Aprendizaje supervisado

Mtro. José Gustavo Fuentes Cabrera

Facultad de Estudios Superiores

Acatlán

Apuntes de Análisis Multivariado

Licenciatura en Actuaría

Índice

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 estrategía 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 caracterísiticas 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 $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ cada uno de los ejemplos en el conjunto de entrenamiento y de manera similar $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ sus correspondientes etiquetas de clasificación. Por tanto, el conjunto de pares ordenados $S_n = \left\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\right\}$. 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: $h : \mathbb{R}^d \to \{+, -\}$. Cuando aplicamos el clasificador a un ejemplo en particular x escribimos $h(x)$ como la etiqueta predicha. Así, cualquier clasificador divide el híperespacio $\mathbb{R}^d$ 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:

$$\mathcal{E}_n(h) = \frac{1}{n} \sum_{i=1}^{n} [\![h\left(x^{(i)}\right) \neq y^{(i)}]\!]$$

donde $[\![verdadero]\!] = 1$ (cuando h comete error) y $[\![falso]\!] = 0$ (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:

$$\mathcal{E}(h) = \frac{1}{n'} \sum_{i=n+1}^{n+n'} [\![h\left(x^{(i)}\right) \neq y^{(i)}]\!]$$

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 algun $\hat{h} \in H$ 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 $\mathcal{E}_n\left(\hat{h}\right)$ y $\mathcal{E}\left(\hat{h}\right)$. 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 $S_n = \{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$, podemos seleccionar:

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

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 ϵ 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 ϵ 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 linea recta, en tres un plano y en alta dimensionalidad un híper-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:

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

Donde $\theta \cdot x = \theta^T x$ y $\theta = [\theta_1, \ldots, \theta_d]^T$ es un vector columna de parámetros de valores reales donde diferentes configuraciones de θ 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 θ el clasificador lineal cambiará su predición 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 θ? 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 θ 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.

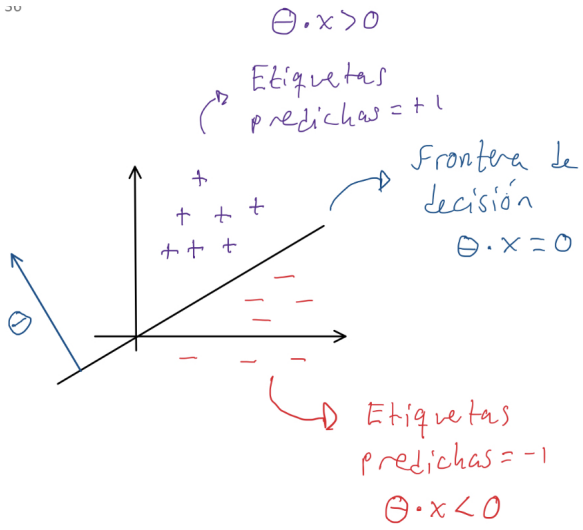

Figura 3.1: Clasificador linear sin sesgo.

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 θ_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:

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

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

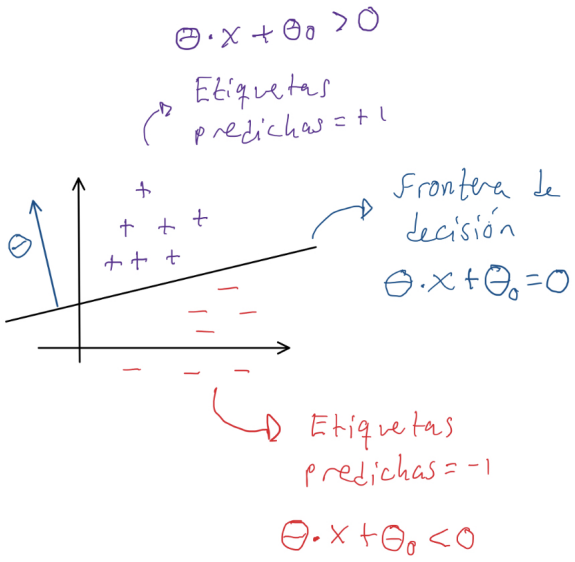

Figura 3.2: Clasificador linear con sesgo.

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 $S_n = \{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$.. Existen fundamentalnente 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 θ tal que el clasificador cometa los menos errores posibles en el conjunto de entrenamiento, esto es:

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

donde $[\![\cdot]\!]$ 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 θ y θ_0 predice incorrectamente la etiqueta. De esta forma, el error de entrenamiento $\mathcal{E}_n\left(\theta, \theta_0\right)$ podrá ser evaluado para cualesquiera elecciones de los parámetros θ y θ_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. $S_n = \{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$ *es* ***linealmente separable con respecto al origen*** *si existe un vector de parámetros* $\hat{\theta}$ *tal que* $y^{(i)}\left(\hat{\theta} \cdot x^{(i)}\right) > 0$ *para todo* $i = 1, \ldots, n$.

