



# Random Forests

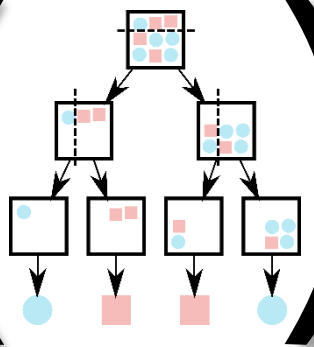
**Álvaro Barbero Jiménez**

[alvaro.barbero.jimenez@gmail.com](mailto:alvaro.barbero.jimenez@gmail.com)

# 1. Random Forests

# Random Forest en una diapositiva

Random Forest  $\equiv$  Bagging (  )





**Afi** Escuela  
de Finanzas

© 2021 Afi Escuela de Finanzas. Todos los derechos reservados.

**¡No tan rápido!**

# 1'. Random Forests (en detalle)

# Motivación Random Forests

¿Por qué bagging de árboles? ¿Qué tienen de especial?

Los árboles de decisión son un clasificador **inestable**

- Pequeños cambios en el dataset de entrenamiento pueden generar árboles muy distintos
- Fomento de la **diversidad** del ensemble

Mediante pre-poda u otros mecanismos puede controlarse la **complejidad** del árbol

- Facilita la construcción de **clasificadores débiles**, pero no demasiado débiles

# Motivación Random Forests

## Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

**Manuel Fernández-Delgado**

MANUEL.FERNANDEZ.DELGADO@USC.ES

**Eva Cernadas**

EVA.CERNADAS@USC.ES

**Senén Barro**

SENEN.BARRO@USC.ES

*CITIUS: Centro de Investigación en Tecnoloxías da Información da USC*

*University of Santiago de Compostela*

*Campus Vida, 15872, Santiago de Compostela, Spain*

**Dinani Amorim**

DINANIAMORIM@GMAIL.COM

*Departamento de Tecnologia e Ciências Sociais- DTCS*

*Universidade do Estado da Bahia*

*Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil*

# Motivación Random Forests

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest (RF)** versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).



# No Free Lunch Theorem

*Given a finite set  $V$  and a finite set  $S$  of real numbers, assume that  $f: V \rightarrow S$  is chosen at random according to uniform distribution on the set  $V^S$  for all possible functions from  $V$  to  $S$ . For the problem of optimizing  $f$  over the set  $V$ , then **no algorithm performs better than blind search***

Interpretación en Machine Learning: en promedio no existe ningún algoritmo mejor que otro si consideramos el espacio de todos los posibles problemas

→ ¿Paradoja?

# No Free Lunch Theorem

La mayoría de funciones del espacio de todas las posibles funciones son **Kolmogorov random**

- No es posible codificar estas funciones de una forma más eficiente que listando todos y cada uno de sus valores

Las funciones (problemas) con los que trabajamos en la práctica tienen alguna **estructura implícita** que permite expresarlos de forma más concisa → no son Kolmogorov random

→ Los algoritmos cuyas **hipótesis** se aproximen más a la estructura del subespacio de “problemas reales” serán más efectivos

- Posible explicación al rendimiento de Random Forest

# Random Forest

Random Forest  $\equiv$  Bagging (  )

+ técnicas para fomentar la diversidad del ensemble

# Fomentando árboles diversos

Random Forest parte de la técnica estándar para fomentar la diversidad en bagging: **muestreo con reemplazo**

- Cada árbol solo ve  $\simeq$  el 63.2% de los datos

☹ Si existe alguna variable con mucho poder explicativo, se escogerá con muy alta probabilidad en todos los árboles

→ Árboles muy correlados

→ Bajo rendimiento del ensemble

Para evitar esto se aplica el **Random Subspace method**

# Random Subspace Method

**Random Subspace Method** (o Attribute Bagging) selecciona un conjunto distinto de **variables explicativas** para entrenar cada componente del ensemble

