Minería de Datos

UTDT

Master in Management + Analytics

Sesión 2

¿Qué vimos hasta ahora?

- Qué es la minería de datos.
- Distintos tipos de aprendizaje automático.
- Aprendizaje supervisado.
- Naïve bayes.
- K-nearest neighbors.

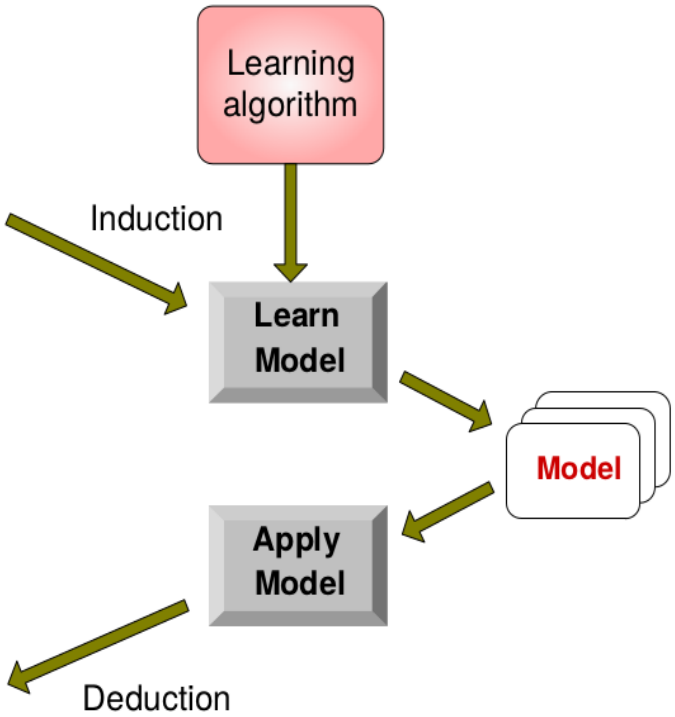
Overfitting & underfitting

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

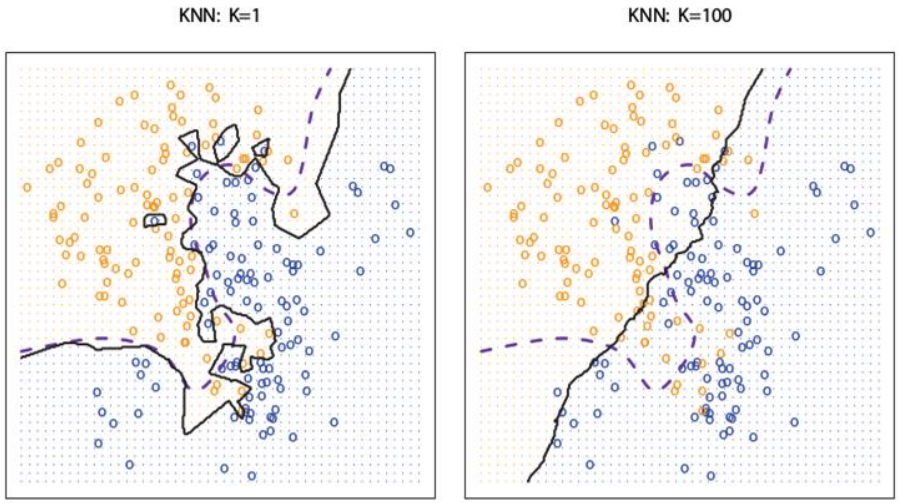
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