Definición 3.2. $S_n = \{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$ *es* ***linealmente separable*** *si existe un vector de parámetros* $\hat{\theta}$ *y un parámetro de sesgo* $\hat{\theta_0}$ *tal que* $y^{(i)}\left(\hat{\theta} \cdot x^{(i)} + \hat{\theta_0}\right) > 0$ *para todo* $i = 1, \ldots, n$.

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 ejemplo, $\theta = \vec{0}$, 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 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

procedure PERCEPTRON($\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$,$T$)
    $\theta \leftarrow \vec{0}$
    for $t = 1, \ldots, T$ **do**
        for $i = 1, \ldots, n$ **do**
            if $y^{(i)}\left(\theta \cdot x^{(i)}\right) \leq 0$ **then**
                $\theta \leftarrow \theta + y^{(i)}x^{(i)}$
    return θ

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, $y^{(i)}\left(\theta^{(k)} \cdot x^{(i)}\right) \leq 0$, 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:

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

En otras palabras, el valor de $y^{(i)}\left(\theta^{(k)} \cdot x^{(i)}\right)$ 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

procedure PERCEPTRON($\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$,$T$)
    $\theta \leftarrow \vec{0}, \theta_0 \leftarrow 0$
    for $t = 1, \ldots, T$ **do**
        for $i = 1, \ldots, n$ **do**
            if $y^{(i)}\left(\theta \cdot x^{(i)} + \theta_0\right) \leq 0$ **then**
                $\theta \leftarrow \theta + y^{(i)}x^{(i)}$
                $\theta_0 \leftarrow \theta_0 + y^{(i)}$
    return θ, θ_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 $x' = [x_1, \ldots, x_d, 1]^T$, y los parámetros $\theta \in \mathbb{R}^d$ a $\theta' \in \mathbb{R}^{d+1}$ tal que $\theta' = [\theta_1, \ldots, \theta_d, \theta_0]^T$.

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* $S_n = \{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$ *es linealmente separable con margen* γ *si existe un vector de parámetros* $\hat{\theta}$ *y un parámetro de sesgo* $\hat{\theta_0}$ *tal que* $y^{(i)}\left(\hat{\theta} \cdot x^{(i)} + \hat{\theta_0}\right) / \|\hat{\theta}\| \geq \gamma$ *para toda* $i = 1, \ldots, n$.

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

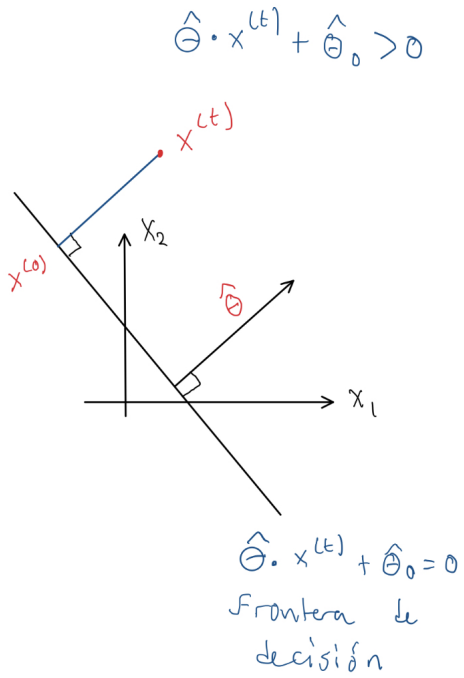

Figura 3.3: Proyección ortogonal sobre la frontera

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

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

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:

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

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

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

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

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

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

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 $\left(x^{(i)}, y^{(i)}\right)$, podemos escribir la siguiente expresión:

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

En la expresión anterior, hemos utilizado la parte **i)** de nuestra hipótesis dado que θ^* clasifica correctamente todos los ejemplos con margen γ. Así, en cada actualización, este término se incrementará en γ unidades y en consecuencia, podrá probarse mediante inducción. Utilizamos entonces $k = 1$ como caso base, $\frac{\theta^{(k-1)} \cdot \theta^*}{\|\theta^*\|} = 0$, 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 γ.