En Random Forest **cada vez que se va a realizar un corte** de un nodo, en lugar de evaluar todas las posibles variables para el corte se analiza únicamente un subconjunto aleatorio de ellas

- Con  $m$  variables normalmente se escogen  $\sqrt{m}$  o  $\log(m)$  variables aleatorias a analizar

✓ Mayor diversidad de árboles en el ensemble

# Bagging de árboles VS Random Forest

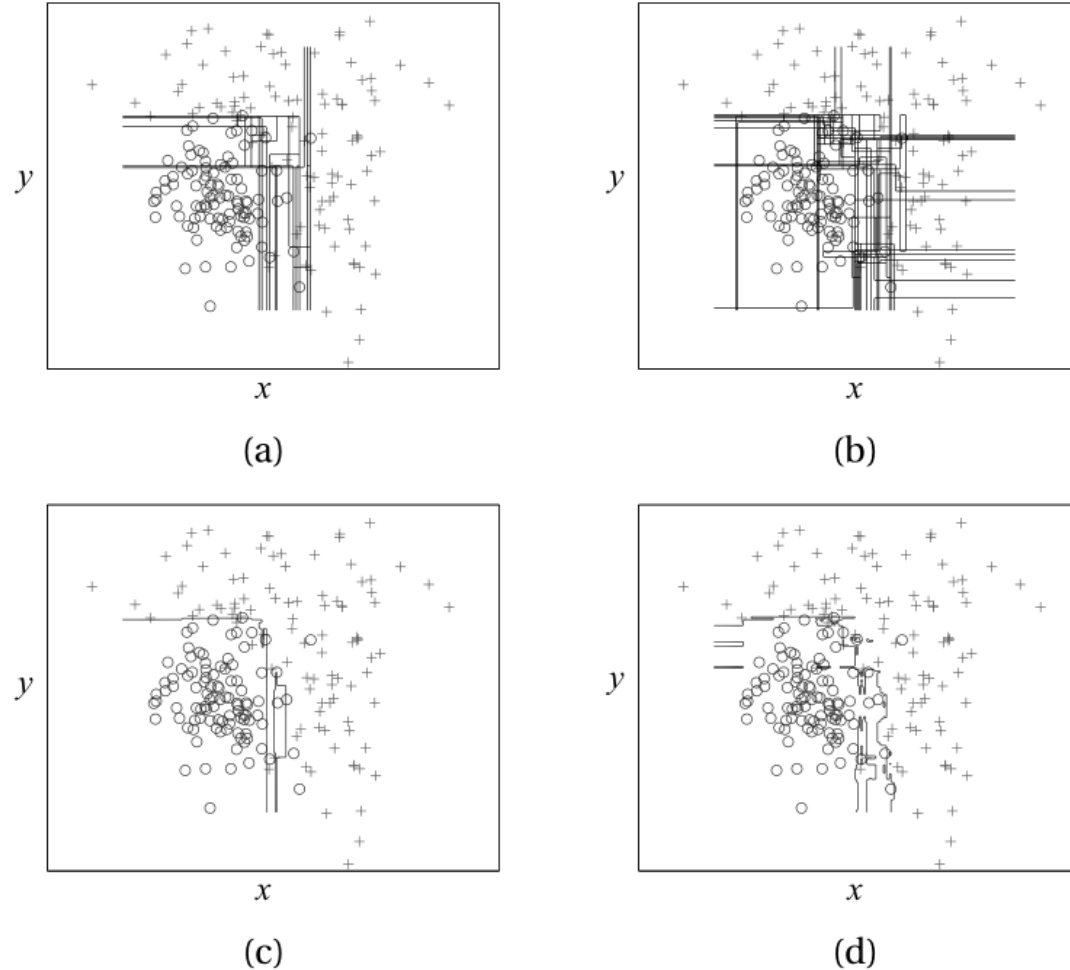


FIGURE 3.8: Decision boundaries on the *three-Gaussians* data set: (a) the 10 base classifiers of Bagging; (b) the 10 base classifiers of RF; (c) Bagging; (d) RF.

