La tarea de clasificación en detalle (II)

Índice

Referencia principal

- Cap. 3 de "Hands-On Machine Learning with Scikit-Learn and TensorFlow"
- Cap. 4 de "Hands-On Machine Learning with Scikit-Learn and TensorFlow"

(Todas las imágenes pertenecen a dicha referencia salvo que se indique lo contrario)

Al terminar este tema tendremos todos los conceptos necesarios para poder construir soluciones ML para casi cualquier tipo de problema.

1. Clasificación No Lineal

Con modelos lineales sólo se pueden clasificar bien conjuntos de datos linealmente separables, pero esta no es la situación habitual ni mucho menos. El problema de la clasificación no lineal es que hay infinidad de posibles modelos de curvas, superficies o en general hipersuperficies; frente a la única posibilidad que ofrece la clasificación lineal.

Por tanto este problema se aborda de un modo diferente. En vez de buscar los parámetros óptimos de un modelo, vamos a transformar los datos en otros que sí sean linealmente separables.

La transformación de un ejemplo $\mathbf{x}$ en $\phi(\mathbf{x})$ puede implicar un aumento de la dimensionalidad. Por ejemplo, en la Figura 1(Der.) se muestra un conjunto de datos bidimensional, $\mathbf{x} = (x_1, x_2)$, que no es linealmente separable. Sin embargo si lo transformamos en $\phi(\mathbf{x}) = (x_1, x_2, x_3 = x_1 x_2)$ entonces todos los ejemplos en los cuadrantes 1 ó 3 quedan en el semiespacio superior, mientras que aquellos en los cuadrantes 2 ó 4 quedan en el semiespacio inferior, y ya es posible la separación lineal, tal y como se muestra en la Figura 1(Izq.).

En este ejemplo se ha aumentado en 1 la dimensionalidad de los datos, pero aún caben dos dimensiones más si nos restringimos a grado 2 (multiplicacion de una característica por si misma o por la otra): $x_4 = x_1^2$ y $x_5 = x_2^2$.

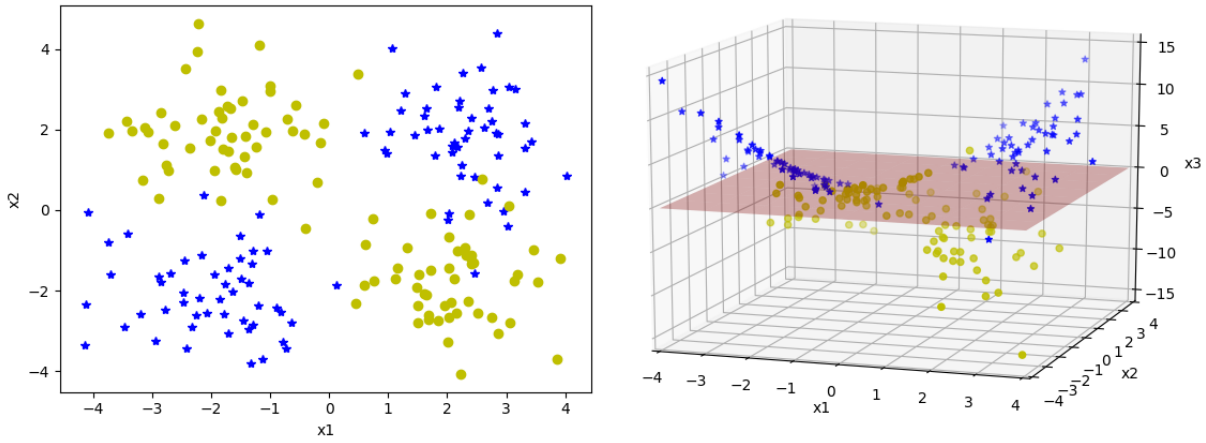


Figura 1: (Izq.) Conjunto de datos no separable linealmente. (Der.) Si a cada ejemplo le añadimos una nueva característica $x_3 = x_1 x_2$, en tres dimensiones sí lo podemos separar linealmente mediante el plano rojo semitransparente. [Fuente: Original de A. Cuesta]

Un modo de sistematizar esta técnica es la **transformación polinómica**. Supongamos, como en el caso anterior, que tenemos ejemplos bidimensionales de modo que $\mathbf{x} = (x_1, x_2)$ y aplicamos la transformación $\phi(\mathbf{x}) = \mathbf{z} = (z_1, z_2, z_2)$ de modo que $z_1 = x_1 x_2$, $z_2 = x_1^2$, $z_3 = x_2^2$. Y a continuación buscamos el clasificador lineal que mejor separe los nuevos puntos $\mathbf{z}$. Pero como estos son tridimensionales, el clasificador será un hiperplano, y tendrá 4 pesos $\mathbf{w} = (w_0, w_1, w_2, w_3)$.