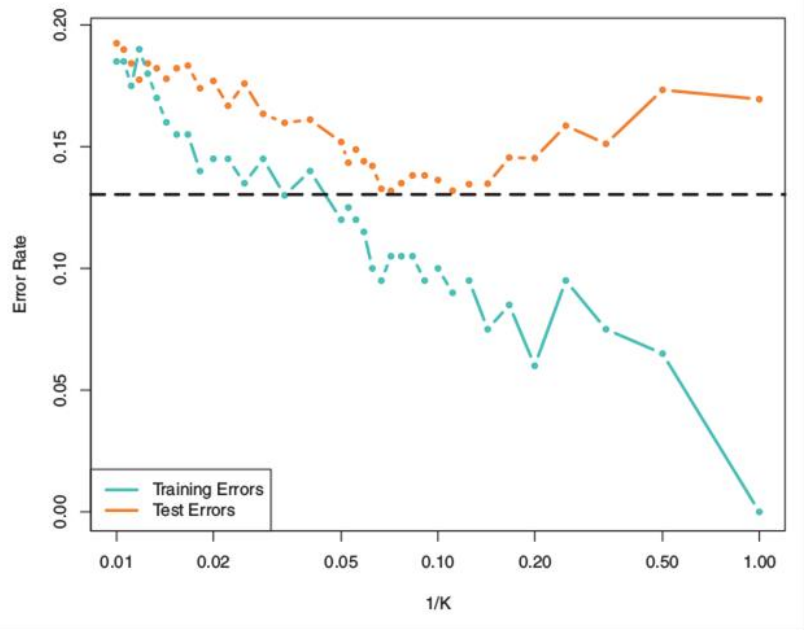
Test Set



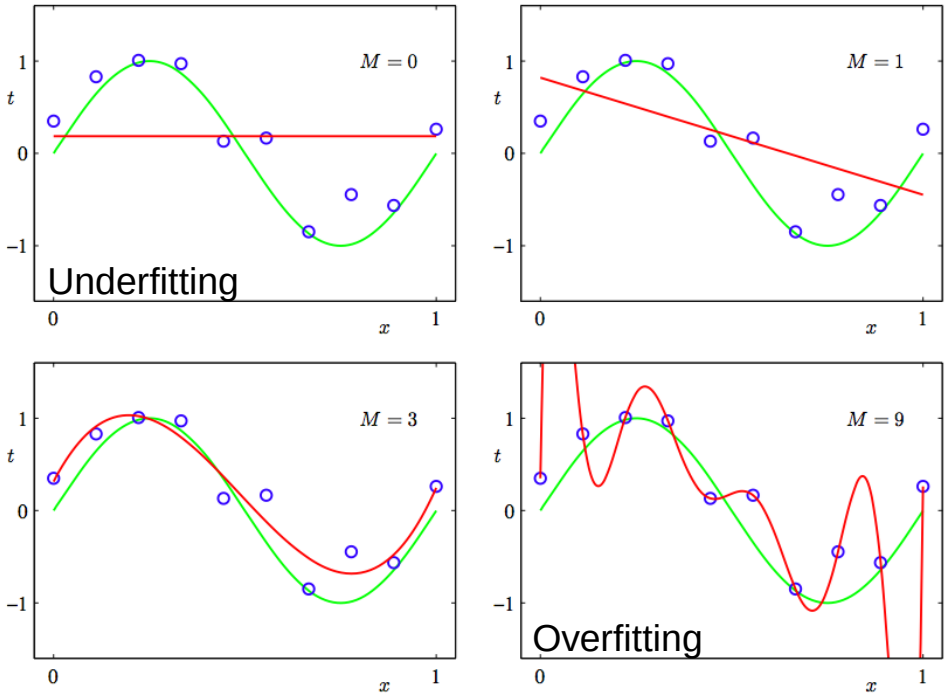
¿Dónde nos interesa predecir bien?

Overfitting & underfitting





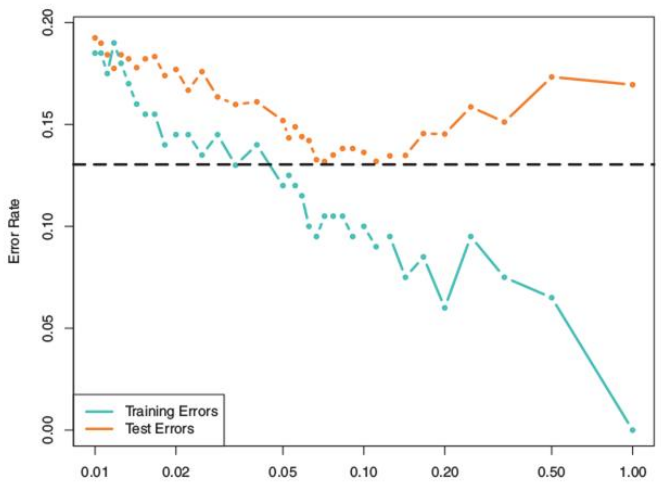
Overfitting & underfitting



Overfitting: se ajusta en exceso la particularidades de los datos de entrenamiento, llevando esto a una mala performance en teste/deployment.

Underfitting: se trata de un algoritmo demasiado rígido que no captura patrones relevantes, consecuentemente tiene mala performance tanto en entrenamiento como en testeo/deployment.

Overfitting & underfitting



¿Overfitting o underfitting?

¿Overfitting o underfitting?

Importante:

1. A medida que aumenta la flexibilidad de un algoritmo, no aumenta necesariamente la performance en testing del mismo (en un momento suele decaer).
2. La flexibilidad de un algoritmo depende en gran medida de hiperparámetros (aunque no depende solo de esto).
3. ¿Dado que no conocemos los valores reales de testing, cómo elegimos buenos valores de hiperparámetros a usar?

Model selection

Al momento de modelar uno tiene muchas decisiones que tomar, por ej.:

- Qué datos tomar como inputs para entrenar modelo.
- Qué preprocesamiento de los datos hacer (clase que viene!).
- Qué modelo usar para predecir (*no free-lunch theorem*).
- Con qué valores de hiperparámetros entrenar el modelo.

Elegir un modelo que tenga buena performance en datos desconocidos es una tarea compleja. En general, vamos a tener muchas observaciones, muchas variables y modelos que fácilmente pueden sobreajustar (combinación peligrosa).

Model selection

IMPORTANTE:

Queremos tener una estimación de cómo impactarán decisiones de modelado en la performance de nuestro modelo, pero en la performance que tendrá en datos que aún no conozco.

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Veamos una metodología para estimar esa performance desconocida.

Model selection

¿Cómo podemos encarar el problema?

Vamos a simular de alguna manera la división entre “datos conocidos” y “datos desconocidos”.

Vamos a usar estos “datos conocidos” para entrenar el modelo con determinadas características (variables, valor de hiperparámetros, etc.).

Vamos a usar estos “datos desconocidos” para validar esas decisiones (de aquí que a este conjunto de datos se le suela llamar *validation set*).

Model selection - Validation/Holdout Set

La manera más simple de simular el comportamiento sobre datos desconocidos es separar una submuestra al azar de observaciones del training set como conjunto de validación.

FIGURE 5.1. *A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.*

¿Una vez definido el modelo final, qué observaciones usamos para entrenarlo?

¡TODAS!

(aunque esto no siempre se hace… por ej., en redes neuronales profundas)

Model selection - Validation/Holdout Set

Potenciales problemas (como estimador de performance futura):

1) La estimación de performance depende de la participación de observaciones en training y validation (es decir, la predicción de performance tiene variabilidad). Una forma de disminuir la variabilidad es repetir el proceso muchas veces y promediar.

2) Es de esperar que a mayor cantidad de datos, un modelo tenga mejor performance. Si el conjunto de entrenamiento quedase con pocas observaciones, la estimación de performance podría ser pesimista. (no es tan grave si uno tiene muchísimos datos).

