uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
APRENDIZAJE AUTOMÁTICO PARA EL ANÁLISIS DE DATOS
GRADO EN ESTADÍSTICA Y EMPRESA
Ricardo Aler

METODOLOGÍA (MACHINE LEARNING PIPELINE):

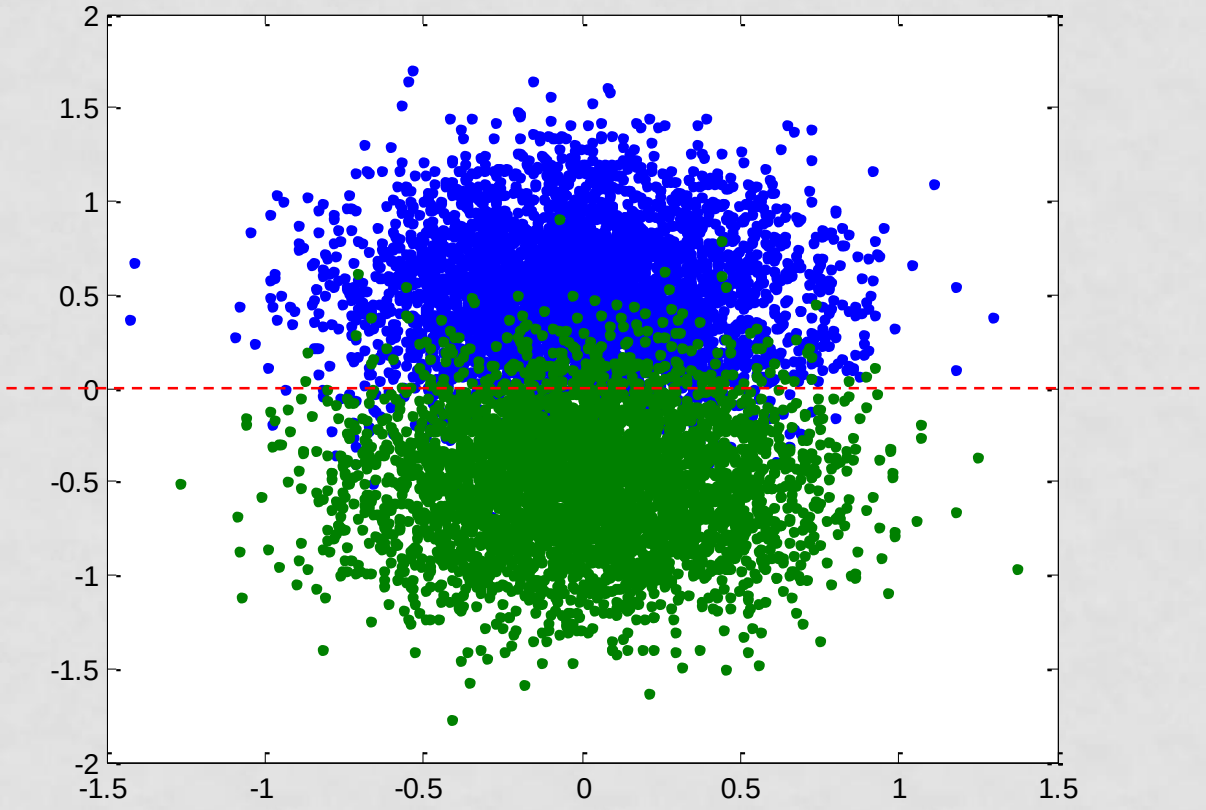
EVALUACIÓN DE MODELOS, AJUSTE DE HIPER-PARÁMETROS

¿POR QUÉ ES NECESARIO EVALUAR MODELOS?

- Evaluar un modelo significa estimar cual va a ser su precisión con datos futuros (es decir, no utilizados durante el entrenamiento del modelo).
- El hecho de que un modelo funcione muy bien con los datos de entrenamiento, no significa necesariamente que vaya a hacerlo con datos futuros:
  - El modelo puede haberse sobreadaptado (overfitting) a los datos de entrenamiento

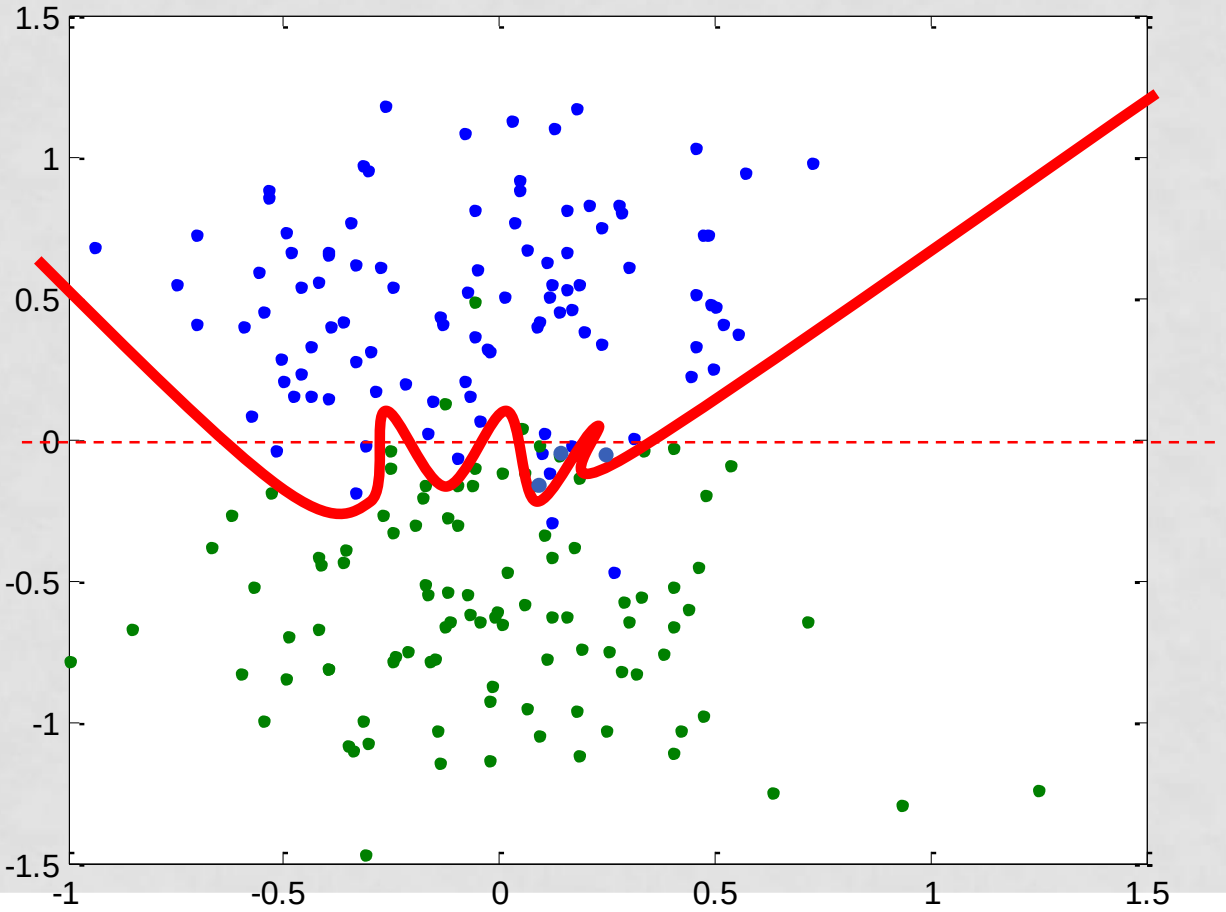
Ejemplo de sobreadaptación en clasificación

- Supongamos que nuestros datos provienen de dos gausianas que generan cada una de las clases (azul y verde)



Sobreadaptación

- Pero supongamos también que sólo disponemos de unos pocos datos para entrenar nuestro modelo (cosa que ocurre en los casos reales). Puede ocurrir que nuestro modelo se ajuste muy bien a los datos de entrenamiento, pero que funcione peor con datos futuros, distintos a los usados en el entrenamiento. Por ello es bueno tener una estimación del comportamiento futuro del modelo.



Evaluación

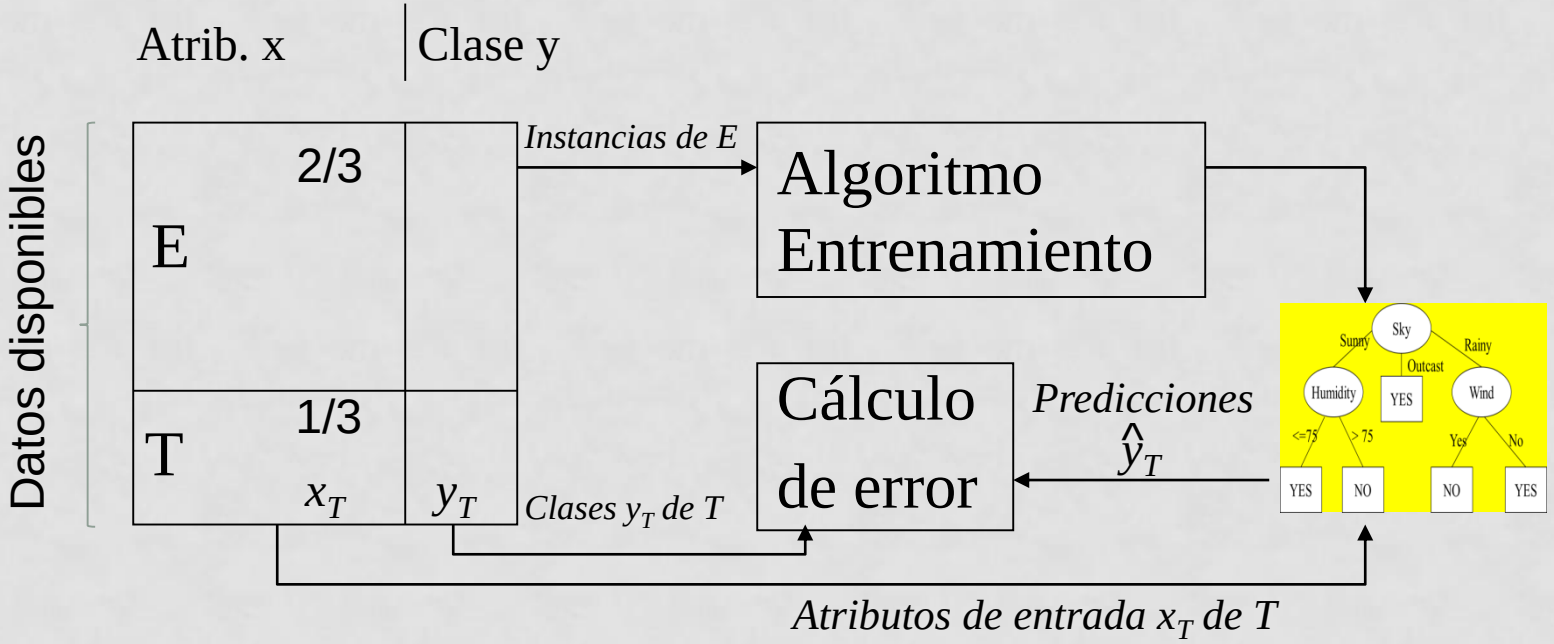
- Otro ejemplo: si a un alumno se le evalúa (examen) con exactamente los mismos problemas con los que aprendió, no se demuestra la capacidad de generalización (tan sólo la capacidad de memorización)

- En resumen, del proceso de aprendizaje, partiendo de un conjunto de datos disponibles queremos obtener:
  - Un modelo
  - Su precisión futura

Evaluación mediante particiones de entrenamiento y test

Holdout partition

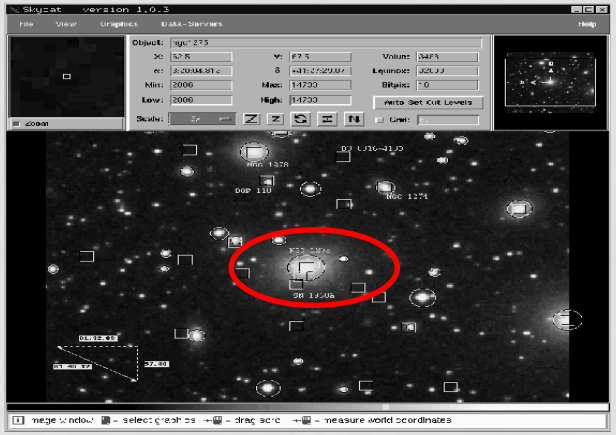
- El conjunto de ejemplos se divide en dos partes **ALEATORIAMENTE**: entrenamiento (E) y test (T)
- Se entrena el modelo con E
- Se estima el error (o tasa de aciertos) que el clasificador comete en el conjunto de test



Particiones estratificadas

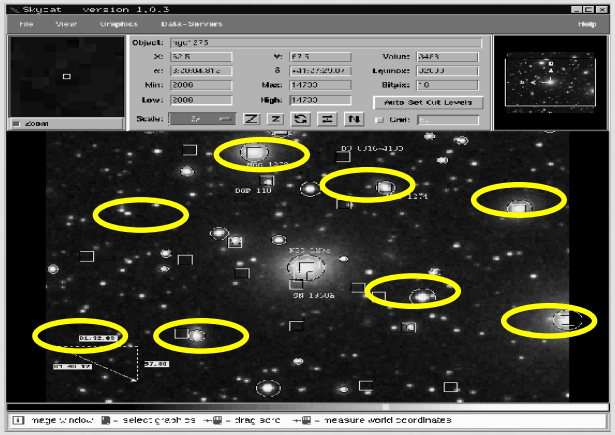
- Para que el test sea mas representativo, es conveniente que las particiones sean estratificadas. Esto es especialmente cierto en problemas desbalanceados (una de las clases tiene muy pocas instancias)
- La proporción entre las clases que existe en el conjunto de datos original, se intenta mantener en los conjuntos de train y test
  - Ejemplo: si en el conjunto original un 99% de los datos pertenecen a la clase negativa y 1% a la positiva, la estratificación intentará que esa proporción se mantenga en train y test

RESUMEN HASTA AHORA:
Comenzamos con este esquema.



?

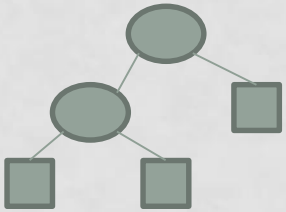
Datos Entrenamiento



Método
(o algoritmo)

Modelo

Uso final del
modelo



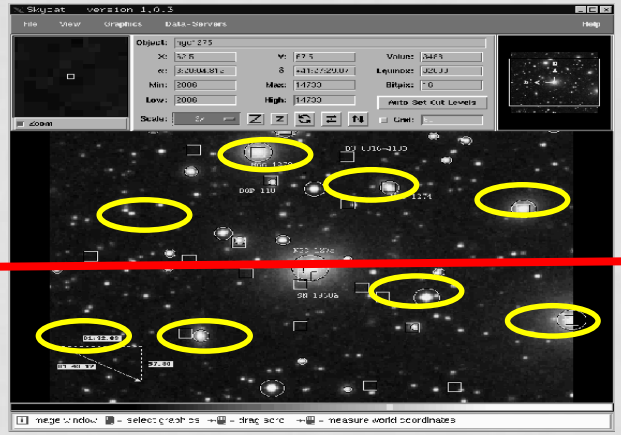
Galaxia espiral

RESUMEN HASTA AHORA:
Ahora, además del modelo, hacemos una estimación del porcentaje de aciertos del modelo con Train (entrenamiento)/Test.

Construcción de un modelo

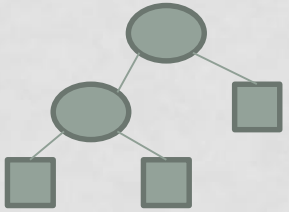
Datos Disponibles

Entrenamiento



Test

Método
(o algoritmo)



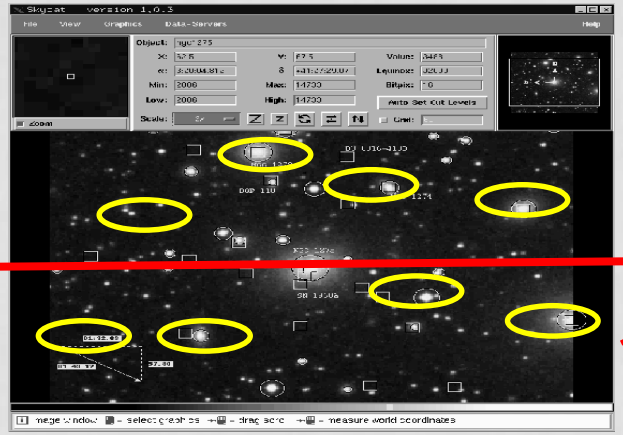
Modelo

RESUMEN HASTA AHORA:
Ahora, además del modelo, hacemos una estimación del porcentaje de aciertos del modelo con Train (entrenamiento)/Test.