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 $\left(x^{(i)}, y^{(i)}\right)$, así como la parte **ii)** de nuestra hipótesis para acotar y dado que $\|\theta^{(k)}\| \leq \sqrt{k}R$, entonces:

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

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. □

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 lo 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

procedure PERCEPTRON MEDIO($\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$,$T$)
  $\theta \leftarrow \vec{0}$, $\theta_0 = 0$
  $\bar{\theta} \leftarrow \vec{0}$, $\bar{\theta}_0 = 0$
  $c \leftarrow 1$
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $y^{(i)}\left(\theta \cdot x^{(i)} + \theta_0\right) \leq 0$ **then**
        $\theta \leftarrow \theta + y^{(i)}x^{(i)}$
        $\theta_0 \leftarrow \theta_0 + y^{(i)}$
        $\bar{\theta} \leftarrow \bar{\theta} + cy^{(i)}x^{(i)}$
        $\bar{\theta_0} \leftarrow \bar{\theta_0} + cy^{(i)}$
      $c \leftarrow c - 1/\left(nT\right)$
  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:

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

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:

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

Para interpretar geométricamente como las distintas elecciones de los parámetros θ y θ_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:

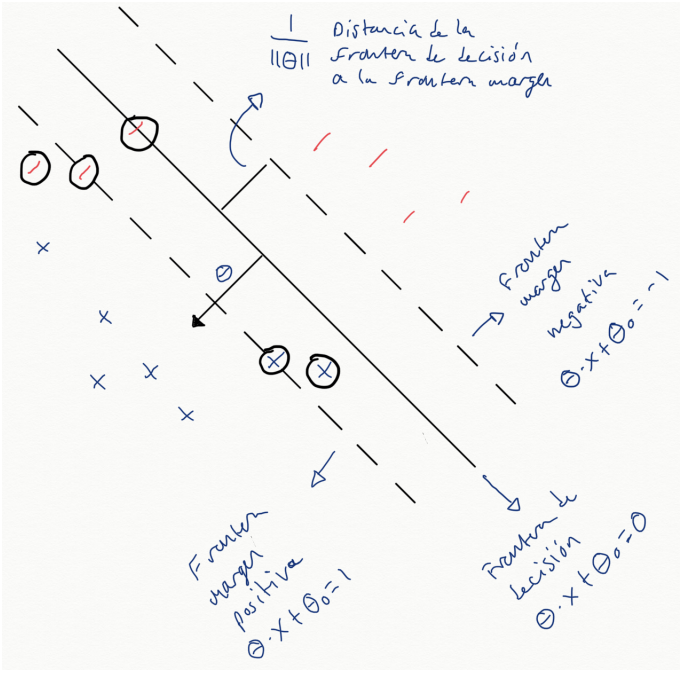

Figura 3.4: Fronteras margen y de decisión para un clasificador lineal. Los puntos resaltados incurren en una pérdida bisagra distinta de cero debido a que violan las fronteras margen.

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 fron-

tera margen. Resultante de esto, $y\left(\theta \cdot x + \theta_0\right) = 1$ y la distancia ortogonal desde la frontera es $y\left(\theta \cdot x + \theta_0\right) / \|\theta\| = 1/\|\theta\|$. Si mantenemos la frontera de decisión sin cambios pero incrementamos $\|\theta\|$, entonces las fronteras margen se acercarían, 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 θ y θ_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 sobreescriba lo que se ha aprendido en los ejemplos previos. El algoritmo tiene garantías similares a las del perceptrón.
Iniciemos consideremos el algoritmo sin el parámetro de sesgo θ_0. Suponiendo que $\theta^{(k)}$ son los parámetros actuales lo cuales fueron inicializadoss 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:

$$\frac{\lambda}{2}\|\theta - \theta^{(k)}\|^2 + Loss_h\left(y\theta \cdot x\right)$$

El parámetro λ determina que tan pasiva es la actualización. Entre mayor el valor de λ más cercanos se encontrarán los parámetros a $\theta^{(k)}$ mientras que valores pequeños de λ permitiran mayores desviaciones para minimizar directamente $Loss_h\left(y\theta \cdot x\right)$. 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 η el cual depende de λ 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:

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

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

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

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 λ producirán valores pequeños de η 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

procedure PASIVO-AGRESIVO($\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$,$\lambda$,$T$)
  $\theta \leftarrow \vec{0}$
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      $\eta \leftarrow \text{mín}\left\{\frac{Loss_h\left(y\theta^{(k)} \cdot x\right)}{\|x\|^2}, \frac{1}{\lambda}\right\}$
      $\theta \leftarrow \theta + \eta y^{(i)} x^{(i)}$
  return θ

Podemos considerar al parámetro λ 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 λ 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 $\frac{1}{nT}\left(\theta^{(1} + \ldots + \theta^{(nT)}\right)$ 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 θ y θ_0 tal que se minimice la expresión:

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

Donde el parámetro λ 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 λ.