Model selection - Validation/Holdout Set

Probémoslo en R. Veamos el primer bloque de código.

Model selection - Leave-one-out Cross-validation

Una alternativa para validar modelos sin tener que reducir tanto el tamaño del conjunto de entrenamiento es *loocv*.

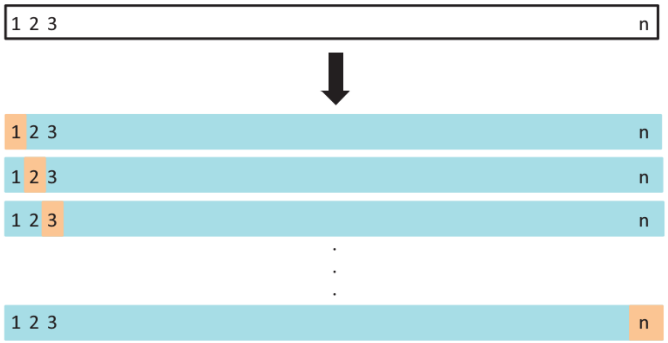


FIGURE 5.3. *A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.*

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Perf_i$$

Ventajas:

- Los resultados no dependen de una partición al azar.
- Cada uno de los modelos es entrenado con prácticamente todas las observaciones, se aprovecha aprender de "muchos" datos.
- Toda observación es usada una única vez para validar.

Desventajas:

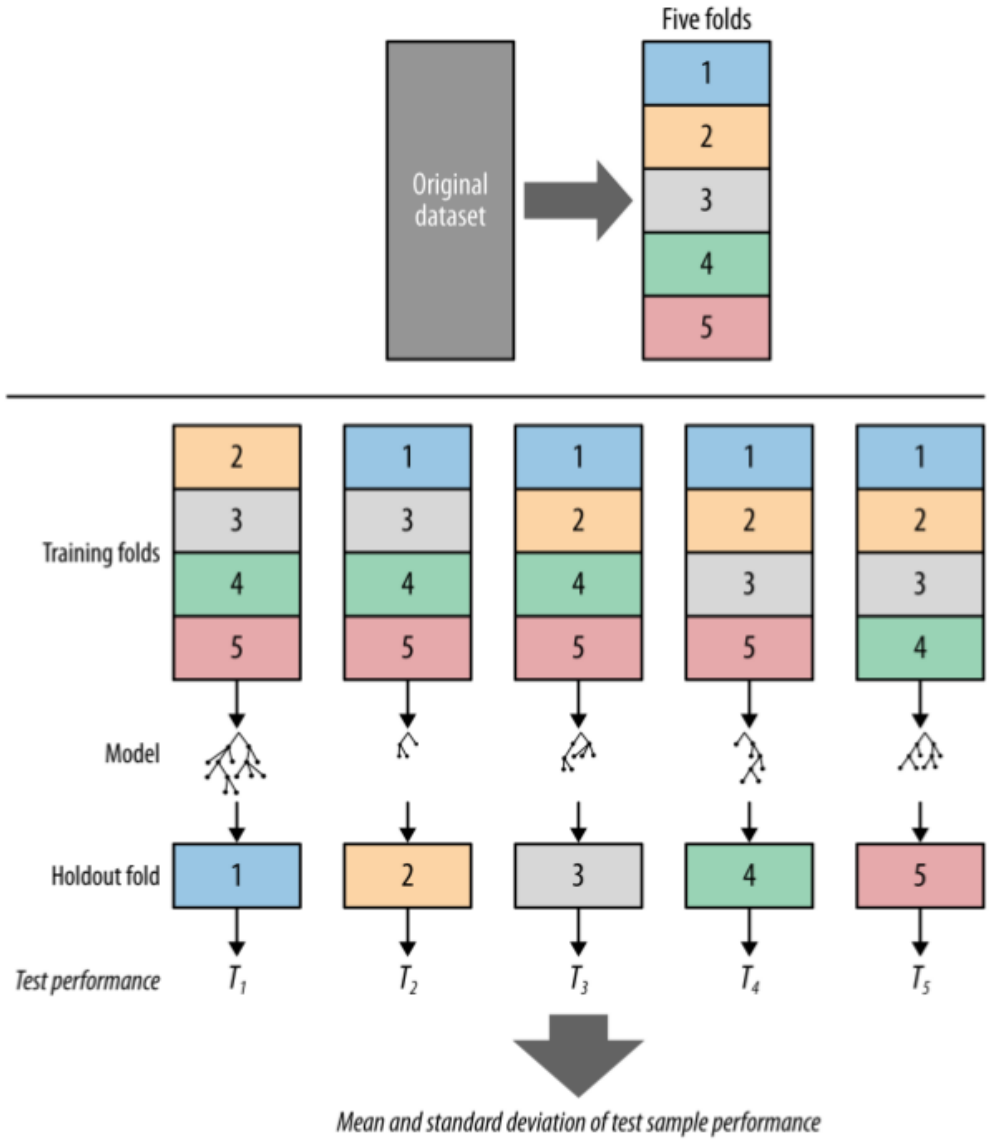
- Requiere de mucho computo (sirve cuando uno tiene poco datos).
- El estimador del error en datos desconocidos tiene alta varianza (ISLR 5.1.4).

Model selection - Leave-one-out Cross-validation

Probémoslo en R. Veamos el segundo bloque de código.

Model selection - *k*-fold cross-validation

Una alternativa "intermedia" es usar *k*-fold cross-validation.