Supongamos que ahora la transformación es $\phi(\mathbf{x}) = \mathbf{z} = (z_1, z_2, z_2, z_3, z_4, z_5, z_6, z_7)$, donde ahora $z_1 = x_1 x_2$, $z_2 = x_1^2$, $z_3 = x_2^2$, $z_4 = x_1^2 x_2$, $z_5 = x_1 x_2^2$, $z_6 = x_1^3$, $z_7 = x_2^3$. Igualmente, ahora buscamos el clasificador lineal para los nuevos puntos $\mathbf{z}$, que tendrá 8 pesos. El numero de pesos aumentaría si además incluimos las características originales, y disminuirá si eliminamos alguna.

Si observamos las componentes de los puntos transformados veremos que son los términos de un polinomio de grado 2, en el primer ejemplo, y de grado 3 en el segundo. También podemos observar que si aumenta el grado del polinomio la dimensión de los puntos $\mathbf{z}$ se dispara. En concreto si $\mathbf{x}$ es n-dimensional y aplicamos una transformación de grado d, la dimensión de los

puntos transformados $\mathbf{z}$, incluyendo las características originales, resulta ser de

$$\frac{(n+d)!}{d!n!}$$

En la Figura 2 se muestra el resultado de aplicar una transformación polinómica de grado 2 sobre los datos del ejemplo-guía en el que pretendíamos aprender a distinguir ceros de unos. Las cinco primeras gráficas muestran la superficie de decisión para diferentes valores del umbral de decisión, mientras que la última muestra el valor de la función de decisión como un continuo, de ahí que se vea un degradado entre el valor más negativo (azul) y el más positivo (rojo).

