Aprendizaje Automático (2014-2015)

Grado en Ingeniería Informática

Universidad de Granada

Cuestionario T3

Mª Cristina Heredia Gómez

31 de mayo de 2015

Índice

Índice de figuras

1. Bootstrap. Suponemos que extraemos una muestra bootstrap de un conjunto de n observaciones.

1.1. ¿Cuál es la probabilidad que la primera extracción de un muestreo por bootstrap no sea la j-ésima observación de la muestra? Justificar la respuesta.

Todas las observaciones tienen la misma probabilidad de salir, que es P=$\frac{1}{n}$, donde n= número de observaciones. Luego la probabilidad que tiene de no salir es 1-P=1 - $\frac{1}{n}$

1.1.1. ¿Cuál es la probabilidad de que la segunda extracción no sea la j-ésima observación de la muestra original?

En la segunda extracción, las observaciones que quedan, tienen, cada una, una probabilidad P=$\frac{1}{n}$ de salir. Por lo tanto, la probabilidad que tiene cada una de no salir es 1-P=1 - $\frac{1}{n}$.

1.2. Mostrar que la probabilidad de que la j-ésima observación no esté en una muestra bootstrap de tamaño n es (1 - $\frac{1}{n}$)^n

Ya dijimos antes que la probabilidad de que una observación **no** esté en el conjunto de booststrap, es:
1 - P=1 - $\frac{1}{n}$
Luego la probabilidad de que dicha obvervación no sea ninguno de los n-elementos del conjunto de booststrap, es el producto de la probabilidad de que no sea la primera observación*la probabilidad de que no sea la segunda*la probabilidad de que no sea la tercera... es decir : (1 - $\frac{1}{n}$) · (1 - $\frac{1}{n}$) · (1 - $\frac{1}{n}$) ·...· (1 - $\frac{1}{n}$) , n-veces.
y eso es (1-$\frac{1}{n}$)^n.

1.3. Con n=5 ¿Cuál es la probabilidad de que la j-ésima observación este en la muestra bootstrap?

Sabemos, por el apartado anterior, que la probabilidad de que la observación **no** esté en el conjunto booststrap es **(1 -** $\frac{1}{n}$**)**^{n}.
Luego, la probabilidad de que sí esté en el conjunto de booststrap es: 1 - [(1-$\frac{1}{n}$)^n]

Si n=5 $\Rightarrow$ P=1 - [(1 - $\frac{1}{5}$)^5]= 1-0.32768=**0.67232**.

1.4. Con n=100, ¿Cuál es la probabilidad de que la j-ésima observación este en la muestra bootstrap?

Procedemos como en el apartado anterior, cambiando el valor de n:
Si n=100 $\Rightarrow$ P=1 - [(1 - $\frac{1}{100}$)^{100}]= 1-0.366032341=**0.63397**.

1.5. Aproximar dicha probabilidad para tamaños muestrales muy grandes $(n > 10^6)$

por ejemplo, n=10^7

Si n=$10^7 \Rightarrow$ P=1 - $\left[\left(1 - \frac{1}{10^7} \right)^{10^7} \right]$= 1-0.9999...$\approx$ **0**.

1.6. Comentar si la probabilidad tiende a 1 cuando crece el tamaño de la muestra o sigue otra conducta

Al parecer, cuando n es muy grande, al dividir 1 por un muy número grande tenemos un número muy muy pequeño(llamémoslo " y") tal que al restar, nos da practicamente 1:
(1-y)$\approx$ 1.
por lo que al hacer ($\approx$ 1)^n nos da 1, aproximadamente, pues cualquier potencia de 1 vale 1. Por lo tanto, al restar 1-($\approx$ 1) tendremos, que la probabilidad de que esté la observación presente es $\approx$ 0.
Dicho de otra forma, a mayor tamaño de la muestra, menos probabilidad tiene la obvervación j-ésima de aparecer en el conjunto de booststrap.

2. Suponga que dispone de una muestra i.i.d para estudiar la predicción del valor de una variable Y para un valor dado del predictor X. Suponga que elige al azar uno de los métodos estudiados. ¿Cómo podríamos estimar la desviación típica de nuestra predicción? Dar todos los detalles de cada paso.

Para estimar el $SE(\hat{}\alpha)$ usaremos **booststrapping** ya que nos permitirá obtener "N"nuevas muestras a partir de la que tenemos, mediante reemplazamiento, por lo que podríamos estimar la variabilidad de $\hat{}\alpha$, ya que no podemos obtener nuevos datos directamente de la población en este caso.
Comenzaríamos creando una serie de conjuntos de booststrapping(los suficientes como para poder calcular el $SE(\hat{}\alpha)$. Para ello, selecionaríamos "x"datos de nuestra muestra y generaríamos un nuevo conjunto de booststrap, mediante reemplazamiento.
Repetimos esto hasta que consideramos que tenemos suficientes conjuntos de booststrapping como para calcular la desviación típica y por último, calculamos dicha desviación usando la fórmula:

$$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'}\right)^2}. \qquad (5.8)$$

Figura 2.1: fórmula para el cálculo del $SE(\hat{}\alpha)$

que nos dará una estimación del $SE(\hat{}\alpha)$ del conjunto original.

