilon \\ -, & \text{en otro caso} \end{cases}$$

Claramente  $\hat{h} \in H$  por ser una función binaria. El clasificador presentado predice + para una región muy pequeña alrededor de cada ejemplo positivo en el conjunto de entrenamiento y – en cualquier otro lugar. Si los ejemplos de entrenamiento son distintos y  $\epsilon$  lo suficientemente pequeño, entonces  $\hat{h}$  tendrá error de entrenamiento nulo, pero esto presenta un problema aunque intuitivamente se presente como un excelente clasificador. Si los ejemplos de validación están balanceados conforme a las etiquetas + y –, y los vectores de características son lo suficientemente diferentes (por lo cual no existirán ejemplos duplicados), entonces el error de validación será exactamente 1/2 (equivalente al azar), esto debido a que todos los ejemplos positivos serán clasificados erróneamente como negativos (no están a una distancia  $\epsilon$  de los ejemplos de entrenamiento positivos). Por otra parte, todos los ejemplos negativos serán clasificados correctamente ya que  $\hat{h}$  predice –

prácticamente donde sea. La meta será entonces, explorar un conjunto de clasificadores  $H$  que es lo suficientemente pequeño para asegurar la generalización pero al mismo tiempo lo suficientemente grande tal que algún  $\hat{h} \in H$  tendrá un error de entrenamiento bajo.

### 3. Clasificadores lineales

Cualquier clasificador  $h$  divide el hiperespacio de entrada en dos mitades basadas en la etiqueta de predicción, es decir, corta  $\mathbb{R}^d$  en dos conjuntos  $\{x : h(x) = 1\}$  y  $\{x : h(x) = -1\}$  los cuales pueden llegar a un alto grado de complejidad. En el caso de los clasificadores lineales, tal división se realiza de una forma geométrica muy simple. En dos dimensiones será una línea recta, en tres un plano y en alta dimensionalidad un hiper-plano por supuesto. Aunque esta familia de clasificadores es limitada, pueden crearse poderosos clasificadores a través de ingeniería de características.

#### 3.1. El conjunto de clasificadores lineales

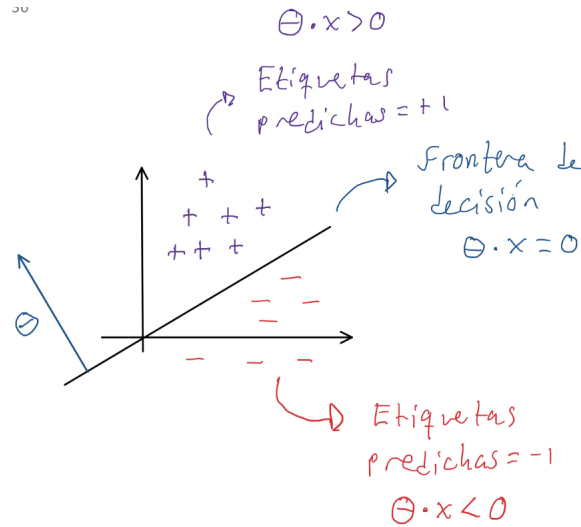
Iniciemos nuestro estudio con los clasificadores lineales, los cuales son mapeos restringidos entre ejemplos y etiquetas. Formalmente, consideraremos clasificadores de la forma:

$$h(x; \theta) = \text{sgn}(\theta_1 x_1 + \dots + \theta_d x_d) = \text{sgn}(\theta \cdot x) = \begin{cases} +1, & \theta \cdot x > 0 \\ -1, & \theta \cdot x \leq 0 \end{cases}$$

Donde  $\theta \cdot x = \theta^T x$  y  $\theta = [\theta_1, \dots, \theta_d]^T$  es un vector columna de parámetros de valores reales donde diferentes configuraciones de  $\theta$  generarán diferentes clasificadores. Se dice entonces, que el conjunto de clasificadores lineales centrados en el origen son parametrizados por  $\theta \in \mathbb{R}^d$ . Desde un punto de vista geométrico, podemos imaginar que para un conjunto fijo de parámetros  $\theta$  el clasificador lineal cambiará su predicción de acuerdo únicamente con el signo de la función  $\theta \cdot x$ . En el espacio de vectores de características, esto corresponde a cruzar la frontera de decisión donde el argumento de la función signo es igual a cero. Como esta superficie es de naturaleza lineal, ¿Qué representa el vector de parámetros  $\theta$ ? Claramente, es la dirección del vector gradiente a la superficie y por tanto, la dirección hacia la cual



la superficie crece más rápido. De esta forma, los ejemplos del lado de la frontera donde apunta  $\theta$  serán clasificados como positivos ( $\theta \cdot x > 0$ ). Esto puede ser visto al reescribir  $\theta \cdot x$  como  $\|x\| \|\theta\| \cos(x, \theta)$  y observar que para toda  $x$  del lado izquierdo de la frontera,  $\cos(x, \theta)$  es positivo como muestra la figura 3.1.



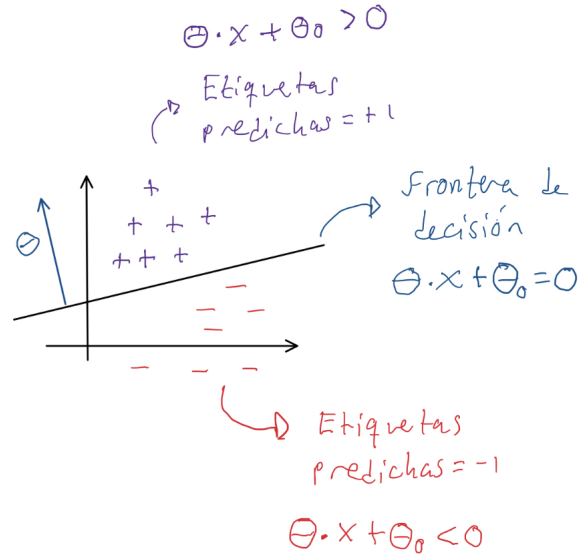
**Figura 3.1:** Clasificador lineal sin sesgo.

Similarmente, puede ser mostrado para el caso negativo. El caso particular donde los puntos se encuentren exactamente sobre la frontera de decisión es un tema puro de definición ( $-1$  en nuestro caso).

Se puede extender el conjunto de clasificadores lineales incluyendo un parámetro de sesgo  $\theta_0$ , lo cual nos permitirá ubicar la frontera de decisión en cualquier parte de  $\mathbb{R}^d$  y no solo anclado al origen. Así, definimos el clasificador lineal con sesgo (o simplemente clasificador lineal) como sigue:

$$h(x; \theta; \theta_0) = \text{sgn}(\theta \cdot x + \theta_0) = \begin{cases} +1, & \theta \cdot x + \theta_0 > 0 \\ -1, & \theta \cdot x + \theta_0 \leq 0 \end{cases}$$

Se obvia que al ser  $\theta_0 = 0$  obtendríamos el clasificador lineal centrado en el origen. Aunque hallamos incluido el parámetro de sesgo, geométricamente tendremos una superficie lineal paralela y por tanto  $\theta$  seguirá siendo ortogonal y la frontera de clasificación seguirá mostrando el mismo comportamiento como muestra la figura 3.2



**Figura 3.2:** Clasificador lineal con sesgo.

### 3.2. Consideraciones sobre algoritmos para aprendizaje para clasificadores lineales

Ahora que hemos elegido un primer tipo de clasificador, nuestra tarea se concentrará en seleccionar uno de forma adecuada para el conjunto de entrenamiento  $S_n = \{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}$ . Existen fundamentalmente dos enfoques para aprendizaje de clasificadores lineales a través de los datos. El primero consiste en diseñar algoritmos en línea que consideran el ejemplo de entrenamiento en turno y ajustan los parámetros ligeramente en respuesta a posibles errores, dichos algoritmos son simples de entender y eficientes computacionalmente debido a que escalan linealmente con respecto al tamaño del conjunto de entrenamiento. Estos algoritmos también no tienen a menudo una función objetivo definida de forma estática que minimiza el error con respecto a los parámetros, sin embargo, cuentan típicamente con garantías teóricas de generalización (nuestro objetivo final). Los algoritmos que revisaremos en esta categoría serán el perceptrón y pasivo-agresivo. El segundo enfoque se concentra en el hecho de definir un método que puntualmente minimice una función objetivo con respecto a los parámetros del clasificador. Dicha función objetivo estará definida en términos del error, costo, pérdida o riesgo de un clasificador lineal.

### 3.3. Error de entrenamiento

Definamos una función que esté basada los errores que comete el clasificador. Para ello, nuestra meta será encontrar  $\theta$  tal que el clasificador cometa los menos errores posibles en el conjunto de entrenamiento, esto es:

$$\mathcal{E}_n(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}[y^{(i)} (\theta \cdot x^{(i)} + \theta_0) \leq 0]$$

donde  $\mathbb{I}[\cdot]$  devuelve 1 si la expresión lógica del argumento es verdadera y 0 en caso contrario. El error de entrenamiento corresponderá en la fracción de ejemplos de entrenamiento en los que el clasificador con parámetros  $\theta$  y  $\theta_0$  predice incorrectamente la etiqueta. De esta forma, el error de entrenamiento  $\mathcal{E}_n(\theta, \theta_0)$  podrá ser evaluado para cualesquiera elecciones de los parámetros  $\theta$  y  $\theta_0$  y en consecuencia, se podrá hacer la optimización con respecto a los mismos. Ahora discutiremos cuál sería un algoritmo razonable para hallar  $\hat{\theta}$  y  $\hat{\theta}_0$  tal que  $\mathcal{E}_n(\theta, \theta_0)$  sea mínimo. Consideremos inicialmente el caso donde existe un clasificador perfecto (lo cual se conoce como factibilidad):

**Definición 3.1.**  $S_n = \{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}$  es *linealmente separable con respecto al origen* si existe un vector de parámetros  $\hat{\theta}$  tal que  $y^{(i)} (\hat{\theta} \cdot x^{(i)}) > 0$  para todo  $i = 1, \dots, n$ .

**Definición 3.2.**  $S_n = \{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}$  es *linealmente separable* si existe un vector de parámetros  $\hat{\theta}$  y un parámetro de sesgo  $\hat{\theta}_0$  tal que  $y^{(i)} (\hat{\theta} \cdot x^{(i)} + \hat{\theta}_0) > 0$  para todo  $i = 1, \dots, n$ .

### 3.4. El perceptrón

#### 3.4.1. Deducción del algoritmo

Este algoritmo forma parte del conjunto de algoritmos llamado *algoritmos en línea impulsados por error*, los cuales garantizan encontrar una solución con error de entrenamiento nulo en el caso factible, sin embargo, en el caso no factible no se tienen garantías de su comportamiento. Consideremos en primera instancia el caso centrado en el origen, en este caso, se inicializa con un clasificador simple, por ejem-

plo,  $\theta = \vec{0}$ , para posteriormente ajustar sucesivamente los parámetros al presentar uno a uno los ejemplos de entrenamiento y de esa manera corregir los errores cometidos. El perceptrón es el algoritmo más simple y antiguo de los algoritmos de este tipo. Fue propuesto por Frank Rosenblatt[Rosenblatt, 1958] un Psicólogo estadounidense reconocido mundialmente por sus aportes al campo de la inteligencia artificial. A continuación, se muestra el algoritmo:

---

**Algorithm 1** Perceptrón sin sesgo

---

```
1: procedure PERCEPTRON( $\{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}, T$ )
2:    $\theta \leftarrow \vec{0}$ 
3:   for  $t = 1, \dots, T$  do
4:     for  $i = 1, \dots, n$  do
5:       if  $y^{(i)} (\theta \cdot x^{(i)}) \leq 0$  then
6:          $\theta \leftarrow \theta + y^{(i)} x^{(i)}$ 
7:   return  $\theta$ 
```

---

El algoritmo consiste en recorrer el conjunto de entrenamiento  $T$  veces, ajustando los parámetros levemente en respuesta a cualquier error de clasificación. Si no existe error, los parámetros no se actualizan. En este entendido, se puede establecer una función de correspondencia entre los parámetros y la aparición de errores. Sea entonces  $\theta^{(k)}$  los parámetros obtenidos después de exactamente  $k$  errores de clasificación en el conjunto de entrenamiento, analicemos el comportamiento del algoritmo.

