2.pdf

destherhada

Inteligencia Artificial II

3º Grado en Ingeniería Informática

Facultad de Informática
Universidad Complutense de Madrid

Contamos con una matriz de datos de n individuos, seguimos distinguiendo entre dos tipos de variables. Un conjunto de m variables de entrada x, y una variable de salida y. Si la variable de salida se trata de una numérica es un *problema de regresión*. Si se trata de una variable categórica es un *problema de clasificación.* El objetivo es ajustar un modelo que relacione las variables de entrada con la de salida, de forma que podamos predecir la respuesta lo mejor posible y entendamos la relación entre variables de entrada y salida.

Debemos comparar los resultados pronosticados por nuestro modelo con los resultados obtenidos. En problemas de regresión se suele utilizar el Error Cuadrático Medio o su raíz cuadrada.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Como medida de error a optimizar/reducir, el RMSE es equivalente al MSE, pero el RMSE es más inteligible ya que está en las mismas unidades de medida que la variable de salida

En problemas de clasificación podemos visualizar el error mediante la matriz de confusión, que para el caso de una clase binaria sería:

		Clase observada	
		1	0
Clase pronosticada	1	*Verdaderos Positivos*	*Falsos Positivos*
	0	*Falsos Negativos*	*Verdaderos Negativos*

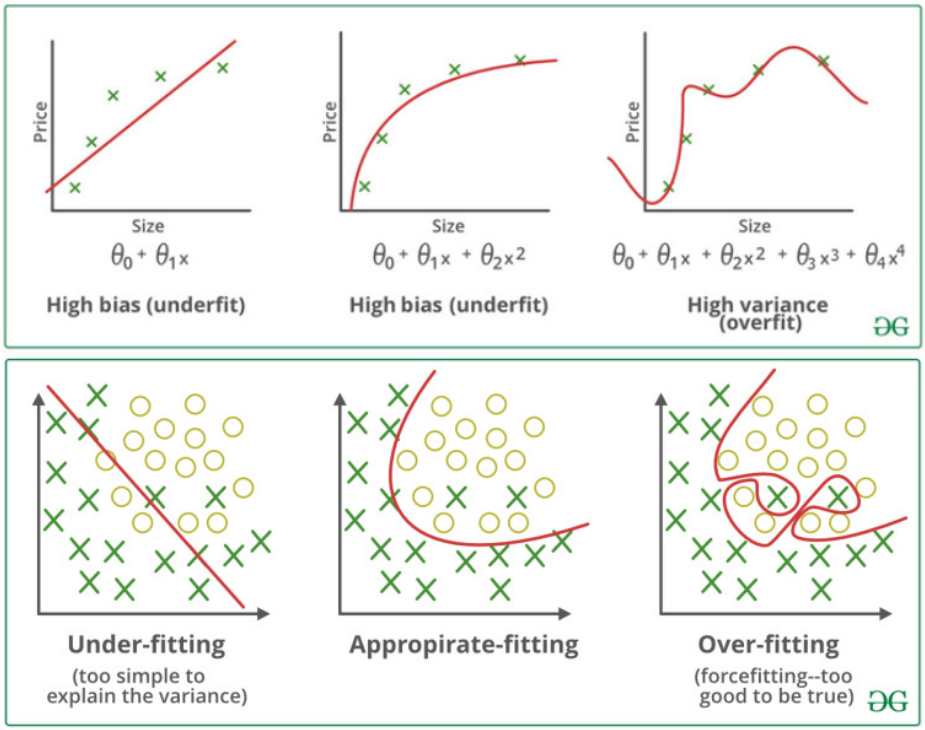
$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$$

Necesitamos medidas que reflejan otros aspectos de la clasificación (por clase):

- Tasa de Verdaderos Positivos (Recall/sensitivity) = VP / VP + FP
- Tasa de Verdaderos Negativos (specificity) = VN / VN + FP
- Valor Predictivo Positivo (precisión) = VP / VP + FP

Podemos obtener clasificadores de muy distinto comportamiento, las medidas de rendimiento nos dicen cómo funcionan. En la fase de aprendizaje, podemos indicar qué medida queremos que optimice el clasificador en su aprendizaje.