# Pruning en Random Forest

Cuando se aplica pruning a los árboles generados por Random Forest:

- ✓ Se mejora la capacidad de generalización de cada árbol
- ☹ Los árboles se vuelven más estables y homogéneos, por lo que se pierde diversidad en el ensemble

En Random Forest no suele aplicarse **post-pruning**.

En algunos casos, especialmente en regresión, es útil usar **pre-pruning**:

- Limitar el número de niveles
- Limitar el número de nodos
- Limitar el número de hojas
- Mínimo de patrones por nodo para hacer un corte

# Pruning VS no pruning

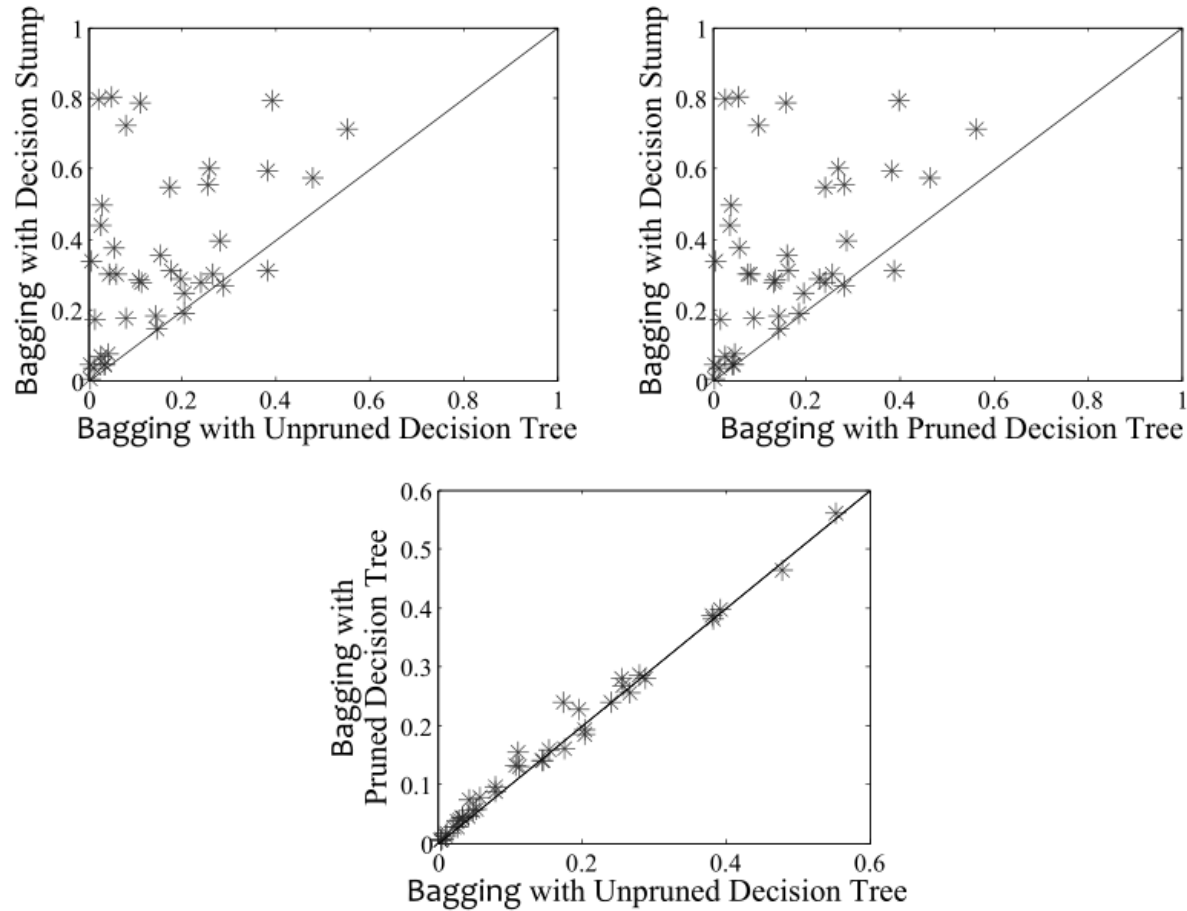


FIGURE 3.4: Comparison of predictive errors of Bagging using decision stumps, pruned decision trees and unpruned decision trees. Each point represents a data set and locates according to the predictive error of the two compared algorithms. The diagonal line indicates where the two compared algorithms have identical errors.



# Validación cruzada en Random Forest

Para ajustar los hiperparámetros de Random Forest (número de árboles, número de variables por corte, criterios de pre-poda, etc.) puede utilizarse el método **out-of-bag**

Debido al bootstrap sampling, cada árbol tiene en promedio un 36.8% del dataset que no ha utilizado para entrenar.