Evaluación de dicho modelo

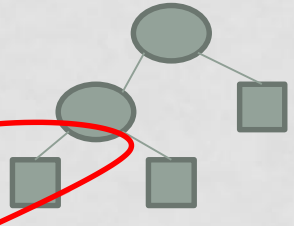
Datos Disponibles

Entrenamiento



Test

Método (o algoritmo)



Modelo

Evalúa → 83%

Problemas train/test (holdout)

- Problema: es posible que por azar, los datos de entrenamiento y/o test estén sesgados, sobre todo si hay pocos datos.
  - Dicho de otra manera, que el conjunto de test no sea representativo del de entrenamiento (es fácil que ocurra si test es pequeño)

Train/test repetido

- Consiste en partir el conjunto de datos totales múltiples veces y calcular el porcentaje de aciertos medio
- Como los sesgos ocurren por azar, la idea es que los sesgos de unas y otras particiones se cancelen
- **Método:**
  - **Repetir múltiples veces:**
    1. **Desordenar el conjunto de datos total aleatoriamente**
    2. **Escoger los primeros 2/3 para entrenamiento y construir el modelo con ellos**
    3. **Escoger los últimos 1/3 para el test y estimar el porcentaje de aciertos**
  - **Calcular el porcentaje de aciertos medio**

ENTRENAMIENTO Y TEST REPETIDO

- Problema: las distintas particiones de test no son independientes (pueden solaparse unas con otras por casualidad)
- Explicación: en el caso extremo, si por casualidad todas las particiones de test contuvieran exactamente los mismos datos, el repetir muchas veces el cálculo en test no nos aportaría ninguna información adicional
- El caso extremo no ocurre, pero siempre hay algún solape entre las particiones de test
- Lo ideal es que las particiones de test no solapen

VALIDACIÓN CRUZADA (CROSSVALIDATION)

- Solución: dividir **varias veces** el mismo conjunto de datos en entrenamiento y test y calcular la media. Así, se probarán con distintas particiones de test y los sesgos se compensarán.
- Se divide el conjunto de datos original en k partes (folds). Con k=3 tenemos los subconjuntos X, Y, Z.
- Tres iteraciones:
  - Aprender con X, Y; test con Z (T1 = % aciertos con Z)
  - Aprender con X, Z; test con Y (T2 = % aciertos con Y)
  - Aprender con Y, Z y test con X (T3 = % aciertos con X)
  - % aciertos esperado T = (T1+T2+T3)/3
- Se suele utilizar k=10

VALIDACIÓN CRUZADA (CROSSVALIDATION)

- Las particiones de test de los distintos ciclos son independientes (no solapan)
- El método de validación cruzada utiliza muy bien los datos al calcular el porcentaje de aciertos esperado, porque todos ellos se utilizan para test (en alguna partición).
- De hecho, todos los datos figuran como entrenamiento o test en alguno de los ciclos de validación cruzada.
- Nota: a cada una de las k divisiones de los datos de entrenamiento se la denomina *fold*

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

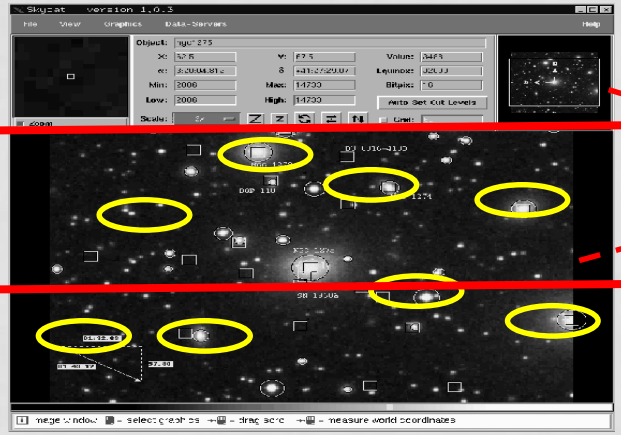
Entrenamos con X e Y, evaluamos con Z

Datos Disponibles

Fold X

Fold Y

Fold Z



Método

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

Entrenamos con X e Y, evaluamos con Z

Datos Disponibles

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

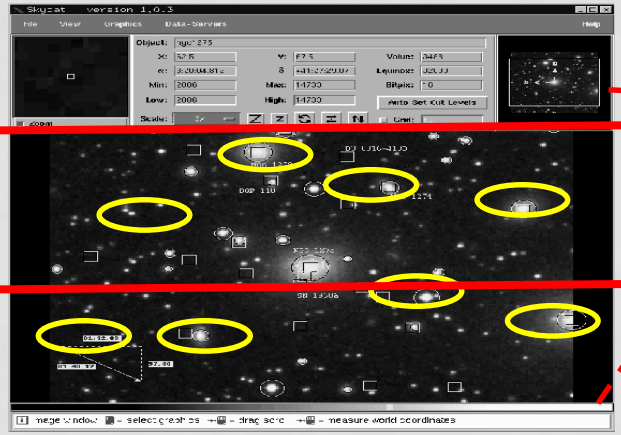
Entrenamos con X, Z; evaluamos con Y

Datos Disponibles

Fold X

Fold Y

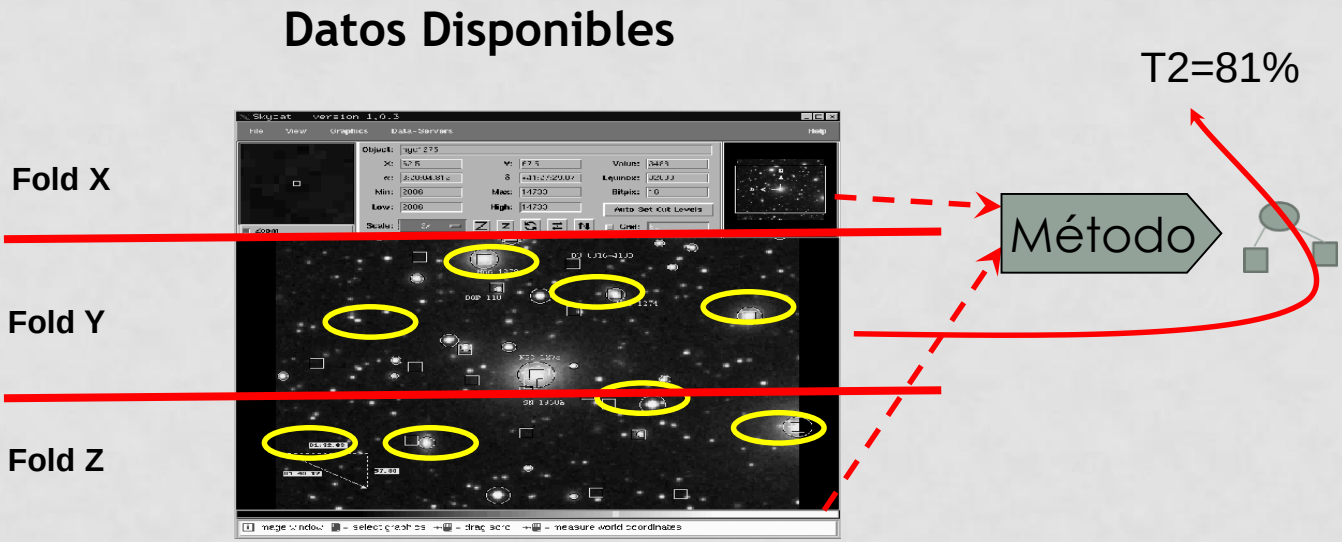
Fold Z



Método

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

Entrenamos con X, Z; evaluamos con Y



RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

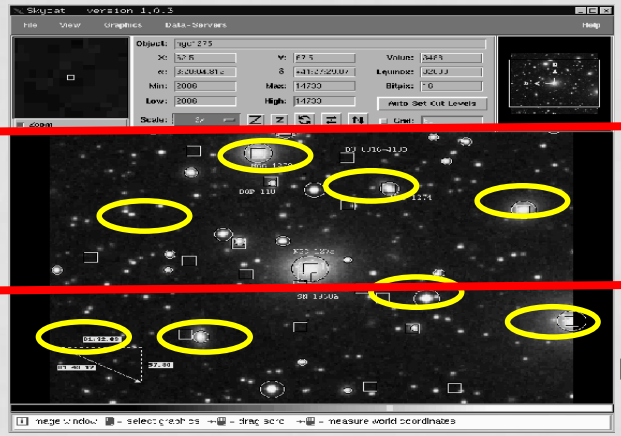
Entrenamos con Y, Z; evaluamos con X

Datos Disponibles

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

Calculamos la evaluación T como la media de los tres folds

Datos Disponibles



Fold X

Fold Y

Fold Z

T1=80%

T2=81%

T3=78%

Evaluación

T= (80%+81%+78%)/3 = 79.7%

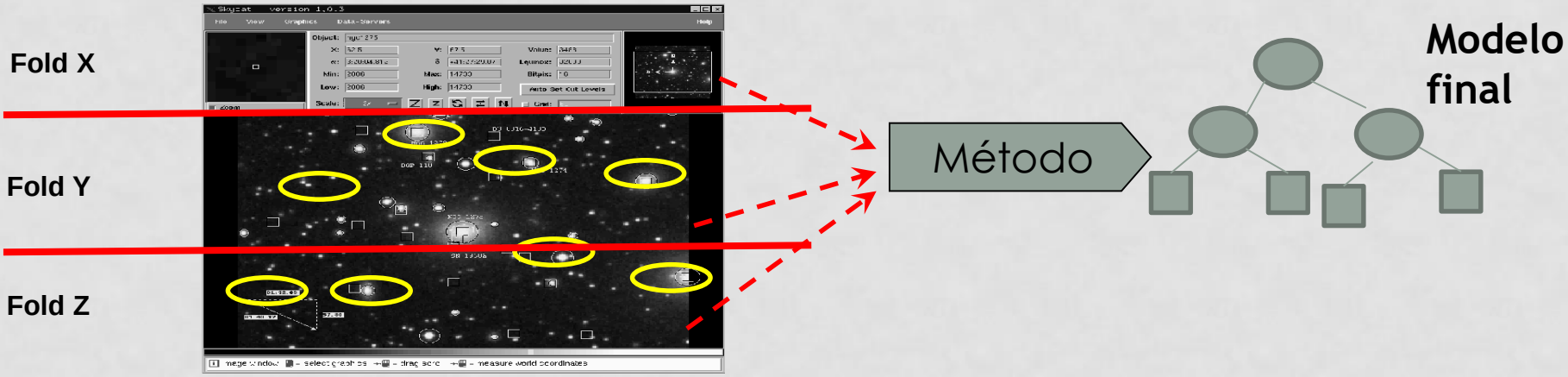
CROSSVALIDATION: ¿QUÉ SE ESTÁ EVALUANDO?

- Hay que pensar en esto …
- En teoría queremos evaluar la precisión futura de un modelo
- ¡Pero con k-fold crossvalidation entrenamos k modelos!
- Aunque los k modelos no serán tan diferentes (sobre todo si k es grande, típicamente 10), dado que los datos de entrenamiento para entrenarlos solapan.
- En cualquier caso, se entrena un clasificador (o modelo) final CF **con todos los datos (los tres conjuntos** X, Y, Z) y se **asume** que T es una estimación del porcentaje de aciertos de CF

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

Construcción del modelo final con todos los datos

Datos Disponibles



Evaluación: T = 79.7%

RESUMEN HASTA AHORA:
Si la evaluación la hacemos con validación cruzada con 3 folds:

?

Uso del modelo

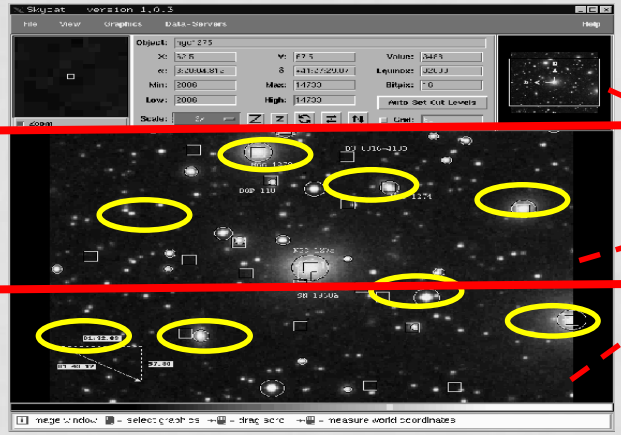
Uso final del modelo

Datos Disponibles

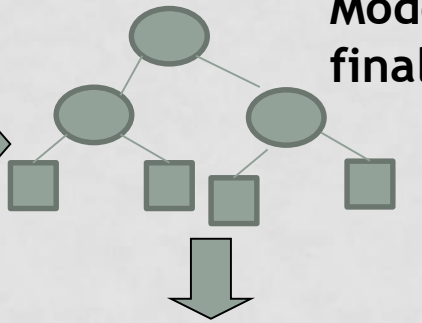
Fold X

Fold Y

Fold Z



Método



Modelo final

Galaxia espiral

Evaluación: T = 79.7%

CROSSVALIDATION (VALIDACIÓN CRUZADA)

- En cualquier caso, se entrena un clasificador final CF **con todos los datos (los tres conjuntos** X, Y, Z) y se **asume** que T es una estimación del porcentaje de aciertos de CF

- Pregunta: el % de aciertos computado con validación cruzada, ¿es una estimación optimista o pesimista del clasificador (modelo) CF?

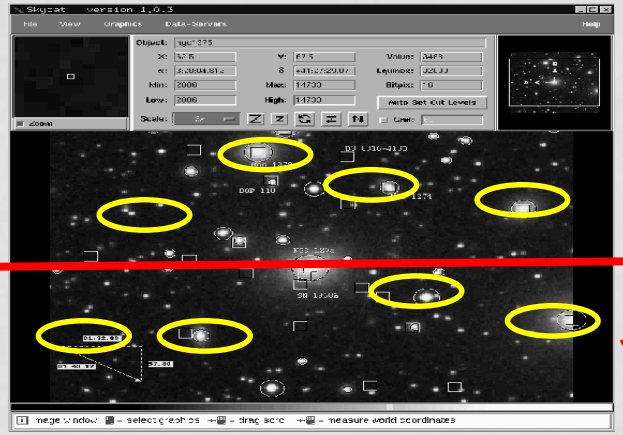
- **Importante**: el método de validación cruzada es el recomendado, pero si tenemos un conjunto de test razonablemente grande, **es suficiente usar train/test**.

También en el caso de train/test construimos un modelo final uniendo todos los datos.

Evaluación de dicho modelo

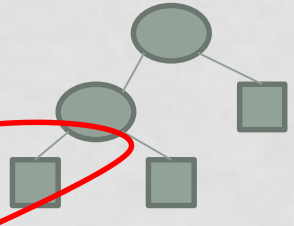
Datos Disponibles

Entrenamiento



Test

Método (o algoritmo)



Modelo

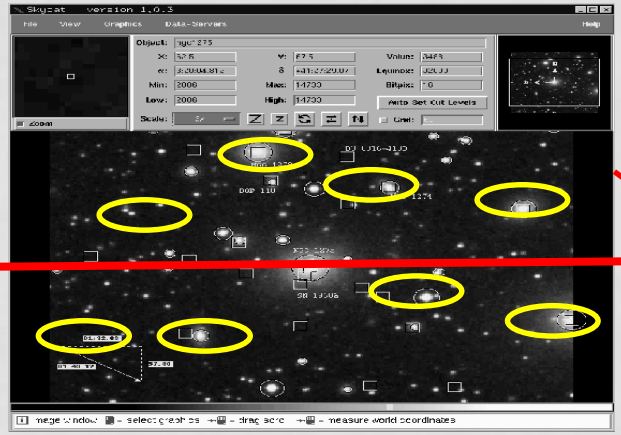
Evalúa 83%

También en el caso de train/test construimos un modelo final uniendo todos los datos.

Construcción del modelo final con todos los datos

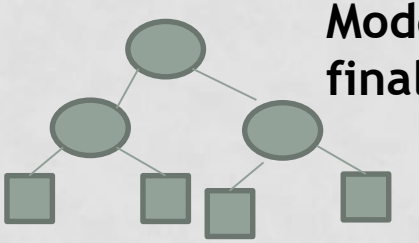
Datos Disponibles

Entrenamiento



Test

Método

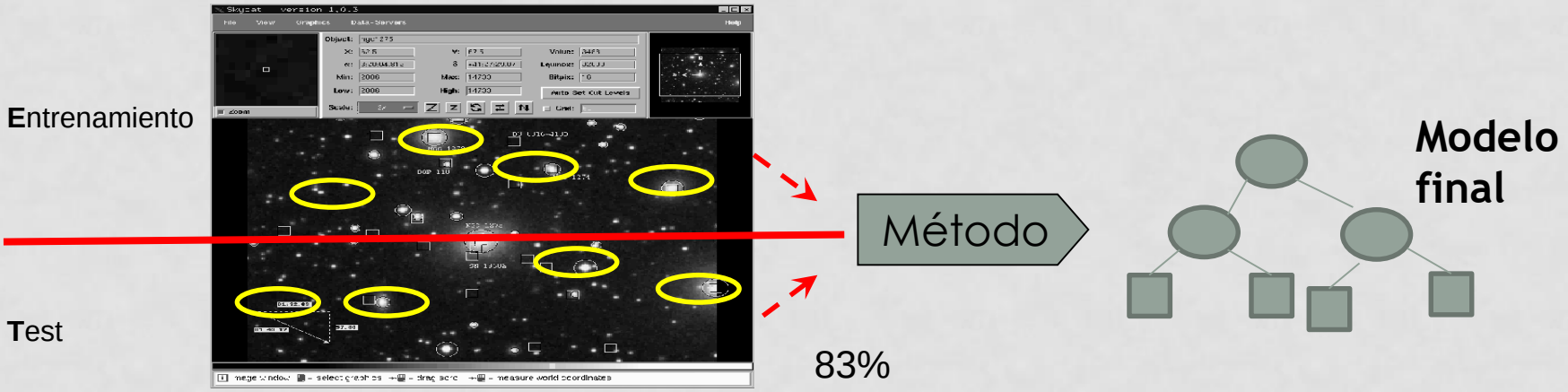


Modelo final

También en el caso de train/test construimos un modelo final uniendo todos los datos.
La estimación que hemos obtenido (83%) también tenderá a ser pesimista para el modelo final, dado que la hemos obtenido a partir de un modelo construido con menos datos que el modelo final.

Construcción del modelo final con todos los datos

Datos Disponibles



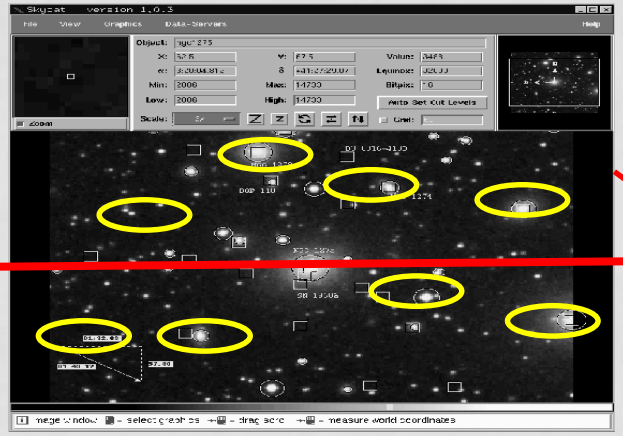
Uso del modelo final

?

Uso final del modelo

Datos Disponibles

Entrenamiento

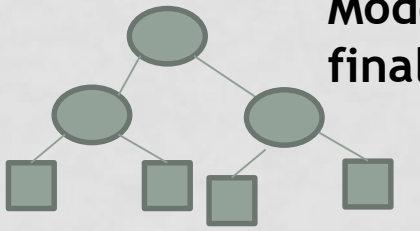


Test

Método

83%

Modelo final



Galaxia espiral

CRITERIOS BÁSICOS PARA EVALUAR

- En problemas de clasificación, si tenemos 2 clases,
  - ¿Cuál es el porcentaje de aciertos a superar?
  - ¿Por qué?
  - ¿Y si tenemos M clases?