Sabemos que cuando se comete un error,  $y^{(i)} (\theta^{(k)} \cdot x^{(i)}) \leq 0$ , supongamos que se comete un error en  $x^{(i)}$ , por tanto, los parámetros se actualizarán de acuerdo a la expresión  $\theta^{(k+1)} = \theta^{(k)} + y^{(i)} x^{(i)}$ , si consideramos clasificar nuevamente el mismo ejemplo  $x^{(i)}$  justo después de la actualización usando los nuevos parámetros  $\theta^{(k+1)}$ , entonces:

$$\begin{aligned} y^{(i)} (\theta^{(k+1)} \cdot x^{(i)}) &= y^{(i)} (\theta^{(k)} + y^{(i)} x^{(i)}) \cdot x^{(i)} \\ &= y^{(i)} (\theta^{(k)} \cdot x^{(i)}) + (y^{(i)})^2 (x^{(i)} \cdot x^{(i)}) \\ &= y^{(i)} (\theta^{(k)} \cdot x^{(i)}) + \|x^{(i)}\|^2 \end{aligned}$$

En otras palabras, el valor de  $y^{(i)} (\theta^{(k)} \cdot x^{(i)})$  se incrementa por el resultado de la actualización. El algoritmo tendrá entonces convergencia garantizada para  $T$  lo suficientemente grande en el caso factible. La convergencia será independiente del orden en que sean presentados los ejemplos, la única diferencia radicará en el número de errores que cometerá el algoritmo antes de converger. El algoritmo se extiende naturalmente al caso sesgado:

---

**Algorithm 2** Perceptrón

---

```
1: procedure PERCEPTRON( $\{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}, T$ )
2:    $\theta \leftarrow \vec{0}, \theta_0 \leftarrow 0$ 
3:   for  $t = 1, \dots, T$  do
4:     for  $i = 1, \dots, n$  do
5:       if  $y^{(i)} (\theta \cdot x^{(i)} + \theta_0) \leq 0$  then
6:          $\theta \leftarrow \theta + y^{(i)} x^{(i)}$ 
7:          $\theta_0 \leftarrow \theta_0 + y^{(i)}$ 
8:   return  $\theta, \theta_0$ 
```

---

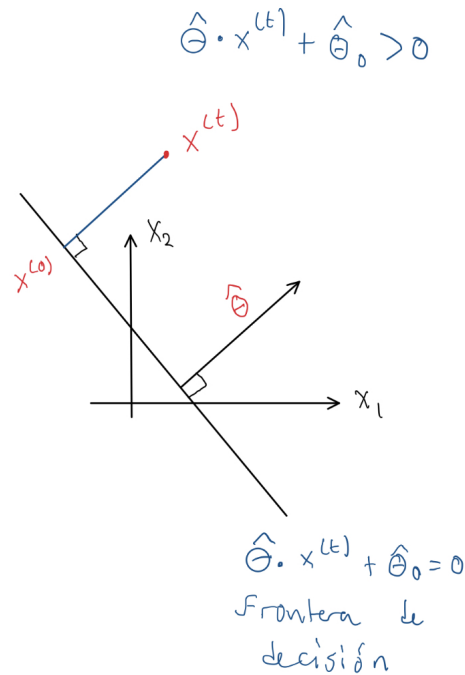
El racional detrás de la actualización del parámetro de sesgo se basa en el hecho de considerar la asignación de una coordenada unitaria a cada ejemplo de entrenamiento, es decir, mapeamos nuestro espacio de características de  $x \in \mathbb{R}^d$  a  $x' \in \mathbb{R}^{d+1}$  tal que  $x' = [x_1, \dots, x_d, 1]^T$ , y los parámetros  $\theta \in \mathbb{R}^d$  a  $\theta' \in \mathbb{R}^{d+1}$  tal que  $\theta' = [\theta_1, \dots, \theta_d, \theta_0]^T$ .

### 3.4.2. Teorema de convergencia del perceptrón

La velocidad de convergencia del perceptrón dependerá de que tan "cerca" se encuentran los ejemplos de entrenamiento de la frontera de decisión, formalizando:

**Definición 3.3.** *El conjunto de entrenamiento  $S_n = \{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}$  es linealmente separable con margen  $\gamma$  si existe un vector de parámetros  $\hat{\theta}$  y un parámetro de sesgo  $\hat{\theta}_0$  tal que  $y^{(i)} (\hat{\theta} \cdot x^{(i)} + \hat{\theta}_0) / \|\hat{\theta}\| \geq \gamma$  para toda  $i = 1, \dots, n$ .*

Previo a la exposición del teorema, tomemos en cuenta algunos razonamientos, de acuerdo con la figura 3.3:



**Figura 3.3:** Proyección ortogonal sobre la frontera

Si consideramos cualquier clasificador lineal definido por  $\hat{\theta}$  y  $\hat{\theta}_0$  que clasifica correctamente los ejemplos de entrenamiento, entonces:

$$\frac{y^{(t)} (\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0)}{\|\hat{\theta}\|}$$

Es la distancia ortogonal entre el punto  $x^{(t)}$  y la frontera de decisión  $\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0 = 0$ . En la figura 3.3 se muestra el punto  $x^{(t)}$  (clasificado positivamente) y su proyección ortogonal en la frontera, el punto  $x^{(0)}$ . Dado que  $y^{(t)} = 1$  y  $\hat{\theta} \cdot x^{(0)} + \hat{\theta}_0 = 0$  ( $x^{(0)}$  se encuentra sobre la frontera), entonces:

$$\frac{y^{(t)} (\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0)}{\|\hat{\theta}\|} = \frac{\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0}{\|\hat{\theta}\|} - \frac{\hat{\theta} \cdot x^{(0)} + \hat{\theta}_0}{\|\hat{\theta}\|} = \frac{\hat{\theta} \cdot (x^{(t)} - x^{(0)})}{\|\hat{\theta}\|} = \|x^{(t)} - x^{(0)}\|$$

Con lo anterior en mente, pasemos a enunciar el teorema de convergencia para el perceptron.

**Teorema 3.1** (Teorema de convergencia del perceptrón). *Dadas las premisas:*

i) Existe  $\theta^*$  tal que  $y^{(i)} (\theta^* \cdot x^{(i)}) / \|\theta^*\| \geq \gamma$  para todo  $i = 1, \dots, n$  y algún  $\gamma > 0$ .

ii) Todos los ejemplos están acotados  $\|x^{(i)}\| \leq R, i = 1, \dots, n$ .

Entonces, el algoritmo del perceptrón hará a lo más  $(R/\gamma)^2$  errores en los ejemplos y etiquetas  $(x^{(i)}, y^{(i)})$ ,  $i = 1, \dots, n$ .

*Demostración.* Nótese que la frontera de decisión  $\{x : \theta \cdot x = 0\}$  depende únicamente de la orientación, mas no de la magnitud de  $\theta$ . Por tanto, será suficiente encontrar un  $\theta$  suficientemente cerca de  $\theta^*$ . Para este fin, analizaremos como la expresión:

$$\cos(\theta^{(k)}, \theta^*) = \frac{\theta^{(k)} \cdot \theta^*}{\|\theta^{(k)}\| \|\theta^*\|}$$

Se comporta como una función de  $k$  (el número de errores que comete) donde  $\theta^{(k)}$  es el vector de parámetros actualizado después de  $k$  errores. Si expresamos el coseno en dos partes  $\theta^{(k)} \cdot \theta^* / \|\theta^*\|$  y  $\|\theta^{(k)}\|$  lo podemos ver como el ratio entre cada una de dichas partes. Si el  $k$ -ésimo error ocurriese en el ejemplo  $(x^{(i)}, y^{(i)})$ , podemos escribir la siguiente expresión:

$$\frac{\theta^{(k)} \cdot \theta^*}{\|\theta^*\|} = \frac{(\theta^{(k-1)} + y^{(i)} x^{(i)}) \cdot \theta^*}{\|\theta^*\|} = \frac{\theta^{(k-1)} \cdot \theta^*}{\|\theta^*\|} + \frac{y^{(i)} x^{(i)} \cdot \theta^*}{\|\theta^*\|} \geq \frac{\theta^{(k-1)} \cdot \theta^*}{\|\theta^*\|} + \gamma \geq k\gamma$$

En la expresión anterior, hemos utilizado la parte i) de nuestra hipótesis dado que  $\theta^*$  clasifica correctamente todos los ejemplos con margen  $\gamma$ . Así, en cada actualización, este término se incrementará en  $\gamma$  unidades y en consecuencia, podrá probarse mediante inducción. Utilizamos entonces  $k = 1$  como caso base,  $\frac{\theta^{(1)} \cdot \theta^*}{\|\theta^*\|} = 0$ , debido a que de acuerdo a la inicialización  $\theta^{(0)} = 0$  y  $\frac{\theta^{(1)} \cdot \theta^*}{\|\theta^*\|} \geq \gamma$  y en cada paso de la inducción, dicho valor será incrementado por  $\gamma$ .

Si expandimos ahora  $\theta^{(k)}$ , tenemos:

$$\begin{aligned} \|\theta^{(k)}\|^2 &= \|\theta^{(k-1)} + y^{(i)} x^{(i)}\|^2 = \|\theta^{(k-1)}\|^2 + 2y^{(i)} \theta^{(k-1)} \cdot x^{(i)} + \|x^{(i)}\|^2 \\ &\leq \|\theta^{(k-1)}\|^2 + R^2 \leq kR^2 \end{aligned}$$

Debido a que se asumió  $\theta^{(0)}$ , se usa explícitamente el hecho de que  $\theta^{(k-1)}$  comete un error en  $(x^{(i)}, y^{(i)})$ , así como la parte ii) de nuestra hipótesis para acotar y dado



que  $\|\theta^{(k)}\| \leq \sqrt{k}R$ , entonces:

$$\cos(\theta^{(k)}, \theta^*) = \frac{\theta^{(k)} \cdot \theta^*}{\|\theta^{(k)}\| \|\theta^*\|} \geq \frac{k\gamma}{\sqrt{k}R} = \sqrt{k} \frac{\gamma}{R}$$

Dado que el coseno está acotado por la unidad, no será posible continuar cometiendo errores (seguir incrementando  $k$ ). Si resolvemos para  $k$  a partir de  $1 \geq \sqrt{k}(\gamma/R)$ , obtendremos que  $k \leq (R/\gamma)^2$  como quería probarse en un principio.  $\square$

### 3.4.3. Perceptron medio

Cuando el conjunto de datos no es linealmente separable, el algoritmo del perceptrón continuará hasta después de  $T$  iteraciones. En este caso, será posible ajustar el algoritmo de tal suerte que se logre obtener un clasificador razonable aunque se presente esta casuística. La estrategia a seguir consistirá en promediar los parámetros del algoritmo en cada iteración y no solamente los parámetros después de la actualización, de esa manera se considerarán aquellos parámetros que funcionan por más tiempo que su complemento. Lo anterior es llamado perceptrón medio, escribimos el algoritmo a continuación:

---

#### Algorithm 3 Algoritmo Perceptrón Medio

---

```

1: procedure PERCEPTRON MEDIO( $\{(x^{(i)}, y^{(i)}), i = 1, \dots, n\}, T$ )
2:    $\theta \leftarrow \vec{0}, \theta_0 = 0$ 
3:    $\bar{\theta} \leftarrow \vec{0}, \bar{\theta}_0 = 0$ 
4:    $c \leftarrow 1$ 
5:   for  $t = 1, \dots, T$  do
6:     for  $i = 1, \dots, n$  do
7:       if  $y^{(i)} (\theta \cdot x^{(i)} + \theta_0) \leq 0$  then
8:          $\theta \leftarrow \theta + y^{(i)} x^{(i)}$ 
9:          $\theta_0 \leftarrow \theta_0 + y^{(i)}$ 
10:       $\bar{\theta} \leftarrow \bar{\theta} + c y^{(i)} x^{(i)}$ 
11:       $\bar{\theta}_0 \leftarrow \bar{\theta}_0 + c y^{(i)}$ 
12:     $c \leftarrow c - 1 / (nT)$ 
13:  return  $\bar{\theta}, \bar{\theta}_0$ 

```

---

### 3.5. Funciones de pérdida

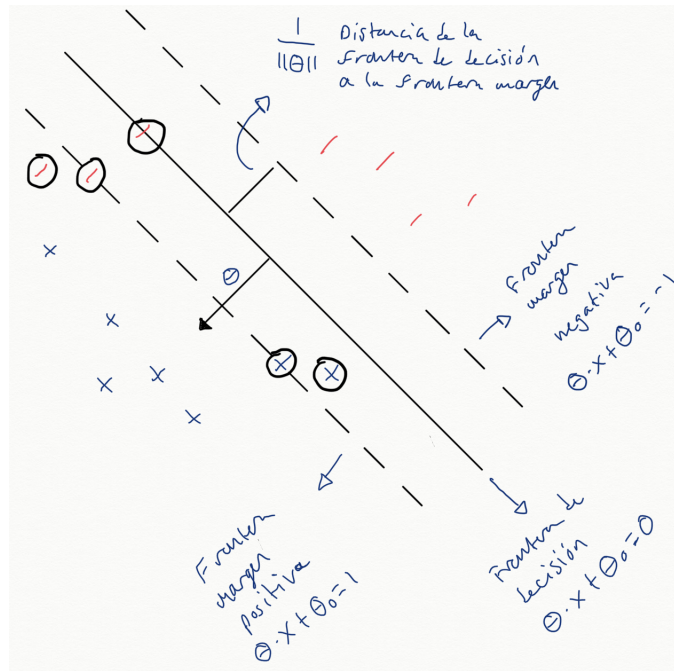
Un enfoque distinto al que hemos revisado se basa en el hecho de que los errores que un clasificador comete no sean tratados de la misma forma. El racional detrás de esto es que aquellos puntos que están muy cerca de la frontera de decisión deberían ser menos penalizados que aquellos que están claramente en el lado incorrecto de la frontera. Formalmente, lo anterior consiste en construir una “Función de pérdida” que de manera simple especifique numéricamente que tan mal hemos clasificado los ejemplos. Por ejemplo, en el caso del perceptrón, se asume una pérdida binaria simple:

$$Loss_{0,1}(y(\theta \cdot x + \theta_0)) = \mathbb{I}[y(\theta \cdot x + \theta_0) \leq 0]$$

Nótese que no se considera la magnitud del error en ningún momento (solamente presencia), por tanto, se propondrá una función de pérdida más sensible al penalizar cualquier predicción para la cual el acuerdo descienda a menos de uno e incrementando la penalización para violaciones más grandes, dicha función se conoce como *función bisagra* definida como sigue:

$$Loss_h(y(\theta \cdot x + \theta_0)) = \max\{0, 1 - y(\theta \cdot x + \theta_0)\} = \begin{cases} 1 - y(\theta \cdot x + \theta_0) & \text{si } y(\theta \cdot x + \theta_0) \leq 1 \\ 0 & \text{en otro caso} \end{cases}$$

Para interpretar geométricamente como las distintas elecciones de los parámetros  $\theta$  y  $\theta_0$  afectan a la función bisagra en la que se incurre en cada ejemplo de entrenamiento, no debemos considerar más las fronteras de decisión, sino las fronteras paralelas a éstas en una vecindad dada conocidas como las fronteras margen positiva y negativa  $\{x : \theta \cdot x + \theta_0 = 1\}$  y  $\{x : \theta \cdot x + \theta_0 = -1\}$  respectivamente. La figura 3.4 ilustra este hecho:



**Figura 3.4:** Fronteras margen y de decisión para un clasificador lineal. Los puntos resaltados incurren en una pérdida bisagra distinta de cero debido a que violan las fronteras margen.