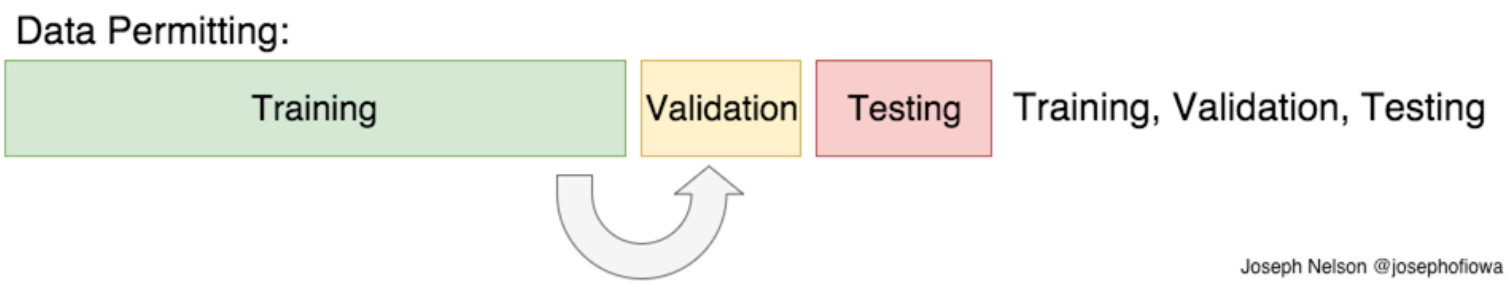
- Típicamente los valores de k que se usan son 3, 5 ó 10.
- Como el resultado también depende particiones al azar, para mejorar la estimación de la performance en testeo, a veces se repite el proceso varias veces.

Model selection - *k*-fold cross-validation

Probémoslo en R. Veamos el tercer bloque de código.

Model selection - training, validation y testing sets.

Si uno realiza distintas pruebas, es posible hacer overfitting sobre el conjunto de validación/holdout. Por este motivo, muchas veces se trabaja con 3 conjuntos: entrenamiento, validación y testeo.



Comúnmente se divide en las siguientes proporciones:

- training 60% (80%)
- validation 20% (10%)
- testing 20% (10%)

Aun así, cuando se tienen muchos datos se suelen usar conjuntos de testing y validation considerablemente menores (por ej. 2.5%).

Model selection - training, validation y testing sets.

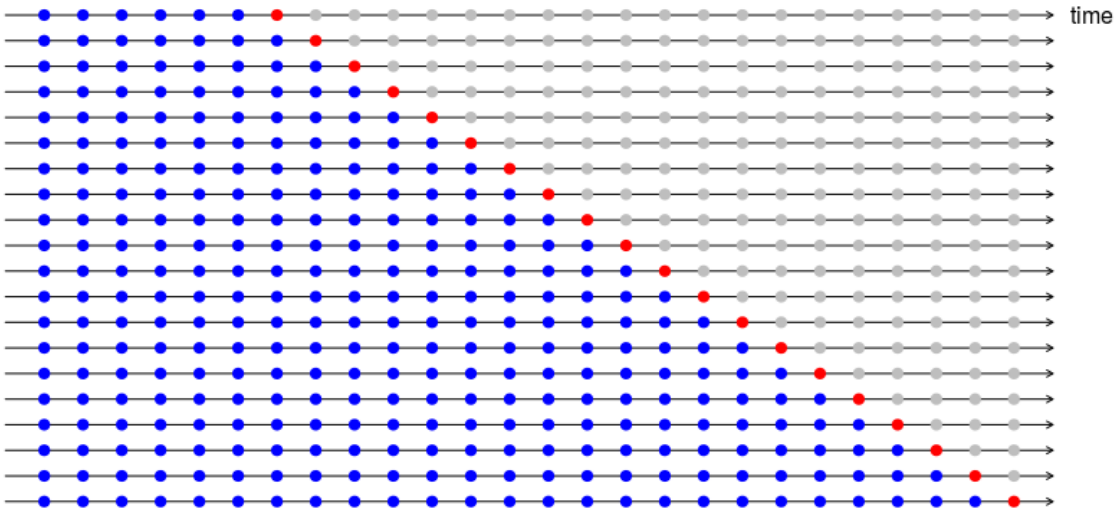
Probémoslo en R. Veamos el cuarto bloque de código.

Model selection

Estos métodos suponen que las observaciones son i.i.d. Lo cual puede no ser cierto (supuesto clave).

¿Un modelo de churn entrenado con datos de 1990 servirá hoy?

Es importante entender qué características tendrán los datos desconocidos. Lo que se debe intentar es replicar lo mejor posible con el esquema de validación aquello que genera los datos desconocidos.



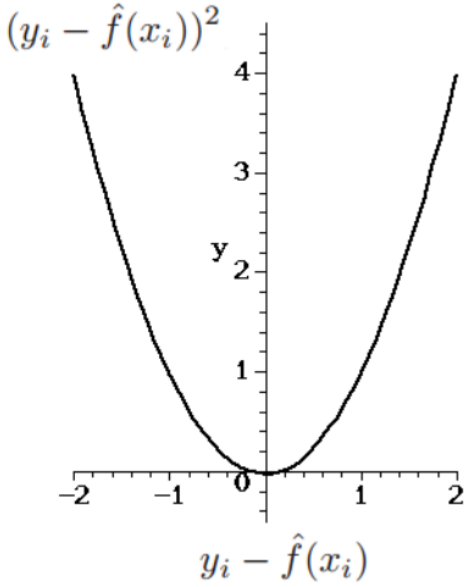
Decision trees

Vamos a ver un modelo adicional de aprendizaje supervisado: árboles de decisión.

La técnica sirve tanto para atacar problemas de clasificación como de regresión.