Para medir la bondad de nuestra solución solamente contamos con nuestro conjunto de datos y debemos usarlo bien. Es posible que nuestro modelo aprenda nuestros datos perfectamente, minimizando el error a cero. En ese caso, nuestro modelo no será muy útil porque ha aprendido demasiado y seguramente ruido. Existen mecanismos para evitar este problema llamado sobreaprendizaje.



La validación de nuestro modelo es una forma de estimar cómo de bueno es nuestro modelo ante datos nuevos. La validación simple consiste en partir nuestro conjunto de datos en dos (entrenamiento ¾ y test ¼). Si el modelo está entrenado correctamente su error en el conjunto de entrenamiento no debe ser muy diferente que en el conjunto test. Esto querrá decir que no ha sobre-aprendido y que el modelo generaliza bien con datos nuevos.

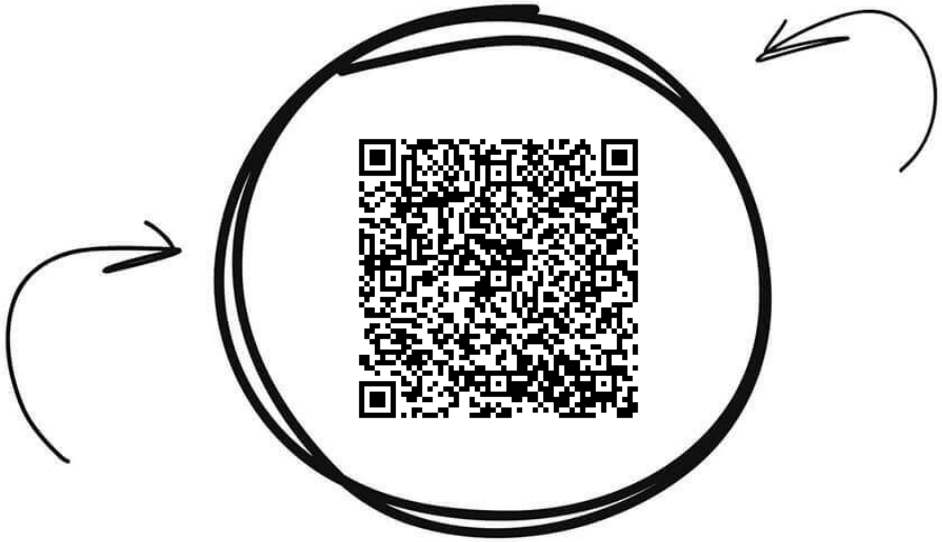
Si tenemos varios modelos y queremos elegir el mejor entre ellos, se opta por partir los datos en tres conjuntos (entrenamiento, validación y test). El tercer conjunto se hace para no sobreaprender ya que las dos primeras particiones las usamos para entrenar y elegir el mejor modelo.

Existen estrategias más sofisticadas para realizar el entrenamiento y validación de un modelo:

LA AUTOESCUELA EN TU UNIVERSIDAD

AUTOESCUELA EUROPEA

Inteligencia Artificial II



Banco de apuntes de la

WUOLAH

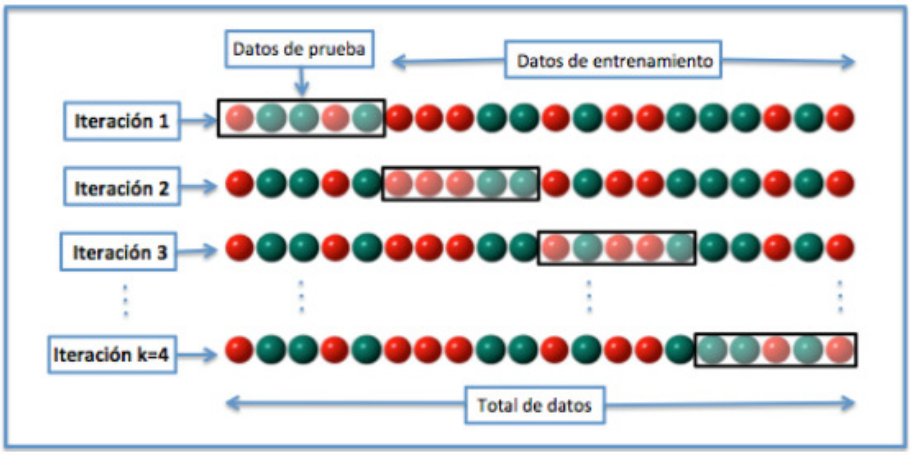
Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