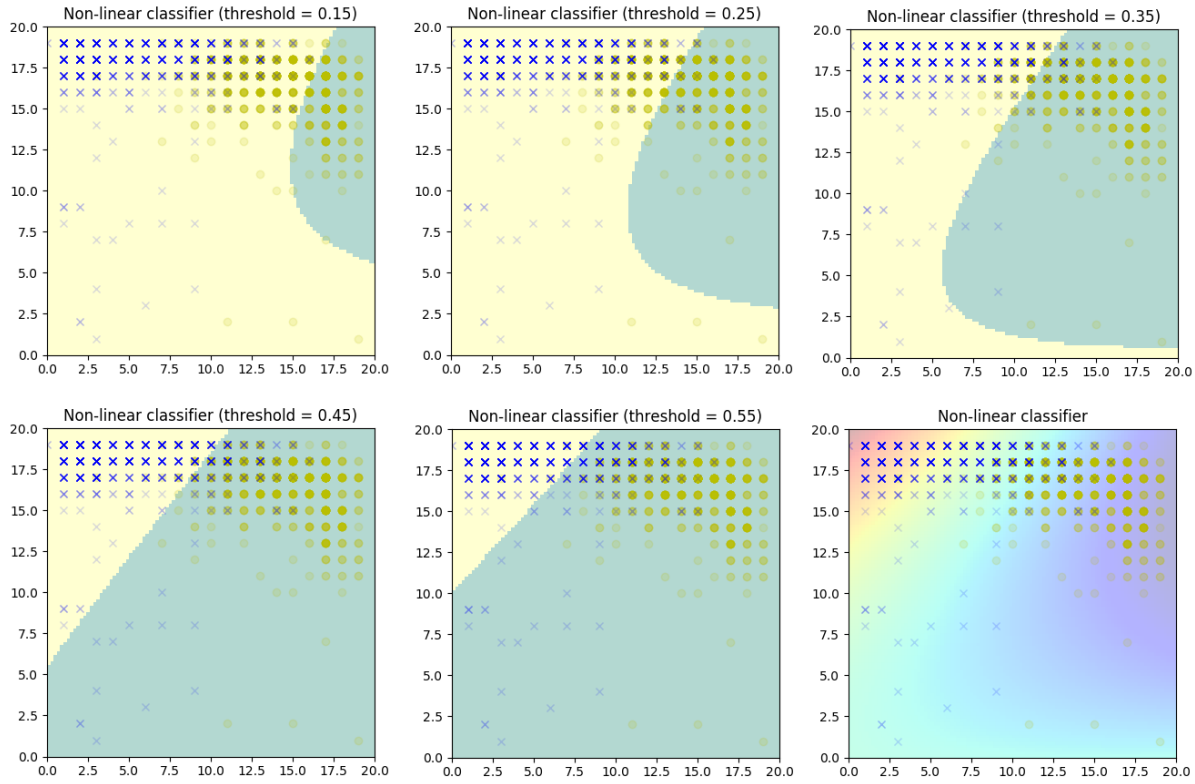


Figura 2: Clasificación no lineal con nuevas características polinómicas de grado 2 para umbrales $\{0.15, 0.25, 0.35, 0.45, 0.55\}$. En la última gráfica se muestra el valor de la función de decisión sobre el conjunto de datos. [Fuente: Original de A. Cuesta]

La transformación polinómica puede causar dos problemas importantes:

- El número de características crece factorialmente (¡más rápido que la exponencial!).
- Aumentar el grado del polinomio produce sobreajuste.

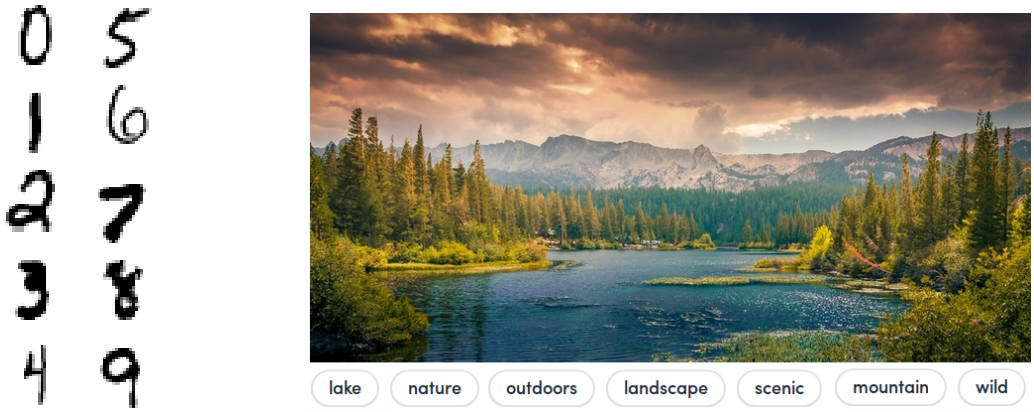
Afortunadamente la primera es tan 'grave' desde el punto de vista computacional, que suele mantener a raya la tentación de aumentar demasiado el grado. Una manera intuitiva de entender porqué grados altos producen sobreajuste es pensar que un polinomio de grado d es capaz de pasar por d puntos sin cometer ningún error. Si lo exageramos, y hacemos d suficientemente grande, entonces lograríamos pasar por todos los puntos, pero eso no es generalizar sino memorizar. En el tema de SVM veremos un 'truco' para clasificación no lineal sin necesidad de hacer transformaciones, aunque sólo válido para esa técnica, mientras que lo explicado arriba es válido en general.

2. Problemas multiclase y multietiqueta

Hasta ahora, sólo hemos visto ejemplos de clasificación, lineal y no lineal, binaria. Es decir cada ejemplo podía pertenecer sólo a una de dos clases posibles, cada una de ellas con una etiqueta. Pero esta tarea se puede generalizar de dos maneras.

Por un lado puede haber más de dos clases posibles. Por ejemplo el conjunto MNIST completo tiene imágenes de 10 clases diferentes, cada una de ellas etiquetada por la cifra que le corresponde.