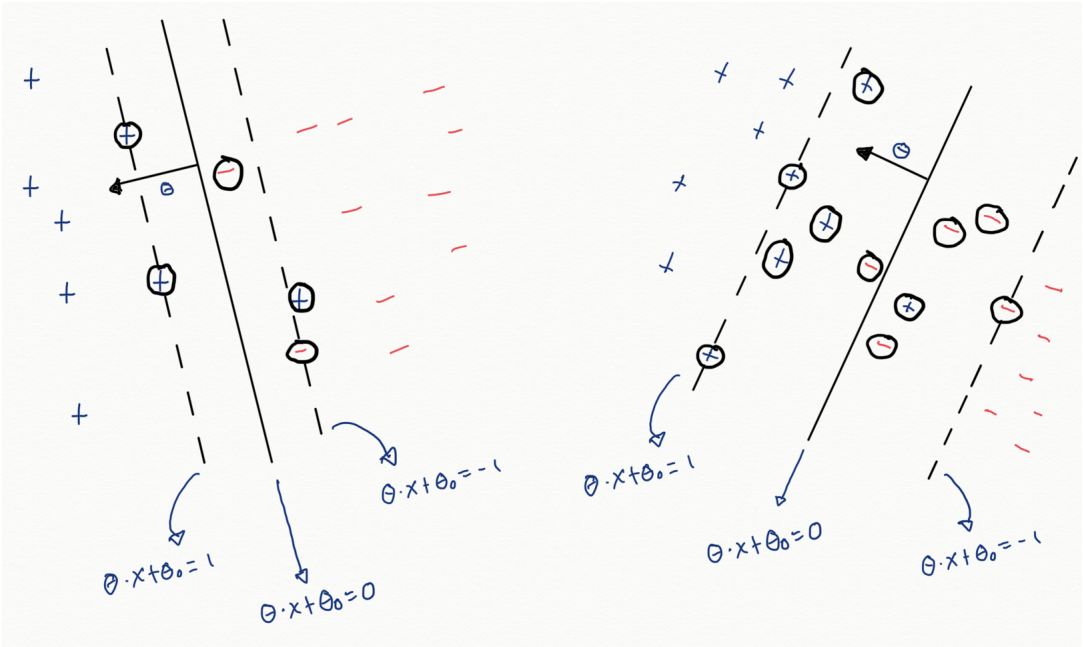

Figura 3.5: El lado izquierdo muestra como se busca minimizar las violaciones con un valor λ pequeño, mientras que el lado derecho valores mayores de λ aceptan violaciones del margen en virtud de incrementar el tamaño del mismo.

Para encontrar los valores de θ y θ_0 que minimicen el error, reformularemos la función objetivo de la máquina de soporte vectorial como un problema de *programación cuadrática*, quedando:

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

$$sujeto\ a:$$

$$y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0\right) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$
$$i = 1, \ldots, n$$

Las variables de holgura ξ_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:

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

Donde η es conocida como tasa de aprendizaje, si se decrementa apropiadamente, las actualizaciones por gradiente descendiente garantizan la convergencia hacia la misma solución de la maquina de soporte vectorial. η_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 movemeremos hacia el óptimo y en consecuencia, el ruido inherente a la selección estocástica tiende a cero. Lo

anterior se consuma en el llamado algoritmo PEGASOS acrónimo inglés para *Primal Estimated sub-GrAdient SOlver for SVM*.

Algorithm 5 Algoritmo PEGASOS

procedure PEGASOS($\{\left(x^{(i)}, y^{(i)}\right), i = 1, \ldots, n\}$,$\lambda$,$T$)
  $\theta \leftarrow \vec{0}$
  for $t = 1, \ldots, T$ **do**
    Elegir al azar $i \in \{1, \ldots, n\}$
    $\eta \leftarrow 1/t$
    if $y^{(i)}\theta \cdot x^{(i)} \leq 1$ **then**
      $\theta \leftarrow (1 - \eta\lambda)\,\theta + \eta y^{(i)} x^{(i)}$
    else
      $\theta \leftarrow (1 - \eta\lambda)\,\theta$
  return θ

Por último, discutiremos lo correspondiente al híperplano de máximo margen para el caso factible. Para ello, haremos un ligero cambio al problema de optimización de la máquina de soporte vectorial, a saber:

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

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

$$\text{mín}\, \frac{1}{2}\|\theta\|^2 \;\; s.a \;\; y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0\right) \geq 1, i = 1, \ldots, n$$