CRITERIOS BÁSICOS PARA EVALUAR

- En problemas de clasificación, si tenemos 2 clases (o M), el porcentaje de aciertos a superar es el 50% (o 100*1/M).
  - De otra manera, sería mejor tirar una moneda (azar) que utilizar el clasificador para predecir
  - Aunque también hay que tener en cuenta que en problemas de clasificación biclase, un clasificador que acierte al 40% es fácil transformarlo en otro que acierte al 60% (simplemente, invirtiendo su salida).

CRITERIOS BÁSICOS PARA EVALUAR

- Sea un problema con muestra desbalanceada (ej: 990 datos negativos y 10 positivos), ¿cuál es el porcentaje de aciertos a superar?

CRITERIOS BÁSICOS PARA EVALUAR

- En problemas de clasificación, si tenemos una clase con muchos más datos que otra, el porcentaje de aciertos a superar es el porcentaje de datos de la clase mayoritaria
  - Ej: Sean dos clases (+ y -). Hay 990 datos - y 10 +. Un clasificador que prediga siempre + (independientemente de los atributos), ya acertará en un 99% (== 990/1000). Hay que hacerlo mejor que eso.
  - ZeroR es el llamado "clasificador de clase mayoritaria": clasifica todas las instancias con la clase que más instancias tiene en los datos de entrenamiento. Es un clasificador "trivial" y nuestro modelo tiene que batirlo con claridad.

MEDIDAS PARA EVALUAR

- Hasta ahora, hemos hablado de clasificación, y la medida adecuada para evaluar un modelo es el porcentaje de aciertos (un número entre 0% y 100%) o tasa de aciertos (*accuracy*) (entre 0 y 1).
- También podemos usar el error de clasificación:
  - error = 1 – tasa de aciertos
- Existen también otras medidas para clasificación, como Kappa o el área bajo la curva ROC (AUC), de las que hablaremos en el tema de curvas ROC
- ¿Qué medida usar en predicción numérica?

MEDIDAS PARA EVALUAR TAREAS DE PREDICCIÓN NUMÉRICA (REGRESIÓN)

- Valores reales: $\{y_1, \ldots, y_n\}$
- Valores predichos por el modelo: $\{p_1, \ldots, p_n\}$

MSE = Mean Squared Error
RMSE = Root-Mean Squared Error

$$MSE: \frac{(p_1 - y_1)^2 + \ldots + (p_n - y_n)^2}{n}; \quad RMSE = \sqrt{MSE}$$

- RMSE es muy común
- Tiene el problema de que aquellos datos muy mal predichos (o sea, $(p_i - y_i)$ grande) tienen mucho peso en la media.

MEDIDAS PARA EVALUAR TAREAS DE PREDICCIÓN NUMÉRICA (REGRESIÓN)

MAE = Mean Absolute Error

$$MAE : \frac{|p_1 - y_1| + ... + |p_n - y_n|}{n}$$

- Con el MAE, los datos muy mal predichos no tienen tanta influencia en la media como con RMSE.
- Pero tanto RMSE como MAE tienen el problema de que su valor es relativo a la escala de la variable de salida
  - Ejemplo: si multiplicamos las variables de salida por 1000, seguramente el valor de RMSE y MAE serán 1000 veces más grandes, sin que eso implique que el modelo es 1000 veces peor
- Esto no pasa en clasificación, donde el porcentaje de aciertos es una medida absoluta.

MEDIDAS PARA EVALUAR TAREAS DE PREDICCIÓN NUMÉRICA (REGRESIÓN)

- Podemos construir medidas relativas, en las que comparamos nuestro modelo con el modelo más sencillo posible: realizar la predicción con una constante, la media de la variable de salida.
- Son números típicamente entre 1 (el modelo acierta igual que la media) y 0 (el modelo no comete errores). Aunque pueden ser mayores que 1 si el modelo es muy malo.

RSE = Relative Squared Error
RRSE = Root Relative Squared Error
RAE = Root Absolute Error

$$RSE: \frac{(p_1 - y_1)^2 + \ldots + (p_n - y_n)^2}{(\bar{y} - y_1)^2 + \ldots + (\bar{y} - y_n)^2}; \quad RRSE = \sqrt{RSE}; \quad \bar{y} = \frac{y_1 + \ldots + y_n}{n}$$

$$RAE = \frac{|p_1 - y_1| + \ldots + |p_n - y_n|}{|\bar{y} - y_1| + \ldots + |\bar{y} - y_n|}$$

Nota: el coeficiente de determinación R^2 = 1- RSE

AJUSTE DE HIPER-PARÁMETROS

HIPER-PARAMETROS

- Se denomina hiper-parámetros, a los parámetros del algoritmo o método que construye los modelos.
  - Ej: número de vecinos (k) en *k-nn*.
  - Ej: profundidad máxima del árbol en árboles.
- Dependiendo de su valor, el algoritmo generará unos modelos u otros, algunos con errores más altos y otros con errores más bajos.
- Es por tanto importante encontrar el valor del (o de los hiper-parámetros) que produzcan los modelos con menor error.
- En principio lo tiene que hacer el usuario, pero aquí estudiaremos métodos que automatizan el proceso.

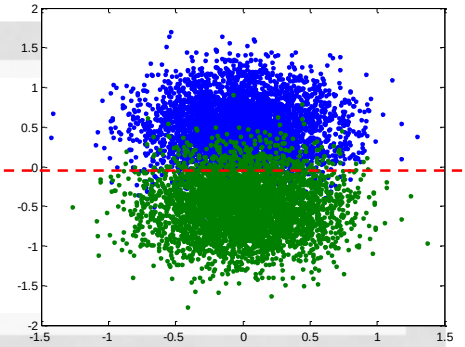
RELACIÓN ENTRE HIPER-PARÁMETROS Y COMPLEJIDAD DEL MODELO

- Supongamos que disponemos de un método de entrenamiento de modelos que es capaz de ajustar polinomios.
- Es decir, la frontera de separación viene dada por un polinomio.
- Pero el método necesita que le digamos el grado del polinomio (este va a ser el hiper-parámetro del algoritmo).

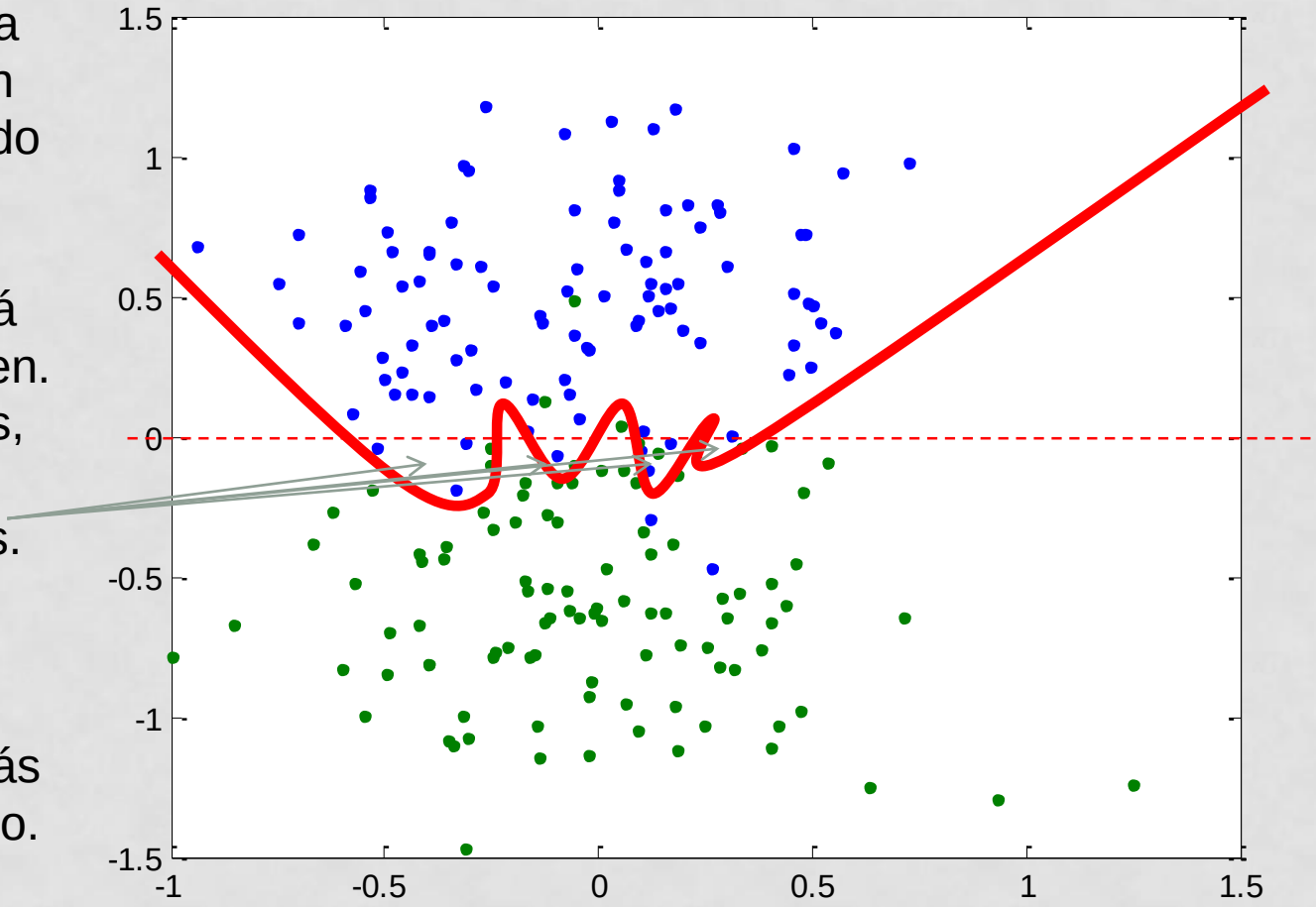
Grado 0: $g(x) = a$
Grado 1: $g(x) = b\,x + a$
Grado 2: $g(x) = c\,x^2 + b\,x + a$
Grado 3: $g(x) = d\,x^3 + c\,x^2 + b\,x + a$

- El grado es el **hiper-parámetro**.
  - El método no sabe encontrar su mejor valor, tenemos que dárselo nosotros
- a, b, c, d, … son los parámetros del modelo concreto.

Complejidad del modelo

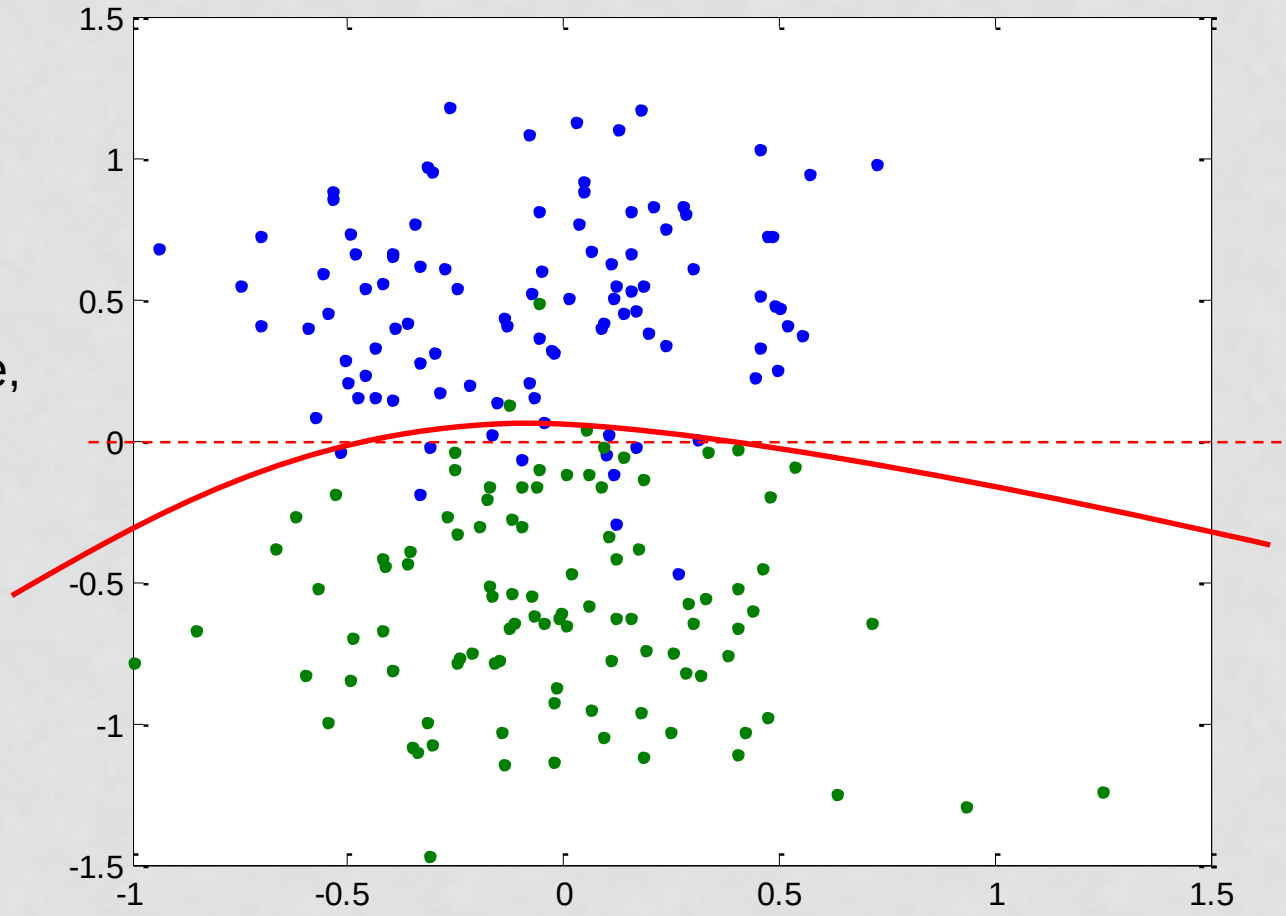


- Este ajuste podría producirse con un polinomio de grado 7 (“6 cambios”).
- El modelo no está generalizando bien. Con datos futuros, cometerá errores en algunas zonas.
- Claramente, en esas zonas, está memorizando, más que generalizando.



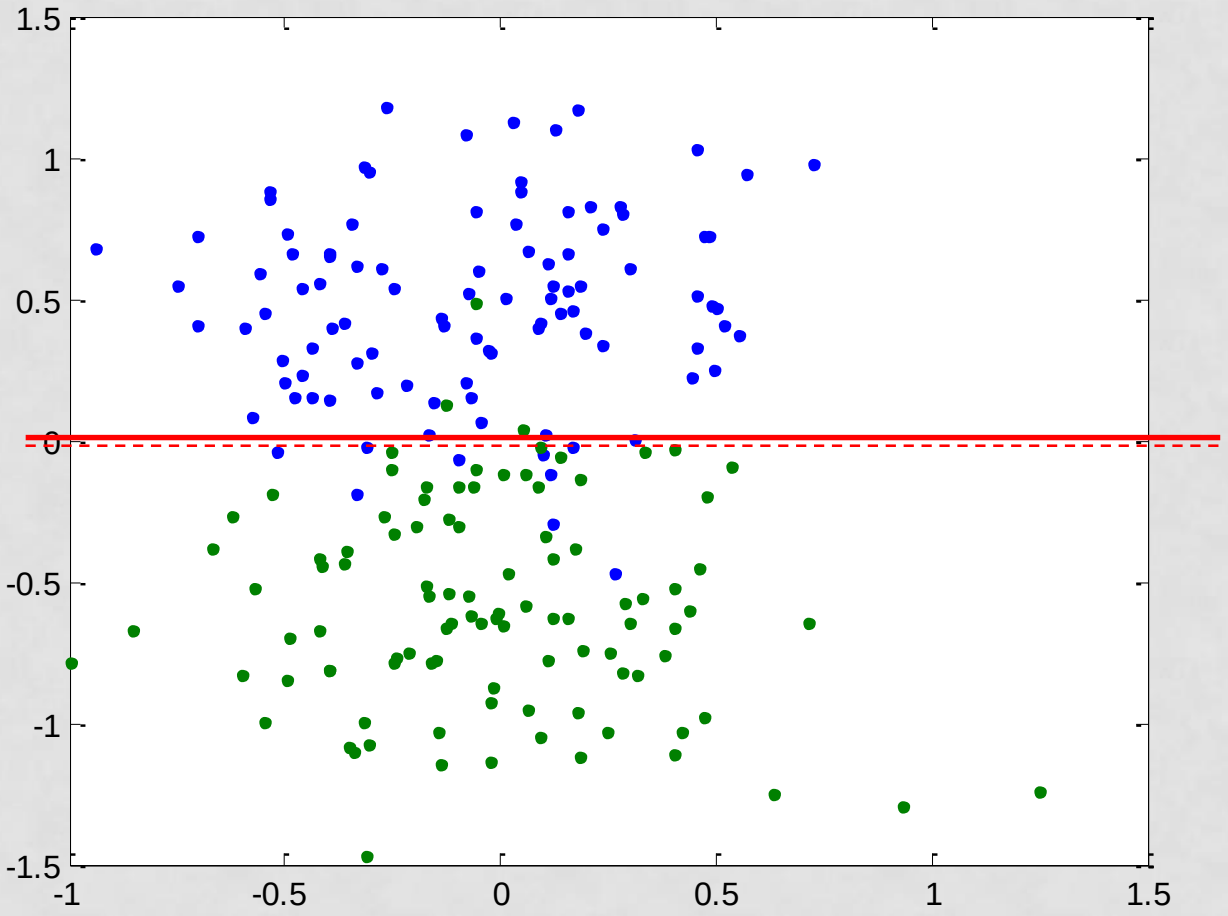
Complejidad del modelo y sobreajuste

- Este ajuste podría producirse con un polinomio de grado 2 (parábola)
- Sigue sin ser el modelo subyacente, pero al ser más simple, no tiene tanta capacidad para sobreajustar.