También puede ocurrir que para un mismo ejemplo haya varias etiquetas posibles. Por ejemplo en una imagen de un paisahe puede aparecer un lago, un bosque, una montaña, etc. Este tipo de problemas hoy en día está muy en boga debido al auge de las redes de detección.



(a) (b)

Figura 3: (a) Problema multiclase (b) Problema multietiqueta [Fuente: Original de A. Cuesta]

2.1. Representación *one-hot* de las etiquetas

La representación *one-hot* presenta todas las etiquetas posibles como columnas en la tabla de datos, y codifica con un 1 aquella etiqueta que se asocia con el ejemplo de la fila. De este modo se pueden representar ambos problemas. En el multiclase habrá un único 1, mientras que el multietiqueta puede tener mas de uno. El resto será siempre 0, aunque normalmente no se escribe, dando lugar a una representación dispersa (*sparse*), que es mucho más eficiente desde el punto de vista computacional que tener una matriz llena de ceros con sólo unos pocos unos.

La Figura 4 muestra un ejemplo de representación *one-hot* para cada problema.

	0	1	2	3	4	5	6	7	8	9
imagen MNIST de un 3				1						

(a)

	river	lake	sea	ocean	indoor	outdoor	landscape	skyline	mountain	city
Figura 3(b)		1				1	1		1	

(b)

Figura 4: (a) *one-hot* multiclase (b) *one-hot* multietiqueta. [Fuente: Original de A. Cuesta]

2.2. El problema multiclase

En el problema hay K clases distintas y el clasificador asigna una única etiqueta $\hat{t} \in \{t_1, t_2, \ldots, t_K\}$ a cada ejemplo. Para algunos algoritmos esto supone un problema porque se han diseñado para clasificación binaria. En ese caso, hay dos posibles estrategias: 'Uno contra todos' (*One vs All*, OvA) y 'Uno contra uno' (*One vs One* OvO)

Uno contra todos (OvA) Tomando el MNIST como conjunto de datos, la estrategia OvA implica entrenar un clasificador para aprender el '0' frente al resto, 'no-0'. Después entrenar otro clasificador para aprender el '1' frente al resto, 'no-1'; y así sucesivamente hasta el '9'. Finalmente, para estimar la etiqueta de un nuevo ejemplo, este se presenta a los 10 clasificadores y se asigna la etiqueta que obtenga la máxima puntación (por ejemplo la máxima probabilidad).

En general, para K clases distintas hay que entrenar K clasificadores. Pero normalmente el conjunto de ejemplos 'negativos' suele ser mucho mayor que el de positivos, lo que provoca un problema de clases desequilibradas por un lado, y lo hace menos apropiado para algoritmos que escalan mal con el tamaño del conjunto de entrenamiento como SVM.

Uno contra uno (OvO) Tomando el mismo ejemplo, con la estrategia OvO habría que entrenar un clasificador binario con ejemplos de '0' frente a '1', luego de '0' frente a '2', luego de '0' frente a '3', y así sucesivamente hasta '8' frente a '9'. Para estimar la etiqueta de un nuevo ejemplo, este se presenta a los 45 clasificadores y se asigna la etiqueta más frecuente (a veces también se dice 'más votada').

En general, para K clases distintas hay que entrenar $K(K-1)/2$ clasificadores, muchos más que en la estrategia OvA. Como contrapartida, el conjunto de entrenamiento se reduce a aquellos ejemplos que pertenecen a las dos clases involucradas, lo que es preferible para SVM. Por otro lado, con la capacidad de paralelización actual, esta estrategia puede ser acelerada enormemente.

Ambas estrategias pueden presentar problemas de clasificación. En la Figura 5 se muestran en verde las regiones que producen una clasificación ambigua.