Que es también un problema de programación cuadrática y geométricamente, 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

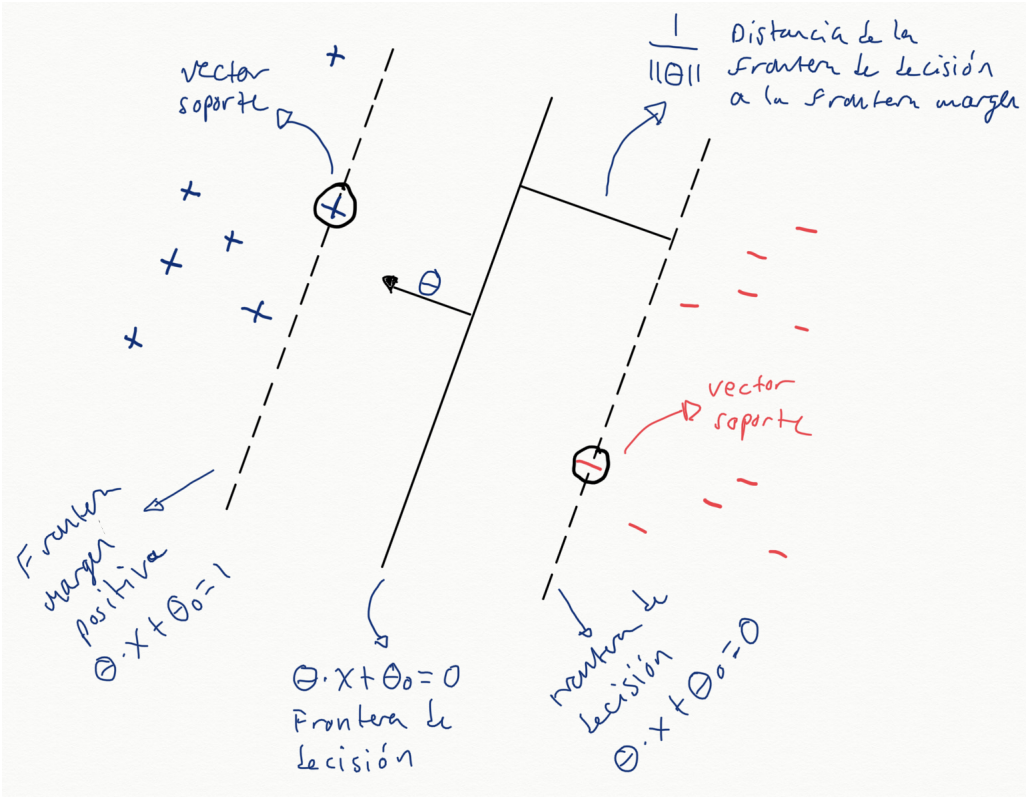

Figura 3.6: Separador lineal de margen máximo

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 $f : \mathbb{R}^d \to \mathbb{R}$ 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:

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

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 $S_n = \{\left(x^{(t)}, y^{(t)}\right), t = 1, \ldots, n\}$ 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*:

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

Por simplicidad, se omite el parámetro θ_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: $Loss(z) = z^2/2$, 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:

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

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

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

Veamos la relación que existe entre $R_n(\theta)$ y $R_{n'}^{test}(\theta)$, 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 subyacentre 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:

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

Por tanto, el algoritmo podrá escribirse como sigue:

Algorithm 6 Algoritmo descenso por gradiente estocástico para mínimos cuadrados

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

Donde η_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 $\left(y^{(t)} - \theta \cdot x^{(t)}\right)$, 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:

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

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 $\hat{\theta} \cdot x^{(t)} = \left(x^{(t)}\right)^T \hat{\theta}$, 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 $X = \left[x^{(1)}, \ldots, x^{(n)}\right]^T$, es decir, X^T tiene cada vector de características como columna, mientras que X los tiene apilados como filas. Si definimos también $\vec{y} = \left[y^{(1)}, \ldots, y^{(n)}\right]^T$ (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 resolvible, esto es:

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

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:

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

y, en consecuencia, el algoritmo podrá escribirse como:

Algorithm 7 Algoritmo descenso por gradiente estocástico para regresión de cresta

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

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, el término de regularización modifica unicamente 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 λ 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 λ, y por tanto, $R_n\left(\hat{\theta}(\lambda)\right)$ incrementará directamente proporcional a λ. 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 λ 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 λ crece demasiado se provoca el efecto contrario al empujar en demasía los parámetros hacia cero.

Referencias

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.