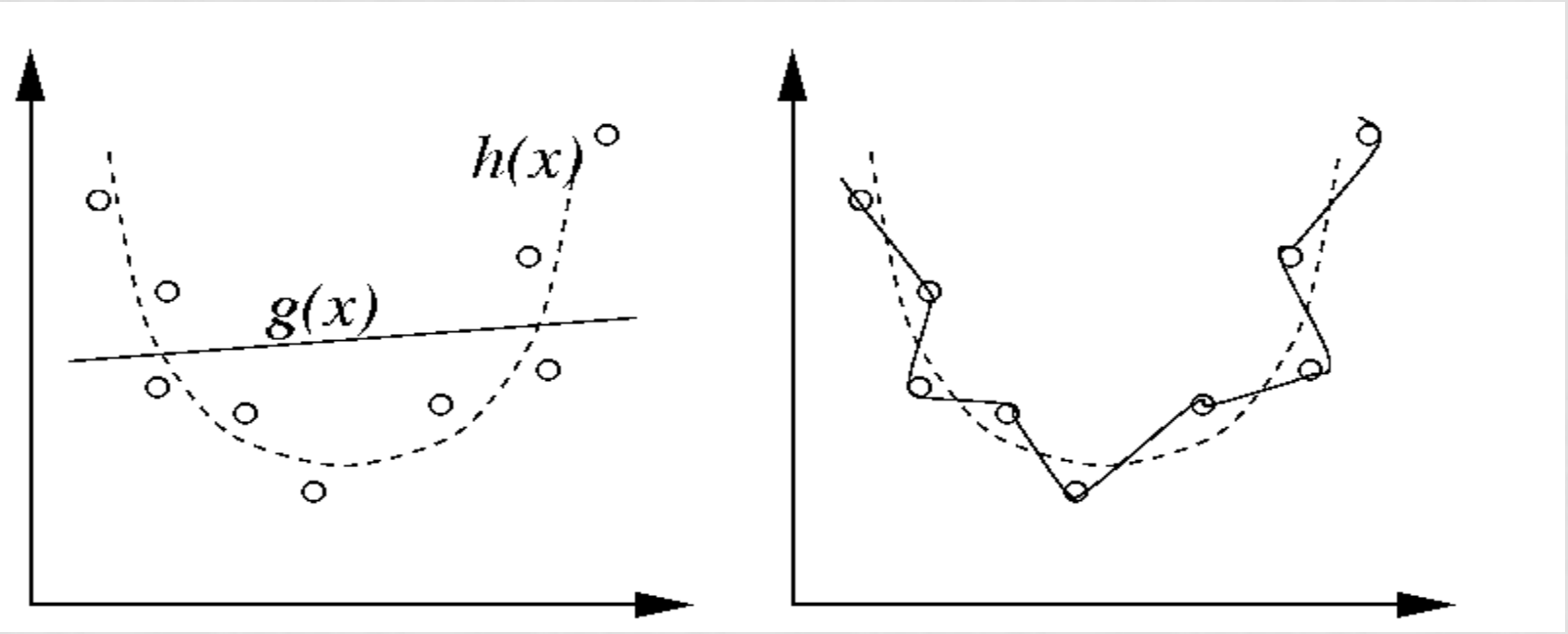
Complejidad del modelo y sobreajuste

- Con un polinomio de grado 1 (recta), prácticamente acierta con el modelo subyacente.

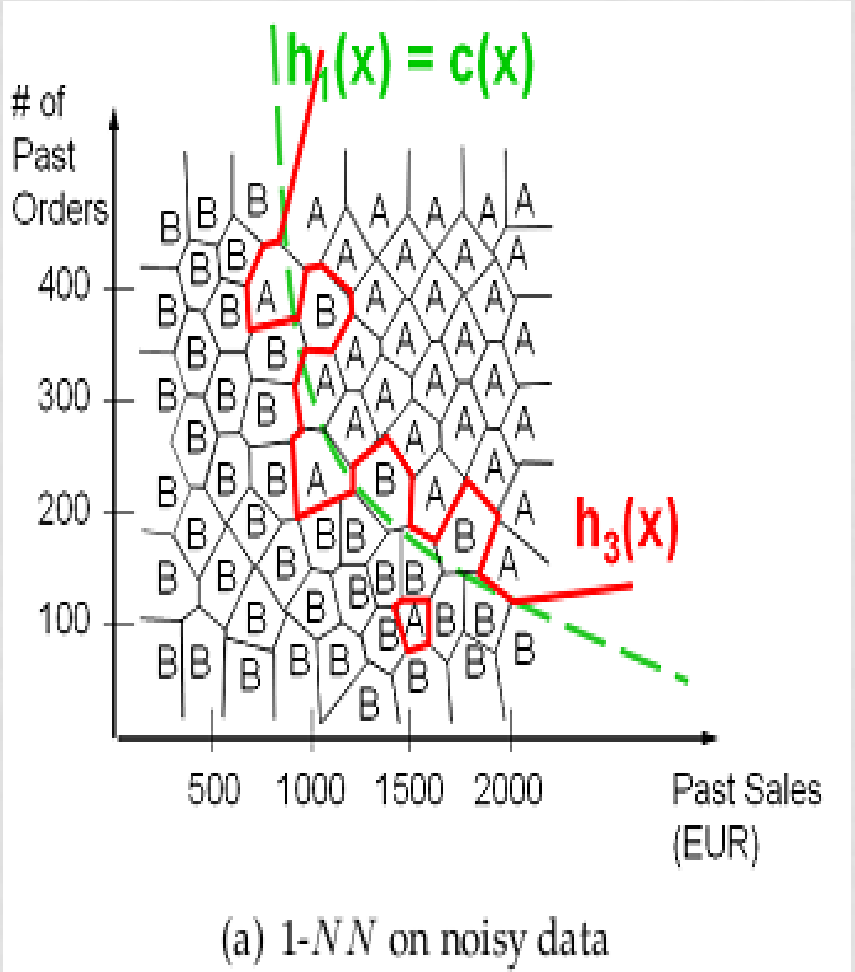


Sobreadaptación en regresión

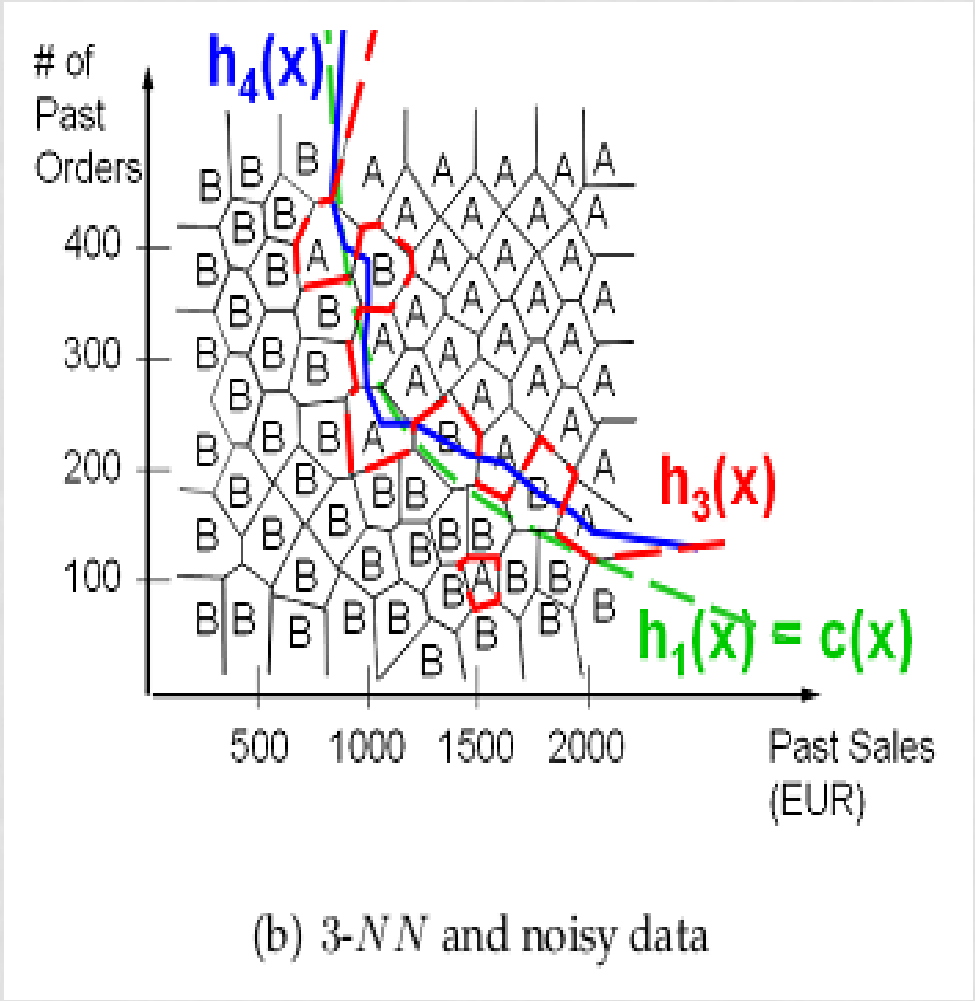
- Derecha: el modelo se ha sobreadaptado al ruido porque es demasiado complejo
- Izquierda: el modelo lineal **g(x)** es demasiado simple para aproximar una parábola y subadapta los datos
- Conclusión: es necesario ajustar bien los hiper-parámetros, los cuales determinan (de manera directa o indirecta) la complejiad del modelo.



HIPERPARÁMETRO K EN KNN



(a) 1-NN on noisy data

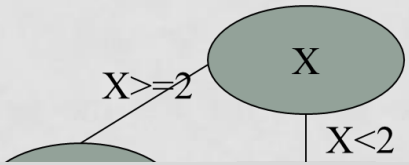


(b) 3-NN and noisy data

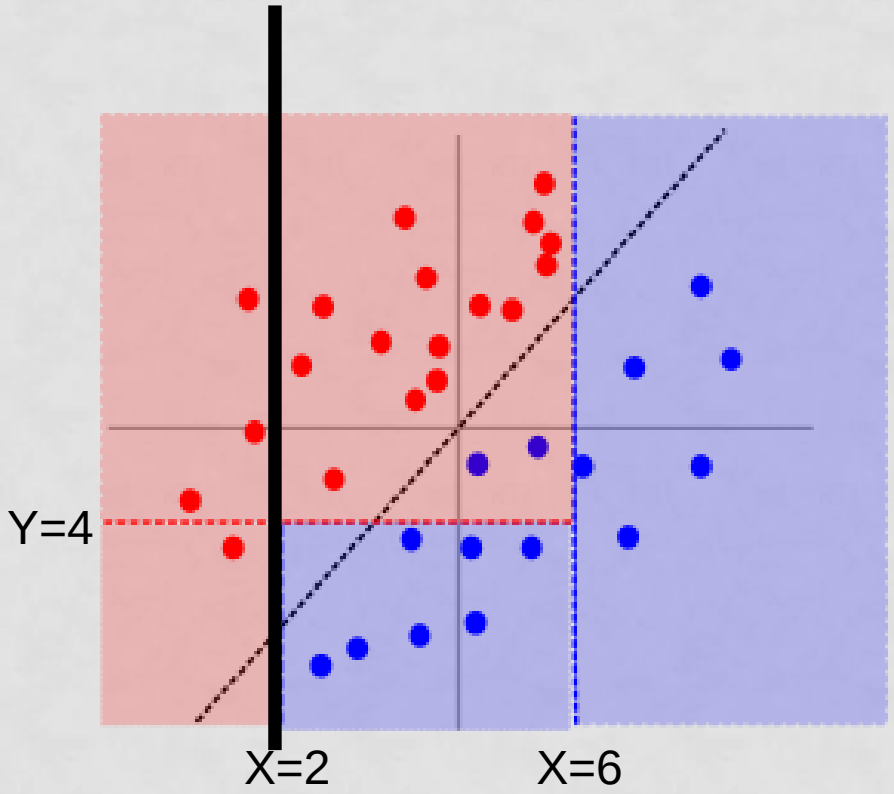
Source: https://en.wikipedia.org/wiki/Voronoi_diagram

HIPER-PARÁMETRO PROFUNDIDAD EN C4.5

- Con profundidad 1, la frontera es una recta.

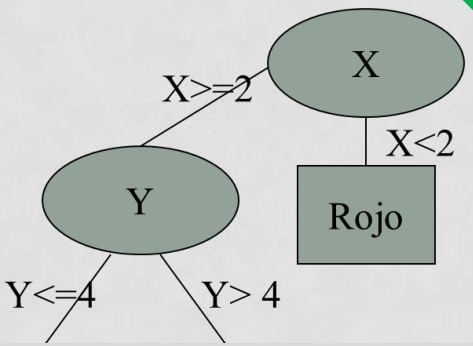


source: https://stats.stackexchange.com/questions/262930/can-a-decision-tree-recreate-the-exact-same-classification-as-a-nearest-neighbor

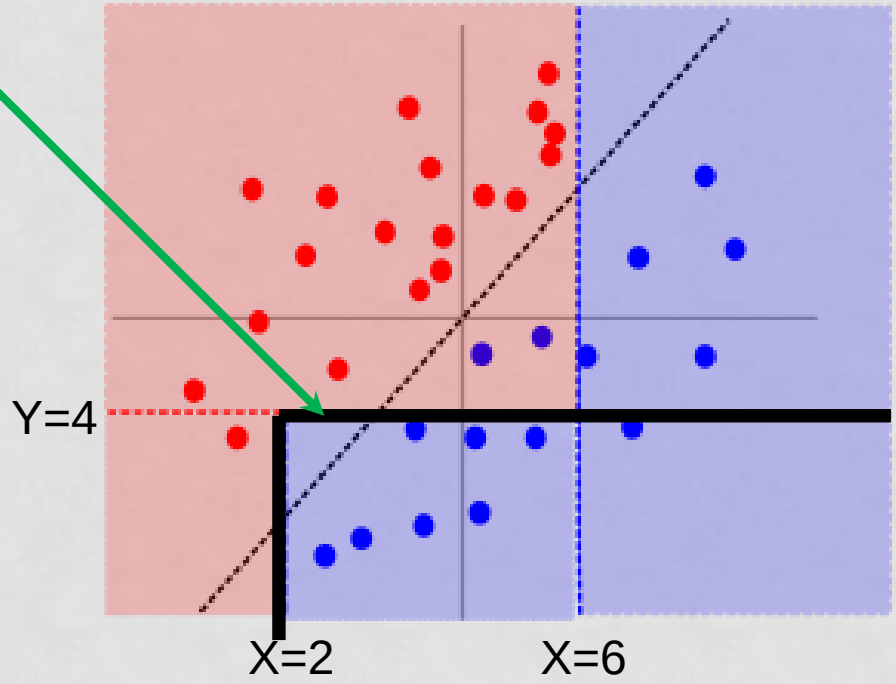


HIPER-PARÁMETRO PROFUNDIDAD EN C4.5

- Con profundidad 2

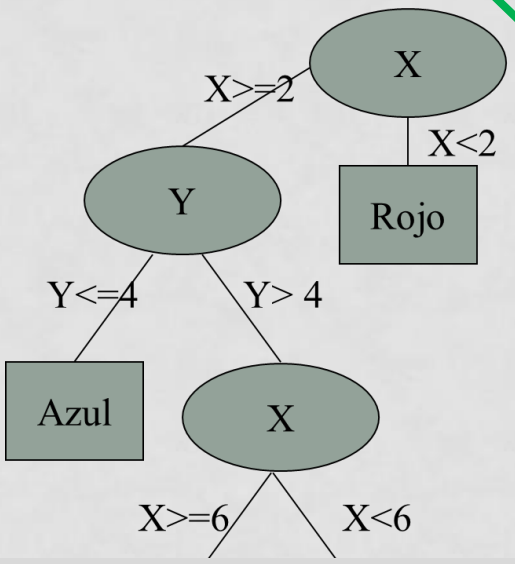


source: https://stats.stackexchange.com/questions/262930/can-a-decision-tree-recreate-the-exact-same-classification-as-a-nearest-neighbor

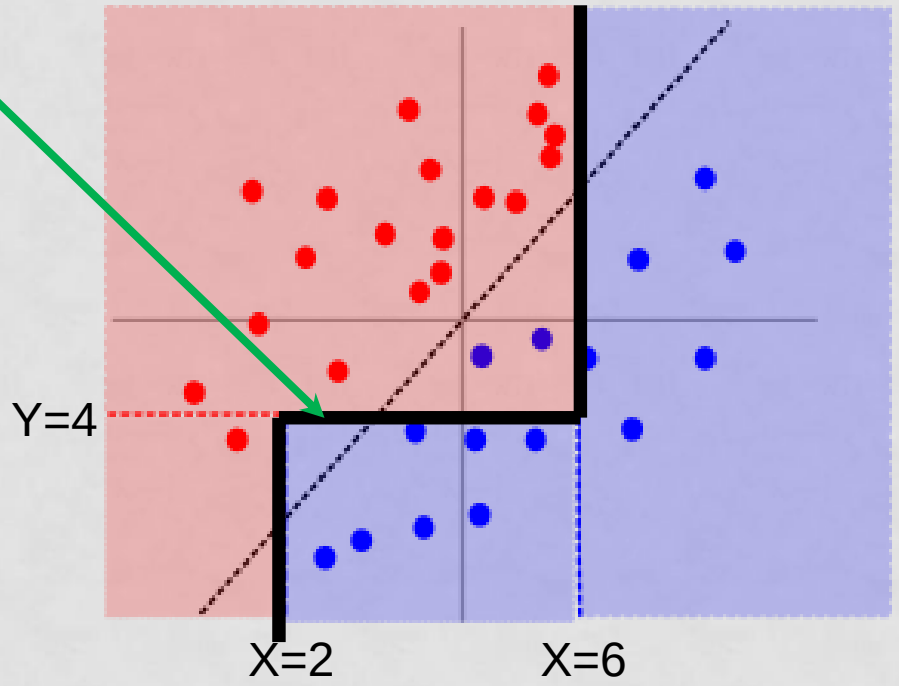


HIPER-PARÁMETRO PROFUNDIDAD EN C4.5

- Con profundidad 3, la frontera es más complicada.
- Etc.



source: https://stats.stackexchange.com/questions/262930/can-a-decision-tree-recreate-the-exact-same-classification-as-a-nearest-neighbor



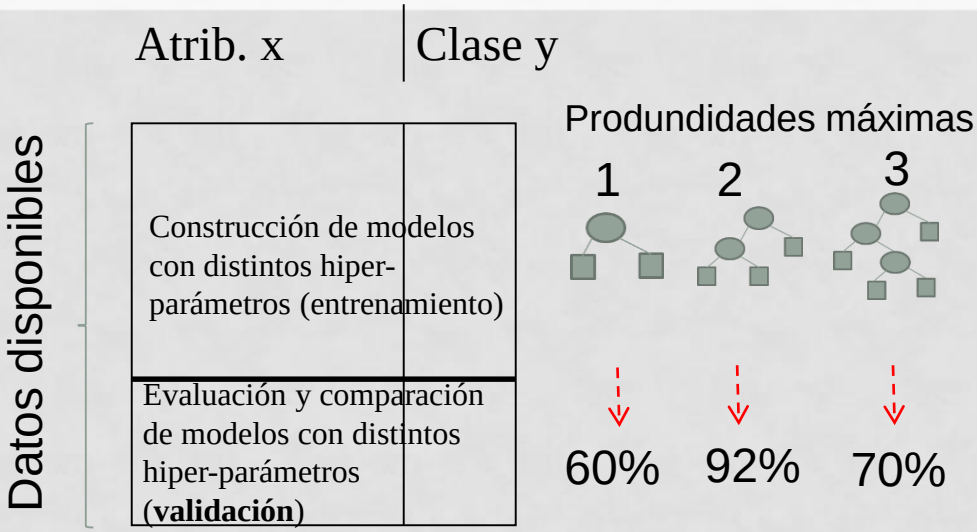
Hiperparámetros

- KNN:
  - K = número de vecinos
  - ¿Usamos ponderación por el inverso de la distancia 1/d?
- Árboles de decisión:
  - Número mínimo de instancias necesario para subdividir
  - Número de nodos
  - Profundidad máxima
  - …
- Polinomios:
  - grado
- Redes de neuronas:
  - Número de neuronas ocultas
  - Ciclos de entrenamiento
  - Learning rate
- SVMs:
  - Coste (C)
  - Desviación (σ)
- No todas las implementaciones de un algoritmo tienen los mismos hiper-parámetros.