3. Describir que problema resuelve y cuál es el fundamento de la técnica de Validación Cruzada de k-partes (k-CV) y porque debe de funcionar.

En CV lo que se pretende es evaluar los resultados de un modelo para garantizar que son independientes de la partición que se ha hecho de los datos a la hora de tomar conjuntos de train y de test.
La Validación Cruzada de k-partes (k-CV) es un tipo de CV en la que los datos se dividen en k-partes, de las cuales, una se toma como test y el resto serán tomados para training. Este proceso se repite k-veces, para cada uno de los posibles subconjuntos de datos test (van rotando).
Por último, cuando ya todos los subconjuntos han sido conjunto test alguna vez y el proceso termina, se hace la media de los resultados que se obtuvieron para cada iteración.
Y, como es de esperar, este proceso funciona siempre precisamente porque se evalúan muchas combinaciones de datos train y test (esactamente, k). El valor de k, es seleccionado a mano, aunque normalmente se suele tomar k=10.

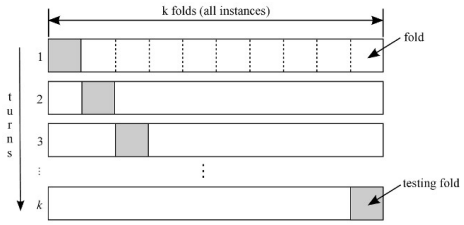

Figura 3.1: validación cruzada con k-particiones

4. Describir las ventajas y desventajas de usar k-CV respecto de usar una aproximación basada en un conjunto de validación o en Leave-One-Out (LOO).

Ventajas de k-CV frente a LOO:

1. k-CV da resultados muy parecidos siempre, independientemente de los datos train y test que se estén considerando(precisión).
2. k-CV es menos costoso computacionalmente, pues divide el conjunto en k-particiones (donde uno será tomado como test y otro para training para cada iteración) frente LOO que toma **una sola muestra** para test y el resto para train, para cada iteración.
3. el número de iteraciones depende de la medida del conjunto de datos, por lo que k-CV siempre iterará menos que LOO.
4. k-CV tiene menos varianza, porque LOO ajusta muchos modelos sobre conjuntos de datos muy parecidos

Desventajas de k-CV frente a LOO:

1. LOO tiene más datos para train: ya en LOO separamos los datos de forma que en cada iteración hay una muestra para test y todo el resto para train.
2. LOO da un error **muy bajo**, pues entrena con más datos.

5. ¿En que beneficia la combinación de múltiples clasificadores frente al uso de un único clasificador ? Justificar la respuesta

Si tenemos un único clasificador, lo que dice ese clasificador es la sentencia final. Sin embargo, si tenemos varios clasificadores entonces el clasificador C_n podría tratar de clasificar bien lo que el clasificador C_{n-1} clasificó mal.(Y así Para N clasificadores). Lo cual nos llevaría seguramente a una reducción de la varianza.

6. ¿Qué es y que aporta el predictor Random Forest frente al uso de Bagging con árboles? Justificar la respuesta

El problema de **bagging** es que funciona bien hasta que hay dependencia entre variables. Si hay variables correladas, no garantiza la disminución de la varianza.
Random Forest aporta, ni más ni menos, que la solución a este problema, haciendo que los árboles que construye bagging no estén correlados.
Para ello, se vale de un "número mágico"(m) ,tal que m=$\sqrt{p}$ y p=número de variables.
Entonces ,Random Forest construye el árbol usando sólo m variables. (Digamos que selecciona un total de m variables a la hora de construir el árbol), introduciendo así una cierta.. " aleatoriedad " que lleva a la disminución de la varianza.

7. Comparar los clasificadores AdaBoost.M1 y Random Forest en el contexto del balance Sesgo-Varianza. Justificar la respuesta

Sabemos que **AdaBoost.M1** se basa en la combinación de clasificadores débiles, sobre los cuales se van construyendo clasificaciones a partir de aquellas en las que el anterior clasificador falló. Finalmente, cuando tenemos "la opinión"de todos los clasificadores, se hace una combinación lineal de ellas, donde a cada opinión se le da un peso:

- alto: si la opinión del clasificador fue buena
- bajo: si la opinión del clasificador no fue buena

por tanto, al estar usando varios clasificadores, seguramente el sesgo no será muy grande. Sabemos, que el error de trainig con AdaBoost.M1 acaba decreciendo hasta 0, y que el de test se puede ir rebajando a medida que se entrena, por tanto, yo creo que en **AdaBoost.M1** el compromiso sesgo-varianza será bueno siempre que los clasificadores débiles sean simples y no excesivamente malos.

