Flujo de resolución de un problema de clasificación

UCO ONLINE

Flujo de resolución de un problema de clasificación

UNIVERSIDAD Ð CÓRDOBA

Flujo de resolución

A la hora de abordar la resolución de un problema de clasificación, cabe destacar los siguientes pasos a seguir, que pese a no ser los únicos, sí que son quizá los más importantes:

- Obtención y exploración de los datos. En primer lugar, es indispensable obtener los datos que vamos a utilizar en el proceso de clasificación. Además, habrá que determinar la variable objetivo a predecir y conocer sus valores. Por último, debemos conocer qué atributos de entrada tenemos (cuántos atributos, su tipo, significado de cada uno de ellos, etc.)
- Pre-procesado de los datos. Una vez tenemos los datos, habrá que analizarlos y pre-procesarlos cuidadosamente. En este curso no se entrará en mucha más profundidad en el pre-procesado de los datos.
- Particionado de los datos. Una vez que tenemos los datos estructurados y limpios, por lo general habrá que seleccionar un método para el particionado de los datos disponibles. El objetivo es dividir los datos en dos (o tres) conjuntos: de entrenamiento y de test (y validación si es necesario). De este modo, la evaluación del modelo de clasificación se realice sobre datos no vistos en entrenamiento. Nótese que los datos de test no deben utilizarse en ningún momento durante el entrenamiento del modelo. En caso de que el propio proceso entrenamiento del clasificador necesite de un subconjunto de datos para validar su rendimiento, se utilizará un conjunto de validación (distinto al de entrenamiento y al de test). Al final de esta lección se presentan algunos de los métodos más utilizados para realizar estas particiones, como *hold-out*, *k-fold cross-validation*, o *leave-one-out*.
- Selección de algoritmo. En este punto, seleccionaremos el algoritmo que deseamos utilizar, pese a que ya debíamos tenerlo en mente desde pasos anteriores. Habrá que considerar si solo queremos utilizar un algoritmo, o en cambio, la opción preferible, utilizar diversos algoritmos y seleccionar el que mejor se ajuste a nuestro problema. En base al algoritmo seleccionado, posiblemente haya que realizar algún pre-procesado específico para dicho algoritmo (por ejemplo, conversión de datos de tipo numérico a categórico, o viceversa, si el algoritmo en cuestión no soporta alguno de los tipos de datos).
- Selección de los hiper-parámetros del algoritmo. No todos, pero algunos algoritmos necesitan de la selección de uno o varios hiper-parámetros para ajustar su funcionamiento a nuestro problema. En este caso, una de las opciones preferibles será seleccionar varias

UCO ONLINE

combinaciones de valores para los hiper-parámetros de nuestro algoritmo, para así posteriormente seleccionar la mejor de ellas. En este proceso, se suele utilizar el subconjunto de validación.

- Evaluar el rendimiento del clasificador sobre el conjunto de test. Para tener una aproximación del rendimiento del clasificador, lo ideal es evaluarlo (en base a cualquiera de las métricas o métodos de evaluación que se verán en el futuro) sobre datos que no fueron utilizados para entrenar el modelo, así evitando el sobre-entrenamiento (*over-fitting*). Además, idealmente este proceso debería repetirse utilizando distintos subconjuntos de entrenamiento y test, para así evitar que los resultados estén sesgados por un único subconjunto dado.
- Por último, seleccionar aquel método (y combinación de hiper-parámetros, en su caso) que reporta un mejor funcionamiento en nuestro caso, y utilizarlo en producción; es decir, para predecir nuevos datos que nos vayan llegando.

Evaluación de modelos

Tal y como se definió en la lección anterior, el objetivo final en la construcción de un clasificador óptimo Ω a partir de un conjunto de datos S es reducir el error de generalización ε. Dicho error se define como el ratio de clasificaciones erróneas sobre la distribución D, o la probabilidad de clasificar erróneamente una instancia seleccionada de acuerdo con dicha distribución D. El error de generalización se define en la siguiente ecuación.

$$\varepsilon(\Omega(S), D) = \sum D(x, y) \cdot L\big(y, \Omega(S)(x)\big)$$

$$L\big(y, \Omega(S)(x)\big) = \begin{cases} 0 \; si \; y = \Omega(S)(x) \\ 1 \; si \; y \neq \Omega(S)(x) \end{cases}$$

Sin embargo, este valor se puede conocer en muy pocos casos (principalmente, en conjuntos de datos sintéticos), ya que la distribución D del espacio de salida no se conoce. Se podría utilizar el error de entrenamiento como un estimador del error de generalización, pero usualmente el error de entrenamiento está sesgado de manera optimista, especialmente en los casos de sobre-entrenamiento de los datos de entrenamiento.