AJUSTE DE HIPER-PARÁMETROS

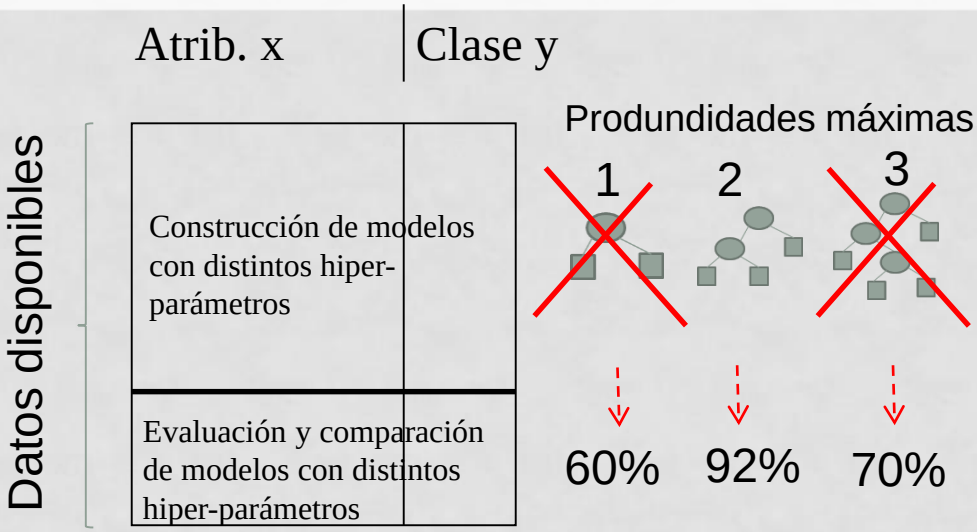
- ¿Cómo encontrar el mejor valor para el hiper-parámetro? (ej: ¿cuál es la profundidad máxima óptima para un árbol de decisión?)
- Idea: construir varios árboles con distintas profundidades, evaluar cada árbol, y quedarse con el mejor.
- Supongamos que tenemos unos datos disponibles. Podemos partirlos en dos partes:
  - Una para construir los distintos modelos con distintas profundidades (entrenamiento)
  - Otra para evaluar y comparar los distintos modelos (**conjunto de validación o de "desarrollo" (dev**elopment))

AJUSTE DE HIPER-PARÁMETROS



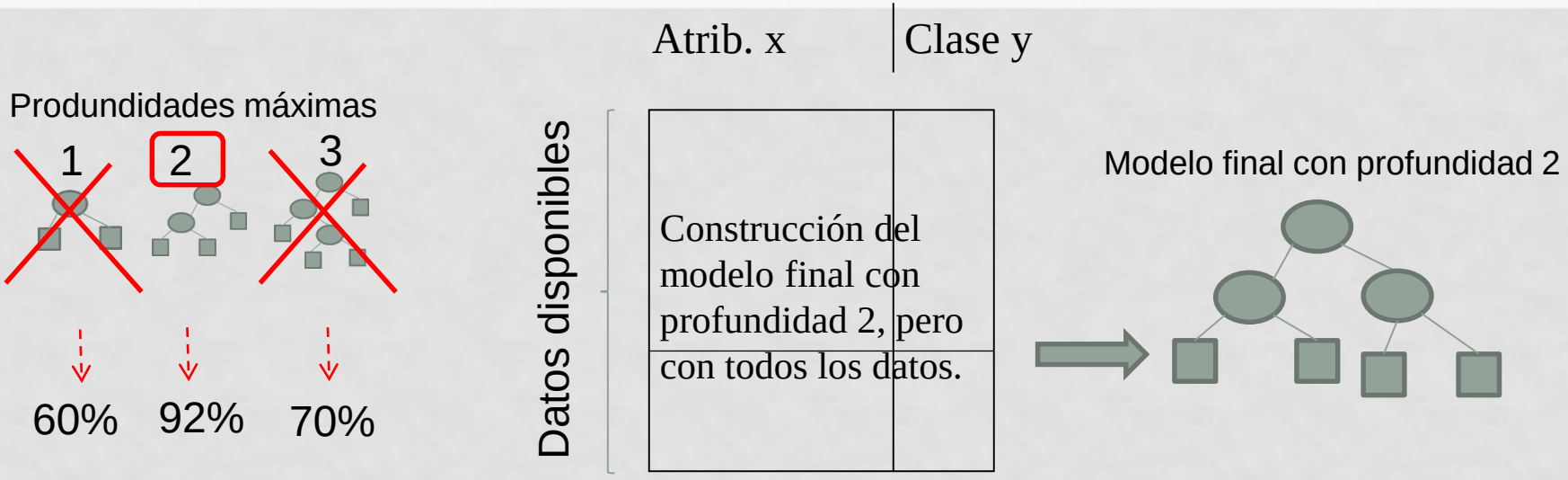
- La mejor profundidad máxima es 2 (92% de aciertos),

AJUSTE DE HIPER-PARÁMETROS



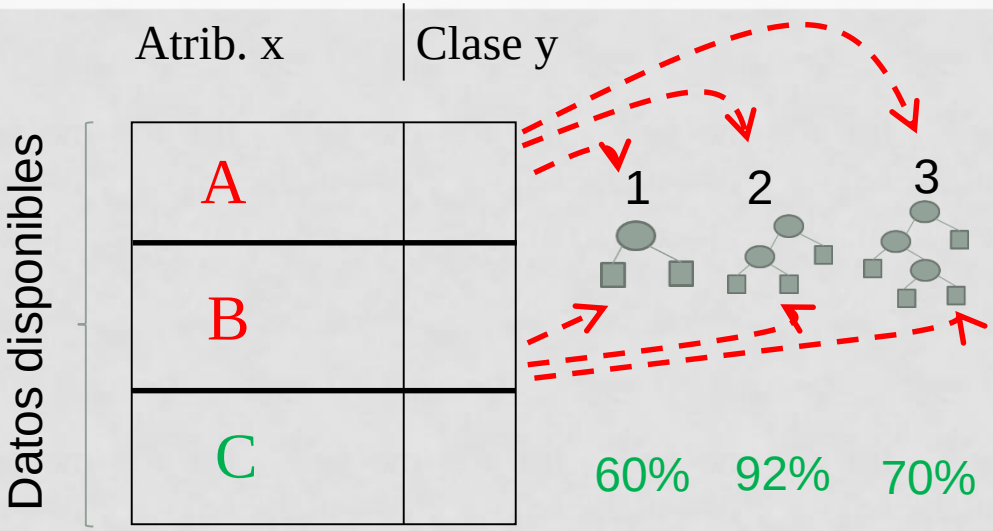
- La mejor profundidad máxima es 2 (92% de aciertos), así que ahora construiremos un árbol de profundidad 2 con **TODOS** los datos.
- ¿Por qué? Una vez que hemos estimado que la profundidad óptima es 2, es mejor usar la mayor cantidad de información posible para construir el modelo final.

AJUSTE DE HIPER-PARÁMETROS



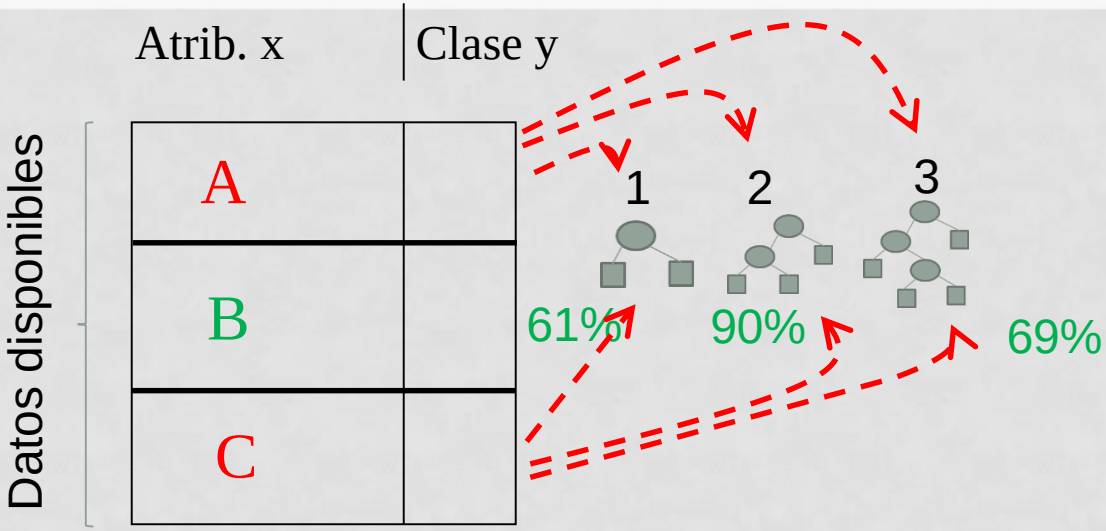
- La mejor profundidad máxima es 2 (92% de aciertos), así que ahora construiremos un árbol de profundidad 2 con **TODOS** los datos.
- ¿Por qué? Una vez que hemos estimado que la profundidad óptima es 2, es mejor usar la mayor cantidad de información (datos) posible para construir el modelo final.

AJUSTE DE HIPER-PARÁMETROS CON VALIDACIÓN CRUZADA



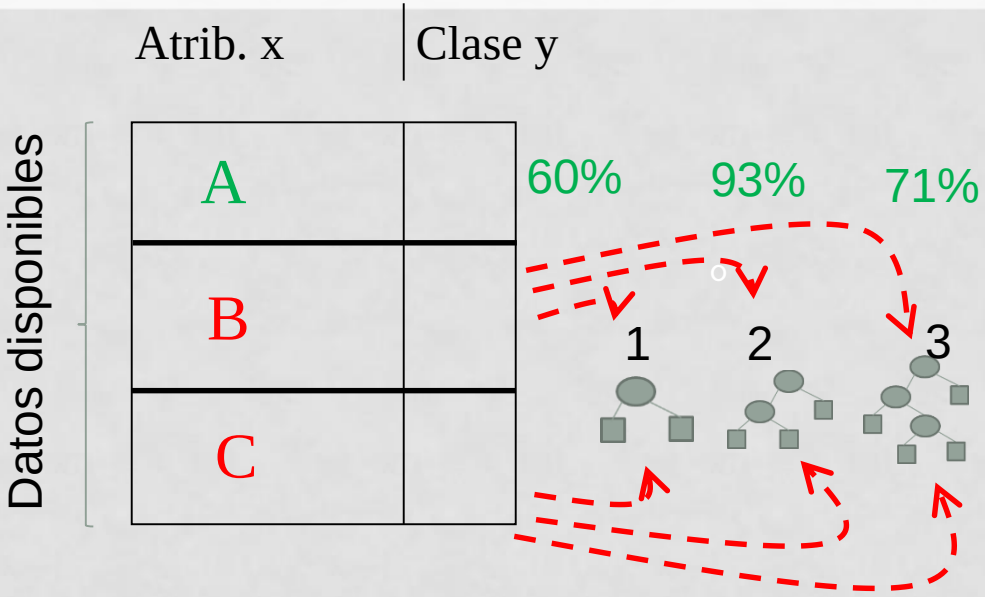
- También podemos usar validación cruzada para estimar la mejor profundidad máxima.
- Es más costoso, pero más preciso.
- Primero entrenamos con A y B, y validamos con C

AJUSTE DE HIPER-PARÁMETROS CON VALIDACIÓN CRUZADA



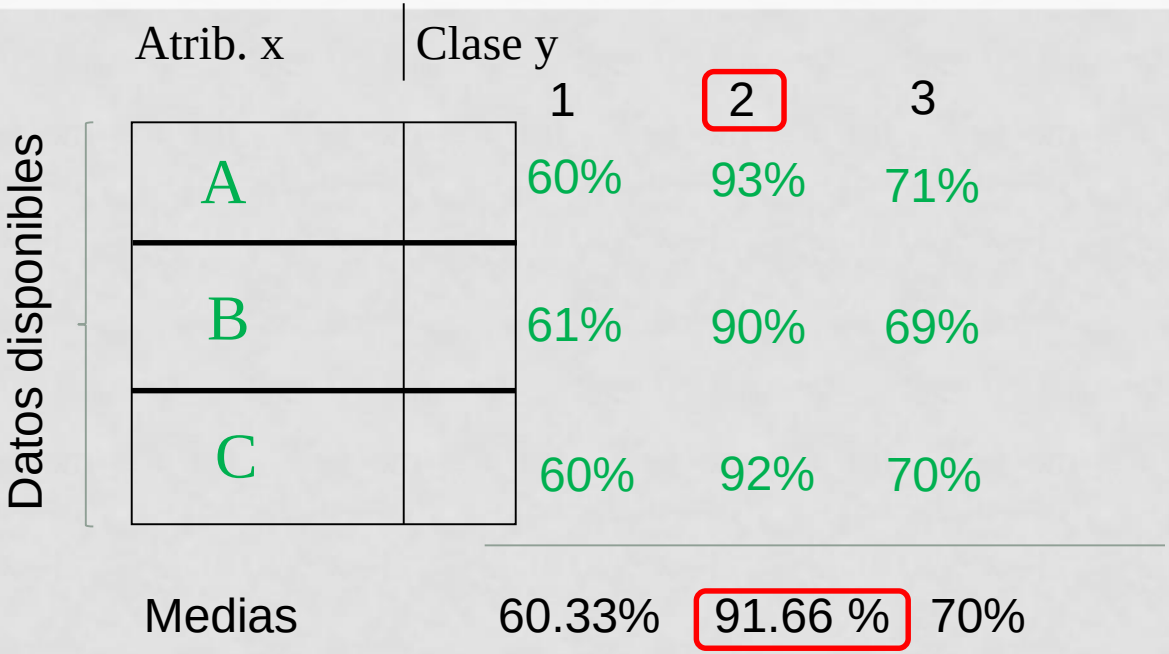
- Después entrenamos con A y C, y validamos con B

AJUSTE DE HIPER-PARÁMETROS CON VALIDACIÓN CRUZADA



- Finalmente, entrenamos con B y C, y validamos con A

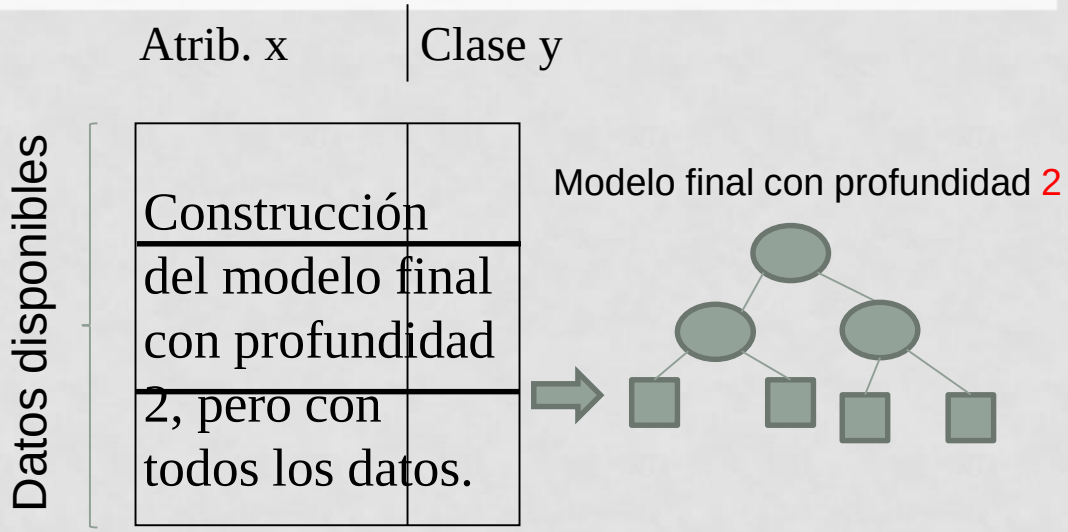
AJUSTE DE HIPER-PARÁMETROS CON VALIDACIÓN CRUZADA



- Por último, estimamos la calidad de cada una de las profundidades haciendo la media de los tres folds.
- La mejor es profundidad 2.

AJUSTE DE HIPER-PARÁMETROS CON VALIDACIÓN CRUZADA

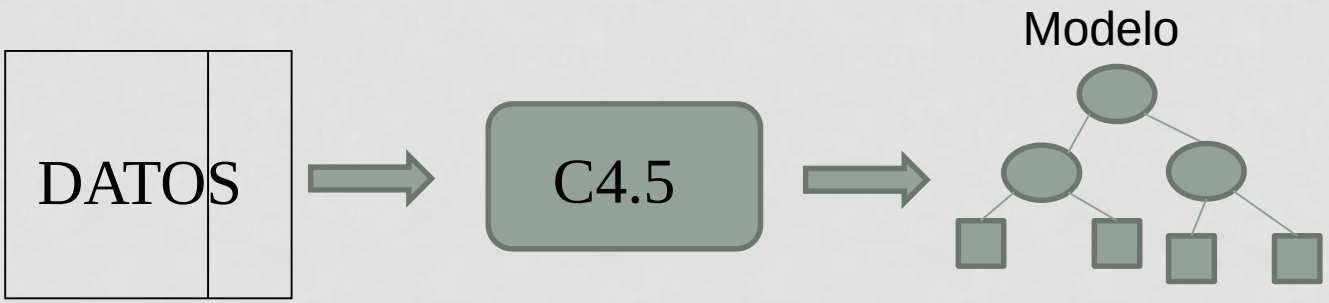
1	2	3
60%	93%	71%
61%	90%	69%
60%	92%	70%
60.33%	91.66 %	70%



- Para terminar, construimos el modelo final con **todos los datos** y profundidad 2.