Podemos definir la **predicción out-of-bag** del ensemble como:

$$H^{oob}(x) = \arg \max_y \sum_{t \in RF} \delta_{h_t(x)=y} \delta_{x \notin D_t}$$

con  $\delta_{a=b} = 1$  si  $a = b$  y 0 en otro caso (delta de Kronecker),  $h_t(x)$  la predicción del árbol  $t$  para el patrón  $x$  y  $D_t$  el conjunto de patrones usado para entrenar el árbol  $t$

# Validación cruzada en Random Forest

Utilizando las predicciones out-of-bag puede estimarse el **error de generalización** del Random Forest como:

$$err^{oob} = \frac{1}{|D|} \sum_{(x,y) \in D} \delta_{H^{oob}(x) \neq y}$$

- ✓ No requiere entrenar modelos por hojas o particiones de validación
- ✓ Es un estimador sobre un modelo que utiliza todo el conjunto de datos disponible para entrenamiento

## 2. Extensiones de Random Forest

# Aún más aleatoriedad

Puede conseguirse aún mayor diversidad en el ensemble aumentando el peso que tiene la componente aleatoria de Random Forest.

Hay varias maneras de hacer esto:

- **Extremely Randomized Trees** (o extra-trees)
- **VR-Trees**
- **Totally Random Trees**

Estas aleatorizaciones producen modelos que además tienen **otros intereses prácticos**.