Para estimar dicho error, la mejor práctica es dividir el conjunto de datos en particiones disjuntas de entrenamiento y de test. De este modo, los datos de entrenamiento serán los que se utilicen

para entrenar el clasificador, y los de test para evaluarlo y calcular el error de dicho modelo, proveyendo una mejor estimación del error de generalización, ya que estos datos no están sujetos al sobre-entrenamiento del clasificador (son instancias de los datos no vistas anteriormente). En algunos casos, será necesario obtener otra partición de validación para evaluar el clasificador en el propio proceso de entrenamiento y así saber cuándo detener el entrenamiento del clasificador. Dicha partición de validación puede considerarse parte de los datos de entrenamiento, y en los siguientes ejemplos no se contempla; bastaría a partir de los datos de entrenamiento siguiendo un proceso similar a los que se muestran.

En la Figura 1 se representan los términos de *underfitting* (o infra-entrenamiento), *overfitting* (o sobre-entrenamiento), y el entrenamiento correcto o ideal. A la izquierda se observa un modelo demasiado simple, que no es capaz de separar los datos de entrenamiento correctamente, por lo que se dice que está infra-entrenado. Se puede observar cómo en este caso los errores tanto de entrenamiento como de validación/test son similares y muy altos. En la parte central se muestra un modelo que sería correcto; pese a que comete algunos errores en entrenamiento, es capaz de capturar la distribución general de los datos. En este caso, el error de entrenamiento es menor que el de validación, pero no muy diferentes. Por último, en la parte derecha se muestra un modelo sobre-entrenado. El error en entrenamiento de este modelo es muy bajo, ya que es capaz de clasificar correctamente todos los datos de entrenamiento, pero hace que posteriormente el error de validación o test crezca cuando se le presentan datos nuevos, por lo que un modelo más simple parece mejor opción.

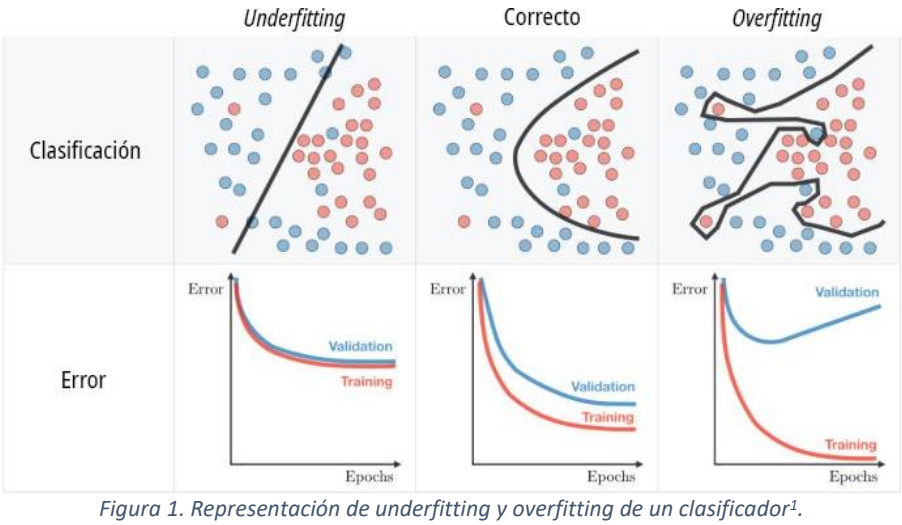

Figura 1. Representación de underfitting y overfitting de un clasificador[1].

[1] Fuente: https://www.kaggle.com/getting-started/166897

Pese a que el error de clasificación es quizá el criterio de evaluación más natural y extendido, no es el único criterio que puede o debe utilizarse a la hora de decantarse por uno u otro clasificador. La complejidad computacional del modelo, es decir, la cantidad de tiempo necesaria para generar el clasificador, actualizarlo con nuevos datos de entrada, o producir salidas para una nueva instancia de los datos, puede ser un factor a tener en cuenta también. Por último, la comprensibilidad o interpretabilidad del clasificador generado, es decir, qué facilidad tendrán los humanos para entender el clasificador entrenado, es también importante. En este punto cabe destacar la diferencia entre los modelos de caja blanca, que no solo se centran en obtener un clasificador con buen rendimiento, sino que también sea interpretable y fácil de conocer por qué se obtiene la predicción; y los modelos de caja negra, que obvian por completo la interpretabilidad del mismo, generalmente a costa de obtener un mejor rendimiento.

Tipos de particionado de datos

En esta sección se muestran algunos de los tipos de particionado de datos en conjuntos disjuntos más utilizados en la literatura: *hold-out*, *k-fold cross-validation*, y *leave-one-out*.

Particionado de datos en *hold-out*

El particionado del conjunto de datos disponible en subconjuntos de entrenamiento y test mediante el método de *hold-out* implica la selección de un porcentaje aleatorio de las instancias para entrenamiento, y el resto para test. En muchos trabajos, un porcentaje aceptable para entrenamiento es el 80% de las instancias, mientras que el 20% restante se utiliza para test; sin embargo, estos porcentajes podrían variar si así se considera necesario. En la Figura 2 se muestra un ejemplo del particionado, donde dado el conjunto de datos completo, los datos de entrenamiento se muestran en azul, y los de test en rojo.

Figura 2. Partición simple en hold-out (80%-20%).