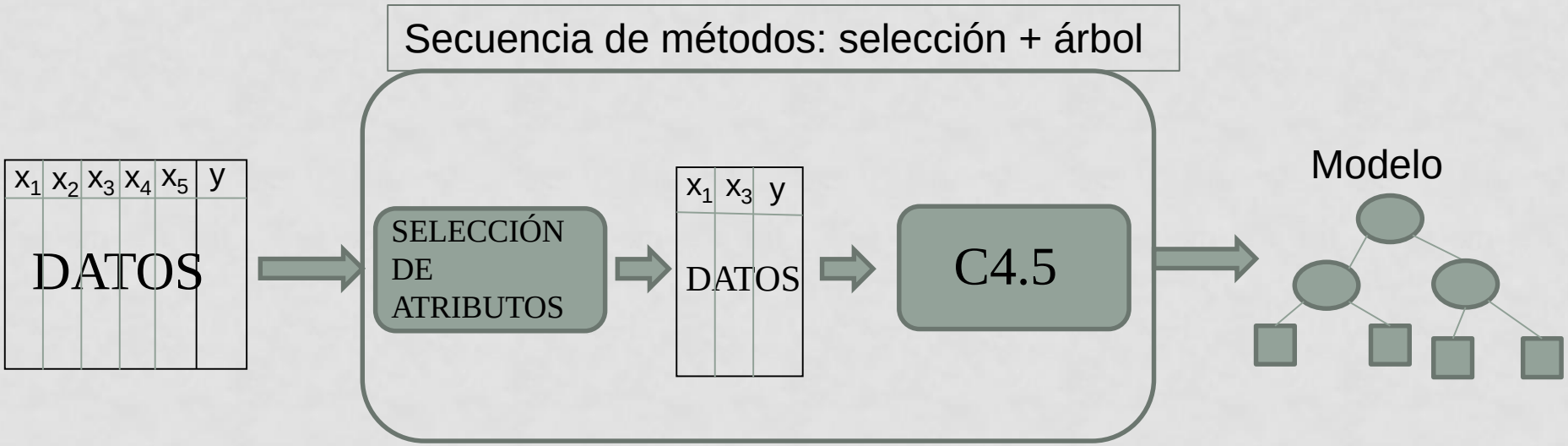
SECUENCIAS DE MÉTODOS

- La manera de programar o implementar el ajuste de parámetros es con una "secuencia de métodos" (o algoritmos)
  - Que también se puede llamar "wrapped learner" (en MLR) o "pipeline"
- Por ejemplo, el algoritmo de construcción de árboles C4.5 (o C5.0) que vimos en clase y que construye árboles de decisión sería un método "puro".



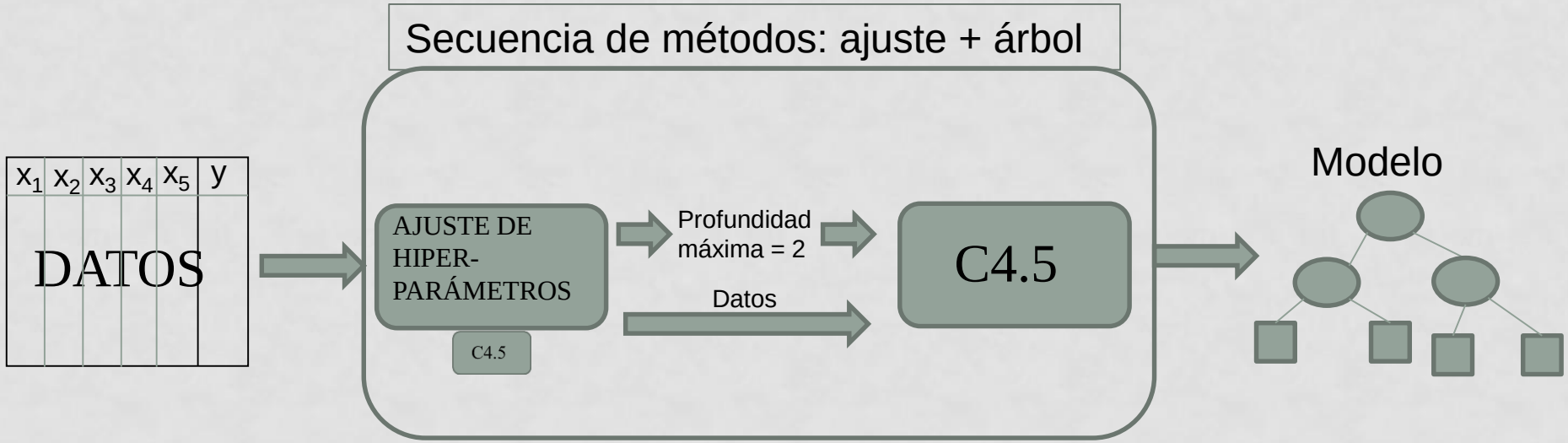
EJEMPLO DE SECUENCIA DE MÉTODOS

- Podemos usar una secuencia de métodos que haga dos cosas:
  - Primero selecciona los atributos más relevantes
  - Después construye el modelo con dichos atributos
- Sigue siendo un método que toma unos datos y devuelve un modelo, como todos.



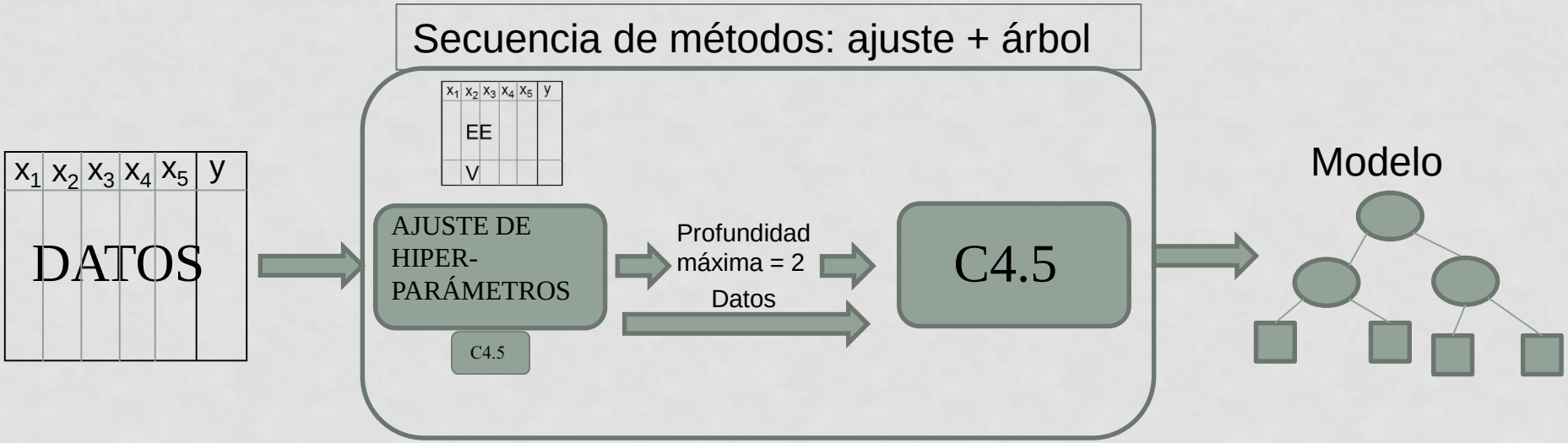
AJUSTE DE PARÁMETROS COMO UNA SECUENCIA DE MÉTODOS

- Ajuste de hiper-parámetros:
  - Primero selecciona los mejores hiper-parámetros
  - Después construye el modelo con dichos hiper-parámetros
- Sigue siendo un método que toma unos datos y devuelve un modelo, como todos.



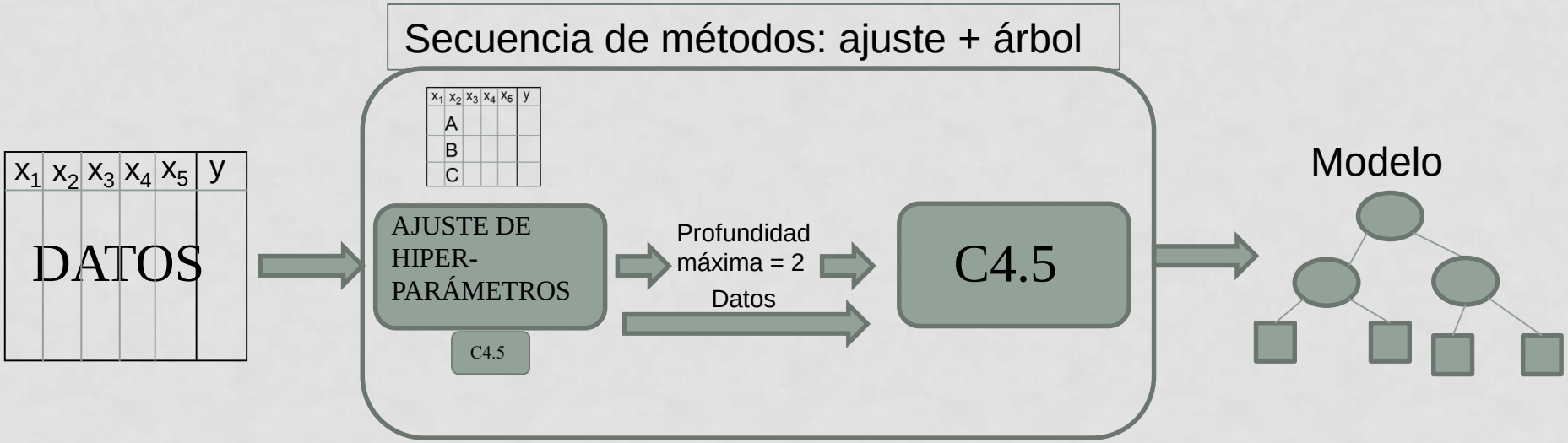
AJUSTE DE PARÁMETROS COMO UNA SECUENCIA DE MÉTODOS

- El ajuste de hiper-parámetros dentro de la secuencia de métodos se puede hacer de dos maneras:
  - **Con entrenamiento y validación**
  - Con validación cruzada



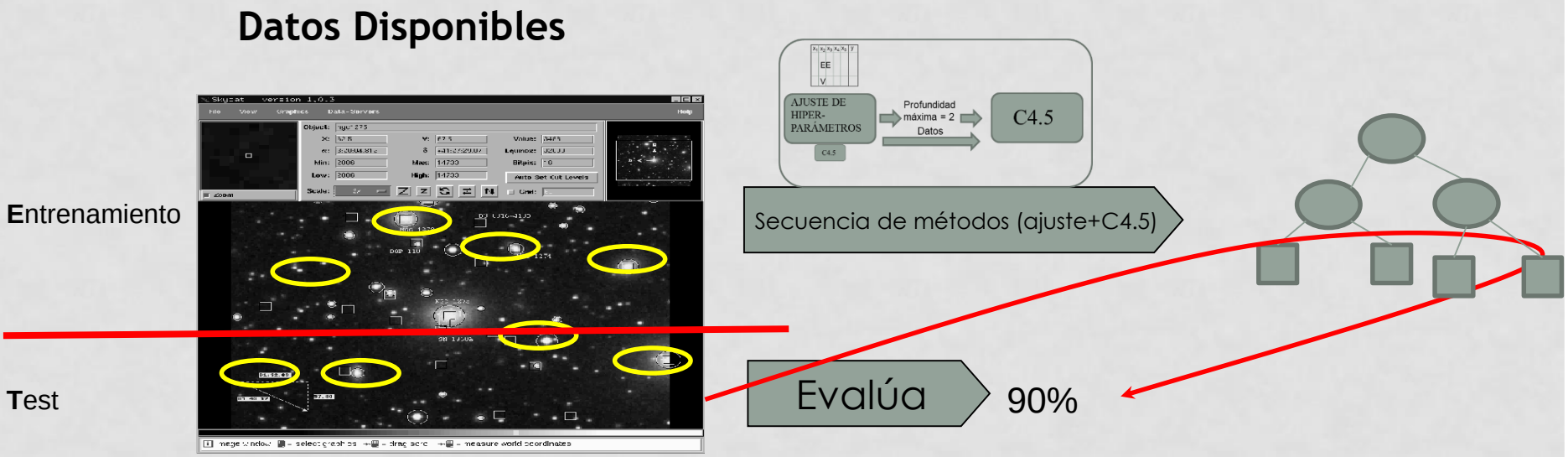
AJUSTE DE PARÁMETROS COMO UNA SECUENCIA DE MÉTODOS

- El ajuste de hiper-parámetros dentro de la secuencia de métodos se puede hacer de dos maneras:
  - Con entrenamiento y validación
  - **Con validación cruzada**



EVALUACIÓN DE UNA SECUENCIA DE MÉTODOS

- Una secuencia de métodos puede ser evaluada como cualquier método, de dos maneras:
  - **Con entrenamiento y test**
  - Con validación cruzada



Evaluación con validación cruzada con 3 folds:

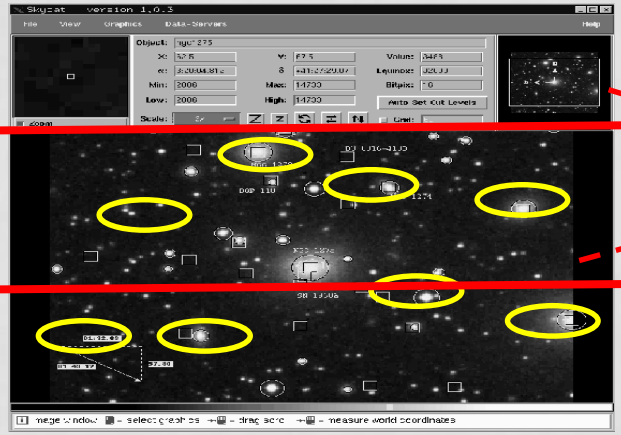
Entrenamos con X e Y, evaluamos con Z

Datos Disponibles

Fold X

Fold Y

Fold Z

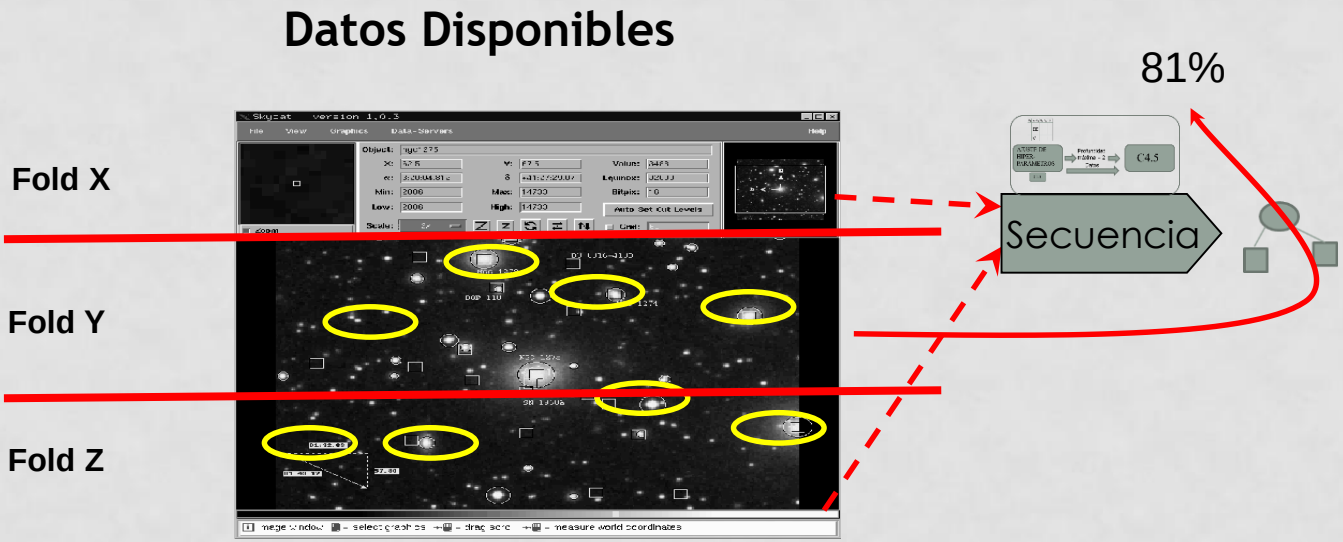


Secuencia

80%

Evaluación con validación cruzada con 3 folds:

Entrenamos con X, Z; evaluamos con Y



Evaluación con validación cruzada con 3 folds:

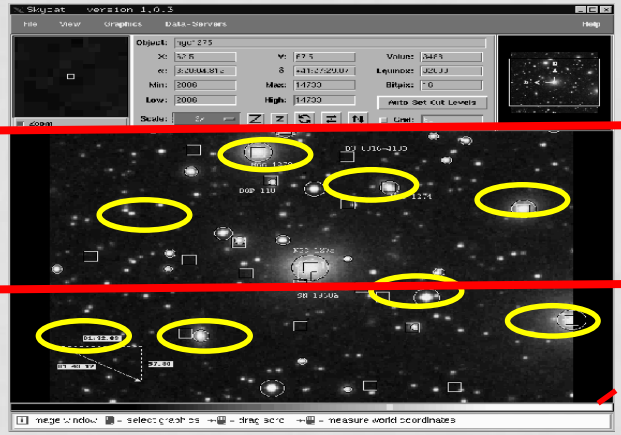
Entrenamos con Y, Z; evaluamos con X

Datos Disponibles

Fold X

Fold Y

Fold Z



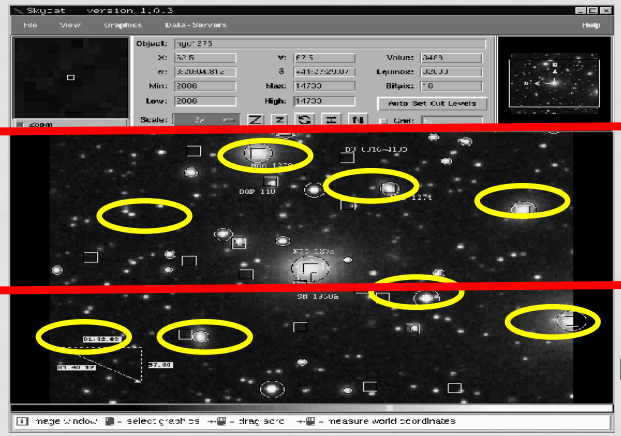
Secuencia

78%

Evaluación con validación cruzada con 3 folds:

Calculamos la evaluación T como la media de los tres folds

Datos Disponibles



Fold X

Fold Y

Fold Z

80%

81%

78%

Evaluación

T= (80%+81%+78%)/3 = 79.7%

Evaluación con validación cruzada con 3 folds:

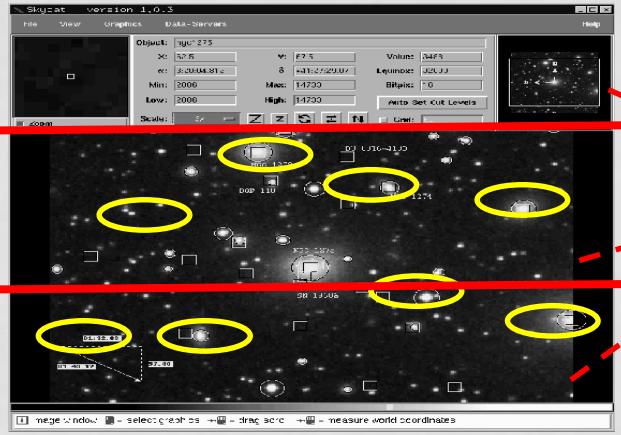
Construcción del modelo final con todos los datos

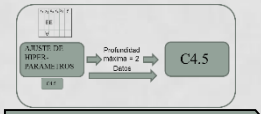
Datos Disponibles

Fold X

Fold Y

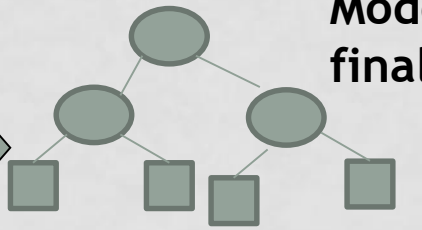
Fold Z



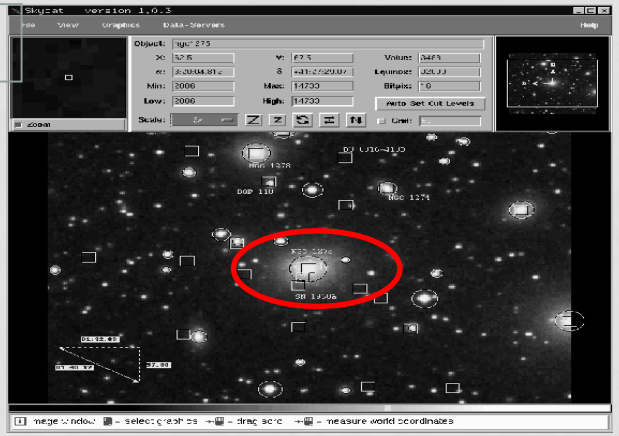


Secuencia de métodos

Modelo final



Evaluación con validación cruzada con 3 folds:



?

Uso del modelo

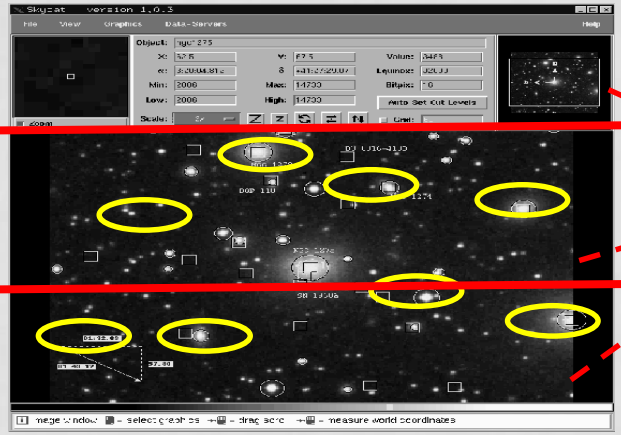
Uso final del modelo

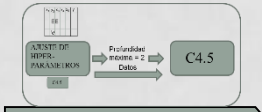
Datos Disponibles

Fold X

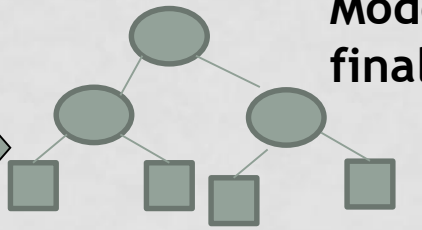
Fold Y

Fold Z





Secuencia de métodos



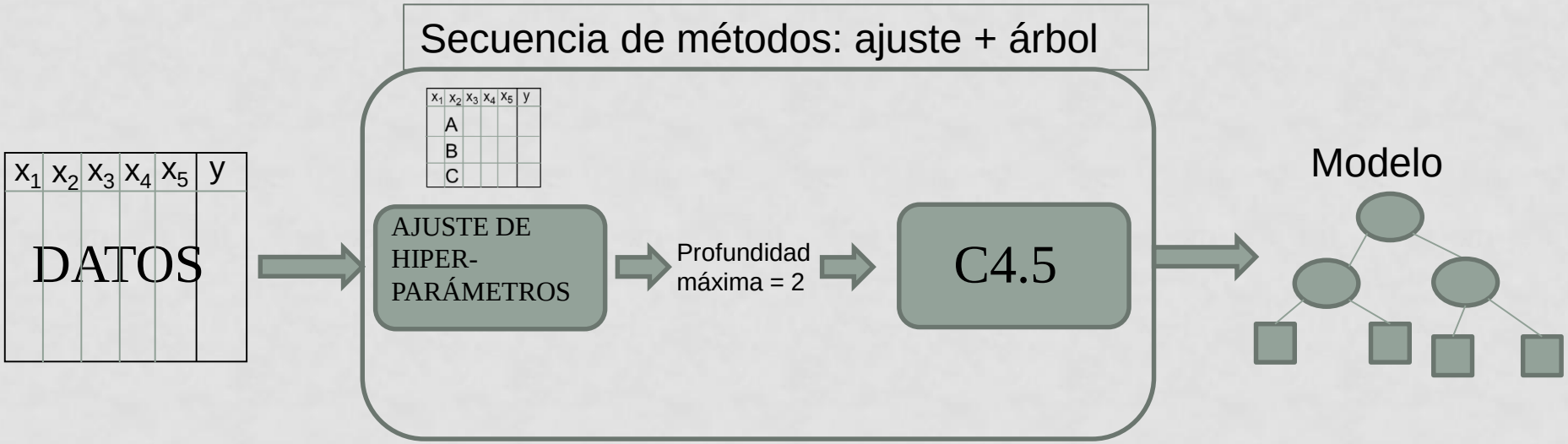
Modelo final

Galaxia espiral

Evaluación T = 79.7%

AJUSTE DE PARÁMETROS COMO SECUENCIA DE MÉTODOS

- En resumen, una secuencia de métodos, internamente puede usar train/validación o validación cruzada para elegir los mejores hiper-parámetros.
- Pero la evaluación de la secuencia de métodos se puede hacer también con train/test o validación cruzada
- Nótese que la validación cruzada para el ajuste de hiper-parámetros y la validación cruzada para la evaluación del método, **son diferentes validaciones cruzadas**.



AJUSTE DE HIPER-PARÁMETROS

- Si hay más de un hiper-parámetro, se sigue el procedimiento sistemático (y bastante costoso computacionalmente) denominado **grid-search** (búsqueda en rejilla).

GRID SEARCH (BÚSQUEDA EN REJILLA)

maxdepth	2	4	6	8
minsplit				
2	(2,2)	(2,4)	(2,6)	(2,8)
4	(4,2)	(4,4)	(4,6)	(4,8)
6	(6,2)	(6,4)	(6,6)	(6,8)

Grid search: probar todas las posibles combinaciones de valores de hiper-parámetros. Para cada una, hacer una validación cruzada y obtener el porcentaje de aciertos. Seleccionar la mejor combinación de hiper-parámetros.

maxdepth	2	4	6	8
minsplit				
2	70%	75%	76%	68%
4	72%	73%	**81%**	70%
6	68%	70%	71%	67%

GRID SEARCH

```
for(maxdepth in c(2,4,6,8)){
  for(minsplit in c(2,4,6)){
    model = train(train_set, maxdepth, minsplit)
    evaluation = "evaluate model with validation set"
  }
}
"Return (maxdepth, minsplit) of model with best evaluation"
```

RANDOM SEARCH

maxdepth	2	4	6	8
minsplit				
2	(2,2)	(2,4)	(2,6)	(2,8)
4	(4,2)	(4,4)	(4,6)	(4,8)
6	(6,2)	(6,4)	(6,6)	(6,8)

Random search: probar **aleatoriamente** sólo algunas de las combinaciones (4, en este caso). Es conveniente hacer esto cuando el número de combinaciones de hiper-parámetros es enorme.

maxdepth	2	4	6	8
minsplit				
2	70%	**75%**	76%	68%
4	72%	73%	81%	70%
6	68%	70%	71%	67%

RANDOM SEARCH

```
budget = 100 # El budget es la cantidad máxima de combinaciones de hiperparámetros que queremos probar
while(budget>0){
   budget = budget – 1 # Ahora tenemos menos budget
   (maxdepth, minsplit) = “get a random combination of hiper-parameter values”
    model = train(train_set, maxdepth, minsplit)
    evaluation <- "evaluate model with validation set"
  }}
"Return (maxdepth, minsplit) of model with best evaluation"
```

BÚSQUEDA ALEATORIA DE HIPER-PARÁMETROS

- Grid-search es obviamente muy costoso computacionalmente.
- Random search permite controlar la cantidad de cómputo destinada a la búsqueda de hiper-parámetros (*budget*), a costa de ser menos exhaustiva.
- Empíricamente, parece funcionar mejor Random Search
- Métodos avanzados: ejemplo, sequential model-based optimization

DEFINICIÓN DE ESPACIOS DE BÚSQUEDA ("REJILLAS")

- Ejemplo de "rejilla" (espacio de búsqueda) con dos hiper-parámetros

maxdepth	**2**	**4**	**6**	**8**
minsplit				
2	(2,2)	(2,4)	(2,6)	(2,8)
4	(4,2)	(4,4)	(4,6)	(4,8)
6	(6,2)	(6,4)	(6,6)	(6,8)

- Los espacios de búsqueda pueden ser más complicados

DEFINICIÓN DE ESPACIOS DE BÚSQUEDA DE HIPER-PARÁMETROS

- Se ve con la sintaxis de MLR, pero es bastante general
- Tipos de hiper-parámetros:
  - Discretos: se especifican los valores concretos. Por ejemplo probar con k= 1, 3, 5, 7
    - `makeDiscreteParam("k", values = c(1, 3, 5, 7))`
  - Enteros: se especifica un rango. Por ejemplo, probar con k entre 2 y 10
    - `makeIntegerParam("k", lower=2, upper=100)`
    - En el caso de RandomSearch, se hará un muestreo de valores en ese rango.
    - En el caso de GridSearch, se puede especificar el paso (o *resolution*). Por ejemplo, con *resolution*=3, se probarán 2, 5, 8, …

DEFINICIÓN DE ESPACIOS DE BÚSQUEDA DE HIPER-PARÁMETROS

- Tipos de hiper-parámetros:
  - Reales: se especifica un rango
    - `makeNumericParam("C", lower=0.0, upper=1.0)`
  - A veces nos interesa que los valores a probar no sean los que se muestrean sino otros transformados (trafo). Ejemplos:
    - Para probar con valores de k impares:
      - `makeIntegerParam("k", lower=2, upper=100, trafo = function(x) 2*x+1)`
    - Para probar con valores de C exponenciales:
      - `makeNumericParam("C", lower=0.0, upper=1.0, trafo = function(x) 2^x)`
  - También hay hiper-parámetros de tipo "cadena de caracteres" (*makeCharacterParam*), pero los veremos cuando sean necesarios

DEFINICIÓN DE ESPACIOS DE BÚSQUEDA DE HIPER-PARÁMETROS

- Para terminar, aquí vemos un espacio de búsqueda con dos hiper-parámetros, para árboles de decisión.

```
ps = makeParamSet(
  makeDiscreteParam("maxdepth", values = c(2, 4, 6, 8)),
  makeDiscreteParam("minsplit", values = c(2, 4, 6))
)
```

ESPACIOS POR OMISIÓN (DEFAULT)

- Aquí se pueden encontrar espacios ya definidos para algunos clasificadores o regresores

http://mlrhyperopt.jakob-r.de/parconfigs

8 code@*.de regr extraTrees

id	type	lower	upper	default
mtry	integer	1	p	max(floor(p/3), 1)
numRandomCuts	integer	1	25	1

9 code@*.de ksvm

id	type	lower	upper	default	trafo
C	numeric	-5	10	0	function (x) 2^x
sigma	numeric	-15	15	structure(expression(kernlab::sigest(as.matrix(getTaskData(task, target.extra = TRUE)[["data"]])), scaled = TRUE), .Names = c("", "scaled")	function (x) 2^x

10 code@*.de glmboost

id	type	lower	upper	default	trafo
mstop	numeric	-3.3219	6.6439	3.3219	function (x) floor(2^x * 10
nu	numeric	0	1	0.1	

11 code@*.de gbm

id	type	lower	upper	default	trafo
interaction.depth	integer	1	10	1	
n.minobsinnode	integer	5	25	10	
n.trees	numeric	0	6.6439	5.6439	function (x) round(2^x * 10
shrinkage	numeric	0.001	0.6	0.001	

12 code@*.de rpart

id	type	lower	upper	default	trafo
cp	numeric	-10	0	-6.6439	function (x) 2^x
maxdepth	integer	3	30	30	
minbucket	integer	5	50	7	
minsplit	integer	5	50	20	

13 code@*.de nnet

id	type	lower	upper	default	trafo
decay	numeric	-5	1	1.0e-05	function (x) 10^x
size	integer	1	20	3	

14 code@*.de glmnet

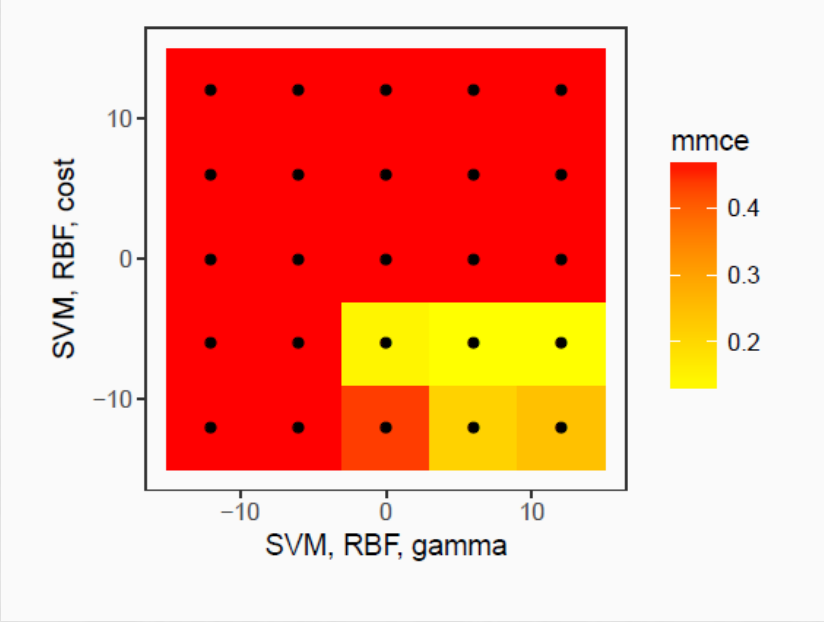
id	type	lower	upper	default	trafo
alpha	numeric	0	1	1	
lambda	numeric	-10	3	0	function (x) 2^x

15 code@*.de xgboost

id	type	lower	upper	default	trafo
colsample_bytree	numeric	0.3	0.7	0.5	

PROBLEMAS DE GRID SEARCH

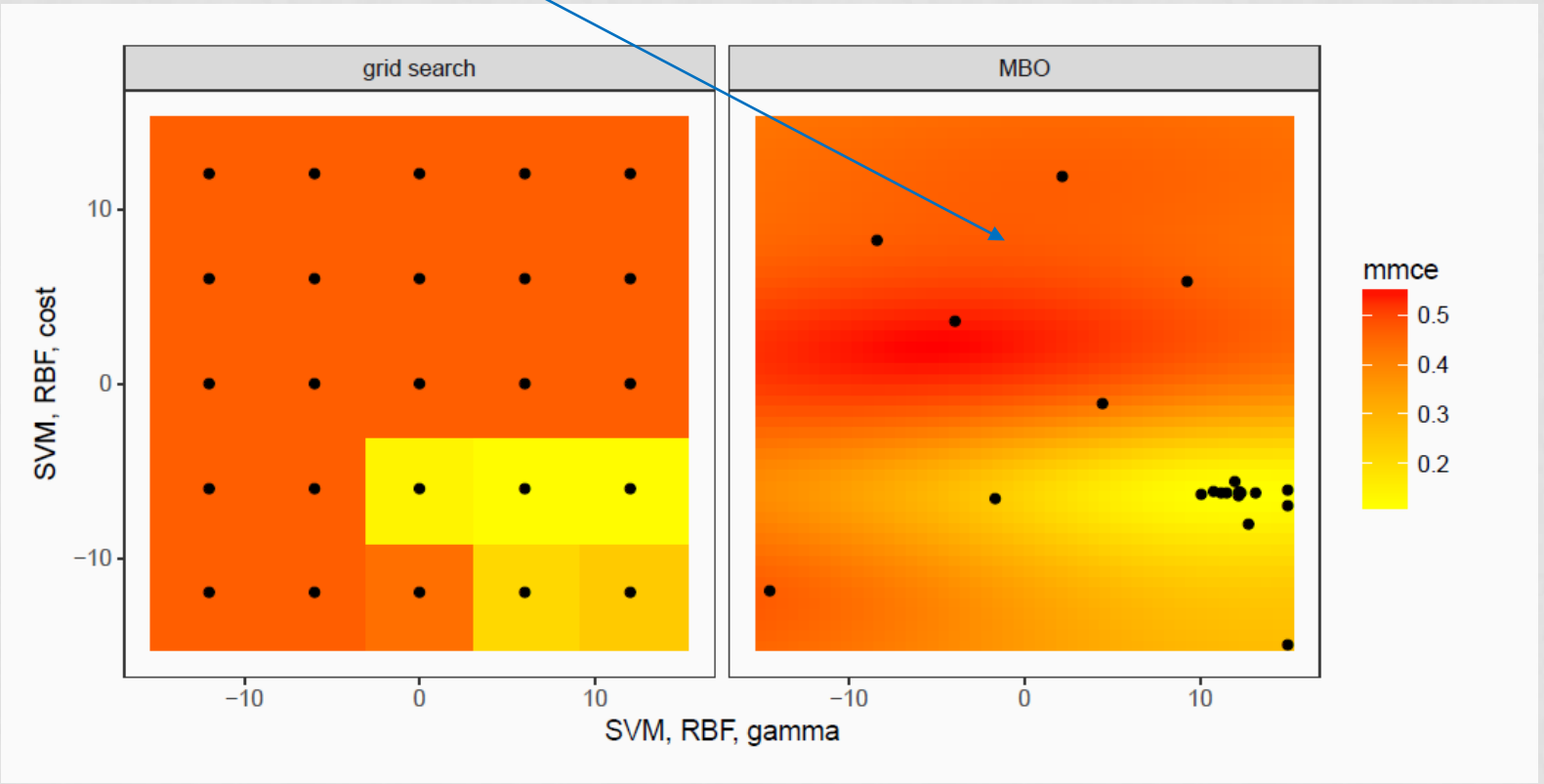
- ¿Qué ocurre si la mejor combinación está fuera de la *grid* (rejilla)?
- Hay que hacer muchas evaluaciones en zonas poco prometedoras



Source: http://ssa.cf.ac.uk/big-data/slides/Archetti-ML-Optimization.pdf

PROBLEMAS DE GRID SEARCH (Y RANDOM SEARCH)

- ¿No sería mejor algo así?