# Extremely Randomized Trees

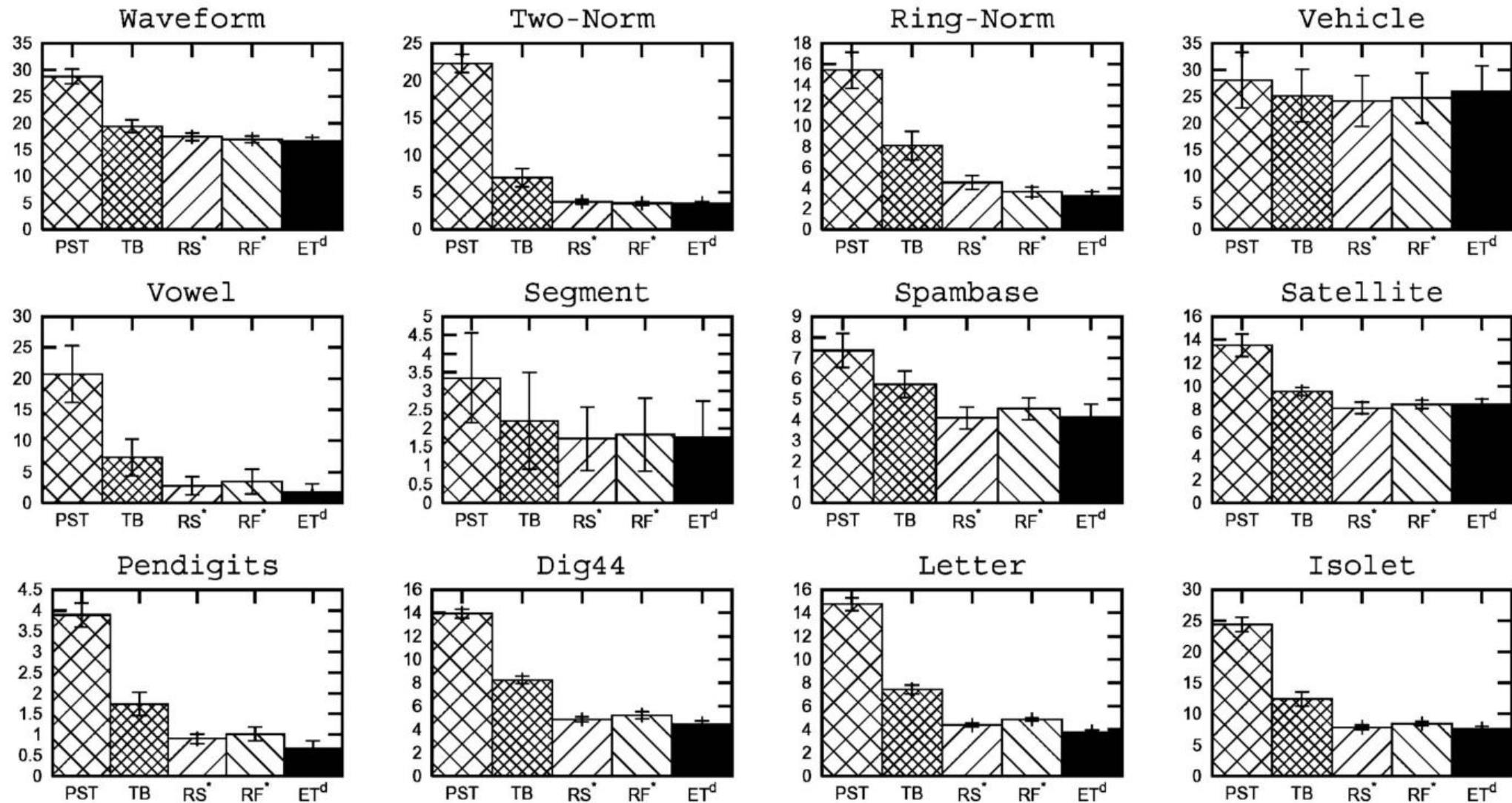
A la hora de realizar un corte:

- Elegir un **subconjunto aleatorio de  $s$  variables**  $v_1, \dots, v_s$
- Para cada variable, elegir un punto de **corte** o pregunta **aleatoria** entre todas las posibles:  $(v_1 < a_1), (v_2 < a_2), \dots, (v_s < a_s)$ , para  $a_1, \dots, a_s$  aleatorios dentro de los posibles para la variable
- Elegir el corte que más reduce la impureza

Todos los árboles se construyen usando todos los datos (sin hacer bootstrap).