Para que un ejemplo etiquetado como positivo obtenga una pérdida bisagra de cero, deberá estar en el lado correcto (positivo) de la frontera margen positiva  $\{x : \theta \cdot x + \theta_0 = 1\}$ . Si moviésemos el ejemplo de entrenamiento más allá de la frontera margen relevante, la pérdida bisagra se incrementaría linealmente como función del grado de violación. Las fronteras margen se encontrarán a una distancia de  $1/\|\theta\|$  unidades de la frontera de decisión. Para ilustrar lo anterior, considérese un punto  $(x, y)$  que está correctamente clasificado y se encuentra exactamente en la frontera margen. Resultante de esto,  $y(\theta \cdot x + \theta_0) = 1$  y la distancia ortogonal desde la frontera es  $y(\theta \cdot x + \theta_0) / \|\theta\| = 1/\|\theta\|$ . Si mantenemos la frontera de decisión sin cambios pero incrementamos  $\|\theta\|$ , entonces las fronteras margen se acercarán, en caso contrario, un valor pequeño de  $\|\theta\|$  implicará que las fronteras margen se alejen, incrementando en consecuencia, la probabilidad de violaciones. En virtud de lo anterior, la búsqueda de los parámetros  $\theta$  y  $\theta_0$  se hará a la par que se mide el desempeño del clasificador con respecto a la función de pérdida bisagra. El objetivo será buscar clasificadores con  $\|\theta\|$  pequeño, esto forzará la selección de clasificadores con mucho “espacio” entre los ejemplos positivos y negativos, es

decir, clasificadores con amplio margen.

### 3.6. Algoritmo Pasivo-Agresivo

El algoritmo Pasivo-Agresivo es un algoritmo en línea al igual que el perceptrón. La diferencia se centra en el hecho de que el algoritmo trata de minimizar la pérdida bisagra de forma explícita en cada ejemplo (Agresivo) y al mismo tiempo mantiene los parámetros cerca de donde se encontraban antes de haber visto el ejemplo (Pasivo). El enfoque pasivo es necesario para evitar que se sobrescriba lo que se ha aprendido en los ejemplos previos. El algoritmo tiene garantías similares a las del perceptrón.

Iniciemos considerando el algoritmo sin el parámetro de sesgo  $\theta_0$ . Suponiendo que  $\theta^{(k)}$  son los parámetros actuales lo cuales fueron inicializados en cero previamente, es decir,  $\theta^0 = \vec{0}$ . Si  $(x, y)$  es el ejemplo de entrenamiento en cuestión, entonces el algoritmo buscará  $\theta = \theta^{(k+1)}$  que minimice:

$$\frac{\lambda}{2} \|\theta - \theta^{(k)}\|^2 + \text{Loss}_h(y\theta \cdot x)$$

El parámetro  $\lambda$  determina que tan pasiva es la actualización. Entre mayor el valor de  $\lambda$  más cercanos se encontrarán los parámetros a  $\theta^{(k)}$  mientras que valores pequeños de  $\lambda$  permitirán mayores desviaciones para minimizar directamente  $\text{Loss}_h(y\theta \cdot x)$ . La actualización resultante se parecerá mucho a la actualización del perceptrón, a saber:

$$\theta^{(k+1)} = \theta^{(k)} + \eta y x$$

Con la única diferencia de que contamos con un parámetro de tasa de aprendizaje (tamaño de paso) llamado  $\eta$  el cual depende de  $\lambda$  y  $\theta^{(k)}$  así como del ejemplo de entrenamiento  $(x, y)$ .

Observemos más a detalle la función de pérdida. El término  $\|\theta - \theta^{(k)}\|^2$  no considera para nada la dirección, únicamente se centra en que tanto nos alejamos de  $\theta^{(k)}$ , por otro lado, el término de pérdida direcciona de manera intrínseca. Para ello debemos considerar el gradiente, cuya dirección positiva indica hacia donde crece más rápido mientras que la dirección negativa nos apuntará hacia el descenso

más rápido, así:

$$-\nabla_{\theta} Loss(y\theta \cdot x) = -\nabla_{\theta} \begin{cases} 1 - y\theta \cdot x & \text{si } y\theta \cdot x \leq 1 \\ 0 & \text{en otro caso.} \end{cases} = \begin{cases} yx & \text{si } y\theta \cdot x \leq 1 \\ 0 & \text{en otro caso.} \end{cases}$$

Nótese que la dirección hacia donde nos movemos no depende en lo absoluto de  $\theta$ , podemos entonces asumir que  $\theta^{(k+1)} = \theta^{(k)} + \eta yx$  para algún  $\eta$  incluso cero. Si minimizamos entonces con respecto al parámetro  $\eta$ , tendremos:

$$\eta = \min \left\{ \frac{Loss_h(y\theta^{(k)} \cdot x)}{\|x\|^2}, \frac{1}{\lambda} \right\}$$

Lo anterior deriva en el hecho de que aquellos ejemplos correctamente clasificados más allá de las fronteras margen no producirán actualización en los parámetros, por otro lado, valores grandes de  $\lambda$  producirán valores pequeños de  $\eta$  debido a que no puede sobrepasar el valor  $1/\lambda$ . Adicionalmente, las actualizaciones resultantes serán siempre mayormente restringidas que las correspondientes al perceptrón. Enlistemos ahora los pasos del algoritmo:

---

**Algorithm 4** Algoritmo Pasivo-Agresivo

---

```

1: procedure PASIVO-AGRESIVO( $\{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}, \lambda, T$ )
2:    $\theta \leftarrow \vec{0}$ 
3:   for  $t = 1, \dots, T$  do
4:     for  $i = 1, \dots, n$  do
5:        $\eta \leftarrow \min \left\{ \frac{Loss_h(y\theta^{(t)} \cdot x)}{\|x\|^2}, \frac{1}{\lambda} \right\}$ 
6:        $\theta \leftarrow \theta + \eta y^{(i)} x^{(i)}$ 
7:   return  $\theta$ 

```

---

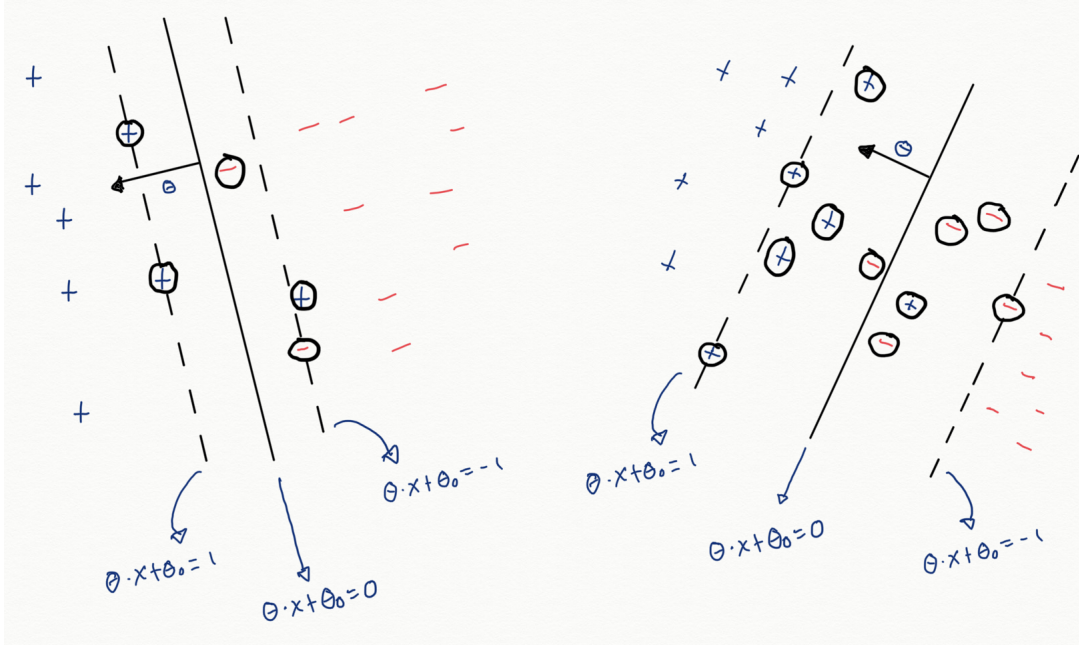
Podemos considerar al parámetro  $\lambda$  en el algoritmo como un *parámetro de regularización* cuyo rol es mantener cada actualización de parámetros pequeña, adicionalmente, dado que los parámetros iniciales son nulos mantiene pequeña la norma del vector de parámetros  $\|\theta\|$  y en consecuencia, buscará una solución de amplio margen. Cuando dicha solución es inexistente, entonces  $\|\theta\|$  se incrementará lentamente conforme crece  $T$ , es importante destacar que tanto  $\lambda$  como  $T$  afectan la calidad de la solución encontrada. Por último, una mejor versión del algoritmo considera un paso de promedio sobre los parámetros (similar al perceptrón medio), es decir, devuelve  $\frac{1}{nT} (\theta^{(1)} + \dots + \theta^{(nT)})$  en lugar de solamente el último vector de parámetros  $\theta^{(nT)}$ .

### 3.7. Máquinas de soporte vectorial

La discusión se ha centrado hasta el momento en enfoques respecto a minimización de errores (perceptrón) o pérdida (pasivo-agresivo). En esta sección nos centraremos en formular el problema directamente como un problema de minimización, considerando todo el conjunto de entrenamiento a la vez y no en línea. En particular, revisaremos lo correspondiente a las máquinas de soporte vectorial (también conocidas como máquinas vector soporte). Las máquinas de soporte vectorial persiguen dos objetivos simultáneos aunque opuestos. En primera instancia, buscan minimizar la pérdida bisagra promedio en los ejemplos de entrenamiento y posteriormente, separar las fronteras margen al reducir  $\|\theta\|$ . Dichos objetivos deberán buscar el balance. Para ello, se formula el problema de encontrar  $\theta$  y  $\theta_0$  tal que se minimice la expresión:

$$\frac{1}{n} \sum_{i=1}^n Loss_h(y^{(i)} (\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2} \|\theta\|^2$$

Donde el parámetro  $\lambda$  será el encargado de balancear ambos objetivos, mientras que el término  $\|\theta\|^2/2$  será el parámetro de regularización que empuja los parámetros hacia una solución por defecto (el vector cero). En la figura 3.5 se muestra como cambia la solución al cambiar el valor de  $\lambda$ .



**Figura 3.5:** El lado izquierdo muestra como se busca minimizar las violaciones con un valor  $\lambda$  pequeño, mientras que el lado derecho valores mayores de  $\lambda$  aceptan violaciones del margen en virtud de incrementar el tamaño del mismo.

Para encontrar los valores de  $\theta$  y  $\theta_0$  que minimicen el error, reformularemos la función objetivo de la máquina de soporte vectorial como un problema de *programación cuadrática*, quedando:

$$\text{mín } \frac{1}{n} \sum_{i=1}^n \xi_i + \frac{\lambda}{2} \|\theta\|^2$$

*sujeto a :*

$$y^{(i)} (\theta \cdot x^{(i)} + \theta_0) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

$$i = 1, \dots, n$$

Las variables de holgura  $\xi_i$  se introducen para representar la pérdida bisagra en

términos de restricciones lineales y en consecuencia, el problema tendrá una función objetivo cuadrática con restricciones lineales. Una consideración importante es que el problema propuesto no escala bien con el número de ejemplos por lo que está limitado a problemas pequeños (del orden de los 10 mil ejemplos).

