Aprendizaje Automático
Cuestionario de teoría 3

Alfonso García Martínez
alfonsogmw@correo.ugr.es

Mayo de 2019

1. ¿Podría considerarse Bagging como una técnica para estimar el error de predicción de un modelo de aprendizaje? Diga sí o no con argumentos. En caso afirmativo compárela con validación cruzada.

   Solución: La técnica de Bagging sí puede usarse para estimar el error de predicción de un modelo, porque con ella se generan conjuntos de entrenamiento a partir del muestreo con reemplazamiento (*bootstrapping*), y los datos no incluidos en esos conjuntos pueden usarse para calcular el error y de esta forma evaluar el modelo.

   Bagging es similar a la validación cruzada en el sentido de que en ambas se generan distintos conjuntos de entrenamiento a partir de la muestra para entrenar el predictor, pero existen diferencias entre ambas:

   - Bagging, como ya hemos mencionado, utiliza muestreo por reemplazamiento, luego puede que hayan datos que se usen más veces que otros, o puede ocurrir que queden datos sin usarse ni una sola vez. La validación cruzada particiona la muestra en k particiones diferentes (subconjuntos disjuntos), y se asegura de que cada subconjunto (y por tanto cada dato) se use en total el mismo número de veces para entrenamiento ($k-1$ veces de las k iteraciones en total). Básicamente, **la validación cruzada utiliza un muestreo sin reemplazamiento**, y esto la hace distinta al Bagging.

   - En validación cruzada, el tamaño de cada subconjunto se ajusta según el k elegido, de forma que cada subconjunto tendrá $\frac{N}{k}$ elementos si el tamaño de la muestra es N. Como no tiene sentido usar $k = 1$ (no tendríamos subconjuntos para validar), el subconjunto que se use cada vez para entrenamiento siempre tendrá un tamaño menor que N.

     En Bagging esto no tiene porqué ser así: como el muestreo es con reemplazamiento, sí es posible tener conjuntos de entrenamiento de N elementos, que o bien tengan exactamente los mismos elementos que la muestra original (es poco probable), o que bien no tenga todos los datos, sino que varios de ellos estén repetidos.

2. Considere que dispone de un conjunto de datos linealmente separable. Recuerde que una vez establecido un orden sobre los datos, el algoritmo perceptron encuentra un hiperplano separador interando sobre los datos y adaptando los pesos de acuerdo al algoritmo

Algorithm 1 Perceptron

Entradas: (x_i, y_i), $i = 1, \ldots, n$, $\mathbf{w} = 0$, $k = 0$
repeat
  $k \leftarrow (k + 1) \bmod n$
  if $\text{sign}(y_i) \neq \text{sign}(\mathbf{w}^T \mathbf{x}_i)$ **then**
    $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
  end if
until todos los puntos bien clasificados

Modificar este pseudo-código para adaptarlo a un algoritmo simple de SVM, considerando que en cada iteración adaptamos los pesos de acuerdo al caso peor clasificado de toda la muestra. Justificar adecuadamente/matemáticamente el resultado, mostrando que al final del entrenamiento solo estaremos adaptando los vectores soporte.

Solución: El caso peor clasificado en una iteración será aquel que, entre los que están mal clasificados, se encuentra a mayor distancia del plano.

Algorithm 2 SVM simple (1^{er} intento)

Entradas: (x_i, y_i), $i = 1, \ldots, n$, $\mathbf{w} = 0$, $k = 0$
repeat
  $peor \leftarrow 1$
  for j desde 1 hasta n **do**
    if $\text{sign}(y_j) \neq \text{sign}(\mathbf{w}^T \mathbf{x}_j + b)$ **then**
      if $\frac{|\mathbf{w}^T \mathbf{x}_j + b|}{||\mathbf{w}||} > \frac{|\mathbf{w}^T \mathbf{x}_{peor} + b|}{||\mathbf{w}||}$ **then**
        $peor \leftarrow j$
      end if
    end if
  end for
  $\mathbf{w} \leftarrow \mathbf{w} + y_{peor} \mathbf{x}_{peor}$
  $b \leftarrow b + y_{peor}$
until todos los puntos bien clasificados

¿Por qué b se actualiza sumándole y_{peor}? Ahora b cumple el papel que cumplía w_0 en la anterior notación. Antes, como todos los puntos $\mathbf{x}_n$ empezaba por un 1 para que se tuviese en cuenta el término independiente al hacer el producto escalar $\mathbf{w}^T\mathbf{x}_n$. Cuando $\mathbf{w}$ se actualiza en el perceptrón según la regla $\mathbf{w} \leftarrow \mathbf{w} + y_n\mathbf{x}_n$, al término independiente w_0 le correspondía la actualización:

$$w_0 \leftarrow w_0 + y_n x_{n0}$$

Que, como ya sabemos, es equivalente a:

$$w_0 \leftarrow w_0 + y_n \cdot 1$$

Como b cumple ahora el papel de w_0, se debe actualizar exactamente igual que él.

Esta primera aproximación, efectivamente, ajusta el vector de pesos $\mathbf{w}$ del hiperplano teniendo en cuenta únicamente los casos peor clasificados de cada iteración entre los que están mal clasificados, pero tal y como está construido no funciona como un SVM, pues no garantiza que, una vez todos los casos estén bien clasificados, el plano se siga ajustando con los vectores de soporte para dar la separación máxima.

Redefinamos lo que significa que un caso sea *el peor clasificado* en una iteración:

- **Si existen puntos mal clasificados**, entonces habrá al menos un punto $\mathbf{x}_i$ que cumpla que $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) < 0$. En este caso, el punto peor clasificado será aquel que, entre todos los mal clasificados, se encuentre más alejado del plano. En otras palabras: buscamos al punto $\mathbf{x}_p$ que, teniendo $y_p(\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b) < 0$ (mal clasificado), tenga el **máximo** $\frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b|}{||\mathbf{w}||}$ (está a la mayor distancia del plano), y por tanto tenga el mayor $|\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b|$.