-validación cruzada en k partes

Es una forma de estimar cómo de bueno es nuestro modelo ante datos nuevos. Consiste en los siguientes pasos:

1. Los datos se dividen aleatoriamente en k subconjuntos del mismo tamaño.
2. Se entrena el conjunto con (k - 1) y se valida con el restante.
3. Se repite k veces el paso 2, cambiando el conjunto que se usa para validar.
4. Como medida de error final se presenta la media de las k medidas de error de validación.



Puede ser interesante estratificar las k partes. En un problema de regresión significa que la media de la variable respuesta sea similar en todas las partes. En un problema de clasificación significa que la proporción de clases en cada parte sea similar.

Si llevamos la validación cruzada al extremo y tomamos k como el número de individuos en nuestro conjunto de datos, estaríamos utilizando la validación Leave One Out validation, es adecuada cuando tenemos pocos datos y no tiene sentido dividirlo en partes mayores.

-los k vecinos más cercanos (k-NN):

Los k vecinos más cercanos es una técnica de aprendizaje supervisado muy sencilla. No calcula ningún modelo y demora todos los cálculos hasta el momento en que se le presenta un ejemplo nuevo. Usa todos los individuos disponibles ante un ejemplo nuevo y recupera los más relevantes para componer la solución.

Dado un ejemplo nuevo con variable de salida desconocida, recupera los k ejemplos que más se le parezcan de los n ejemplos disponibles. Devuelve la media de la solución de los k ejemplos recuperados. Se puede también devolver la media ponderada, donde se da más peso a los vecinos según su cercanía. Si la variable de salida es categórica, devuelve el valor más frecuente entre los valores solución de los k individuos recuperados.
Aunque el k-NN no estime ningún modelo, debemos determinar el valor de k. Si es muy pequeño corre el riesgo de sobreaprender y si es muy grande corre el riesgo de generalizar demasiado. Para que no pase ni uno ni lo otro, podemos usar una estrategia de validación cruzada para encontrar el valor de k que minimiza el error de validación.

-árboles de decisión

Los árboles de decisión son una técnica de aprendizaje supervisado, pueden usarse en problemas de clasificación o de regresión. Permite el aprendizaje de conceptos de forma inductiva.

El árbol de decisión se construye recursivamente, a partir de un conjunto de ejemplos de entrenamiento. En cada nodo se pregunta por el valor de una variable que busca disminuir la **entropía** de los nodos hijos. Se selecciona aquel atributo que mayor disminución de entropía genera. La **ganancia de información** o **disminución de la entropía** es la diferencia entre la entropía del conjunto original y la de los subconjuntos obtenidos. El atributo usado en el nodo no se volverá a usar en otros niveles, se elimina del conjunto de atributos.

La terminación de un árbol de decisión ocurre cuando todos sus ejemplos pertenecen a la misma clase (la entropía es nula) o cuando no quedan más atributos para expandir nodos. Siempre existe el riesgo de sobreaprendizaje, para evitar esto, hay alternativas que no generan árboles completos como usar **mecanismos de poda** o utilizar un umbral de entropía por debajo de la cual no ramificar. Además siempre conviene evaluar el sobreaprendizaje de un árbol mediante una estrategia de validación.

El árbol construido sirve para clasificar ejemplos nuevos del conjunto de validación. Para ello se coloca el ejemplo nuevo en la raíz y respondiendo las preguntas desciende por las ramas hasta llegar a un nodo hoja.