Para resolver esta situación, recurriremos a técnicas aproximadas mucho más simples y escalables conocidas como *métodos de gradiente estocástico*. Dichos métodos funcionan igual que los algoritmos en línea, considerando uno a uno los ejemplos de entrenamiento, mueven los parámetros ligeramente en dirección del gradiente negativo y continúan con el siguiente ejemplo. Aunque es parecido al algoritmo Pasivo-Agresivo, la diferencia sustancial consiste en que el término de regularización controla explícitamente la norma  $\|\theta\|$  en lugar de intentar mantener pequeña la magnitud de las actualizaciones. Escribamos primero la función objetivo de la máquina de soporte vectorial como un promedio de términos. En esta forma, podemos tomar de forma aleatoria un término y aplicar una actualización mediante descenso por gradiente estocástico, específicamente:

$$\begin{aligned}
\theta &= \theta - \eta \nabla_{\theta} \left[ \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)})) + \frac{\lambda}{2} \|\theta\|^2 \right] \\
&= \theta - \eta \nabla_{\theta} [\text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)}))] - \eta \nabla_{\theta} \left[ \frac{\lambda}{2} \|\theta\|^2 \right] \\
&= \theta - \eta \nabla_{\theta} [\text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)}))] - \eta \lambda \theta \\
&= (1 - \lambda \eta) \theta - \eta \nabla_{\theta} [\text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)}))] \\
&= (1 - \lambda \eta) \theta + \eta \begin{cases} y^{(i)} x^{(i)} & \text{si } y^{(i)}(\theta \cdot x^{(i)}) \leq 1 \\ 0 & \text{en otro caso} \end{cases}
\end{aligned}$$

Donde  $\eta$  es conocida como tasa de aprendizaje, si se decrementa apropiadamente, las actualizaciones por gradiente descendiente garantizan la convergencia hacia la misma solución de la máquina de soporte vectorial.  $\eta_k$  puede cambiar después de la  $k$ -ésima actualización tal que  $\sum_{k=1}^{\infty} \eta_k = \infty$  y  $\sum_{k=1}^{\infty} \eta_k^2 < \infty$ . Estas condiciones garantizan que sin importar donde inicia o termina la optimización nos moveremos hacia el óptimo y en consecuencia, el ruido inherente a la selección estocástica tiende a cero. Lo anterior se consume en el llamado algoritmo PEGASOS acrónimo inglés para *Primal Estimated sub-GrAdient SOLver for SVM*.

Por último, discutiremos lo correspondiente al hiperplano de máximo margen para el caso factible. Para ello, haremos un ligero cambio al problema de optimiza-



---

**Algorithm 5** Algoritmo PEGASOS

---

```
1: procedure PEGASOS( $\{(x^{(i)}, y^{(i)}) , i = 1, \dots, n\}, \lambda, T$ )
2:    $\theta \leftarrow \vec{0}$ 
3:   for  $t = 1, \dots, T$  do
4:     Elegir al azar  $i \in \{1, \dots, n\}$ 
5:      $\eta \leftarrow 1/t$ 
6:     if  $y^{(i)}\theta \cdot x^{(i)} \leq 1$  then
7:        $\theta \leftarrow (1 - \eta\lambda)\theta + \eta y^{(i)}x^{(i)}$ 
8:     else
9:        $\theta \leftarrow (1 - \eta\lambda)\theta$ 
10:  return  $\theta$ 
```

---

ción de la máquina de soporte vectorial, a saber:

$$\left(\frac{1}{\lambda n}\right) \sum_{i=1}^n \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{1}{2}\|\theta\|^2$$

Llamaremos  $C$  al término  $(\frac{1}{\lambda n})$ , este parámetro nos indicará que tanto énfasis se hará en las pérdidas del entrenamiento. En el caso cuando  $C \rightarrow \infty$ , si forzamos la inexistencia de violaciones de margen, el problema se reduce a:

$$\min \frac{1}{2}\|\theta\|^2 \quad \text{s.a.} \quad y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1, i = 1, \dots, n$$

Que es también un problema de programación cuadrática y geoméricamente, al minimizar  $\|\theta\|$  se empujan las fronteras margen hasta eliminar las violaciones de las restricciones. Esto arrojará un separador lineal cuya frontera de decisión está lo más lejos posible de los ejemplos de entrenamiento como se muestra en la figura 3.6

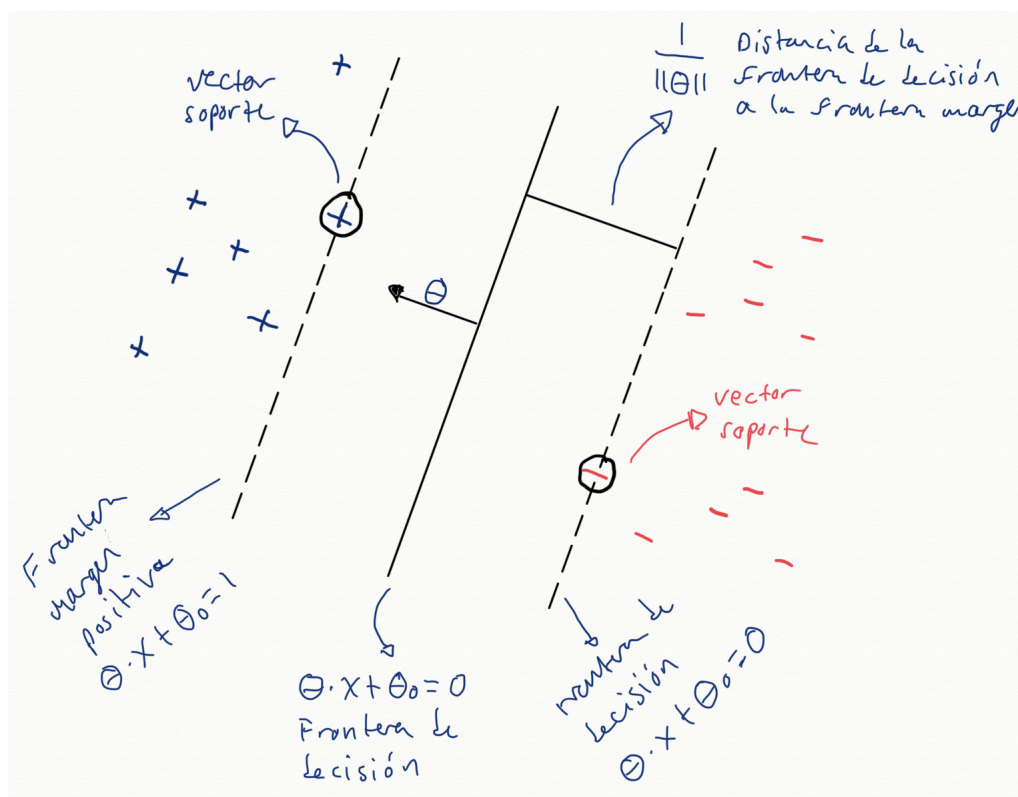


Figura 3.6: Separador lineal de margen máximo

## 4. Regresión Lineal

Es nuestro interés ahora predecir un valor continuo en lugar de solo una etiqueta. En virtud de lo anterior, aquellos problemas de modelación supervisada donde la variable objetivo es de naturaleza continua, se conocen como problemas de regresión. Formalmente, la tarea consistirá en encontrar una función  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  tal que  $f(x) \approx y$  para todos los ejemplos de prueba  $x$ . De igual forma que en clasificación, seleccionaremos  $f$  basado en un conjunto finito de ejemplos de entrenamiento antes de utilizarlo en ejemplos de prueba. Para lograr la generalización del modelo, debemos acotar la familia de funciones a considerar y al igual que en problemas de clasificación, iniciaremos con funciones lineales simples. Una función de regresión lineal es simplemente una función lineal de los vectores de características, a

saber:

$$f(x; \theta; \theta_0) = \theta \cdot x + \theta_0 = \sum_{i=1}^d \theta_i x_i + \theta_0$$

Las diferentes elecciones de los parámetros  $\theta \in \mathbb{R}^d$  y  $\theta_0 \in \mathbb{R}$  darán lugar al conjunto de funciones  $\mathcal{F}$ . Por tanto, nuestra tarea de aprendizaje consistirá en elegir una función  $f \in \mathcal{F}$  definida por los parámetros  $\theta \in \mathbb{R}^d$  y  $\theta_0 \in \mathbb{R}$  basados en el conjunto de entrenamiento  $S_n = \{(x^{(t)}, y^{(t)})\}, t = 1, \dots, n\}$  donde  $y^{(t)} \in \mathbb{R}$ .

## 4.1. Mínimos Cuadrados

Para medir el error de entrenamiento, se utilizará la pérdida promedio, también conocida como *riesgo empírico*:

$$R_n(\theta) = \frac{1}{n} \sum_{t=1}^n \text{Loss}(y^{(t)} - \theta \cdot x^{(t)})$$

Por simplicidad, se omite el parámetro  $\theta_0$ . Nótese que a diferencia de los problemas de clasificación, la función de pérdida depende de la diferencia entre el valor real  $y^{(t)}$  y su correspondiente predicción lineal  $\theta \cdot x^{(t)}$ . Aunque existen muchas formas posibles de definir la función de pérdida, usaremos una versión simple basada en el error cuadrático:  $\text{Loss}(z) = z^2/2$ , donde el multiplicar por  $1/2$  es elegido por convenciencia matemática. La idea detrás de este racional es permitir pequeñas discrepancias al mismo tiempo que castigamos severamente grandes diferencias, en consecuencia:

$$R_n(\theta) = \frac{1}{n} \sum_{t=1}^n \text{Loss}(y^{(t)} - \theta \cdot x^{(t)}) = \frac{1}{n} \sum_{t=1}^n \frac{(y^{(t)} - \theta \cdot x^{(t)})^2}{2}$$

Debemos recordar que nuestra meta es la generalización del modelo mas no la minimización de  $R_n(\theta)$ ; simplemente es una estrategia razonable debido a que no tenemos acceso al error de validación (generalización):

$$R_{n'}^{\text{test}}(\theta) = \frac{1}{n'} \sum_{t=n+1}^{n+n'} (y^{(t)} - \theta \cdot x^{(t)})^2$$

Veamos la relación que existe entre  $R_n(\theta)$  y  $R_{n'}^{\text{test}}(\theta)$ , en primera instancia, hallaremos  $\hat{\theta}$  al minimizar  $R_n(\theta)$ , sin embargo, el desempeño será medido conforme al

error de validación  $R_{n'}^{test}(\hat{\theta})$ . Dicho error puede ser grande debido principalmente a dos razones. La primera esta relacionada al *error de estimación* que consiste en que aunque la relación verdadera entre  $x$  y  $y$  sea lineal, esta será difícil de estimar en virtud de contar un conjunto de entrenamiento  $S_n$  pequeño o con ruido y por tanto nuestros parámetros estimados  $\hat{\theta}$  no serán del todo correctos. La segunda razón tiene que ver con el llamado *error estructural*, lo cual significa que aunque estemos estimando un mapeo de tipo lineal entre  $x$  y  $y$ , la relación subyacente entre ellos sea de naturaleza no lineal. Si este fuese el caso, incluso un conjunto de entrenamiento grande no sería de ayuda. Para reducir el error estructural, requeriremos ampliar el conjunto de funciones  $\mathcal{F}$ , por ejemplo, al incluir características polinómicas como coordenadas adicionales. El problema será que si contamos solamente con un conjunto con ruido  $S_n$ , será muy difícil seleccionar la función correcta dentro del amplio conjunto  $\mathcal{F}$  y el error de estimación necesariamente crecerá, es por ello que para lograr un aprendizaje efectivo debemos siempre balancear entre el error estructural y el error de estimación.

Cuando el problema de regresión lineal se formula desde un punto de vista estadístico, suponemos que la variable objetivo es generada por alguna función subyacente con el añadido de un componente de ruido aleatorio. En este caso el error estructural puede ser considerado como un “Sesgo”, es decir, que tan lejos de la verdadera función nos encontramos al considerar diferentes elecciones del conjunto de entrenamiento. Por otro lado, el error de estimación corresponde a la “varianza” de la función o del estimador del parámetro  $\hat{\theta}(S_n)$ . Los parámetros  $\hat{\theta}$  se obtienen mediante el conjunto de entrenamiento  $S_n$  y por tanto pueden ser vistos como funciones de  $S_n$ . Un estimador no es mas que un mapeo desde los datos hacia los parámetros, si consideramos un conjunto de funciones grande, nuestro sesgo será bajo, sin embargo, los parámetros estimados variaran ligeramente de un conjunto de entrenamiento a otro (alta varianza) debido a que puede encontrarse fácilmente una función que ajuste el ruido sin menor complicación.