- **Si no existen puntos mal clasificados**, o sea, todos están en ese momento bien clasificados, entonces para cualquier punto $\mathbf{x}_i$ de la muestra se cumple que $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) > 0$. En este caso, el punto peor clasificado será aquel que, estando bien clasificado y sin que haya ningún otro punto mal clasificado, esté más cerca del hiperplano. Dicho de otra forma: buscamos al punto $\mathbf{x}_p$ que, teniendo $y_p(\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b) > 0$ (bien clasificado), tenga el **mínimo** $\frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b|}{||\mathbf{w}||}$ (está a la menor distancia del plano), y por tanto tenga el menor $|\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b|$.

Tal y como lo hemos descrito, en ambas situaciones **el punto peor clasificado es el punto $\mathbf{x}_p$ que minimiza** $y_p(\mathbf{w}^{\mathrm{T}}\mathbf{x}_p + b)$.

Algorithm 3 SVM simple (definitivo)

Entradas: (x_i, y_i), $i = 1, \ldots, n$, $\mathbf{w} = 0$, $k = 0$
repeat
  $peor \leftarrow 1$
  for j desde 1 hasta n **do**
    if $y_j(\mathbf{w}^{\mathrm{T}}\mathbf{x}_j + b) < y_{peor}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_{peor} + b)$ **then**
      $peor \leftarrow j$
    end if
  end for
  $\mathbf{w} \leftarrow \mathbf{w} + y_{peor}\mathbf{x}_{peor}$
  $b \leftarrow b + y_{peor}$
until todos los puntos bien clasificados $\wedge$ hiperplano deja mismo margen para ambas clases

Esta nueva definición de punto peor clasificado tiene una importante consecuencia, y es que una vez se hayan conseguido clasificar correctamente todos los puntos, el algoritmo seguirá ajustando el hiperplano con ejemplos bien clasificados. Veamos qué efectos provoca esto:

Comportamiento con ejemplos mal clasificados

En el cuestionario 1 ya demostramos que, en cada paso, el algoritmo perceptrón *acercaba* el hiperplano hacia ese caso mal clasificado para intentar pasarlo al otro lado y que , de esta forma, pasase a estar bien clasificado.

Aquí pasa exactamente lo mismo, la únicas diferencias están en la notación (usamos b en lugar de w_0) y en que no ajusta con cualquier punto mal clasificado, sino con el peor de todos.

- Si $y(t) = 1$, la estimación para un punto $\mathbf{x}$ en este caso será $\text{sign}(\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t)+b(t)) = -1$, y entonces a $\mathbf{w}(t)$ estaremos **sumándole** $\mathbf{x}(t)$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{x}(t)$$
$$b(t+1) = b(t) + 1$$

con lo cual, al hacer la nueva predicción sobre el mismo dato

$$\mathbf{w}(t+1)^{\mathrm{T}}\mathbf{x}(t) + b(t+1) = (\mathbf{w}(t) + \mathbf{x}(t))^{\mathrm{T}}\mathbf{x}(t) + (b(t)+1) =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + \mathbf{x}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) + 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + ||\mathbf{x}(t)||^2 + b(t) + 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) + (||\mathbf{x}(t)||^2 + 1) =$$

Como a $\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t)$, que en este caso es negativo, le estamos sumando $||\mathbf{x}(t)||^2 + 1$ que es positivo, entonces estaremos contribuyendo a que $\mathbf{w}(t)^T\mathbf{x}(t)$ sea *menos negativo*, o sea, que lo desplazamos en sentido positivo para que se acerque al 0 o incluso lo sobrepase, en cuyo caso pasaría a estar bien clasificado.

- Si $y(t) = -1$, la estimación en este caso será $\text{sign}(\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t)) = 1$, y entonces a $\mathbf{w}(t)$ estaremos **restándole** $\mathbf{x}(t)$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{x}(t)$$
$$b(t+1) = b(t) - 1$$

con lo cual, al hacer la nueva predicción sobre el mismo dato

$$\mathbf{w}(t+1)^{\mathrm{T}}\mathbf{x}(t) + b(t+1) = (\mathbf{w}(t) - \mathbf{x}(t))^{\mathrm{T}}\mathbf{x}(t) + (b(t) - 1) =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) - \mathbf{x}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) - 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) - ||\mathbf{x}(t)||^2 + b(t) - 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) - (||\mathbf{x}(t)||^2 + 1) =$$

Como a $\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t)$, que en este caso es positivo, le estamos restando $||\mathbf{x}(t)||^2 + 1$ que es positivo, entonces estaremos contribuyendo a que $\mathbf{w}(t)^T\mathbf{x}(t)$ sea *menos positivo*, o sea, que lo desplazamos en sentido negativo para que se acerque al 0 o incluso lo sobrepase.

Comportamiento con ejemplos bien clasificados

En una iteración t con todos los ejemplos bien clasificados:

- Si $y(t) = 1$, la estimación para un punto $\mathbf{x}$ en este caso será $\text{sign}(\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t)+b(t)) = 1$, y entonces a $\mathbf{w}(t)$ estaremos **sumándole** $\mathbf{x}(t)$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{x}(t)$$
$$b(t+1) = b(t) + 1$$

con lo cual, al hacer la nueva predicción sobre el mismo dato

$$\mathbf{w}(t+1)^{\mathrm{T}}\mathbf{x}(t) + b(t+1) = (\mathbf{w}(t) + \mathbf{x}(t))^{\mathrm{T}}\mathbf{x}(t) + (b(t)+1) =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + \mathbf{x}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) + 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + ||\mathbf{x}(t)||^2 + b(t) + 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) + (||\mathbf{x}(t)||^2 + 1) =$$

Como a $\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t)$, que en este caso es positivo, le estamos sumando $||\mathbf{x}(t)||^2 + 1$ que también es positivo, entonces estaremos contribuyendo a que $\mathbf{w}(t)^T\mathbf{x}(t)$ sea aún mayor, *aún más positivo*, o sea, que lo desplazamos en sentido positivo para que se aleje aún más del 0.