# Extremely Randomized Trees

## Classification problems



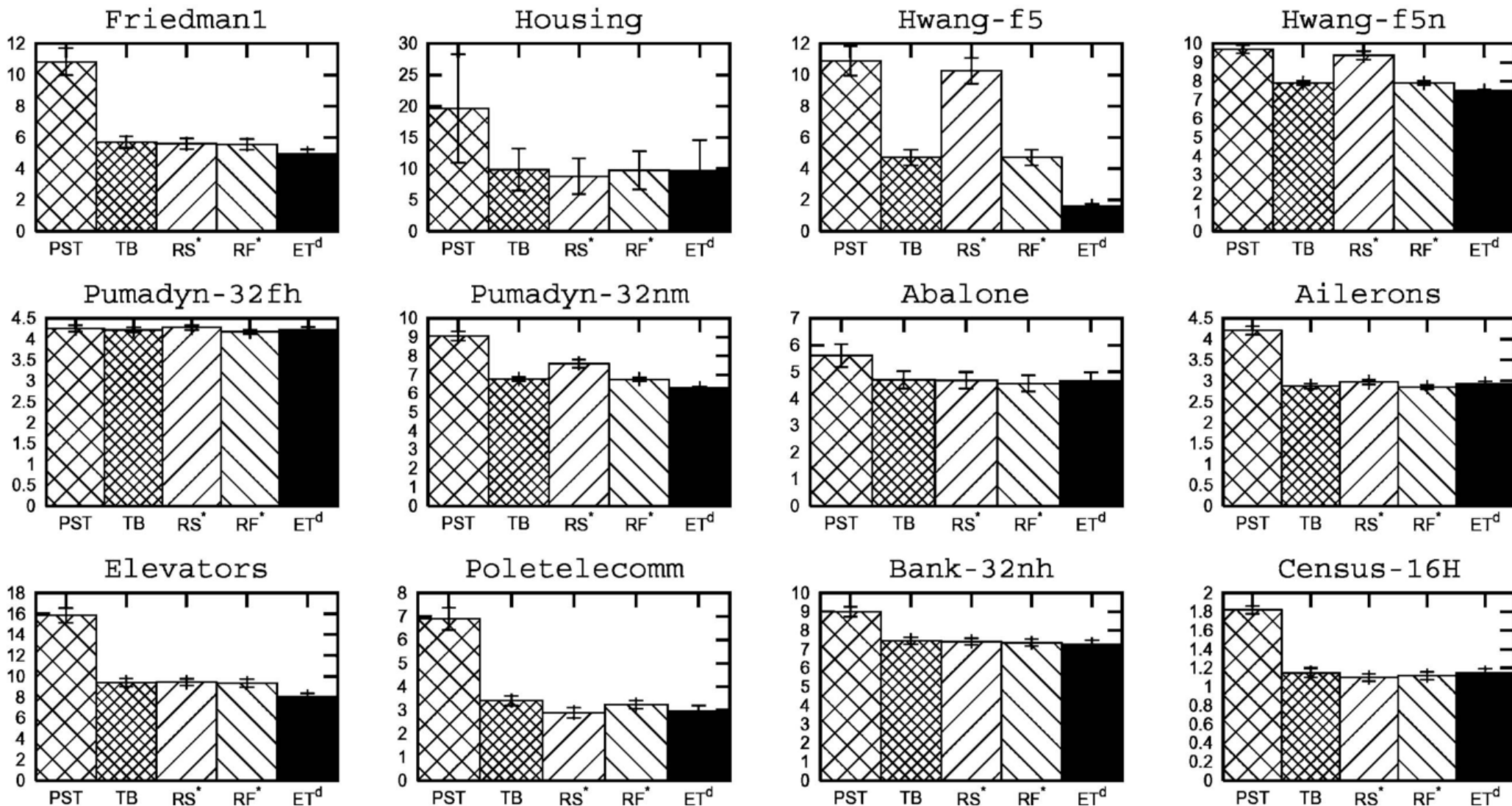
Legenda: PST = Single CART Tree, TB = Tree Bagging, RS= Random Subspace, RF = Random Forest, ET Extremely Randomized Trees

Geurts et al – Extremely randomized trees



# Extremely Randomized Trees

## Regression problems



Legenda: PST = Single CART Tree, TB = Tree Bagging, RS= Random Subspace, RF = Random Forest, ET Extremely Randomized Trees

Geurts et al – Extremely randomized trees

# Extremely Randomized Trees

En teoría, Extremely Randomized Trees es superior a Random Forests porque:

- ✓ No usa bootstrap → estimador con menor sesgo
- ✓ Más aleatoriedad → ensemble más diverso
- En la práctica:
  - ~ Rendimiento más o menos similar a Random Forest
  - ✓ Entrenamiento más rápido
  - 😞 Árboles más grandes → más lentitud en predicción



# Totally Random Trees

En el caso extremo de Extra Trees considerando una única variable por nodo, se obtiene el algoritmo de **Totally Random Trees**.