El árbol aprendido es muy interpretable y debe ser analizado por un experto para entender la clasificación. A veces es tan grande que es difícil de interpretar.

En los árboles de decisión la entropía juega un papel muy importante, vamos a hablar de ella. La **entropía** mide la ausencia de homogeneidad de un conjunto de ejemplos con respecto a su clase. La **entropía inicial** (antes de clasificar) de un nodo se define como:

$$E(N) = -\sum_{i=1}^{p} P(s_i) \log_2 P(s_i),$$

siendo $P(s_i)$ la probabilidad de que un ejemplo tenga la clase s_i en el nodo N, y calculándose únicamente para las clases observadas en el nodo N (para evitar el cero en el logaritmo)

La entropía final de un nodo N tras usar el atributo A que tiene q valores (aj) se define como:

$$E(N|A) = \sum_{j=1}^{q} P(a_j) \left(-\sum_{i=1}^{p} P(s_i|a_j) \log_2 P(s_i|a_j) \right),$$

siendo $P(s_i|a_j)$ la probabilidad de que un ejemplo del nodo hijo con $A = a_j$ tenga la clase s_i

Ahora vamos a hablar de los diferentes **mecanismos de poda**. Hemos comentado anteriormente que la poda se realiza para simplificar el árbol y ganar en capacidad de interpretación y legibilidad, tanto como para evitar el sobreaprendizaje y mejorar la capacidad de generalización.

-Poda mediante reducción del error:

Este tipo de poda requiere un conjunto de ejemplos no usados para generar el árbol.

1. Clasifica dichos ejemplos y calcula el error que cometen todas las hojas
2. El número de errores de un subárbol es la suma de los errores de todas sus hojas
3. Calcula el número de errores que cometería el nodo que origina el subárbol si no se expandiera
4. Si los errores sin expandir son iguales o menores que expandido, se poda el subárbol
    - Se recalcula el número de errores de los subárboles que contenían dicho subárbol

– **Ventaja**: Evita el sobreajuste de forma efectiva
– **Inconveniente**: Requiere de nuevos ejemplos

-Poda pesimista:

1. Clasifica un conjunto de ejemplos (usado o no para generar el árbol)
2. Para cada nodo hoja, calcula los errores que comete y añade un valor prefijado r
3. El número de errores de un subárbol es la suma de los errores de todas sus hojas
4. Calcula el número de errores que cometería el nodo que origina el subárbol si no se expandiera e increméntalo también con r
5. Si los errores sin expandir son iguales o menores que expandido, se poda el subárbol
    - Se recalcula el número de errores de los subárboles que contenían dicho subárbol

– Ventaja: No necesita ejemplos extra
– Inconveniente: El número de elementos en la hoja no afecta

-redes neuronales:

Una red neuronal es una función capaz de representar complejas relaciones no lineales entre los datos de entrada y la variable a predecir. Las redes neuronales se basan en perceptrones multicapa.

Un perceptrón multicapa (MLP) es una red neuronal de tipo perceptrón con al menos tres capas. Tiene una **capa de entrada** con tantas neuronas como variables de entrada, una o más **capas ocultas** con un número variable de neuronas, cada una con su función de activación, y una **capa de salida** con una o más neuronas con su función de activación. Las capas ocultas son capaces de modelar relaciones no lineales entre las variables de entrada y salida.

Las **funciones de activación** son funciones no lineales que permiten modelar funciones complejas, calculan combinaciones lineales de funciones no lineales. Las funciones de activación pueden ser de distintos tipos: logística, tangente hiperbólica, rectificada lineal, identidad y softmax. Usaremos un tipo u otro depende de los rangos de las variables que estamos modelando.

- Logística: Se usa cuando la variable de salida es una probabilidad. Es una función sigmoidal que varía entre 0 y 1.

- Tangente hiperbólica: Se usa cuando la variable de salida es un número entre el -1 y el 1. Es una función sigmoidal que varía entre -1 y 1.

- Identidad: Se usa en la capa de salida para problemas de regresión no acotados.

- Rectificada linear (ReLU): Se usa cuando la variable de salida es un número positivo. Es una función con dos tramos, uno lineal que varía entre el 0 y el infinito (muy usada en Deep learning).