Sólo por simplicidad, primero encararemos el problema de regresión. Para ello, definimos una medida de "calidad de la predicción" (función de costo). Sin pérdida de generalidad, usaremos el error cuadrático medio (MSE).

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



Decision trees

Nuestro objetivo es minimizar el ECM en datos desconocidos, pero vamos a usar los datos de entrenamiento para guiar la búsqueda de patrones con poder predictivo que esperamos generalicen bien.

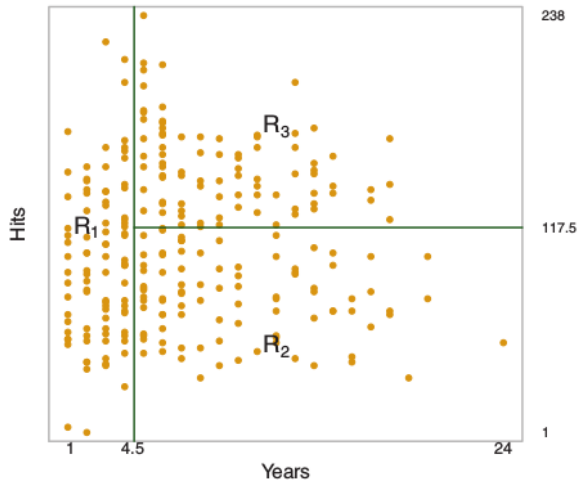
En árboles de decisión la construcción de los mismos estará guiada por minimizar el error/costo en entrenamiento (aun cuando no sea lo que en verdad nos interesa).

Esta idea se aplica a muchos modelos de aprendizaje supervisado, por ej.:

- Regresión lineal
- Ridge regression
- LASSO
- Regresión logística
- Redes neuronales
- Support vector machines
- Gradient Boosting Machines
- XGBoost

Decision trees

Vamos a proponer un modelo que particione el espacio de atributos en regiones. Las regiones vendrán definidas por si se cumplen o no una serie de reglas.



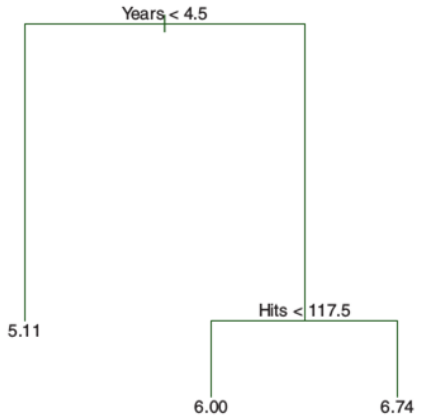


FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

Particionar de esta manera al espacio de atributos define una estructura de árbol (con nodos internos, nodos hoja y ramas)

Decision trees

Primero veamos cómo usar una partición ya armada para predecir.

Prediction via Stratification of the Feature Space

We now discuss the process of building a regression tree. Roughly speaking, there are two steps.

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Decision trees

En teoría las regiones podrían tener cualquier forma, por simplicidad tendrán forma de rectángulos (o “cajas”) en altas dimensiones.

El objetivo será encontrar las cajas que minimicen el ECM. (de entrenamiento), es decir:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

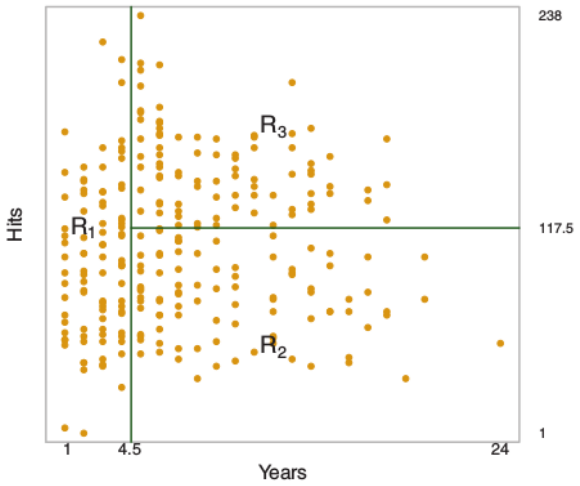
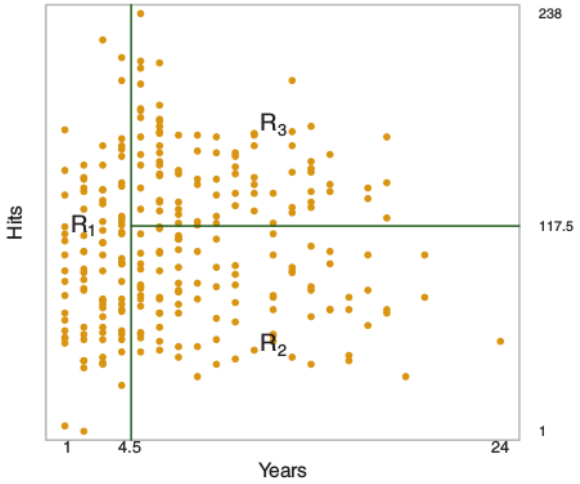


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

Decision trees

¿Cómo podremos encontrar buenas particiones?

Vamos a utilizar un paradigma que se llama *divide & conquer* (consiste en dividir recursivamente un problema en subproblemas más pequeños que conservan la misma estructura).



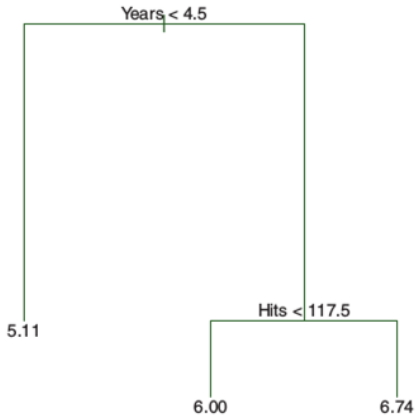


FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

Decision trees

Seguiremos una estrategia de particiones recursivas (*top-down* y *greedy*).