Por su parte, sabemos que **Random Forest** disminuye la varianza, pues para solucionar el error que presentaba Baggin, construye los aŕboles de forma que no estén correlados. ¿pero qué pasa con el sesgo en Random Forest? Si tenemos que el árbol está muy pegado a los datos de train, podemos tener una varianza buena pero un sesgo malo, y por tanto un mal compromiso sesgo-varianza.

8. Si tenemos dos métodos que son capaces de separar linealmente un problema de dos clases y uno de ellos es SVM-lineal. ¿Hay alguna razón que nos llevaría a preferir la técnica SVM frente al otro método? Justificar la respuesta

Pues sí se me ocurre una razón por la que podríamos preferir usar SVM frente a otra técnica, y es si queremos dar con el hiperplano óptimo o el hiperplano de máxima separación.
Pues ante ese problema, buscaríamos, de entre todos los hiperplanos, el que haga que la distancia de un punto al hiperplano, la cual se define como :

$$\gamma = \frac{1}{||\omega||}$$

sea máxima. Ahora bien, para ello **SVM** lo que hacía era identificar el interés de los puntos, y los que son interesantes los tiene en cuenta, mientras que los que no son interesantes no los considera. (digamos que, en los que no importan, el valor α=0 que es el que influye a ω , y por lo tanto esa observación es como si no se tuviera en cuenta).

RLG, por ejemplo, también hacía algo así, pero en RLG para calcular el hiperplano se consideran todos los puntos, ya sean relevantes o no, e incluso podían quedar puntos situados en el hiperplano.

9. ¿Cuál son las razones principales para usar técnicas de núcleo en un problema dado? Describir los casos y justificar la respuesta.

En un problema dado el principal motivo para usar técnicas de núcleo es si **nos encontramos que tenemos un problema que no es separable por un hiperplano, al menos en una dimensión baja**.
Entonces podemos aplicar las técnicas de núcleo para ver si es separable en dimensiones más grandes.
¿Cómo?

Bueno, las técnicas de núcleo lo que hacen es encontrar funciones tal que el **producto de los vectores vale lo mismo en el espacio dado que en espacios más grandes**(dimensiónes superiores).
Eso sí, a la hora de usar técnicas de núcleo será necesario que nuestro problema disponga de muchos puntos, sino no podría aplicarse, pues de hacerlo iríamos de cabeza al sobreajuste.
Y también será necesario que los algoritmos usen el producto escalar, pues sino tampoco podría aplicarse.

10. En un laboratorio de biológicos se procesan muestras de material genético para obtener un modelo de predicción de cáncer. Debido al coste de procesamiento solo se pueden procesar un bajo número de muestras, sin embargo cada muestra proporciona un vector de variables de considerable longitud. Los investigadores son capaces de identificar que variables son relevantes como predictores y cuales como predicción, pero no saben que técnica sería más conveniente aplicar en este caso. Discutir el problema y proponer y justificar soluciones adecuadas desde el punto de vista metodológico

El problema es que por el coste de preprocesamiento, no podemos procesar un gran número de muestras, pero sin embargo, cada muestra proporciona un vector de longitud considerable. Probablemente una buena idea sería aplicar **Bootstrapping** , porque así dado un conjunto con un bajo número de muestras, podría llegar a sacar *N* muestras, mediante reemplazamiento.
Con esto podríamos llegar a una muestra más grande, y por tanto una varianza más chica, pues sabemos que :

$\mu = \frac{\Sigma x_i}{N}$
y que: Var[μ]=$\frac{\sigma^2}{N}$, por lo que si N$\uparrow \Rightarrow$ Var$\downarrow$

Pero ésto es en la teoría. En la práctica no siempre sale así, pues puede ser que hagamos bootstrapping y que luego hagamos la media de los árboles ajustados (lo que se conoce como **Bagging**) y que no se verifique que si N$\uparrow \Rightarrow$ Var$\downarrow$.
¿Porqué? ésto depende de si las variables están correladas o no, pues si están correladas, baggin no es el método más acertado, pues constiría árboles conrrelados y no podríamos disminuir la varianza.
Por eso, lo más acertado para este problema yo creo que sería usar **Random forest**, pues este método es como un baggin pero seleccionando variables, y consigue que los árboles que se construyan no estén correlados, por lo que sí se conseguiría disminuir la varianza.