- Si $y(t) = -1$, la estimación en este caso será $\text{sign}(\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t)) = 1$, y entonces a $\mathbf{w}(t)$ estaremos **restándole** $\mathbf{x}(t)$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{x}(t)$$
$$b(t+1) = b(t) - 1$$

con lo cual, al hacer la nueva predicción sobre el mismo dato

$$\mathbf{w}(t+1)^{\mathrm{T}}\mathbf{x}(t) + b(t+1) = (\mathbf{w}(t) - \mathbf{x}(t))^{\mathrm{T}}\mathbf{x}(t) + (b(t)-1) =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) - \mathbf{x}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) - 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) - ||\mathbf{x}(t)||^2 + b(t) - 1 =$$
$$= \mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t) - (||\mathbf{x}(t)||^2 + 1) =$$

Como a $\mathbf{w}(t)^{\mathrm{T}}\mathbf{x}(t) + b(t)$, que en este caso es negativo, le estamos restando $||\mathbf{x}(t)||^2 + 1$ que es positivo, entonces estaremos contribuyendo a que $\mathbf{w}(t)^T\mathbf{x}(t)$ sea aún menor, *aún más negativo*, o sea, que lo desplazamos en sentido negativo para que se aleje aún más del 0.

¿Cuál es la conclusión intuitiva que podemos sacar de esta demostración? Básicamente, que el algoritmo acerca el hiperplano a los ejemplos mal clasificados, y lo aleja de los bien clasificados. ¿De qué forma afecta esto a la obtención del plano óptimo?

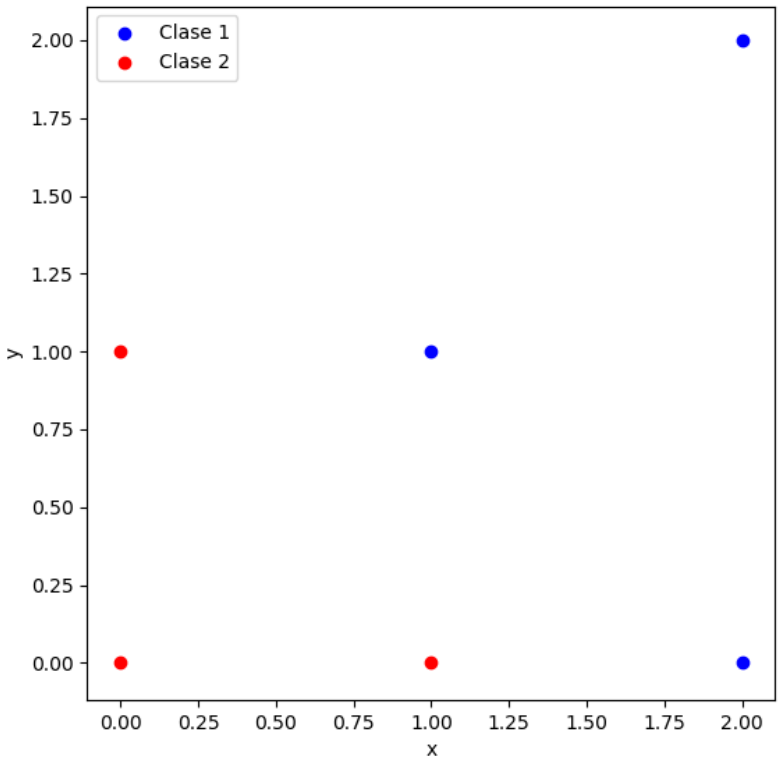
Al principio, el algoritmo se encargará de acercar sucesivamente el hiperplano a los ejemplos mal clasificados más lejanos, y continuará haciéndolo hasta que no queden casos mal clasificados.

En las últimas iteraciones, cuando todos los datos estén bien clasificados, el algoritmo cogerá los puntos más cercanos al plano (los **vectores de soporte**), y alejará el hiperplano de ellos, reajustándolo para dejar una separación o **margen**. El algoritmo continuará haciendo esto hasta que el margen sea el mismo tanto para los vectores de soporte de una clase como para los de la otra clase.

3. Considerar un modelo SVM y los siguientes datos de entrenamiento: Clase-1:(1,1),(2,2),(2,0), Clase-2:(0,0),(1,0),(0,1)

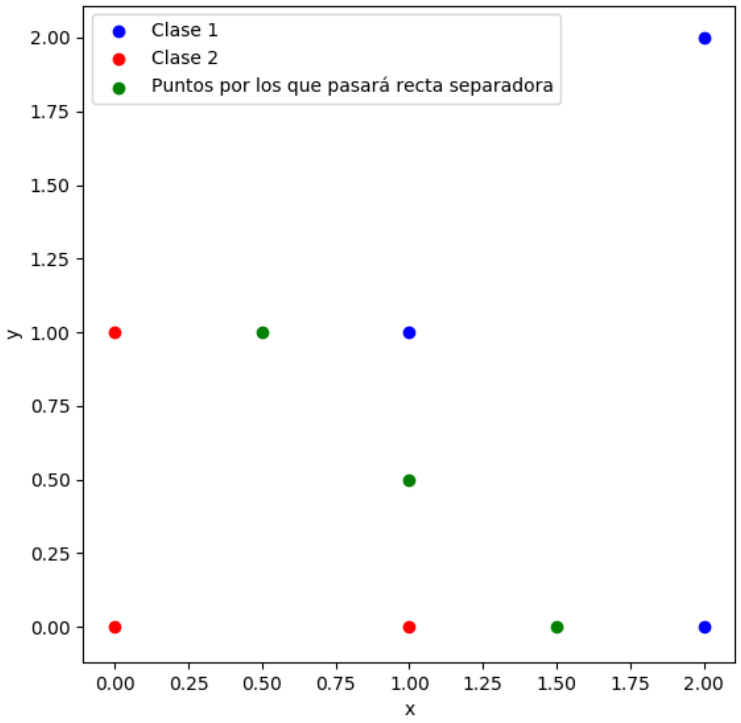
   a) Dibujar los puntos y construir por inspección el vector de pesos para el hiperplano óptimo y el margen óptimo.