Para cada variable *j* y cada punto de corte *s*, definimos las siguientes regiones:

$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\}$$

Elegiremos los valores de *j* y *s* que minimicen la siguiente expresión:

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

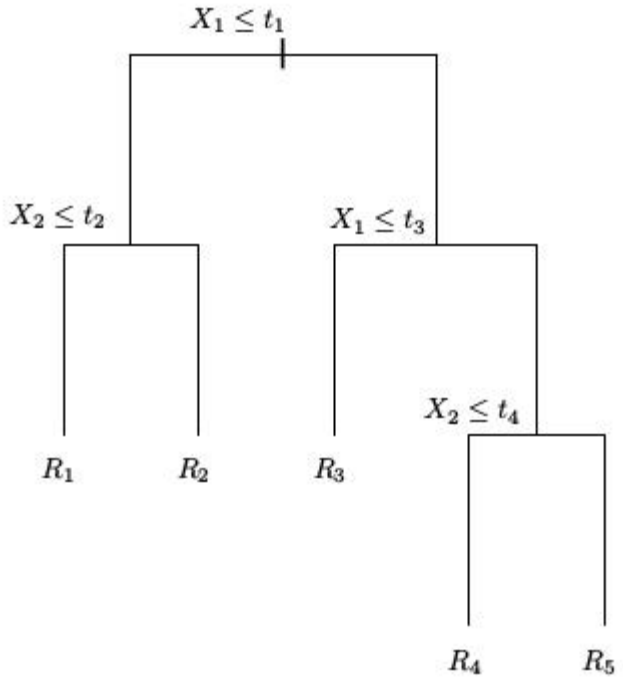
Luego esto se repite para cada subregión (paso recursivo) hasta cumplir algún criterio de parada (por ej., que en la región quede una única observación).

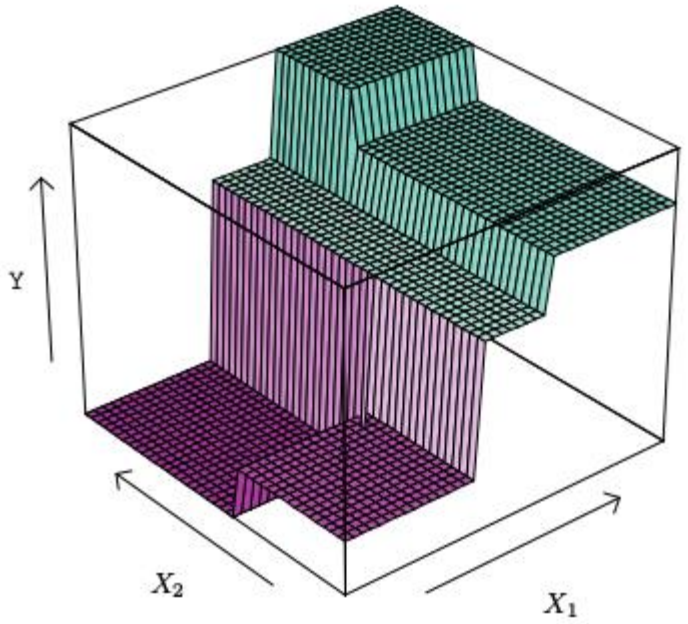
Decision trees

Veamos un ejemplo:

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

Decision trees



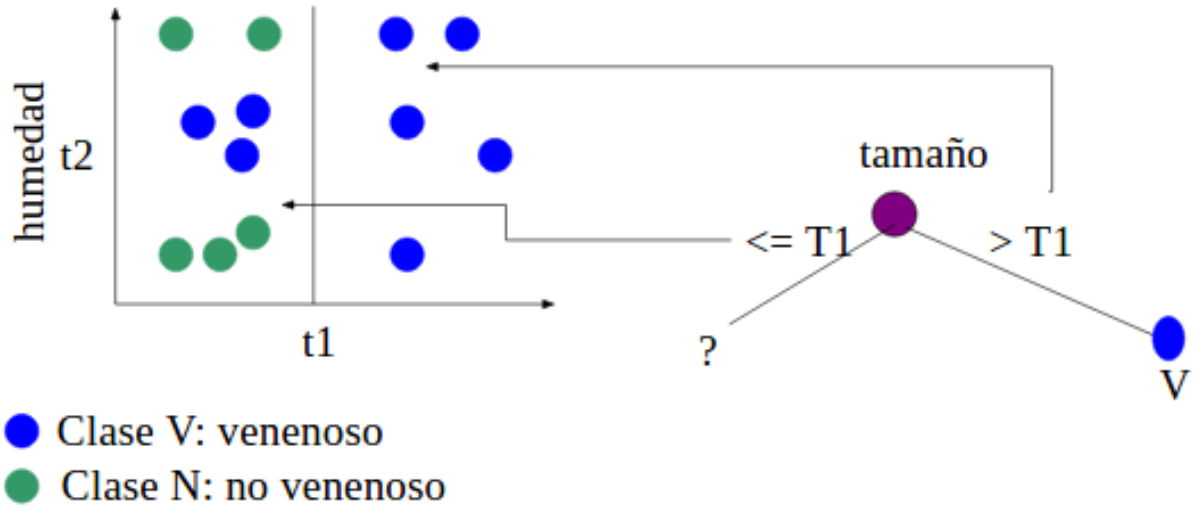


¿Pueden identificar en la figura de la derecha cada región?