Procedamos ahora a optimizar la función objetivo de mínimos cuadrados  $R_n(\theta)$ , para ello nos valdremos método de descenso por gradiente estocástico revisado previamente en el contexto de clasificación. En este caso será mas simple debido a la diferenciabilidad en todo punto de la función objetivo. El algoritmo seleccionará de manera aleatoria un ejemplo de entrenamiento y actualizará ligeramente los parámetros en dirección opuesta al gradiente, esto es:

$$\nabla_{\theta} \frac{(y^{(t)} - \theta \cdot x^{(t)})^2}{2} = (y^{(t)} - \theta \cdot x^{(t)}) \nabla_{\theta} (y^{(t)} - \theta \cdot x^{(t)}) = - (y^{(t)} - \theta \cdot x^{(t)}) x^{(t)}$$

Por tanto, el algoritmo podrá escribirse como sigue:

---

**Algorithm 6** Algoritmo descenso por gradiente estocástico para mínimos cuadrados
 

---

```

1: procedure LSSGD( $\{(x^{(t)}, y^{(t)}) , t = 1, \dots, n\}, T$ )
2:    $\theta \leftarrow \vec{0}$ 
3:   for  $k = 1, \dots, T$  do
4:     Elegir al azar  $t \in \{1, \dots, n\}$ 
5:      $\theta^{(k+1)} \leftarrow \theta^{(k)} + \eta_k (y^{(t)} - \theta \cdot x^{(t)}) x^{(t)}$ 
6:   return  $\theta$ 

```

---

Donde  $\eta_k$  es la tasa de aprendizaje. Recordemos que, por ejemplo, en el caso del algoritmo del perceptrón, las actualizaciones solamente se realizaban en caso de error. Ahora por el contrario, la actualización es proporcional a la discrepancia  $(y^{(t)} - \theta \cdot x^{(t)})$ , así que por más pequeño que sea, cualquier error contribuirá al entrenamiento.

Otra aproximación será la correspondiente a minimizar directamente  $R_n(\theta)$  al hacer el gradiente igual a cero. Formalmente buscamos  $\hat{\theta}$  para el cual  $\nabla R_n(\theta)_{\theta=\hat{\theta}} = 0$ , esto es:

$$\begin{aligned}
 \nabla R_n(\theta)_{\theta=\hat{\theta}} &= \frac{1}{n} \sum_{t=1}^n \nabla_{\theta} \{ (y^{(t)} - \theta \cdot x^{(t)})^2 / 2 \}_{|\theta=\hat{\theta}} \\
 &= \frac{1}{n} \sum_{t=1}^n \{ - (y^{(t)} - \hat{\theta} \cdot x^{(t)}) x^{(t)} \} \\
 &= -\frac{1}{n} \sum_{t=1}^n y^{(t)} x^{(t)} + \frac{1}{n} \sum_{t=1}^n (\hat{\theta} \cdot x^{(t)}) x^{(t)} \\
 &= \underbrace{-\frac{1}{n} \sum_{t=1}^n y^{(t)} x^{(t)}}_{=b} + \underbrace{\frac{1}{n} \sum_{t=1}^n x^{(t)} (x^{(t)})^T \hat{\theta}}_{=A} \\
 &= A\hat{\theta} - b = 0
 \end{aligned}$$

donde hemos usado el hecho de que  $\hat{\theta} \cdot x^{(t)}$  es un escalar y por tanto, puede ser movido a la derecha del vector  $x^{(t)}$ . Adicionalmente, se reescribió el producto interno como  $\hat{\theta} \cdot x^{(t)} = (x^{(t)})^T \hat{\theta}$ , dando como resultado que la ecuación de los parámetros pueda ser expresada en términos de un vector columna  $b$  de dimensión  $d \times 1$  y una matriz  $A_{d \times d}$ . Cuando  $A$  es no singular, podemos entonces resolver directamente para los parámetros de acuerdo con la expresión  $\hat{\theta} = A^{-1}b$ .

A menudo utilizamos una notación distinta al referirnos a problemas de regresión lineal. Para ello, definimos la matriz  $X = [x^{(1)}, \dots, x^{(n)}]^T$ , es decir,  $X^T$  tiene cada vector de características como columna, mientras que  $X$  los tiene apilados como filas. Si definimos también  $\vec{y} = [y^{(1)}, \dots, y^{(n)}]^T$  (vector columna), podemos verificar fácilmente que:

$$b = \frac{1}{n} X^T \vec{y}, \quad A = \frac{1}{n} X^T X$$

## 4.2. Regularización

En el caso de que la matriz  $A$  sea singular, se modificará el criterio de estimación al añadir un *término de regularización* al error cuadrático medio. El propósito de dicho término será sesgar los parámetros hacia una respuesta por defecto, por ejemplo, cero. El término de regularización buscará contraponerse a establecer los parámetros lejos del vector cero, incluso si los datos nos indican lo contrario (aunque sea débilmente). Esta contraposición es muy útil para asegurar una correcta generalización, donde la intuición indica que proveeremos la respuesta más "simple" cuando la evidencia esté ausente o sea débil. Entre la multitud de posibles términos de regularización, elegiremos  $\|\theta\|^2/2$  como penalización en virtud de mantener el problema de optimización fácilmente resoluble, esto es:

$$J_{n,\lambda}(\theta) = \frac{\lambda}{2} \|\theta\|^2 + \frac{1}{n} \sum_{t=1}^n (y^{(t)} - \theta \cdot x^{(t)})^2 / 2$$

Donde el parámetro de regularización  $\lambda \geq 0$  cuantifica la compensación entre mantener los parámetros pequeños minimizando la norma cuadrada  $\|\theta\|^2/2$  y ajustar a los datos de entrenamiento minimizando el riesgo empírico  $R_n(\theta)$ . El usar esta función objetivo modificada se conoce como *regresión de cresta*.

El utilizar el término de regularización produce pequeños pero importantes cambios en los algoritmos de estimación. En el caso del gradiente estocástico, nos moveremos en cada caso en dirección opuesta del gradiente:

$$\nabla_{\theta} \left\{ \frac{\lambda}{2} \|\theta\|^2 + (y^{(t)} - \theta \cdot x^{(t)})^2 / 2 \right\}_{|\theta=\theta^{(k)}} = \lambda \theta^{(k)} - (y^{(t)} - \theta^{(k)} \cdot x^{(t)}) x^{(t)}$$

y, en consecuencia, el algoritmo podrá escribirse como:

El nuevo factor  $(1 - \lambda \eta_k)$  que multiplica a los parámetros  $\theta^{(k)}$  los contrae hacia cero en cada actualización. En cuanto a la resolución directa de los parámetros,

---

**Algorithm 7** Algoritmo descenso por gradiente estocástico para regresión de cresa

---

```

1: procedure RRSKD( $\{(x^{(t)}, y^{(t)}) , t = 1, \dots, n\}, T$ )
2:    $\theta \leftarrow \vec{0}$ 
3:   for  $k = 1, \dots, T$  do
4:     Elegir al azar  $t \in \{1, \dots, n\}$ 
5:      $\theta^{(k+1)} \leftarrow (1 - \lambda \eta_k) \theta^{(k)} + \eta_k (y^{(t)} - \theta \cdot x^{(t)}) x^{(t)}$ 
6:   return  $\theta$ 

```

---

el término de regularización modifica únicamente la matriz  $A_{d \times d}$  de acuerdo a la expresión  $A = \lambda I + \frac{1}{n} X^T X$  donde la matriz resultante será siempre no singular mientras  $\lambda > 0$ .

El término de regularización se enfocará entonces en quitar énfasis al conjunto de entrenamiento, como resultado, esperaríamos que valores grandes de  $\lambda$  tengan un impacto negativo en el error de entrenamiento. Específicamente, sean  $\hat{\theta} = \hat{\theta}(\lambda)$  los parámetros encontrados al minimizar la función objetivo regularizada  $J_{n,\lambda}(\theta)$ , se observa que  $\hat{\theta}(\lambda)$  es una función de  $\lambda$ , y por tanto,  $R_n(\hat{\theta}(\lambda))$  incrementará directamente proporcional a  $\lambda$ . Esto nos lleva a cuestionarnos el beneficio del término de regularización al castigar el error de entrenamiento; la respuesta está en la generalización, ya que al incrementar el valor de  $\lambda$  se va volviendo más difícil el poder sobreajustar los parámetros al conjunto de entrenamiento, volviéndose resilientes a datos con ruido, sin embargo, si el valor de  $\lambda$  crece demasiado se provoca el efecto contrario al empujar en demasía los parámetros hacia cero.

## 5. Kernels

Consideremos inicialmente cómo es que podemos usar clasificadores lineales para realizar predicciones de naturaleza no lineal. Para esto, nuestra primera aproximación será mapear todos los ejemplos  $x \in \mathbb{R}^d$  hacia otro espacio  $\phi(x) \in \mathbb{R}^p$  donde usualmente  $p$  es muchísimo mayor que  $d$ . Dicho esto, una vez que hemos aprendido las reglas de nuestro clasificador lineal, estaremos en posibilidad de producir clasificadores no lineales para las coordenadas originales. El mapeo puede hacerse de múltiples formas, por ejemplo, un vector bidimensional  $x = [x_1, x_2]^T$  puede

expandirse a un vector de 6 dimensiones como sigue:

$$\phi(x) = [x_1, \sqrt{x_1}, x_2, x_1x_2, x_1^2, x_2^3]^T$$

A partir de ahí, podemos entrenar un clasificador “lineal” en las nuevas  $\phi$ -coordenadas:

$$y = \text{sgn}(\theta \cdot \phi(x) + \theta_0)$$

al mapear cada ejemplo de entrenamiento  $x(t)$  con su correspondiente vector de características  $\phi(x(t))$ , es decir, nuestro conjunto de entrenamiento será  $S_n^\phi = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \dots, n\}$ . Los parámetros estimados  $\hat{\theta}$  y  $\hat{\theta}_0$  definirán una frontera lineal de decisión en las  $\phi$ -coordenadas pero una frontera de decisión no lineal en las  $x$ -coordenadas:

$$\hat{\theta} \cdot \phi(x) + \hat{\theta}_0 = 0 \Leftrightarrow \hat{\theta}_1 x_1 + \hat{\theta}_2 \sqrt{x_1} + \hat{\theta}_3 x_2 + \hat{\theta}_4 x_1 x_2 + \hat{\theta}_5 x_1^2 + \hat{\theta}_6 x_2^3 + \hat{\theta}_0$$

El problema con esta aproximación, es que los vectores de características  $\phi(x)$  pueden explotar en dimensionalidad, por lo que usaremos una estrategia de representación basada en los productos internos de los ejemplos de entrenamiento.

Todos los métodos lineales que hemos revisado hasta el momento pueden transformarse a su versión no lineal al mapear los ejemplos de entrenamiento  $x \in \mathbb{R}^d$  al espacio  $\phi(x) \in \mathbb{R}^p$  con  $p > d$ . Con esto en mente, las versiones no lineales correspondientes serían simplemente:

$$\text{Perceptrón} : y = \text{sgn}(\theta \cdot \phi(x) + \theta_0)$$

$$\text{SVM} : y = \text{sgn}(\theta \cdot \phi(x) + \theta_0)$$

$$\text{Regresión Lineal} : y = \theta \cdot \phi(x) + \theta_0$$

Los cuales difieren en la aproximación del entrenamiento basado en el conjunto expandido  $S_n = \{(\phi(x^{(t)}), y^{(t)}), t = 1, \dots, n\}$ , por lo que los parámetros  $\hat{\theta}$  y  $\hat{\theta}_0$  serán diferentes incluso entrenados en el mismo conjunto por razones intrínsecas a la naturaleza de cada modelo.

## 5.1. Perceptrón de kernel

Podemos ejecutar el algoritmo de perceptrón en el espacio transformado (considerando separabilidad lineal en el nuevo espacio). Recordando los pasos del algoritmo, tenemos:



1. Inicializar  $\theta \leftarrow \vec{0}$  y  $\theta_0 \leftarrow 0$
2. Iterar sobre los ejemplos de entrenamiento  $t = 1, \dots, n$ 
  - si  $y^{(t)} (\theta \cdot \phi(x^{(t)}) + \theta_0) \leq 0$  (error)
  - entonces  $\theta \leftarrow \theta + y^{(t)} \phi(x^{(t)})$  y  $\theta_0 \leftarrow \theta_0 + y^{(t)}$

A partir de lo anterior, es posible representar los parámetros  $\theta$  y  $\theta_0$  en cualquier momento del algoritmo de acuerdo a lo siguiente:

$$\theta = \sum_{i=1}^n \alpha_i y^{(i)} \phi(x^{(i)})$$

$$\theta_0 = \sum_{i=1}^n \alpha_i y^{(i)}$$

Donde  $\alpha_i$  es el número de veces que hemos cometido errores en el correspondiente ejemplo de entrenamiento  $(\phi(x^{(i)}), y^{(i)})$ . Nuestro objetivo entonces, será reescribir el algoritmo de tal suerte que solo actualicemos las  $\alpha_i$ , en lugar de construir explícitamente  $\theta$  que puede ser de alta dimensionalidad, para ello, expresaremos en términos de productos internos como sigue:

$$\theta \cdot \phi(x) = \sum_{i=1}^n \alpha_i y^{(i)} (\phi(x^{(i)}) \cdot \phi(x)) = \sum_{i=1}^n \alpha_i y^{(i)} K(x^{(i)}, x)$$

Donde el producto interno  $K(x^{(i)}, x)$  es conocido como *función kernel*, es una función con dos argumentos y siempre se define como el producto interno de vectores de características correspondientes a argumentos de entrada. Una función kernel se considera válida si existe un vector de características  $\phi(x)$  tal que  $K(x, x') = \phi(x) \cdot \phi(x')$  para todo  $x$  y  $x'$ . En este orden de ideas, buscamos que el algoritmo del perceptron se exprese únicamente usando valores kernel, es decir, comparaciones entre ejemplos, para este fin, escribiremos la función discriminante  $\theta \cdot \phi(x) + \theta_0$  únicamente en términos de la función kernel y  $\alpha$ :

$$\theta \cdot \phi(x) + \theta_0 = \sum_{i=1}^n \alpha_i y^{(i)} K(x^{(i)}, x) + \sum_{i=1}^n \alpha_i y^{(i)} = \sum_{i=1}^n \alpha_i y^{(i)} [K(x^{(i)}, x) + 1]$$