   Solución: Primero, dibujamos los puntos indicados en un plano:



   Es fácil ver a partir de aquí cuáles son los puntos de distintas clases más cercanos entre. Si obtenemos los puntos medios entre cada par de puntos más cercanos, podremos saber por dónde pasará el hiperplano:

   - Entre los puntos (1,0) y (2,0) $\rightarrow$ (1.5,0)
   - Entre los puntos (0,1) y (1,1) $\rightarrow$ (0.5,1)
   - Además, entre los puntos (1,0) y (1,1) $\rightarrow$ (1,0.5)



Con dos de esos tres puntos, podemos calcular la ecuación punto-pendiente de la recta que separaría a los puntos en el plano. Tomaremos, por ejemplo, los puntos (0.5,1) y (1,0.5). La pendiente de la recta sería:

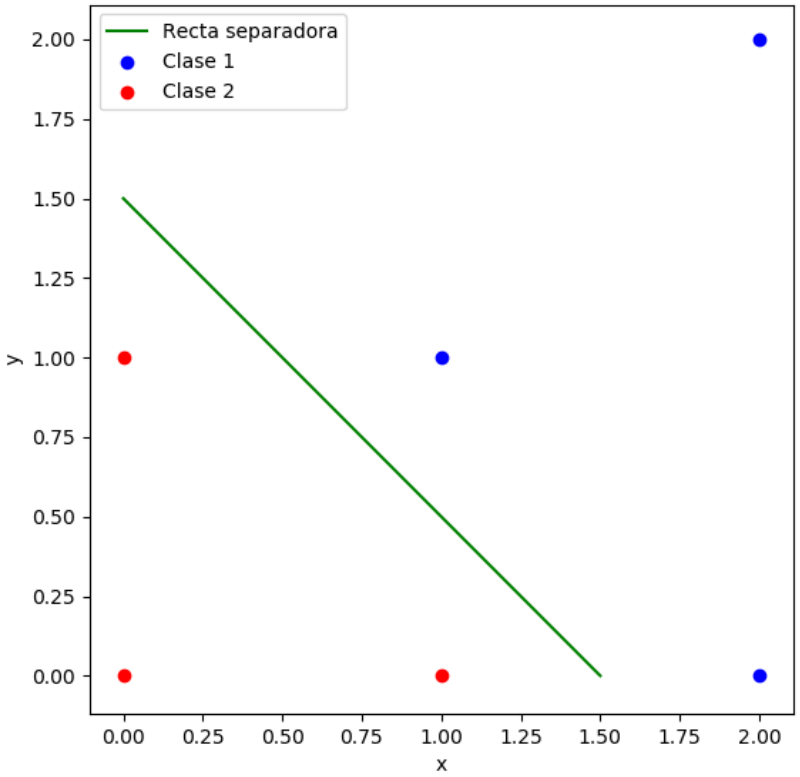
$$m = \frac{1 - 0{,}5}{0{,}5 - 1} = \frac{0{,}5}{-0{,}5} = -1$$

Y la ecuación de la recta:

$$y - y_0 = m(x - x_0)$$

Si tomamos como (x_0, y_0) al punto (0.5, 1):

$$y - 1 = -(x - 0{,}5)$$
$$y = -x + 1{,}5$$



Aún nos queda encontrar los pesos del modelo. Recordemos que, si contamos las etiquetas (+1 para Clase1, -1 para Clase2), el separador es un **plano tridimensional**, por lo que la recta que acabamos de pintar es una intersección. Teniendo en cuenta los valores de las etiquetas, podemos interpretar que la recta es la intersección con el plano X-Y, donde $z = 0$. Tenemos:

$$y = -x + 1{,}5$$

$$-y - x + 1{,}5 = 0$$

Aquí es fácil ver la interpretación de que $z = 0$. Hay que tener en cuenta que, para una recta intersección, existen infinitos planos que pasan por ella:

$$\rho(-y - x + 1{,}5) = 0$$

Con lo que podemos definir el conjunto de planos que pasan por esa intersección

$$H(x,y) = \rho \cdot h(x,y) = \rho(1{,}5 - x - y) \qquad \rho \in \mathbb{R}$$

Para ello, definamos el vector de pesos:

$$\mathbf{w} = (w_0, w_1, w_2) = (1{,}5\rho, -\rho, -\rho)$$

O, si usamos la notación en la que en vez de w_0 tenemos el umbral b:

$$\mathbf{w} = (w_1, w_2,) = (-\rho, -\rho)$$
$$b = 1{,}5\rho$$

La cuestión ahora es: ¿cuál de esos planos es el óptimo? Si queremos encontrar el valor óptimo de ρ, no nos queda más remedio que resolver el problema de optimización cuadrático asociado. El objetivo es minimizar

$$\frac{1}{2}||\mathbf{w}||^2 = \frac{1}{2}(w_1^2 + w_2^2)$$

Bajo las restricciones:

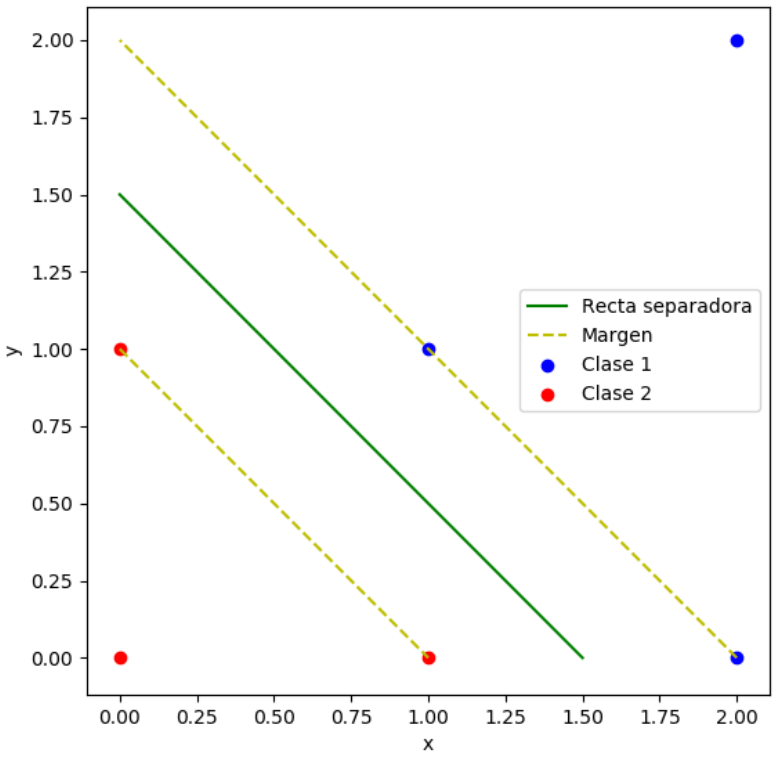
$$y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \qquad n = 1, 2, \ldots, N$$

De forma que, para los 6 datos que tenemos:

1) $w_1 + w_2 + b \geq 1$
2) $2w_1 + 2w_2 + b \geq 1$
3) $2w_1 + b \geq 1$
4) $-b \geq 1$
5) $-w_1 - b \geq 1$
6) $-w_2 - b \geq 1$

b) ¿Cuáles son los vectores soporte?

Solución: Son los puntos de los cuales calculamos los puntos medios para saber por donde tenía que pasar el plano, y que a raíz de esto son los puntos que están más cerca del hiperplano. Se tratan de los puntos (1,0), (0,1), (1,1) y (2,0).



c) Construir la solución en el espacio dual. Comparar la solución con la del apartado (a).

Solución: Para este apartado tendremos que usar un *Quadratic Programming Solver* al que podamos introducirle como entrada los datos de este problema y nos devuelva una salida que podamos comparar con la solución *informal* que acabamos de construir por inspección. Por ejemplo, usaremos el paquete *quadprog* que hay disponible para Python.

Definiendo las matrices $\mathbf{Q}$ y $\mathbf{A}$, y los vectores $\mathbf{p}$ y $\mathbf{c}$ adaptados a nuestro problema, obtemenos el vector de pesos:

$$\mathbf{u} = \mathbf{w}^* = (w_0^*, w_1^*, w_2^*) = (-3, 2, 2)$$

Luego, para la expresión que obtuvimos en el apartado *a*) para definir el plano, $\rho = -2$.

4. ¿Cúal es el criterio de optimalidad en la construcción de un árbol? Analice un clasificador en árbol en términos de sesgo y varianza. ¿Qué estrategia de mejora propondría?

   Solución: Encontrar el árbol óptimo que clasifique a la perfección un conjunto de datos es una tarea sumamente costosa en términos de complejidad computacional. En su lugar, se utilizan enfoques heurísticos, aproximativos, con los que se toman decisiones conforme se va construyendo el árbol.

   La idea intuitiva detrás de esta forma de construir árboles consiste en que, en cada nodo que se forma, se debe elegir el atributo o característica que mejor separe los datos con la ramificación que surge a partir de ese nodo, es decir, el que de lugar a la separación más homogénea posible.

   Por ejemplo: supongamos que durante la construcción de un árbol, tenemos dos variables, A y B, ambas binarias. Una de ellas, A, deja casi todos los casos positivos para uno de sus valores, y a casi todos los casos negativos para el otro. La variable B, por el contrario, deja aproximadamente la misma cantidad de casos positivos y negativos tanto para un valor como para otro. ¿Cuál de las dos variables se debería escoger para el nodo actual? Efectivamente, A es la mejor opción.

   Existen distintas métricas para elegir la siguiente variable entre las restantes. Una de las más usadas se basa en los conceptos de entropía y ganancia de información. Sin entrar en mucho detalle, la entropía mide matemáticamente la incertidumbre o *grado de sorpresa* de una variable, y la ganancia de información a su vez mide cuánto disminuye la entropía al elegir esa variable respecto cómo teníamos los datos antes de elegir esa variable. Así pues, lo que se busca en este enfoque es **elegir la variable con mayor ganancia de información**.

   Es perfectamente posible, incluso usando estos métodos aproximativos, que nuestro árbol se ajuste perfectamente a la muestra de datos, de forma que podríamos acabar con un árbol en el que cada uno de los nodos hoja correspondería a un dato distinto. Luego, en teoría, un árbol tiene el potencial para **separar cualquier conjunto de datos**, lo que se traduce en un bajo nivel de sesgo, pero por otra parte provoca que la

dimensión de Vapnik-Chervonenkis sea infinita.

Precisamente esta gran capacidad de separar los datos hace que los árboles sean terriblemente sensibles a cualquier cambio en los datos, lo que se traduce en una alta tendencia al *overfitting* y, en definitiva, en un **alto grado de varianza**.