Decision trees

La idea se puede adaptar fácilmente a clasificación.

Se predice como probabilidad condicional de una clase *k*, ($P(C_k \mid x_i)$), a la proporción de observaciones de la clase *k* que cae en la región correspondiente a x_i.

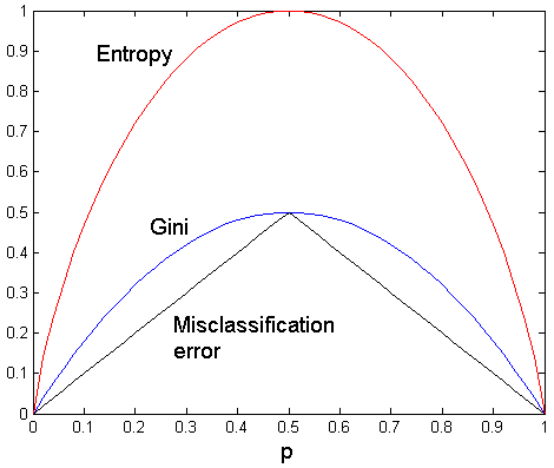


Decision trees

En clasificación, en vez de guiar la construcción usando MSE, se utilizan medidas de impureza.

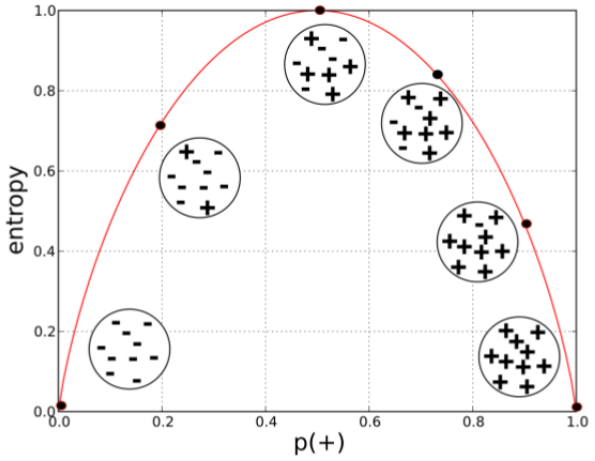
The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$



An alternative to the Gini index is *cross-entropy*, given by

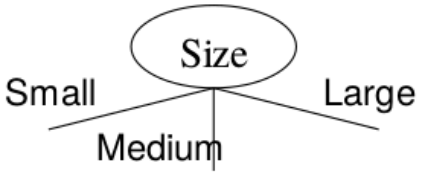
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$



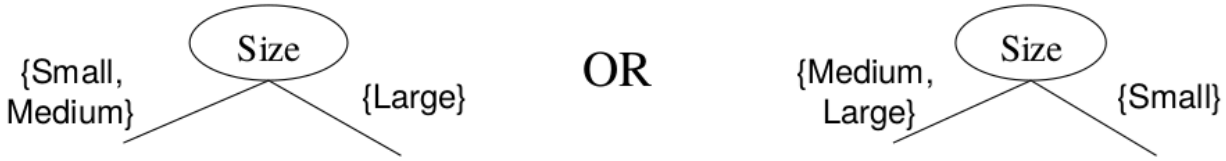
Decision trees

Los árboles pueden trabajar de manera muy natural con variables categóricas.

Cortes múltiples:



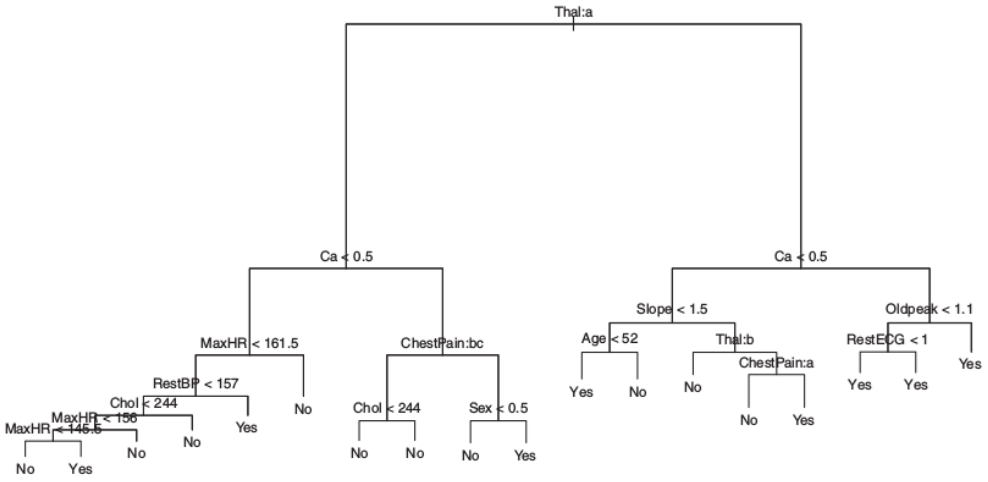
Cortes binarios:

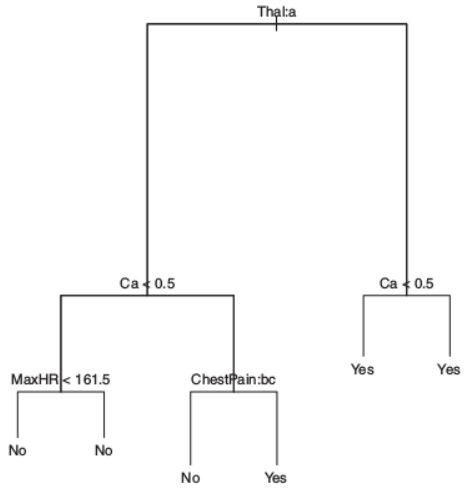


(No todas la librerías tiene esto bien implementado, por ej. sci-kit learn tiene implementado árboles que sólo trabajan con atributos numéricos)

Decision trees

¿Cuál de estos dos árboles ajusta más los datos?