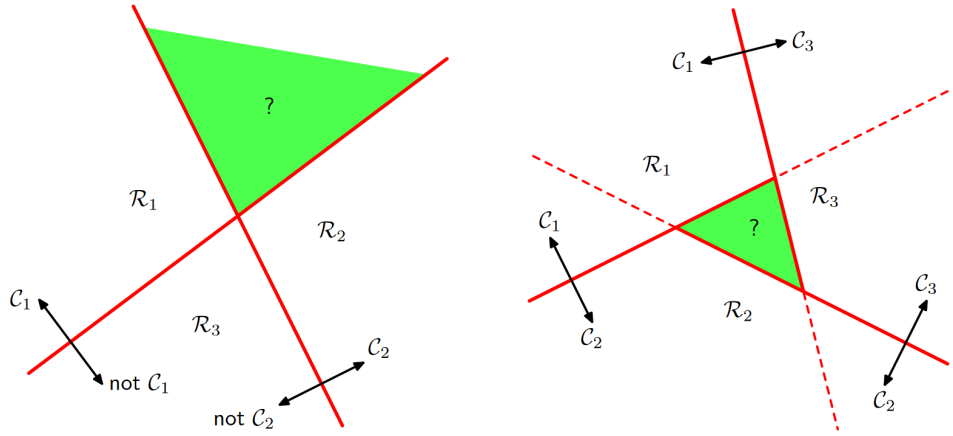


Figura 5: La región verde produce problemas de clasificación con estrategía OvA (Izq.), y con la estrategia OvO (Der.) [Fuente: Original de "Pattern Recognition and Machine Learning", C.M. Bishop, 2006]

Regresión Softmax Una solución alternativa para problemas multiclase consiste en generalizar el método de regresión logística utilizando la función *Softmax*.

Ya vimos que la función softmax asignaba una probabilidad de pertenencia $\hat{p}$ a una clase (y por tanto la probabilidad de pertenecer a la otra es $1-\hat{p}$),

$$\hat{p} = S(\mathbf{w}^T\mathbf{x}).$$

Si en vez de dos clases tenemos K, la probabilidad de pertenencia a la clase k se obtiene con la función softmax generalizada:

$$\hat{p}_k = \frac{\exp\left(\mathbf{w}_k^T\mathbf{x}\right)}{\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^T\mathbf{x}\right)},$$

donde ahora cada clase tiene su propio vector de pesos $\mathbf{w}_k$, y que podemos agrupar todos los vectores en una matriz $\mathbf{W}$ de K columnas.

La etiqueta que se asigna entonces es aquella que maximiza la probablilidad, o equivalentemente la que maximiza el *score* $\mathbf{w}_k^T\mathbf{x}$

$$\hat{t} = \arg\max_k(\hat{p}_k) = \arg\max_k \left(\frac{\exp\left(\mathbf{w}_k^T\mathbf{x}\right)}{\sum_{k=1}^{K} \exp\left(\mathbf{w}_k^T\mathbf{x}\right)} \right) = \arg\max_k \left(\mathbf{w}_k^T\mathbf{x}\right).$$

La regresión softmax permite tratar todas las clases al mismo tiempo. Para ello también se debe generalizar la función de coste *log-loss* que se utilizó en regresión logística; de modo que ahora:

$$L(\mathbf{W}, \mathbf{X}, \mathbf{t}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} t_k^{(i)} \log\left(\hat{p}_k^{(i)}\right),$$

donde $t_k^{(i)} = 1$ sólo si la clase del ejemplo i-esimo es k; lo cual es directo si se emplea la representación *one-hot*. Esta función de coste generalizada se denomina **Entropía cruzada**, y mide el grado de solapamiento de una función de probabilidad estimada respecto de la real, con la única condición de que ambas sean discretas[1]. El término 'entropía' se explicará más adelante, cuando estudiemos árboles de decisión; pero no es lo mismo que la entropía cruzada.

Este método es igualmente válido si se utiliza un *score* diferente. Sea $f_{\text{score}}(\mathbf{w}, \mathbf{x}, k) = s_k$, la función que mide el *score* de unos parámetros $\mathbf{w}$ con unos datos $\mathbf{x}$ de la clase k; y s_k el *score* obtenido, entonces simplemente

$$\hat{p}_k = S(s_k) = \frac{\exp\left(s_k\right)}{\sum_{k=1}^{K} \exp\left(s_k\right)},$$

y así, tanto la función de coste L como la asignación de la etiqueta $\hat{t}$ son igualmente válidas.