¿Qué puede hacerse para mitigar este defecto? Por ejemplo, se puede podar los árboles para *reducir su potencia* y por tanto su varianza, ya sea deteniendo prematuramente el proceso de construcción, o podando el árbol hasta cierta profundidad una vez ya está construido.

Otros métodos muy usados consisten en utilizar distintos árboles ajustados con distintos subconjuntos de entrenamiento y combinarlos de alguna forma (Bagging, Random Forest). Esta opción es especialmente buena, porque es demostrable que combinar predictores disminuye la varianza, pero se conserva el bajo nivel de sesgo.

5. ¿Cómo influye la dimensión del vector de entrada en los modelos: SVM, RF, Boosting and NN?

   Solución: Un problema con una alta dimensionalidad fuerza a que se usen modelos de alta compejidad que tengan un sesgo lo suficientemente bajo como para que *den la talla* y puedan ajustarse lo suficientemente bien a una muestra dada.

   El problema de usar modelos muy complejos es que, al tener tanta potencia y capacidad de moldearse a los datos de la muestra, tienden al sobreajuste. Este no es sino uno de los problemas fundamentales del Machine Learning: cómo obtener un modelo lo suficientemente complejo y potente como para que pueda ajustarse bien a los datos sin que por ello pierda demasiada capacidad de generalización.

   Existen modelos que inherentemente, por cómo han sido diseñados, no solo buscan disminuir el error sino que también procuran mantener la capacidad de generalización. Son el caso del SVM y el Boosting, pues ambos son modelos que buscan no solo clasificar correctamente los datos de la muestra, sino que además consiguen dejar el mayor margen posible entre las diferentes clases para que sea más probable clasificar bien los datos de fuera de la muestra. Por ello, **SVM y Boosting son modelos que responden bien a datos de alta dimensionalidad**.

   Por otra parte, las NN son modelos con muchísima potencia. Si se saben usar bien, teóricamente son capaces de ajustarse a casi cualquier función. Pero esto es un arma de doble filo: cuanto más compleja sea, más probable será que se ajuste tan bien a la muestra que se pegue a ella. Por este motivo, **hay que tener precaución al usar NNs en problemas de alta dimensionalidad**, pues a mayor dimensionalidad, más nodos habrá en la capa deentrada, con lo que tendremos que controlar la complejidad limitando los nodos de las capas ocultas, o directamente aplicando regularización.

   Otro modelo, los RF, también tienen su forma de afrontar el exceso de complejidad. Los árboles de decisión, por sí solos, también tienen gran capacidad de ajuste, lo que hace que su varianza se dispare. RF combina árboles de decisión (no correlados) y disminuye de esta forma su varianza, lo que también lo convierte en un modelo robusto frente a

problemas complejos con datos de muchas dimensiones.

6. El método de Boosting representa una forma alternativa en la búsqueda del mejor clasificador respecto del enfoque tradicional implementado por los algoritmos PLA, SVM, NN, etc. a) Identifique de forma clara y concisa las novedades del enfoque; b) Diga las razones profundas por las que la técnica funciona produciendo buenos ajustes (no ponga el algoritmo); c) Identifique sus principales debilidades; d) ¿Cuál es su capacidad de generalización comparado con SVM?

   Solución:

   a) La novedad que introduce el Boosting es que, a parte de ser un método agregativo donde se combinan clasificadores débiles, **la distribución de la muestra varía para cada clasificador débil**, de forma que se les da más peso a los datos mal clasificados en una iteración y menos peso a los bien clasificados, para que el clasificador de la siguiente iteración de más importancia a los datos que no se clasificarion bien en anteriores ajustes.

   b) La clave del Boosting está en combinar de una cierta manera clasificadores débiles. Es importante que los clasificadores sean débiles (no mucho mejores que uno aleatorio), pues la distribución irá variando y no queremos de ninguna manera la más mínima presencia de sobreajuste.

   Cada clasificador débil, como hemos explicado más arriba, altera la muestra de datos con la que se ajusta, aumentando el peso o importancia de los ejemplos mal clasificados y disminuyendo la de los clasificados correctamente.

   Por ejemplo, en el caso de AdaBoos (un algoritmo concreta que utiliza técnicas de Boosting), para cada clasificador (en cada iteración del método), conocemos ϵ_t, la proporción de datos mal clasificados. Así pues, la intensidad con la que se cambian los pesos de los datos se determina en función de un parámetro α_t dependiente de ϵ_t. A su vez, la α_t de cada clasificador se utiliza para combinar los clasificadores, de forma que al final se les dará más importancia a los clasificadores más exitosos, que son aquellos con mayor α_t. Esta forma de combinar los clasificadores hace que el clasificador resultante sea muy eficaz.

c) El inconveniente del Boosting es que su rendimiento depende mucho de la clase de funciones que elijamos para los clasificadores débiles: si es demasiado compleja, los clasificadores dejarán de ser débiles y tenderán al *overfitting*. Si, por el contrario, la clase es demasiado simple, puede que los clasificadores débiles no se ajusten lo suficientemente bien a los datos, e incluso puede que hasta pierdan capacidad de generalización y, de nuevo, tiendan al *overfitting*.

Además, el Boosting también depende mucho de los datos, lo que lo hace muy sensible al ruido: el ruido mal clasificado altera notablemente la distribución de la muestra, provacando que el resultado final al combinar los clasificadores empeore.

d) El error de *training* solo tiene en cuenta las muestras mal clasificadas. Si tenemos en cuenta tanto ejemplos mal clasificados como ejemplos bien clasificamos al promediar los clasificadores tendremos una manera de estimar la *confianza* del clasificador, que se traduce en una forma de **medir el margen**. Por ello, Boosting goza de una gran capacidad de generalización, pues cuantos más clasificadores débiles se combinen en la solución final, más énfasis de hará en dejar un mayor margen y, consecuentemente, podremos esperar que el error de test disminuya.