¿Cuál tendrá más riesgo de sobreajustar los datos?

Decision trees

Hay muchos criterios para evitar que los árboles sean excesivamente profundos, por ej.:

- Definir una profundidad máxima (`maxdepth`).

- Definir un número mínimo de observaciones en un nodo de decisión para realizar un split (`minsplit`).

- Definir un número mínimo de observaciones para aceptar una hoja (`minbucket`).

Decision trees

Complexity parameter:

- Dado un valor de α hacer crecer un árbol a máxima profundidad y buscar el sub-árbol que minimice la siguiente expresión (ISLR, algoritmo 8.1).

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

¿Cómo afecta α la flexibilidad del modelo?

α es un hiperparámetro que al aumentar quita flexibilidad al modelo (al proceso de quitar flexibilidad con hiperparámetros se lo llama "regularizar").

En la práctica (`?rpart.control`):

"*Any split that does not decrease the overall lack of fit by a factor of cp is not attempted…. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile…*"

Decision trees

Importancia de variables

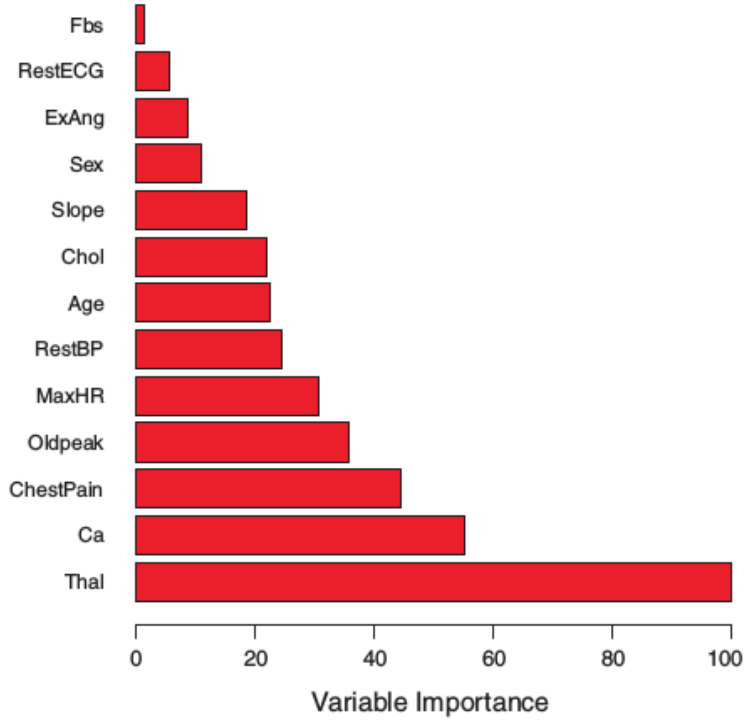


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Decision trees

Puntos a favor:

- Hábiles en seleccionar atributos relevantes.
- Buen manejo de no linealidades.
- Buen manejo de interacciones entre variables.
- Manejo elegante de atributos categóricos (seǵun la implementación).
- Manejo elegante de datos faltantes (según la implementación).
- Interpretables si no son excesivamente profundos.

Puntos en contra:

- No tienen una gran capacidad predictiva.
- Su estructura puede cambiar mucho ante pequeños cambios de los datos de entrenamiento (prueben cambiar la semilla en el código de clases y vean cómo cambia la estructura del árbol).
- Pueden ser poco interpretables si tienen gran profundidad.

Decision trees

Probémoslo en R. Veamos el quinto bloque de código.

Lecturas recomendadas para los temas vistos hoy

Selección de modelos:

- ISLR (Cap 5)

Árboles de decisión:

- ISLR (Cap 8), Tan (Sección 4.3)

Práctica de laboratorio

Para hacer:

Entiendan el problema de predicción que se plantea en https://archive.ics.uci.edu/ml/datasets/Adult. Una vez que lo hayan entendido, vayan a "Data Folder" y bajen los archivos "adult.data" y "adult.test" (noten que los nombres de columnas se encuentran en un tercer archivo).

Se pide:

- En base a los datos que se encuentran en adult.data entrene un modelo de árbol de decisión, que tenga profundidad máxima y valores de 1 en `minsplit` y `minbucket`. Pero para el que sí se prueben diferentes valores de `cp`. Para seleccionar un buen valor de cp siga un esquema de k-fold cross validation con 5 folds. ¿Cuál es el mejor valor de performance obtenido?
- Pruebe entrenar el modelo utilizando todos los datos de entrenamiento y la mejor configuración que encontró en el punto 1.
- Analice si la performance que estimó en el punto 1 fue similar a la que efectivamente obtuvo en el punto 2.

Práctica teórica

¿Es cierto que cuando uno hace overfitting tanto el error en entrenamiento como el validación suelen ser altos? Justifique su respuesta.

¿Cuál es el objetivo final de usar técnicas como la de holdout set?

¿Es cierto que tanto en holdout set como en k-fold cross validation al reordenar los datos de manera aleatoria se garantiza que uno va a tener una buena estimación de la performance del modelo en datos desconocidos? Justifique su respuesta.

¿Puede darse la situación en que el valor de `minbucket` afecte cómo el árbol parte un nodo particular pero el valor de `minsplit` no? Piense y explique un ejemplo concreto.

¿Es cierto que árboles de decisión es un modelo lazy? Justifique su respuesta.