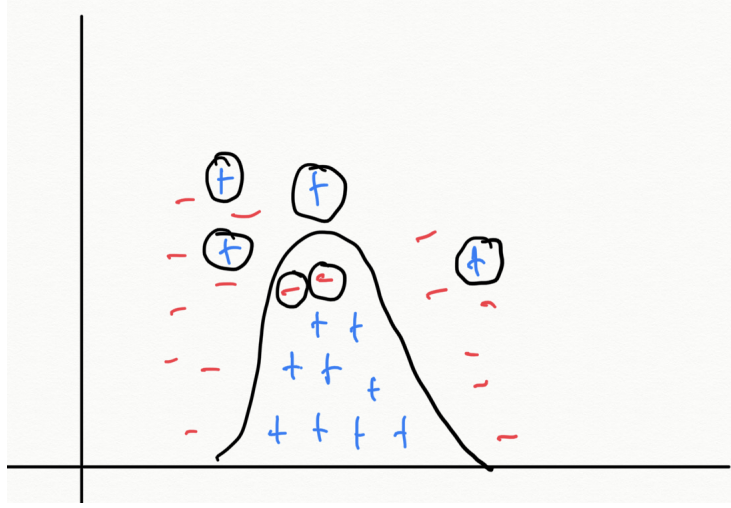
Con lo anterior, tenemos suficiente para evaluar la presencia de un error en el ejemplo de entrenamiento  $(\phi(x)^{(t)}, y^{(t)})$  y escribir el algoritmo en términos de  $\alpha$ ,

actualizando su valor en respuesta a cada error, en consecuencia, el algoritmo *perceptrón de kernel* será:

1. Inicializar  $\alpha_t \leftarrow 0, t = 1, \dots, n$
2. Iterar sobre los ejemplos de entrenamiento  $t = 1, \dots, n$   
 si  $y^{(t)} \left( \sum_{i=1}^n \alpha_i y^{(i)} [K(x^{(i)}, x^{(t)}) + 1] \right) \leq 0$  (error)  
 entonces  $\alpha_t \leftarrow \alpha_t + 1$

Este algoritmo puede entrenarse con cualquier kernel válido, en la figura 5.1 puede observarse un ejemplo gráfico del trabajo del clasificador para la frontera de decisión:

$$\sum_{i=1}^n \alpha_i y^{(i)} [K(x^{(i)}, x) + 1] = 0$$



**Figura 5.1:** Ejemplo de perceptrón de kernel

## 5.2. Regresión lineal de kernel

En virtud de la simplicidad, tomaremos nuevamente el caso donde  $\theta_0 = 0$ , por tanto, la función de regresión que estimaremos tendrá la forma  $\theta \cdot \phi(x)$ , con ello, el criterio de estimación de los parámetros  $\theta$  será:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \theta \cdot \phi(x^{(i)}))^2 / 2 + \frac{\lambda}{2} \|\theta\|^2$$

El mínimo de la función previamente expuesta encontrará en el gradiente nulo, en consecuencia:

$$\begin{aligned}
\frac{\partial}{\partial \theta} J(\theta) &= -\frac{1}{n} \sum_{t=1}^n (y^{(t)} - \theta \cdot \phi(x^{(t)})) \phi(x^{(t)}) + \lambda \theta \\
&= -\frac{1}{n} \sum_{t=1}^n n \lambda \alpha_t \phi(x^{(t)}) + \lambda \theta \\
&= -\lambda \sum_{t=1}^n \alpha_t \phi(x^{(t)}) + \lambda \theta \\
&= 0
\end{aligned}$$

Donde se observa que  $\theta = \sum_{t=1}^n \alpha_t \phi(x^{(t)})$  con  $\alpha_t$  proporcional a los errores en la predicción, lo cual es posible sí y solo si  $\alpha_t$  y  $\theta$  están relacionadas específicamente, es decir:

$$\begin{aligned}
n \lambda \alpha_t &= y^{(t)} - \theta \cdot \phi(x^{(t)}) \\
&= y^{(t)} - \left( \sum_{i=1}^n \alpha_i \phi(x^{(i)}) \right) \cdot \phi(x^{(t)}) \\
&= y^{(t)} - \sum_{i=1}^n \alpha_i K(x^{(i)}, x^{(t)})
\end{aligned}$$

Si escribimos la ecuación en su forma vectorial,  $\vec{\alpha} = [\alpha_1, \dots, \alpha_n]^T$ ,  $\vec{y} = [y^{(1)}, \dots, y^{(n)}]^T$  y dado que  $K(x^{(i)}, x^{(t)}) = K(x^{(t)}, x^{(i)})$ , tenemos:

$$\sum_{i=1}^n \alpha_i K(x^{(i)}, x^{(t)}) = \sum_{i=1}^n K(x^{(t)}, x^{(i)}) \alpha_i = [K \vec{\alpha}]_t$$

Donde  $K$  es una matriz de  $n \times n$  y el elemento  $ij$  es  $K(x^{(i)}, y^{(i)})$ , también conocida como *Matriz de Gram*. De esta forma, tenemos finalmente:

$$n \lambda \vec{\alpha} = \vec{y} - K \vec{\alpha} \Rightarrow (n \lambda I + K) \vec{\alpha} = \vec{y}$$

Lo cual es fácilmente resoluble a través de la expresión  $\vec{\alpha} = (n \lambda I + K)^{-1} \vec{y}$ . De esta forma, los coeficientes  $\hat{a}_t$  pueden ser calculados en términos de la función kernel

y los valores de la variable objetivo y por tanto, jamás tendremos que construir explícitamente vectores de características  $\phi(x)$ . Por último, una vez calculados los coeficientes, las predicciones para un nuevo punto  $x$  serán fácilmente calculables a través de la expresión:

$$\hat{\theta} \cdot \phi(x) = \sum_{i=1}^n \hat{\alpha}_i \phi(x^{(i)}) \cdot \phi(x) = \sum_{i=1}^n \hat{\alpha}_i K(x^{(i)}, x)$$

### 5.3. Funciones kernel

Todos los métodos discutidos son aplicables con cualquier función kernel válida  $K(x, x')$ . Una función kernel es válida si y solo si existe un mapeo de características  $\phi(x)$  tal que  $K(x, x') = \phi(x) \cdot \phi(x')$ , no es mandatorio conocer la forma de  $\phi(x)$ , solamente debemos garantizar que existe. Basado en las siguientes cuatro reglas, podemos construir muchas de las funciones kernel más comunes:

1.  $K(x, x') = 1$  es una función kernel.
2. Sea  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  cualquier función real de  $x$ . Entonces, si  $K(x, x')$  es una función kernel también lo será  $\tilde{K}(x, x') = f(x)K(x, x')f(x')$ .
3. Si  $K_1(x, x')$  y  $K_2(x, x')$  entonces  $K_1(x, x') + K_2(x, x')$  es un kernel.
4. Si  $K_1(x, x')$  y  $K_2(x, x')$  entonces  $K_1(x, x')K_2(x, x')$  es un kernel.

Exploremos más a profundidad las reglas anteriores mediante la relación a los mapeos de características subyacentes. En el caso del Kernel constante  $K(x, x')$ , el mapeo corresponde simplemente a  $\phi(x) = 1$  para toda  $x \in \mathbb{R}^d$ . De manera similar, si  $\phi(x)$  es el mapeo de características para  $K(x, x')$ , entonces  $\tilde{K}(x, x') = f(x)K(x, x')f(x')$  corresponderá a  $\tilde{\phi}(x) = f(x)\phi(x)$ . Para lo correspondiente a la adición de kernels, tendremos que para los kernels  $K_1(x, x')$  y  $K_2(x, x')$  y sus respectivos mapeos  $\phi^{(1)}(x)$  y  $\phi^{(2)}(x)$ :

$$\begin{aligned} K(x, x') &= \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix} \cdot \begin{bmatrix} \phi^{(1)}(x') \\ \phi^{(2)}(x') \end{bmatrix} \\ &= \phi^{(1)}(x) \cdot \phi^{(1)}(x') + \phi^{(2)}(x) \cdot \phi^{(2)}(x') \\ &= K_1(x, x') + K_2(x, x') \end{aligned}$$

En el caso del producto, consideremos adicionalmente a  $f_i(x)$  como el valor de la  $i$ -ésima característica bajo el mapeo  $\phi^{(1)}(x)$  y a  $g_i(x)$  como el valor de la  $i$ -ésima característica bajo el mapeo  $\phi^{(2)}(x)$ , luego entonces:

$$\begin{aligned}
K_1(x, x')K_2(x, x') &= (\phi^{(1)}(x) \cdot \phi^{(1)}(x')) (\phi^{(2)}(x) \cdot \phi^{(2)}(x')) \\
&= \left( \sum_i f_i(x) f_i(x') \right) \left( \sum_j g_j(x) g_j(x') \right) \\
&= \sum_{i,j} f_i(x) f_i(x') g_j(x) g_j(x') \\
&= \sum_{i,j} (f_i(x) g_j(x)) (f_i(x') g_j(x'))
\end{aligned}$$

Definamos el mapeo de características  $\phi^{(3)}(x)$  mediante la característica  $h_{i,j}(x) = f_i(x)g_j(x)$ , por tanto:

$$K_1(x, x')K_2(x, x') = \phi^{(3)}(x) \cdot \phi^{(3)}(x')$$

y en consecuencia,  $K_1(x, x')K_2(x, x')$  es un kernel.

Muchos kernels típicos se construyen basados en estas reglas, por ejemplo  $K(x, x') = x \cdot x'$  es un kernel basado en las reglas 1,2 y 3. Para ver esto, sea  $f_i(x) = x_i$  la  $i$ -ésima coordenada del mapeo, entonces:

$$x \cdot x' = x_1x'_1 + \dots + x_dx'_d = f_1(x)1f_1(x') + \dots + f_d(x)1f_d(x')$$

Un kernel interesante es el siguiente:

$$K(x, x') = \exp(x \cdot x') = 1 + (x \cdot x') + \frac{1}{2!}(x \cdot x')^2 + \dots$$

El cual está construido bajo las mismas reglas, con la particularidad de que está expresado en forma de suma infinita y por tanto, su representación de características será de dimensión infinita. Un kernel muy popular en la práctica es el llamado *kernel de base radial* el cual también tendrá una representación de características infinita, a saber:

$$\begin{aligned}
K(x, x') &= \exp(-\|x - x'\|^2/2) \\
&= \exp(-\|x\|^2/2) \exp(x \cdot x') \exp(-\|x'\|^2/2) \\
&= f(x) \exp(x \cdot x') f(x')
\end{aligned}$$

Donde  $f(x) = \exp(-\|x\|^2/2)$ .

El kernel de base radial es especial debido a que vuelve linealmente separables cualquier par de ejemplos de entrenamiento así como tener diversas ventajas computacionales en su cálculo.

## 6. Análisis Discriminante

El análisis discriminante (en lo sucesivo AD) es una técnica estadística que permite asignar o clasificar nuevos individuos dentro de grupos previamente definidos. AD intenta realizar la misma tarea que la regresión lineal múltiple al predecir una salida, sin embargo, la regresión está limitada a los casos donde la variable dependiente es tal que la combinación de sus predictoras, a través de la ecuación de regresión, producirá los valores numéricos estimados de la media mediante combinaciones ponderadas de los valores de las variables independientes. El objetivo del AD es producir una regla o esquema de clasificación tal que nos permita predecir la población a la que es más probable pertenecer una nueva observación (donde se conocen previamente dichas poblaciones). El AD se presenta desde dos enfoques:

- Por obtención de funciones discriminantes: Se construyen ecuaciones similares a las de regresión lineal múltiple
- Canónico: Emplea técnicas de correlación canónica y componentes principales

El problema del AD para clasificar en dos categorías fue resuelto por [Fisher, 1940]. Sean  $k$  variables explicativas para clasificar individuos separados en 2 y sólo 2 categorías. Se propone una función discriminatoria dada por:

$$D = \sum_{i=1}^k u_i X_i \quad (1)$$

Que es conocida como función discriminante de Fisher.