Podemos pensarlo de la siguiente manera: los puntos más cercanos a la verdadera frontera de decisión serán los más problemáticos, es decir, serán los que más veces serán clasificados incorrectamente. Por tanto, se les penalizará más y tendrán mucho peso en las distribuciones, por lo que serán los más determinantes a la hora de ajustar el clasificador. Cuantas más ejecuciones (y clasificadores débiles) se usen, más oportunidad tendrá el algoritmo de Boosting de clasificar bien esos puntos problemáticos, y para ello procurará dejar el mayor márgen posible.

Las SVM, como sabemos, también poseen una gran capacidad de generalización al igual que el Boosting. En el ámbito de las SVM, los puntos problemáticos de los que hemos hablado **corresponderían a los vectores de soporte**, que en su caso son también los que determinan en última instancia cómo será el clasificador para que deje el mayor margen posible entre ellos.

7. Discuta pros y contras de los clasificadores SVM y Random Forest (RF). Considera que SVM por su construcción a través de un problema de optimización debería ser un mejor clasificador que RF. Justificar las respuestas.

   Solución:

   SVM

   Pros:

   - Maximiza el margen, gran capacidad de generalización
   - Garantiza mejor resultado posible si datos son separables

   Contras:

   - Difícil saber elegir un kernel apropiado si datos no son separables linealmente (que es casi siempre)
   - Tarda mucho en ser entrenado/ajustado (hay que resolver un problema de optimización cuadrática)

   Random Forest

   Pros:

   - Baja varianza debido a composición de clasificadores (árboles) y por decorrelación de estos
   - Sesgo aceptable por los árboles que lo componen
   - Parametrizable: podemos elegir número de árboles y su profundidad

   Contras:

   - Poca interpretabilidad (tenemos muchos árboles, muy difícil saber qué variables son las más importantes)

Es cierto que SVM, por estar construido en base a un problema de clasificación, en teoría ofrece muy buenos resultados. Aún así, SVM ofrece un modelo verdaderamente óptimo **si los datos son linealmente separables**. En otro caso, hay que hacer uso del truco del kernel para transformar el espacio del problema, y dado lo difícil que resulta encontrar un kernel que realmente nos ayude, llegados a ese punto es normal plantearse el uso de otros clasificadores como el RF.

8. ¿Cuál es a su criterio lo que permite a clasificadores como Random Forest basados en un conjunto de clasificadores simples aprender de forma más eficiente? ¿Cuales son las mejoras que introduce frente a los clasificadores simples? ¿Es Random Forest óptimo en algún sentido? Justifique con precisión las contestaciones.

   Solución: Un árbol por sí solo tiene un bajo sesgo, pero demasiada varianza ($d_{VC} = +\infty$). Random Forest combina diversos árboles simples, y encuentra la clase mayoritaria entre las predicciones que todos ellos producen. El promediar los resultados hace que la varianza disminuya, lo que lo convierte en un clasificador muy eficaz en comparación con los simples.

   Pero Random Forest no se queda ahí, pues **minimiza la varianza lo máximo posible combinando árboles decorrelados**. Después de todo, si los árboles se parecen entre sí, las predicciones también lo serán, y la varianza no disminuirá tanto como podría.

9. En un experimento para determinar la distribución del tamaño de los peces en un lago, se decide echar una red para capturar una muestra representativa. Así se hace y se obtiene una muestra suficientemente grande de la que se pueden obtener conclusiones estadísticas sobre los peces del lago. Se obtiene la distribución de peces por tamaño y se entregan las conclusiones. Discuta si las conclusiones obtenidas servirán para el objetivo que se persigue e identifique si hay algo que lo impida.

   Solución: Las conclusiones que saquemos en este experimento no serían válidas porque **estamos introduciendo un posible sesgo en la muestra**.

   Si capturamos los peces tirando una red, estaremos obteniendo la muestra de una parte concreta del lago, no de todo el lago. Es posible que los peces de la zona donde se ha echado la red sean más grandes (o más pequeños) que los peces del resto del lago, es decir, que tengan una distribución de tamaños distinta al resto de peces. Por tanto, la muestra no sería representativa de todo el lago.

   La mejor forma de extraer una muestra aleatoria que sí sea representativa sería pescar cada pez individualmente en una zona distinta del lago cada vez, y que además esas zonas estén lo más uniformemente distribuidas a lo largo del lago, sin pescar más peces en un sitio que en ningún otro.

   Otro posible sesgo en la muestra podría estar causado por la propia red: si la red no es lo suficientemente fina, los peces más pequeños del lago escaparían de ella y ninguno de ellos formaría parte de la muestra, con lo cual no estarían representados en nuestro experimento.

10. Identifique qué pasos daría y en qué orden para conseguir con el menor esfuerzo posible un buen modelo de red neuronal a partir una muestra de datos. Justifique los pasos propuestos, el orden de los mismos y argumente que son adecuados para conseguir un buen óptimo. Considere que tiene suficientes datos tanto para el ajuste como para el test.