En cada corte se escoge una variable y un punto de corte (pregunta) de forma aleatoria.

Los árboles crecen hasta que solo queda un patrón en cada nodo.

El target de clasificación/regresión no se usa para nada en la construcción de los árboles.

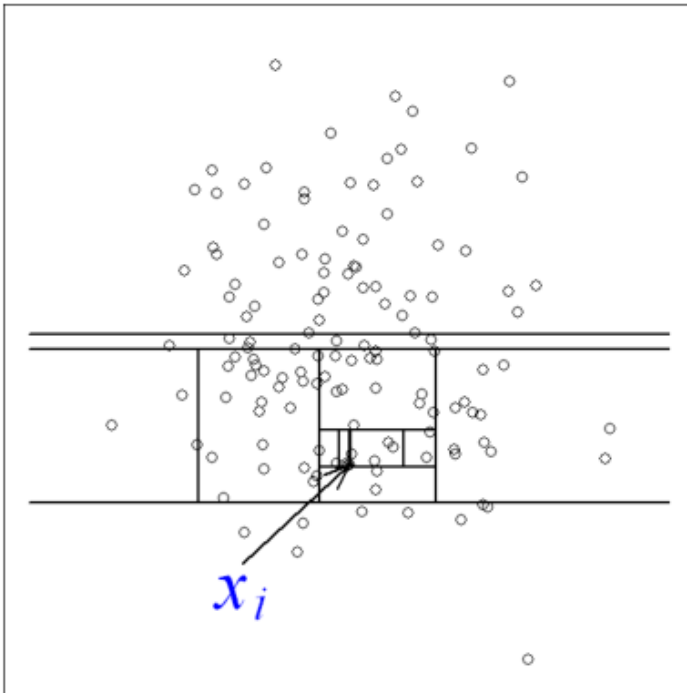
# Totally Random Trees

- ☹ Los Totally Random Trees no valen para clasificar ni hacer regresión (porque no usan el target para decidir nada).
- ✓ Pero sirven para hacer **estimación de densidad** de los datos

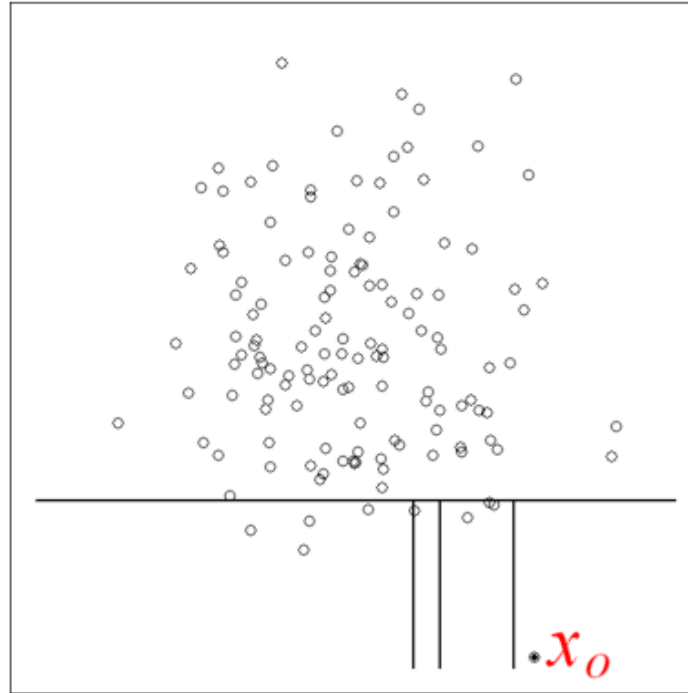
Intuitivamente, si hacemos particiones aleatorias del conjunto de datos:

- Patrones en zonas muy aisladas tienen mayor probabilidad de quedarse solos tras un corte aleatorio.  
→ mayor probabilidad de acabar en nodos de poca profundidad.
- Patrones muy próximos a otros acabarán en nodos profundos.