2.3. El problema multietiqueta

El modo de tratar este problema es sencillamente entrenar un clasificador para cada una de las posibles etiquetas. Si recurrimos a la representación *one-hot* y al ejemplo de la Figura 3(b), para el mismo conjunto de detos deberíamos entrenar un clasificador binario para la etiqueta 'river', otro para 'lake', otro para 'sea', y así con todas las que haya.

Evidentemente el problema multiclase se puede abordar de la misma manera, pero las otras estrategias estudiadas son mucho más eficaces.

2.4. Analisis de errores

La matriz de confusión se construye exactamente igual que para clasificación binaria, aunque ahora habrá más filas y columnas.

[1] Si fueran continuas los sumatorios serían integrales. Cuando estudiemos modelos generativos repasaremos conceptos fundamentales de probabilidad. En este punto es suficiente con concer la función de coste y su nombre, para poder usar SciKit-Learn

3. Variantes del descenso de gradiente

El descenso del gradiente es una regla de actualización de pesos en la que el nuevo vector de pesos es igual al anterior menos una cierta cantidad que viene determinada por la dirección y módulo del gradiente de la función de coste. Formalmente escribíamos:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla J(\mathbf{w}; \mathbf{X}, \mathbf{t})$$

El parámetro η es el ratio o paso de aprendizaje y, en la semana pasada veíamos el efecto que produce en la búsqueda del óptimo. Aunque no es posible dar una norma general para elegirlo, hay estrategias para hacerlo variar desde una valor grande a un valor pequeño, de manera que al comienzo los saltos sean muy grandes de manera que, si hay varios mínimos o mesetas, explore varias regiones, y poco a poco se centre. Una de estas estrategias es el **templado simulado**(*simmulated annealing*), llamado así porque imita el proceso de templado de metales. Para que un metal no tenga imperfecciones internas, se sube su temperatura hasta la fundición y después se enfría muy despacio. De manera similar, en el templado simulado se asigna un valor η muy alto y después se disminuye muy lentamente.

La disminución de η de una iteración a la siguiente se produce de acuerdo con una **curva de enfriamiento** o regla de aprendizaje (*learning schedule*). Si η' representa nuevo valor para la siguiente iteración, y η el valor actual, entonces dos reglas clásicas son:

- 'Lineal': η' es igual a la actual menos N, es decir $\eta' = \eta - N$.
- 'Fraccional': η' es igual una fracción N de la actual, es decir $\eta' = \eta / N$.

Normalmente se fija el intervalo de η que deseamos barrer con la regla de aprendizaje y el número de iteraciones, y con ello obtenemos el valor de N.

Por otro lado, a la hora de implementar el descenso del gradiente también hay algunas variantes que conviene conocer para optimizar el rendimiento del sistema ML. Para explicarlo vamos a utilizar $J(\mathbf{w}; \mathbf{X}, \mathbf{t}) = \text{MSE}(\mathbf{w}; \mathbf{W}, \mathbf{t})$. Entonces

$$\nabla J = \frac{2}{m} \mathbf{X}^T \left(\mathbf{X}\mathbf{w} - \mathbf{t} \right).$$

Esto significa que podemos actualizar los pesos utilizando todos los ejemplos en cada iteración, en vez de tener que presentarlos uno a uno. Pero para ello se necesita trasponer y multiplicar la matriz $\mathbf{X}$ por si misma, y es de suponer que será muy grande.

- **Descenso del gradiente estocástico (SGD)** Una estrategia para ir más rápido es calcular el gradiente utilizando un sólo ejemplo, elegido aleatoriamente, en cada iteración.
- **Descenso del gradiente en mini-lotes** Otra estrategia es no calcular el gradiente sobre todo el lote (conjunto) de datos sino sobre un subconjunto (mini-lote) que tiene un tamaño mucho menor, en cada iteración.

Ambas estrategias son subóptimas respecto del descenso del gradiente utilizando todos los datos. Por este motivo es necesario ejecutar muchas más iteraciones. Se denomina **época** (*epoch*) a cada vez que el algoritmo se ha ejecutado sobre el conjunto de datos.