Solución:

a) En primer lugar, como debe hacerse siempre, separaría los datos en los conjuntos de test y *training*, en una proporción de 20 % y 80 % respectivamente, procurando que hubiese más o menos la misma distribución de valores de $\mathcal{Y}$ en ambos.

b) Definiría un criterio de parada y un vector de pesos inicial. Para el criterio de parada, fijaría un número máximo de iteraciones y un valor de E_{in} mínimo cercano a cero, por ejemplo 10^{-8}. Para el valor inicial, usaría pesos aleatorios cercanos a cero para no saturar la función de activación. Los pesos además tampoco deberían ser iguales a 0, pues al calcular las derivadas en la *backpropagation*, todo acabaría con valores nulos.

c) Una vez definidos estos criterios, procedería a probar la red sucesivas veces, tanto con todo el conjunto de *training* tal cual como con validación cruzada con $k = 5$, para que cada partición conformase el 20 % del conjunto de *training*. Empezaría probando la red neuronal con, por ejemplo, 1 (ó 0) capas ocultas, e iría añadiendo cada vez más capas y/o más elementos a las capas si observase errores altos en general, tanto en E_{in} como en E_{CV}.

d) Cuando alcanzase un valor lo suficientemente bajo en E_{in}, me fijaría en si E_{CV} fuese demasiado alto en comparación. En tal caso, la red estaría empezando a ser demasiado compleja y estaríamos ante un caso de *overfitting*, por lo que aplicaría alguna técnica de regulación. Probaría con *weight decay*, *dropout* de neuronas o reajustaría la red con una parada prematura.

e) Una vez hubiese elegido el modelo con menor error tanto en *training* como en validación, usaría ese modelo final con el conjunto de test. Si hubiésemos usado el conjunto de test alguna otra vez previamente para elegir o ajustar el modelo, no serviría de nada haber separado los datos en un conjunto de test, pues los estaríamos usando en tal caso como validación.

BONUS

1. Suponga que durante 5 semanas seguidas, recibe un correo postal que predice el resultado del partido de futbol del domingo, donde hay apuestas substanciosas. Cada lunes revisa la predicción y observa que la predicción es correcta en todas las ocasiones. El día de después del quinto partido recibe una carta diciendole que si desea conocer la predicción de la semana que viene debe pagar 50.000€. ¿Pagaría?

   a) ¿Cuántas son las posibles predicciones gana-pierde para los cinco partidos?

   Solución: La combinatoria nos dice que existen 2^n combinaciones posibles para n resultados binarios, luego en este caso tenemos $2^5 = 32$ posibles predicciones.

   b) Si el remitente desea estar seguro de que al menos una persona recibe de él la precicción correcta sobre los 5 partidos, ¿Cuál es el mínimo número de cartas que deberá de enviar?

   Solución: La primera semana, el remitente deberá enviar como mínimo una carta por cada posible combinación, es decir, debe enviar 2^5 cartas, la mitad de ellas prediciendo un resultado, y en la otra prediciendo el otro.

   La siguiente semana se habrá equivocado en la mitad de las predicciones, luego deberá seguir enviando cartas a la otra mitad restante, o sea, $\frac{2^5}{2} = 2^4 = 16$.

   Así sucesivamente, hasta que en la quinta semana, la última, deberá mandar solo dos cartas, de forma que habrá mandado todas las predicciones acertadas al destinatario de una de esas dos cartas. Por tanto, para asegurarse de que una persona reciba todas las predicciones acertadas, deberá mandar como mínimo el siguiente número de cartas:

   $$\sum_{i=1}^{5} 2^i = 62$$

   c) Después de la primera carta prediciendo el resultado del primer partido, ¿a cuantos de los seleccionados inicialmente deberá de

enviarle la segunda carta?

Solución: Como hemos dicho antes, si en la primera semana mandamos tantas cartas como posibles combinaciones, después deberá mandársela a la mitad de personas para las que mandó el primer resultado, que son $2^4 = 16$.

d) ¿Cuantas cartas en total se habrán enviado depués de las primeras cinco semanas?

Solución: $\sum_{i=1}^{5} 2^i = 62$, tal y como se ha demostrado en el apartado *b*), más la carta en la que le hace la oferta por 50.000€, con lo que serían 63.

e) Si el coste de imprimir y enviar las cartas es de 0.5€por carta, ¿Cuanto ingresa el remitente si el receptor de las 5 predicciones acertadas decide pagar los 50.000€?

Solución: En total ha gastado $0{,}5 \cdot 62 = 31$€ ($0{,}5 \cdot 63 = 31{,}50$€ si contamos la carta donde hace la oferta). Su beneficio entonces sería de $50000 - 31 = 49969$€, o de $49968{,}5$€ contando la carta de la oferta.

f) ¿Puede relacionar esta situación con la función de crecimiento y la credibilidad del ajuste de los datos?

Solución: El remitente de nuestro problema, mandando las suficientes cartas, es capaz de acertar en al menos un caso la secuencia de predicciones correctas. Podemos decir que, de alguna forma, puede encontrar la secuencia correcta entre las 2^N posibles mandando las suficientes cartas.

Podemos interpretar que **las secuencias de predicciones son dicotomías**, con lo que la función de crecimiento del remitente es $m_{\mathcal{H}}(N) = 2^N$, y su dimensión $d_{VC} = +\infty$.

De la misma forma en que no es fiable la predicción fuera de la muestra de un clasificador con $d_{VC} = +\infty$, no podemos fiarnos tampoco del remitente, porque el hecho de que pueda probar todas las combinaciones de resultados y acertar en una de ellas *por*

fuerza bruta no implica en absoluto que vaya a acertar a la primera todos los resultados de una serie de partidos la próxima vez.

En vista de todo esto, no pagaría ni un euro si me viese en una situación similar.

2. Considere un modelo de red neuronal con dos capas totalmente conectadas: d unidades de entrada, n_H unidades ocultas y c unidades de salida. Considere la función de error definida por $J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - c_k)^2 = \frac{1}{2}||\mathbf{t} - \mathbf{z}||$, donde el vector $\mathbf{t}$ representa los valores de la etiqueta, $\mathbf{z}$ los valores calculados por la red y $\mathbf{w}$ los pesos de la red. Considere que las entradas de la segunda capa se calculan como $z_k = \sum_{j=0}^{N_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}$ donde el vector $\mathbf{y}$ representa la salida de la capa oculta.

   a) Deducir con todo detalle la regla de adaptación de los pesos entre la capa oculta y la salida.

   b) Deducir con todo detalle la regla de adaptación de los pesos entre la capa de entrada y la capa oculta.

   Usar θ para notar la función de activación.