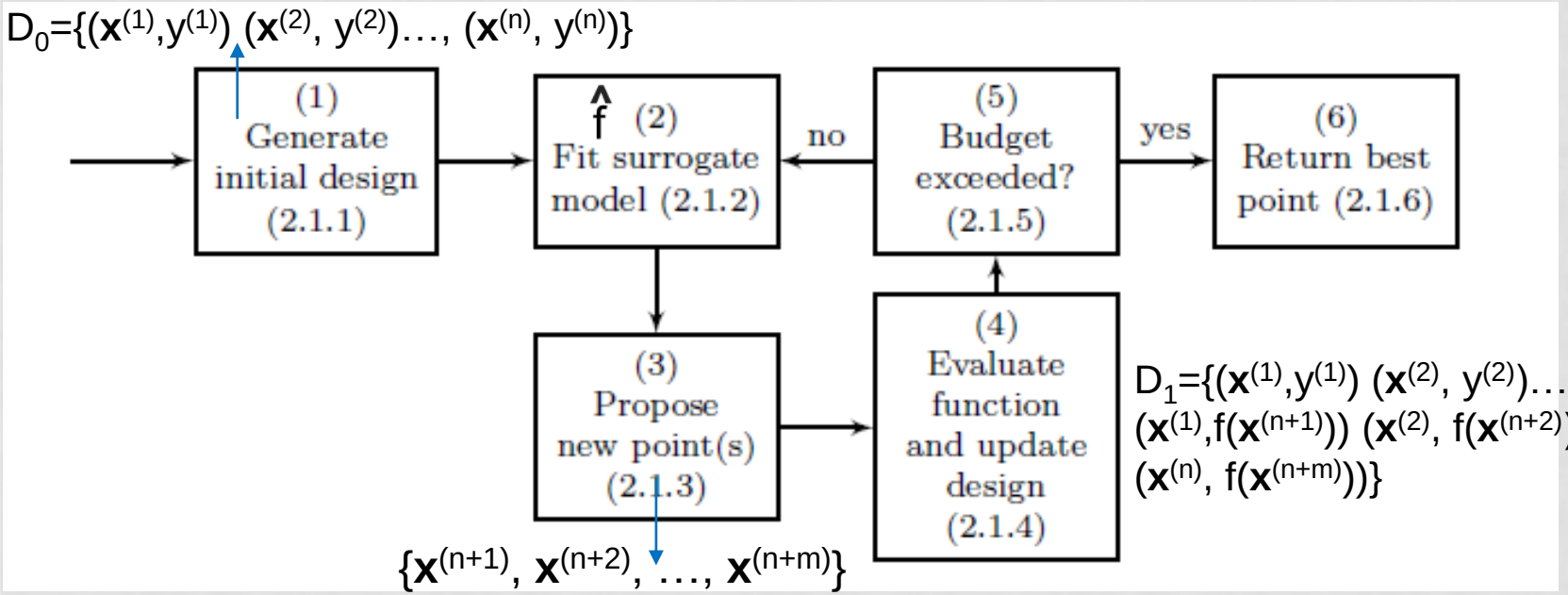
Source: http://ssa.cf.ac.uk/big-data/slides/Archetti-ML-Optimization.pdf

SEQUENTIAL MODEL-BASED OPTIMIZATION

- Sea $\mathbf{X}=\mathbf{R}^d$ el espacio de los valores de d hiper-parámetros (suponiendo de momento que todos tienen valores reales). Para (maxdepth, minsplit) sería $\mathbf{X}=\mathbf{R}^2$
- Idea de SMO:
  - Construye un "diseño" inicial de n puntos $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\} \subseteq \mathbf{X}$
  - Evalúa dichos puntos: $y = f(\mathbf{x}^{(i)})$
    - Es decir, entrena un modelo con el conjunto de entrenamiento y calcula una medida (ej: error de clasificación) con el de validación (también xvalidation)
    - Tendremos $D=\{(\mathbf{x}^{(1)},y^{(1)}) (\mathbf{x}^{(2)}, y^{(2)})\ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$
  - Construimos un modelo de regresión usando como entrenamiento D
    - $\hat{f} = \text{train}(D)$
    - A $\hat{f}$ se le llama *surrogate model* (modelo sustituto). Suele usarse *Kriging*
    - Es una estimación de f, que permite calcular f(x) en mucho menos tiempo
  - *Infill*: Usamos $\hat{f}$ para determinar los $\mathbf{x}^{(i+j)}$ prometedores que merece la pena explorar a continuación.
    - $\mathbf{x}^{(i+j)} = \text{argmin } \hat{f}(\mathbf{x})$

SEQUENTIAL MODEL-BASED OPTIMIZATION

- Método completo:



Bischl, Bernd, et al. "mlrMBO: A modular framework for model-based optimization of expensive black-box functions." *arXiv preprint arXiv:1703.03373* (2017).

SMO CON FOCUS SEARCH

- Una posible variante es *focus search*
- Cada vez que se generan los puntos siguientes prometedores $\{\mathbf{x}^{(n+1)}, \mathbf{x}^{(n+2)}, \ldots, \mathbf{x}^{(n+m)}\}$, se focaliza el espacio de búsqueda.
- Supongamos que un hiper-parámetro x_1 tiene un rango de variación de [0,100]
  - En la primera iteración, se muestrean valores de [0,100]
  - En la segunda iteración se disminuye el rango a ¼ y se muestrea alrededor del mejor valor encontrado
    - Si el mejor valor encontrado es $x_1^* = 30$, las siguientes muestras se buscarán en [30- ¼ *(100-0), 30+ ¼*(100-0)]
    - En la siguiente iteración se volverá a reducir el rango a 1/8
    - Etc.

SMO: ¿SURROGATE MODEL?

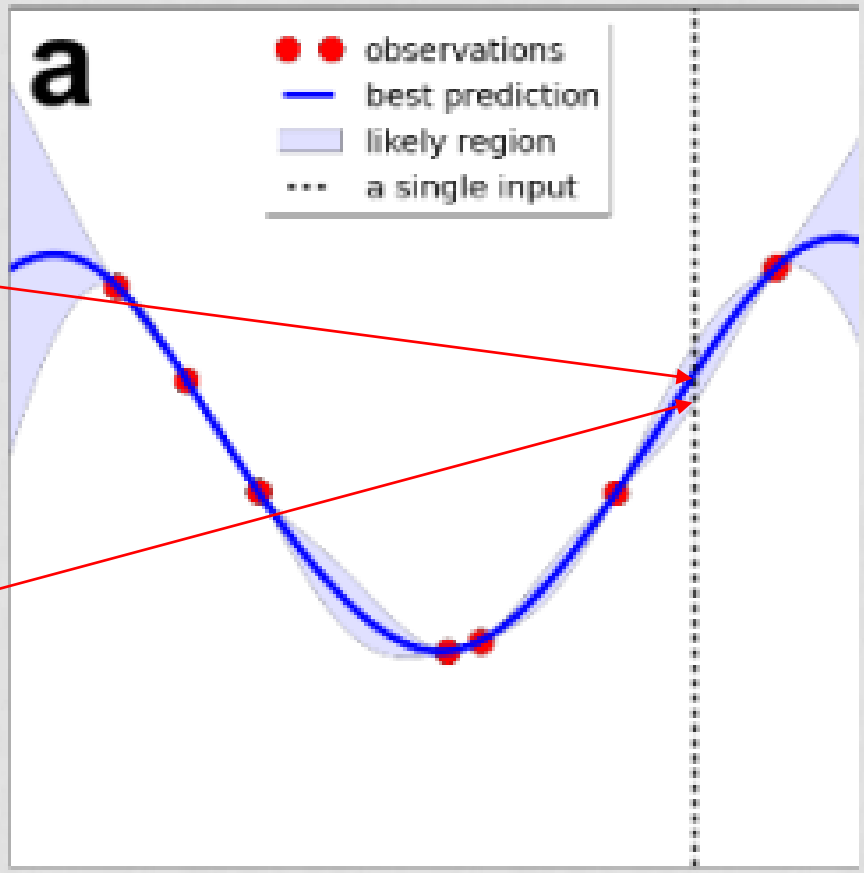
- $\hat{f}$ es un modelo de regresión, que se construye a partir de $D=\{(\mathbf{x}^{(1)},y^{(1)})\ (\mathbf{x}^{(2)}, y^{(2)})\ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$
- En principio, cualquier modelo de regresión valdría, incluyendo uno lineal:
  - $\hat{f}$ = lm(D)

- Normalmente se utilizan modelos más sofisticados como Kriging (interpolación espacial)
- Kriging (o Gaussian process regression) es un modelo que devuelve la media *u*, pero también la desviación *se*
- Usando media y desviación, SMO puede proponer puntos prometedores calculando el llamado *expected improvement* (EI)

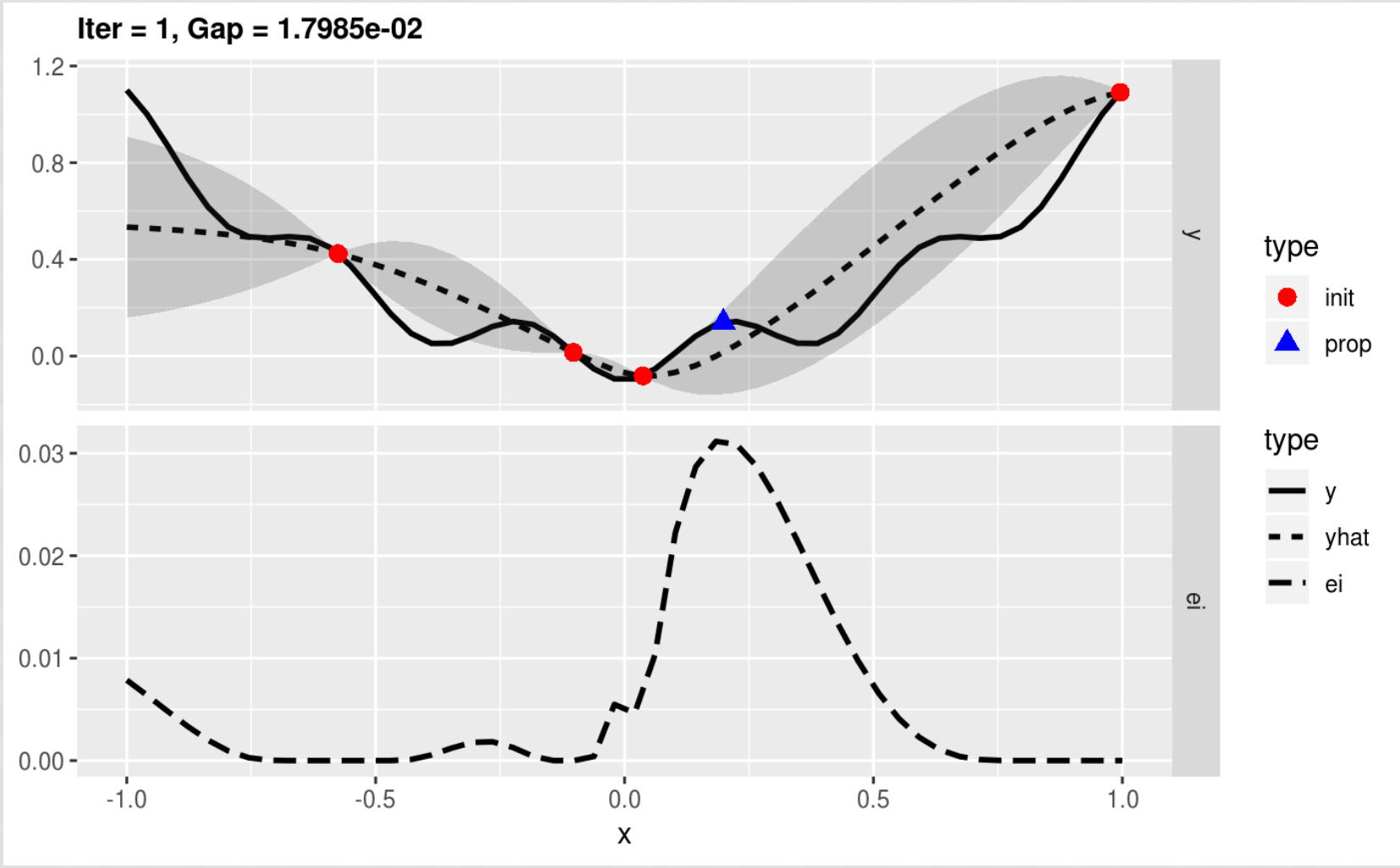
KRIGING

Media *u*

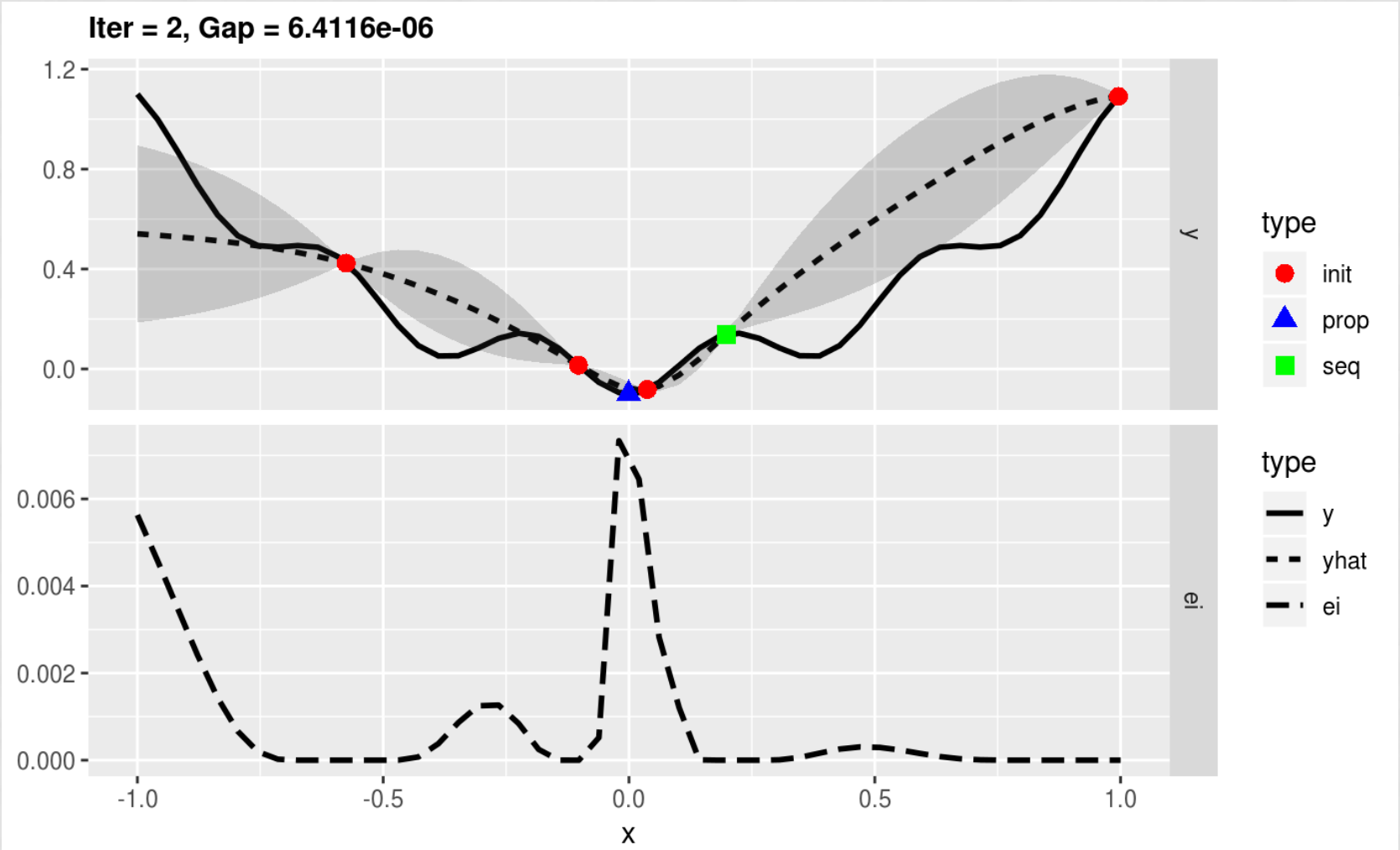
Desviación *se*



Bischl, Bernd, et al. "mlrMBO: A modular framework for model-based optimization of expensive black-box functions." *arXiv preprint arXiv:1703.03373* (2017).



Bischl, Bernd, et al. "mlrMBO: A modular framework for model-based optimization of expensive black-box functions." *arXiv preprint arXiv:1703.03373* (2017).



Bischl, Bernd, et al. "mlrMBO: A modular framework for model-based optimization of expensive black-box functions." *arXiv preprint arXiv:1703.03373* (2017).