La intención es obtener los coeficientes de ponderación  $u_i$  por medio de  $n$  observaciones, de esta manera, podemos expresar una función discriminante para cada una de ellas como sigue:

$$D_j = \sum_{i=1}^k u_i X_{ij} \quad j = 1, \dots, n \quad (2)$$

$D_j$  representa la puntuación discriminante correspondiente a la  $j$ -ésima observación. Si estandarizamos las variables con respecto a la desviación estándar, podemos expresar la relación anterior de forma matricial:

$$\begin{bmatrix} D_1 \\ \vdots \\ D_n \end{bmatrix} = \begin{bmatrix} X_{1,1} & \cdots & X_{k,1} \\ \vdots & \ddots & \vdots \\ X_{1,n} & \cdots & X_{k,n} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_k \end{bmatrix}$$

que puede escribirse en notación compacta como:

$$\vec{d} = X\vec{u}$$

Para expresar la variabilidad de la función discriminante recurrimos a la suma de cuadrados de las variables discriminantes estandarizadas:

$$\vec{d}^T \vec{d} = \vec{u}^T X^T X \vec{u}$$

donde  $X^T X$  es una matriz simétrica que contiene la suma de cuadrados total de las variables explicativas, en consecuencia, podrá descomponerse en la suma de las matrices entre grupos  $F$  e intragrupos  $W$ :

$$X^T X = T = F + W$$

y por tanto:

$$\vec{d}^T \vec{d} = \vec{u}^T X^T X \vec{u} = \vec{u}^T T \vec{u} = \vec{u}^T F \vec{u} + \vec{u}^T W \vec{u}$$

De la expresión anterior podemos calcular las matrices  $T$ ,  $F$  y  $W$  a partir de los datos muestrales, los coeficientes  $u$  son las incógnitas a determinar. Para poder determinar dichas incógnitas, Fisher planteó maximizar la razón de la variabilidad entre grupos respecto de la variabilidad intragrupos. Dicha razón provocará que las distribuciones estén lo más separadas entre sí y al mismo tiempo dentro de cada una exista la menor dispersión posible. Matemáticamente, expresamos esto de la siguiente manera:

$$\lambda = \frac{\vec{u}^T F \vec{u}}{\vec{u}^T W \vec{u}}$$

Donde el  $\lambda$  máximo estará dado por:

$$\begin{aligned} \frac{\partial \lambda}{\partial \vec{u}} &= \frac{\vec{u}^T W \vec{u} \frac{\partial}{\partial \vec{u}} (\vec{u}^T F \vec{u}) - \vec{u}^T F \vec{u} \frac{\partial}{\partial \vec{u}} (\vec{u}^T W \vec{u})}{(\vec{u}^T W \vec{u})^2} \\ &= \frac{\vec{u}^T W \vec{u} [2F\vec{u}] - \vec{u}^T F \vec{u} [2W\vec{u}]}{(\vec{u}^T W \vec{u})^2} = 0 \end{aligned}$$

En consecuencia:



$$2F\vec{u}(\vec{u}^T W \vec{u}) = 2W\vec{u}(\vec{u}^T F \vec{u})$$

$$\frac{2F\vec{u}}{2W\vec{u}} = \frac{\vec{u}^T F \vec{u}}{\vec{u}^T W \vec{u}} = \lambda$$

Así:

$$F\vec{u} = \lambda W\vec{u}$$

$$W^{-1}F\vec{u} = \lambda\vec{u}$$

$$(W^{-1}F - \lambda I)\vec{u} = 0$$

Esto nos indica que para obtener los ejes discriminantes habremos de calcular los valores propios de la matriz  $W^{-1}F$  y al ser  $\lambda$  el ratio a maximizar, escogeremos siempre los valores propios en orden descendente así como su vector propio asociado  $\vec{u}$ .

Definiremos ahora las puntuaciones discriminantes como:

$$D = \sum_{i=1}^k u_i X_i$$

Lo anterior equivale a proyectar cada vector del espacio de variables de  $k$  dimensiones sobre el eje discriminante.

Por otro lado, debemos definir los centroides de cada categoría dentro de la variable nominal que corresponden a los estadígrafos que resumirán la información de cada una de ellas, a saber:

$$\bar{x}_I = \begin{bmatrix} \bar{X}_{1,I} \\ \bar{X}_{2,I} \\ \vdots \\ \bar{X}_{k,I} \end{bmatrix} \quad \bar{x}_{II} = \begin{bmatrix} \bar{X}_{1,II} \\ \bar{X}_{2,II} \\ \vdots \\ \bar{X}_{k,II} \end{bmatrix}$$

Con estos vectores es posible calificar el punto de corte discriminante dado por:

$$C = \frac{(D_I + D_{II})}{2}$$

Donde:

$$D_I = \sum_{i=1}^k u_i \bar{X}_{i,I} \quad y \quad D_{II} = \sum_{i=1}^k u_i \bar{X}_{i,II}$$

Así, si  $D_i < C$  clasificamos al individuo  $i$  en el grupo  $I$  y si  $D_i > C$ , clasificaremos al individuo en el grupo  $II$

El segundo enfoque se basa en la obtención de las componentes principales dentro del conjunto de datos. Sabemos que las componentes principales son una combinación lineal de las variables originales que explican la mayor variabilidad posible y además son ortogonales entre sí. Si tomamos en cuenta que la primera componente  $C$  está asociada al mayor valor propio de la matriz de datos, entonces las componentes sucesivas tendrán asociados los valores propios en orden descendente y, en consecuencia, podríamos utilizar dichas componentes como ejes discriminantes, donde la función discriminante equivaldría a la ecuación del componente principal de acuerdo a la cardinalidad de la variable dependiente.

## 7. Regresión Logística

La regresión logística es uno de los modelos más utilizados en la industria, esto debido a la facilidad de implementación aunado a propiedades favorables en la curva de probabilidad que produce. Para iniciar nuestro estudio, repasaremos tres métricas de desempeño que, aunque son transversales a todos los clasificadores binarios, son de particular relevancia en cuanto a la regresión logística se refiere.

### 7.1. Métricas de desempeño

Iniciemos con *accuracy score* y *ROC*, para ello recurrimos a lo siguiente:

Sea la matriz:

		Valor predicho		total
		p	n	
Valor real	p'	Verdadero Positivo	Falso Negativo	P'
	n'	Falso Positivo	Verdadero Negativo	N'
total		P	N	

El accuracy score estará definido mediante la siguiente expresión:

$$accuracy = \frac{VerdaderoPositivo + VerdaderoNegativo}{P + N}$$

que no es mas que la proporción de clases correctamente clasificada. Para este indicador, el corte de probabilidad para determinar la clase se estila ajustar en el punto de máxima entropía de la dicotomía (cuando  $P = 1/2$ ). Para el caso de ROC, el procedimiento para su obtención se basa en la misma matriz de confusión con el agregado de que el corte de probabilidad para determinación de clase se mide en  $k$  puntos distintos, proporcionando las coordenadas bidimensionales para la formación de una curva (Curva ROC) donde posteriormente, el indicador se obtendrá a partir del cálculo del área bajo dicha curva, donde el valor de referencia 0.5 indica una clasificación idéntica al efecto del azar, 1.0 indica clasificación perfecta, por tanto, valores más cercanos a 1.0 indican un modelo con mejor poder predictivo.

Por último, el estadígrafo de la prueba de Kolmogorov-Smirnov (KS) está basado en la máxima diferencia entre las distribuciones acumuladas de una distribución empírica y una teórica. Donde la clasificación positiva y negativa toman dichos roles de manera indistinta pero no simultánea. Al ser una prueba de bondad de ajuste, existirá diferencia significativa para un valor de confiabilidad  $\alpha$  dado en

caso de que rechace la hipótesis de igualdad de las distribuciones, luego entonces, a mayor el valor de KS, más potente será un modelo, siendo cero su valor mínimo ante distribuciones idénticas.

## 7.2. Especificación del modelo

La regresión logística nos permite modelar un evento dicotómico en forma probabilística donde  $p$  representa la probabilidad de éxito. Dependiendo del contexto, “éxito” puede interpretarse de manera relativa. Comúnmente, “éxito” se referirá a aquellos casos que representan una característica de interés, por ejemplo: tomar un producto, incumplir un préstamo, enfermarse, etc. El complemento de  $p$  se denota por  $q = 1 - p$ . El enfoque de modelado de probabilidad se basará en el cociente denominado momio(odd's) dado por:

$$\frac{p}{q} = \frac{p}{1 - p} \quad (3)$$

que representa cuánto más probable es el éxito que el fracaso.

Estamos interesados en modelar linealmente la probabilidad de cierto evento en función de un conjunto de variables. Supóngase que cada variable independiente  $x_i$  contribuye proporcionalmente a la variable objetivo, al linealizar los momios podemos proponer:

$$\ln \left( \frac{p}{1 - p} \right) = \beta_0 + \sum_{i=1}^n \beta_i x_i \quad (4)$$

Generando una expresión para  $p$ , tenemos:

$$p = \left( 1 + e^{-\beta_0 - \sum_{i=1}^n \beta_i x_i} \right)^{-1} \quad (5)$$

Que es conocida como función logística (sigmoidal), la cual devuelve valores entre 0 y 1 quienes serán interpretados como la probabilidad de ocurrencia del evento para cada vector presentado. Los parámetros  $\beta_i$  pueden estimarse a través del método de máxima verosimilitud.

### 7.3. Estimación de parámetros

La regresión logística forma parte de un subconjunto de modelos llamado Modelos Lineales Generalizados (MLG). Un modelo lineal generalizado describe la relación existente entre la media de la variable dependiente  $y$  y una variable independiente  $x$ . La regresión logística es un MLG tal que la variable dependiente  $y \sim Be(p)$ . El principal supuesto es que la probabilidad  $p$  está relacionada al modelo lineal de la siguiente forma:

$$\ln \left( \frac{p}{1-p} \right) = \alpha + \beta x$$

El logaritmo linealiza el momio de éxito en el lado izquierdo de la expresión dejando en igualdad de condiciones ambos miembros. Si expresamos  $p$  en términos de  $x_i$  obtendremos:

$$p(x) = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}}$$

El hecho de que  $0 < p < 1$  apunta en la dirección correcta, sin embargo, los valores extremos  $p(x) = 0$  ó  $p(x) = 1$  que se alcanzan conforme  $x \rightarrow \pm\infty$ .

Examinemos ahora la derivada de  $p(x)$ :

$$\frac{d}{dx} p(x) = \beta p(x) (1 - p(x))$$

Dado que el signo de  $p(x) (1 - p(x))$  es siempre correspondiente al signo de  $\beta$ , en el caso particular  $\beta = 0$  no existe relación entre  $p$  y  $x$ . Para entender el significado de los parámetros  $\alpha$  y  $\beta$ , consideremos el caso donde  $x = 0$ :

$$p(0) = \frac{1}{1 + e^{-\alpha}} \rightarrow \alpha = \frac{p}{1-p}$$

Observamos que  $\alpha$  no es mas que el logaritmo del momio de éxito, en el caso de  $\beta$ , incrementemos una unidad a  $x$ :

$$\ln \left( \frac{p(x+1)}{1-p(x+1)} \right) - \ln \left( \frac{p(x)}{1-p(x)} \right) = \alpha + \beta(x+1) - \alpha - \beta(x) = \beta$$

, así el cambio en  $\beta$  es el cambio en el logaritmo del momio de éxito correspondiente al incremento de una unidad en  $x$ , en consecuencia,  $e^\beta$  es el cambio multiplicativo en los momios de éxito correspondiente al incremento de una unidad en  $x$ .

Una vez revisados los fundamentos, en la práctica se requiere la estimación de los parámetros del modelo a través de datos. Dado que  $y_i \sim Be(p_i)$  no existe una conexión directa entre  $y_i$  y  $\alpha + \beta x_i$ , por tanto, el método de estimación de parámetros recaerá en el método de máxima verosimilitud. De manera general, podemos expresar  $p(x) = F(\alpha + \beta x)$ , haciendo  $F_i = F(\alpha + \beta x_i)$ , entonces la función de verosimilitud será:

$$L(\alpha, \beta | \vec{y}) = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i} = \prod_{i=1}^n F_i^{y_i} (1 - F_i)^{1-y_i}$$

Donde la log-verosimilitud es:

$$\ln L(\alpha, \beta | \vec{y}) = \sum_{i=1}^n \left[ \ln(1 - F_i) + y_i \ln \left( \frac{F_i}{1 - F_i} \right) \right]$$

Para maximizar esta función, precisamos derivar para obtener las ecuaciones a resolver para estimar los parámetros, así:

$$\frac{\partial \ln(1 - F_i)}{\partial \alpha} = -\frac{f_i}{1 - F_i} = -\frac{F_i f_i}{F_i (1 - F_i)}$$

y

$$\frac{\partial}{\partial \alpha} \ln \left( \frac{F_i}{1 - F_i} \right) = \frac{f_i}{F_i (1 - F_i)}$$

En consecuencia:

$$\frac{\partial}{\partial \alpha} \ln L(\alpha, \beta | \vec{y}) = \sum_{i=1}^n (y_i - F_i) \frac{f_i}{F_i (1 - F_i)}$$

Mediante un procedimiento similar, se llega a:

$$\frac{\partial}{\partial \beta} \ln L(\alpha, \beta | \vec{y}) = \sum_{i=1}^n (y_i - F_i) \frac{f_i}{F_i(1 - F_i)} x_i$$

En el caso particular del enlace logístico donde  $F(x) = \frac{e^x}{1+e^x}$  las ecuaciones anteriores se simplifican y al resolverlas para  $\alpha$  y  $\beta$  se obtienen los parámetros estimados que garantizan la máxima verosimilitud debido a las propiedades de concavidad de la función de verosimilitud. Las ecuaciones son de naturaleza no lineal y por tanto deberán ser resueltas por métodos numéricos.