Por ejemplo, si el conjunto de entrenamiento constara de 1000 ejemplos y utilizamos una estrategia de mini-lotes de 20 ejemplos, entonces necesitaríamos $1000/20 = 50$ iteraciones para haber ejecutado el algoritmo sobre todo el conjunto de datos, con lo que habríamos cubrido 1 época. La recomendación es lanzar el algoritmo varias épocas. Aunque puede parecer que no hay una ventaja computacional real, estas dos estrategias se pueden paralelizar y ejecutar en sistemas con muy poca capacidad de cálculo.

4. El potencial generalizador de un modelo

Cuando estimamos los parámetros del modelo normalmente encontraremos un valor óptimo o subóptimo para el conjunto de datos que tenemos, pero ¿cómo podemos evaluar el potencial generalizados del modelo obtenido? En otras palabras, supongamos que hubiera un conjunto de parámetros 'absolutamente' óptimo, mejor que cualquier otro conjunto, representado por θ^*. Entonces, para cualquier otro conjunto de parámetros óptimo obtenido como resultado de un aprendizaje sobre un conjunto de datos, $\hat{\theta}$, habría una discrepancia que se puede medir, por ejemplo $e = (\hat{\theta} - \theta^*)^2$.

Puesto que el conjunto de datos de entrenamiento no contiene todos los datos posibles, podemos suponer que existe una distribución $p(\hat{\theta})$ para la cual podemos calcular el valor esperado $\overline{\theta} = \mathbb{E}(\hat{\theta})$. El valor esperado de e sería [2] entonces:

$$\begin{aligned}
\mathbb{E}\left[(\hat{\theta} - \theta^*)^2\right] &= \mathbb{E}\left[(\hat{\theta} - \overline{\theta} + \overline{\theta} - \theta^*)^2\right] \\
&= \mathbb{E}\left[(\hat{\theta} - \overline{\theta})^2\right] + 2(\overline{\theta} - \theta^*)\mathbb{E}\left[\hat{\theta} - \overline{\theta}\right] + (\overline{\theta} - \theta^*)^2 \\
&= \mathbb{E}\left[(\hat{\theta} - \overline{\theta})^2\right] + (\overline{\theta} - \theta^*)^2 \\
&= \text{variance}(\hat{\theta}) + \text{bias}^2(\hat{\theta})
\end{aligned}$$

Es decir, el error que comete un modelo al generalizar se puede expresar como la suma de tres errores diferentes:

- Desvío (*Bias*)

  Se debe a hipotesis de partida erroneas. Por ejemplo asumir que el modelo es lineal o que los datos tienen una distribución normal.

  Generalmente, un modelo con un error por sesgo alto tiende a subajustar, es decir generaliza demasiado. Por tanto se produce con modelos muy sencillos.

  ¡ No hay que confundir este *bias* con el término independiente del modelo lineal !

- Varianza (*Variance*)

  Se debe a un exceso de sensibilidad del modelo a pequeñas variaciones en el conjunto de entrenamiento. Por ejemplo, si estamos haciendo validación cruzada y clasificación no lineal con polinomios de orden muy alto, nos ajustaremos demasiado a los datos del subconjunto de entrenamiento. Cuando probemos otro subconjunto, las pequeñas diferencias provocaran grandes diferencias en el clasificador.

  Obviamente, este error tiende a sobreajustar, por lo que se suele producir en modelos demasiado complejos.

- No eliminable (*Irreducible*)

  Se debe al ruido propio de los datos. El único modo de reducir este error es 'limpiarlos' o mejorar la fuente de los datos. En cualquier caso no depende del modelo, y por eso no aparece en la expresión de arriba.

El compromiso entre un modelo sencillo y uno complicado, o lo que es lo mismo entre el error de Sesgo y el de Varianza, se conoce en inglés como *bias-variance tradeoff*.

[2] La siguiente derivación se ha incluido simplemente a título explicativo. Más adelante entratemos en conceptos de probabilidad donde recordaremos la función valor esperado y sus propiedades. Para aclarar el paso de la ecuación 2^a a la 3^a hay que recordar que $\overline{\theta} = \mathbb{E}(\hat{\theta})$, es decir que ya no tiene variaciones porque es un 'valor medio'. Por tanto $\mathbb{E}\left[\hat{\theta} - \overline{\theta}\right] = \mathbb{E}\left[\hat{\theta}\right] - \overline{\theta} = \overline{\theta} - \overline{\theta} = 0$

Índice alfabético