- Softmax: Se usa en la capa de salida en problemas de clasificación multiclase. Interpreta la probabilidad de que el ejemplo pertenezca a cada una de las n clases. Valores entre el 0 y 1 que suman 1.

A la hora de entrenar una red neuronal, el objetivo es configurar la red de forma que se minimice el error total cometido. Existe la **función de error (función de pérdida)** que calcula el error cometido por la red neuronal. Esta función de error depende de los pesos y bias de las neuronas. Dada una función de error diferenciable, podemos encontrar uno de sus mínimos locales mediante el **algoritmo de descenso de gradiente**.

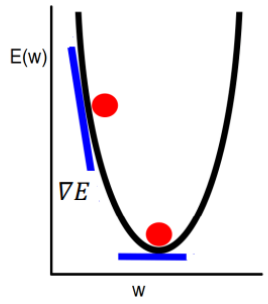
Partimos de un punto aleatorio *p* con un cierto error *E(p)*

Calculamos el gradiente de la función $\nabla E = (\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_s})$ en el punto p

- El vector gradiente indica la dirección en la que la función de error crece más rápido

Calculamos un nuevo punto p = p $-\alpha \nabla E$ y volvemos al paso anterior.

- α indica cuanto nos desplazamos y se conoce como tasa de aprendizaje (*learning rate*)



En cada paso del algoritmo disminuimos el error hasta llegar a un mínimo local donde el gradiente será 0.

Si la función no es convexa no se garantiza encontrar un mínimo global.

El **algoritmo de retropropagación** disminuye el error de la red mediante el algoritmo de descenso de gradiente. Funciona de la siguiente manera, se inicializan los pesos de forma aleatoria y con valores pequeños y para cada ejemplo k del conjunto de entrenamiento se hacen dos fases:

- Hacia delante: Calculamos la salida de la red para un ejemplo y calculamos el error entre la salida real de la red y la salida deseada para ese ejemplo.
- Hacia atrás: Calculamos el gradiente de la función de error con respecto a los pesos de la red para el ejemplo k y ajustamos los pesos para que reduzcan el error respecto al ejemplo k.

De esta forma se reduce el error con respecto al ejemplo k, pero en realidad nos interesa reducir el error total con respecto a todos los ejemplos. Así que calculamos el gradiente medio y modificamos los pesos de la red en dirección contraria. La actualización de la red para todo el conjunto de entrenamiento recibe el nombre de época.

Para evitar el sobreaprendizaje usamos **técnicas de regularización**. La que usaremos nosotros es la regularización L2. La idea de esta técnica es penalizar los pesos demasiado grandes para que la red generalize mejore. Usa una variable que permite controlar cuánto queremos regularizar.

A la hora de **configurar un perceptrón multicapa** debemos tener varias cosas en cuenta:

- Los datos de entrada deben ser numéricos, pueden ser valores sin normalizar o sin estandarizar. No pasa nada por utilizar variables irrelevantes, ya que se les acabara asignando peso cero.

- Determinar el número de capas y de neuronas puede hacerse probando varias configuraciones y determinando la mejor mediante validación cruzada. Debemos entender que un MLP con una única capa oculta suficientes neuronas puede

resolver igual un problema que un MLP con más capas ocultas. Las capas ocultas añaden representación de los datos con un nivel de abstracción amyor y en principio es más fácil llegar a la solución a partir de ellas.

- A la hora de determinar qué función de activación utilizar tendremos en cuenta el tipo de problema.

- A la hora de dar un valor al parámetro de tasa de aprendizaje, debemos tener en cuenta que cuanto más pequeño sea su valor, mejores resultados podemos encontrar, pero más tiempo tarda el entrenamiento.

- Sobre qué tipo de técnica de regularización usar ya hemos determinado anteriormente que la única que vamos a utilizar es la regularización L2.

- A la hora de usar validación cruzada nos sirve cualquier estrategia, lo importante es no usar el mismo subconjunto de datos para entrenar, configurar y evaluar.