## 7.4. Método de Newton

A través de este método podemos estimar los parámetros de máxima verosimilitud. Si partimos de su forma más simple, la cual consiste en minimizar una función escalar de una sola variable, por ejemplo,  $f(\theta)$ , nos daremos entonces a la tarea de encontrar el mínimo global  $\theta^*$ . Si se cumplen los supuestos de convexidad de la función, podemos expandir en serie de Taylor como sigue:

$$f(\theta) \approx f(\theta^*) + \frac{1}{2}(\theta - \theta^*)^2 \frac{d^2 f}{d\theta^2} \Big|_{\theta=\theta^*}$$

El método minimiza la aproximación cuadrática en la que estamos interesados. Para ello, si incorporamos la expansión completa de segundo orden en serie de Taylor, alrededor de un punto inicial  $\theta_0$ , tenemos:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \frac{df}{d\theta} \Big|_{\theta=\theta_0} + \frac{1}{2}(\theta - \theta_0)^2 \frac{d^2 f}{d\theta^2} \Big|_{\theta=\theta_0}$$

Ahora simplemente derivamos con respecto a  $\theta$ , igualamos a cero en un punto subsecuente  $\theta_1$  para obtener:

$$f'(\theta_0) + \frac{1}{2}f''(\theta_0)2(\theta_1 - \theta_0) = 0 \Rightarrow \theta_1 = \theta_0 - \frac{f'(\theta_0)}{f''(\theta_0)}$$

La nueva aproximación es mejor que la anterior, en consecuencia, podemos poner una regla de correspondencia iterativa que eventualmente convergerá al óptimo deseado  $\theta^*$ :

$$\theta_{k+1} = \theta_k - \frac{f'(\theta_k)}{f''(\theta_k)}$$

Lo anterior es fácilmente extendible a más dimensiones, considérese ahora la función  $f(\vec{\theta})$ , entonces la aproximación de Newton se actualizará en cada iteración de acuerdo a la expresión:

$$\vec{\theta}_{k+1} = \vec{\theta}_k - H^{-1}(\vec{\theta}_k) \nabla f(\vec{\theta}_k)$$

Donde H es la matriz Hessiana de la función  $f$ .

## 7.5. Transformación WoE

Un tópico importante con respecto a la regresión logística está íntimamente relacionado con la teoría de la información, en particular, con una forma de codificación de las variables basada en la divergencia de Kullback-Leibler[Kullback, 1997]. La divergencia de Kullback-Leibler es también conocida como *entropía relativa* y se define de acuerdo a lo siguiente:

Sean  $p$  y  $q$  dos distribuciones de probabilidad, la medida de disimilitud entre ambas (divergencia de Kullback-Leibler) está dada por:

$$KL(p||q) := \sum_k p_k \log \frac{p_k}{q_k}$$

Para el caso discreto, para el caso continuo, simplemente se sustituye la suma por una integral.

La divergencia de KL representa el número promedio de bits *extra* que se requieren para codificar la información. Nótese que si  $p = q$  la divergencia de KL será nula.

Dado lo anterior, utilizaremos una codificación especial cuya característica fundamental radica en que convierte el riesgo asociado a una elección particular en una escala lineal fácil de interpretar por el ser humano. Adicional a esto, permite incorporar variables discretas a nuestro análisis en virtud de que la codificación se basa en el movimiento de probabilidad dado cierto atributo y no al valor numérico perse. En el caso de las variables continuas, procederemos inicialmente a discretizarlas en intervalos para que la cantidad de información relativa al riesgo se



acumule. La codificación descrita es conocida como *Weight of Evidence* (WoE), su cálculo es muy simple, veamos:

Sean  $P$  el número de ocurrencias de cierto evento en particular y  $N$  el número de no ocurrencias. La codificación WoE para el  $i$ -ésimo atributo asociado a una característica particular está dado por:

$$WoE_i = \log \left( \frac{\frac{N_i}{\sum_{i=1}^n N_i}}{\frac{P_i}{\sum_{i=1}^n P_i}} \right)$$

La codificación WoE es adecuada para la regresión logística en virtud de que posee correspondencia lineal con la función sigmoide.

Otra característica interesante del WoE es que nos permitirá medir el poder predictivo de cada variable explicativa a través de la métrica conocida como *Information Value* (IV), el cual se calcula mediante la expresión:

$$IV = \sum_{i=1}^n \left[ \left( \frac{N_i}{\sum_{i=1}^n N_i} - \frac{P_i}{\sum_{i=1}^n P_i} \right) WoE_i \right]$$

Que no es más que la divergencia de Kullback-Leibler entre las distribuciones de evento y no evento.

Una vez codificadas todas las variables predictoras a WoE, las nuevas variables fungirán como predictores continuos para la regresión logística.

## 8. Redes Neurales Artificiales

Las redes neurales consisten en un gran número de unidades de cómputo simples llamadas neuronas (clasificadores lineales), que en conjunto procesan el vector de entrada  $x$  hacia una salida específica tanto para tareas de clasificación como para tareas de regresión.

## 8.1. Redes neurales alimentadas hacia adelante

Cuando organizamos una red neuronal en capas, donde cada capa procesa su entrada y la transmite a su consecuente se le conoce como *red neural alimentada hacia adelante*. La inspiración para dicha arquitectura esta basada en nuestro sistema de visión, donde las señales luminosas se procesan en paralelo para producir un resultado cognitivo en nosotros. En este tipo de redes, se cuenta con los siguientes componentes:

1. *Capa de entrada*: Cada neurona corresponde a una coordenada del vector de entrada, no se realiza cómputo alguno, los valores unicamente se propagan hacia adelante.
2. *Capas ocultas*: Unidades de neuronas que representan transformaciones complejas de la señal de entrada, pueden ser una o varias capas ocultas. Su activación se determina mediante la agregación de la capa precedente.
3. *Capa de salida*: Comúnmente es una sola unidad que toma como entrada a las precedentes para tomar la decisión final en cuanto a clasificación o regresión.

En la figura 8.1 se aprecia una arquitectura básica de red alimentada hacia adelante.

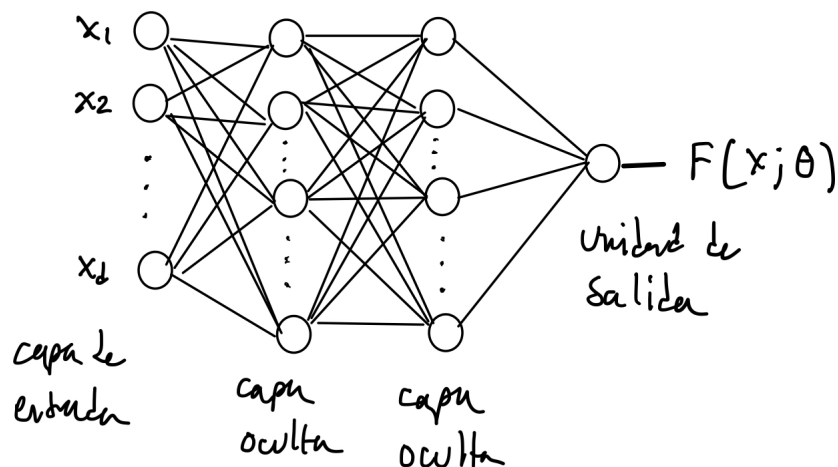
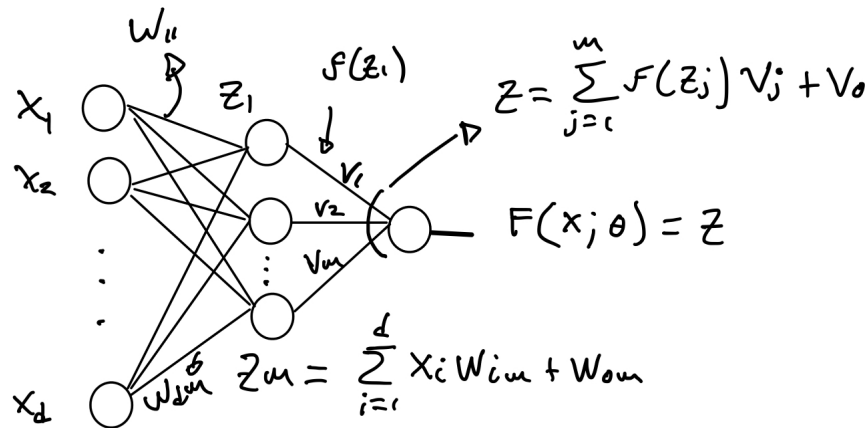


Figura 8.1: Red alimentada hacia adelante con dos capas ocultas

Revisemos matemáticamente el modelo de red neural con una capa oculta como

se muestra en la figura 8.2.



**Figura 8.2:** Red alimentada hacia adelante con una capa oculta

En principio las neuronas de la capa de entrada simplemente limitan a adoptar los valores de las coordenadas del vector de entrada, por el contrario, cada una de las  $m$  neuronas ocultas evalúan su salida en dos pasos:

$$z_j = \sum_{i=1}^d x_i w_{ij} + w_{0j} (\text{entrada ponderada de la } j - \text{ésima neurona oculta}).$$

$$f(z_j) = \max\{0, z_j\} (\text{función de activación lineal rectificadora})$$

En este ejemplo particular se utilizó la función de activación lineal rectificadora, sin embargo, en este paso puede usarse cualquier función de activación deseada, algunas de las más destacadas son:

- signo:  $\text{sign}(z)$
- sigmoide:  $(1 + e^{-z})^{-1}$
- tangente hiperbólica:  $\tanh(z)$
- softsign:  $\frac{z}{1+|z|}$

Finalmente, la neurona de salida no procesará los valores originales de entrada sino las activaciones de las neuronas ocultas y en consecuencia la salida  $z$  de la red será:

$$z = \sum_{j=1}^m f(z_j)V_j + V_0 = F(x; \theta)$$

Donde  $\theta$  constituye el conjunto de todos los parámetros de la red (los pesos).

## 8.2. Propagación hacia atrás

Dado el conjunto de entrenamiento  $\{(x^{(i)}, y^{(i)}), i = 1, \dots, n\}$  de ejemplos  $x \in \mathbb{R}^d$  y etiquetas  $y \in \{-1, 1\}$ , estamos interesados en estimar los parámetros  $\theta$  de cierta arquitectura de red neural tal que la pérdida media sobre los ejemplos de entrenamiento se minimice:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \text{Loss}(y^{(i)} F(x^{(i)}; \theta))$$

Donde se asume que la pérdida es la pérdida bisagra  $\text{Loss}(z) = \max\{0, 1 - z\}$ . Debido a la posible complejidad del modelo, es prudente añadir un término de regularización, digamos,  $1/2\|\theta\|^2$  para forzar la nulidad de los parámetros si no destacan en la clasificación. Para minimizar la pérdida promedio, recurriremos un procedimiento simple de optimización estocástica, dicho algoritmo es conocido como *Descenso por Gradiente Estocástico*:

1. Inicializar  $\theta$  a valores aleatorios pequeños.
2. Seleccionar  $i \in \{1, \dots, n\}$  al azar.
3.  $\theta \leftarrow \theta - \eta_k \nabla_{\theta} \text{Loss}(y^{(i)} F(x^{(i)}; \theta))$

Donde se itera hasta convergencia en los pasos 2 y 3. El gradiente  $\nabla_{\theta}$  tiene la misma dimensión que el vector de parámetros y apunta en la dirección dentro del espacio paramétrico donde la función de pérdida crece más rápido, por ello, empujamos los parámetros suavemente en dirección opuesta. La tasa de aprendizaje  $\eta_k$  deberá decrecer lentamente proporcionalmente al número de actualizaciones, usualmente se estima que la tasa de aprendizaje esté dada mediante la expresión

$$\eta_k = \eta_0 / (k + 1)$$

A pesar de lo simple de los cálculos, a medida que agregamos capas ocultas o arquitecturas más complejas, la optimización se vuelve más difícil (por la explosión

de parámetros) y por ende, el poder de cómputo requerido se incrementará considerablemente, es por ello que en la práctica se recurre a hardware específico para cálculos paralelos como unidades de procesamiento gráfico (GPU). Cuando el tamaño de las redes es considerablemente grande, al tiempo que se usan transformaciones o arquitecturas complejas, se dice que estamos ante un problema de aprendizaje profundo (deep learning), el cual es trascendental en los desarrollos modernos de inteligencia artificial avanzada.

## Referencias

- [Bolton et al., 2010] Bolton, C. et al. (2010). *Logistic regression and its application in credit scoring*. PhD thesis, Citeseer.
- [Casella and Berger, 2002] Casella, G. and Berger, R. L. (2002). *Statistical inference*, volume 2. Duxbury Pacific Grove, CA.
- [Fisher, 1940] Fisher, R. A. (1940). The precision of discriminant functions. *Annals of Human Genetics*, 10(1):422–429.
- [Kullback, 1997] Kullback, S. (1997). *Information theory and statistics*. Courier Corporation.
- [Massey Jr, 1951] Massey Jr, F. J. (1951). The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78.
- [Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- [Nisbet et al., 2009] Nisbet, R., Elder, J., and Miner, G. (2009). *Handbook of statistical analysis and data mining applications*. Academic Press.
- [Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.
- [Shalev-Shwartz and Ben-David, 2014] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.
- [Siddiqi, 2012] Siddiqi, N. (2012). *Credit risk scorecards: developing and implementing intelligent credit scoring*, volume 3. John Wiley & Sons.
- [Sokolova et al., 2006] Sokolova, M., Japkowicz, N., and Szpakowicz, S. (2006). Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence*, pages 1015–1021. Springer.