Para evitar que los valores de evaluación de rendimiento no estén sesgados por los datos escogidos en el conjunto de test, se suele repetir dichas particiones, cada vez escogiendo un 80% de instancias aleatorias para entrenamiento y el restante para test. Posteriormente, por lo general los resultados de las métricas de evaluación se promedian entre todos los obtenidos de las distintas particiones, teniendo una aproximación del rendimiento global del clasificador. Sin embargo, en este caso no tenemos control del número de veces que cada instancia aparece en el subconjunto de

test (algunas instancias podrían no aparecer, mientras que otras podrían aparecer muy frecuentemente).

Particionado de datos en *k-fold cross-validation*

La partición de los datos en *k-fold* significa que los datos se dividen en *k* subconjuntos disjuntos de similar tamaño, siendo 5 o 10 valores típicos para dicho *k*, pero que pueden depender de cada caso específico. Posteriormente, de esos *k* subgrupos, *k-1* se utilizan para entrenar el modelo, y el subconjunto restante como partición de test, repitiendo el proceso hasta que todas las particiones se han utilizado en una ocasión para testear el modelo (lo que le da el nombre de *cross-validation*, o validación cruzada). En la Figura 3 se muestra un ejemplo de partición en *5-fold cross-validation*; en cada caso la partición de test (en rojo) es distinta, mientras que el resto de las particiones (en azul) se utilizan en conjunto para el entrenamiento del modelo.

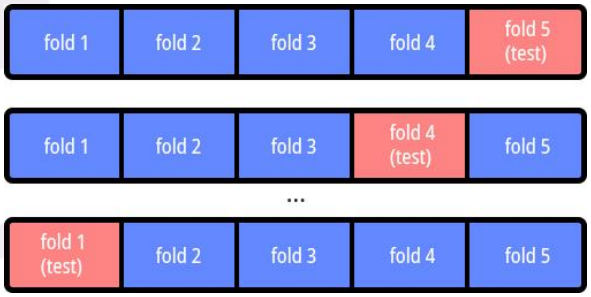

Figura 3. Partición de los datos en 5-fold cross-validation.

Igual que al repetir varias veces un *hold-out*, al tener varias ejecuciones en este caso también habrá que promediar (por lo general) los valores de las métricas obtenidas en las particiones de test para obtener una aproximación del rendimiento global. Además, dado que los resultados pueden estar sesgados por las particiones iniciales realizadas, es común repetir el proceso de *k-fold cross-validation* varias veces, y promediando los resultados en todos los casos.

Al realizar un *k-fold cross-validation* sí que nos aseguramos de que todos los datos se utilizan el mismo número de veces (si no se repite el proceso de particionado, una única vez), no sesgando tanto la estimación del rendimiento por los datos seleccionados en este subconjunto. Obviamente, el sesgo será menor si el proceso se repite varias veces con distintas particiones. Por lo general, este proceso es preferible al *hold-out*, aunque *hold-out* también es aceptable si el tamaño de los datos es muy grande o estamos trabajando con modelos preliminares.

Particionado de datos *leave-one-out*

Cuando el tamaño de los datos es muy pequeño, utilizar un subconjunto del mismo aún más pequeño para entrenar el modelo puede hacer que el clasificador no sea lo preciso que debiera, dado que no tiene la suficiente información para entrenarse. En estos casos, el proceso de ç validación cruzada *leave-one-out*, donde se utiliza una única instancia de los datos para evaluar el modelo y el resto para entrenarlo, puede ser útil. En la Figura 4 se muestra un ejemplo del particionado *leave-one-out*; en cada iteración, una única instancia (en rojo) se utiliza para validar el modelo, mientras que el resto (en azul) se utilizan para entrenar.

Figura 4. Particionado leave-one-out.

Pese a la ventaja de poder utilizar el máximo posible de datos en entrenamiento, uno de los inconvenientes es el coste computacional, dependiendo del tamaño de los datos. En este caso, habrá que entrenar el modelo tantas veces como instancias existan en los datos (en cada caso, una instancia será utilizada como test y el resto para entrenar), lo que puede ser prohibitivo en muchos problemas. Por otro lado, en ocasiones también puede caer en sobre-entrenamiento (si se utiliza teniendo un gran número de datos de entrenamiento), al solo tener un único dato sobre el que testear. Por lo general, el método *k-fold cross-validation* es preferible, si tenemos los suficientes datos.

Referencias

[Agg15] Aggarwal, C. C. (2015). Data Classification. Algorithms and Applications. *Chapman and Hall/CRC*.

[Lar14] Larose, D. T., & Larose, C. D. (2014). *Discovering knowledge in data: an introduction to data mining* (Vol. 4). John Wiley & Sons.

[LeC98] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

[LeC10] LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database. Disponible en: http://yann.lecun.com/exdb/mnist/. Última visita: 23/08/2021.

[Mai10] Maimon, O., & Rokach, L. (Eds.). (2010). Data mining and knowledge discovery handbook, 2nd edition. *Springer*.

[Wit11] Witten, I. H., Frank, E., & Hall, M. A. (2011). Data mining: practical machine learning tools and techniques, 3rd edition. *Morgan Kaufmann*.