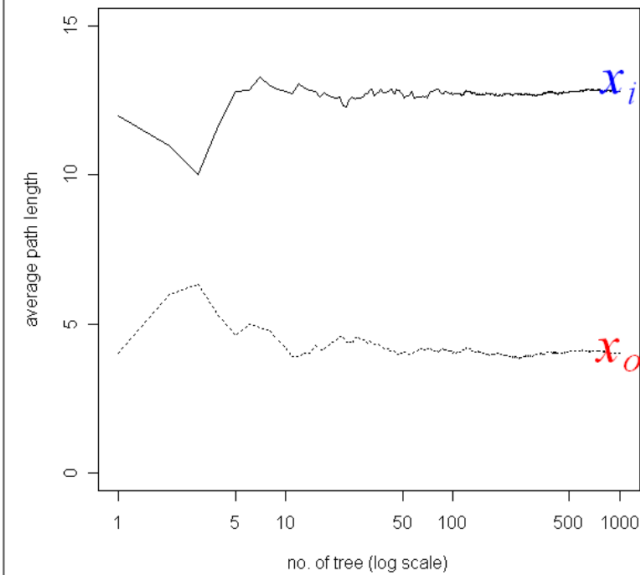
# Totally Random Trees



(a) Isolating  $x_i$



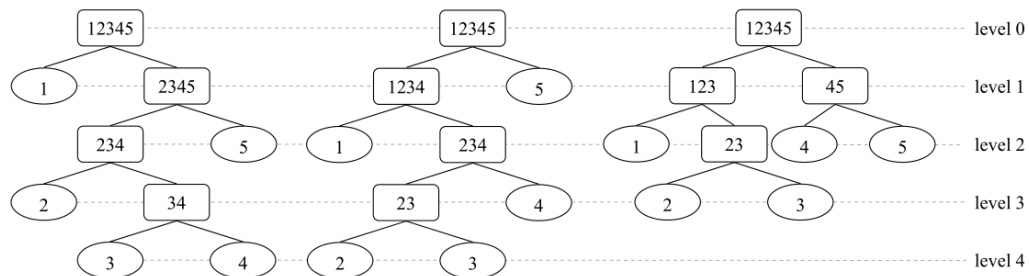
(b) Isolating  $x_o$



# Totally Random Trees

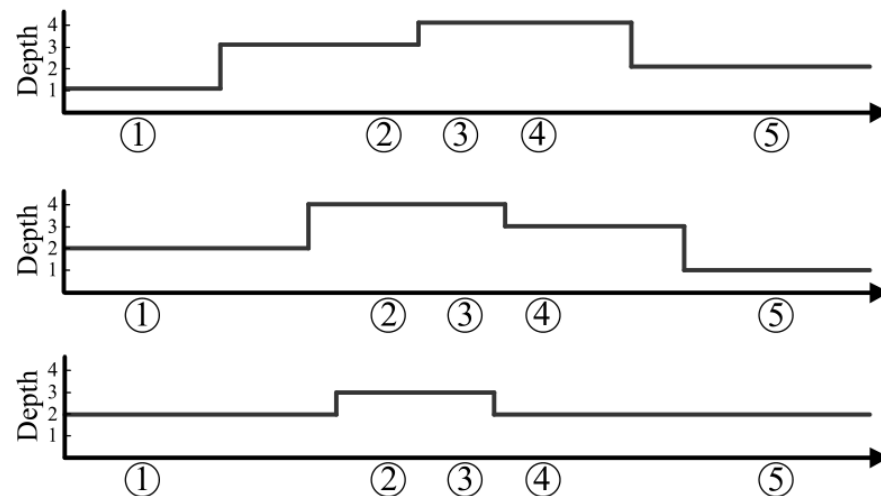


(a) Five one-dimensional data points

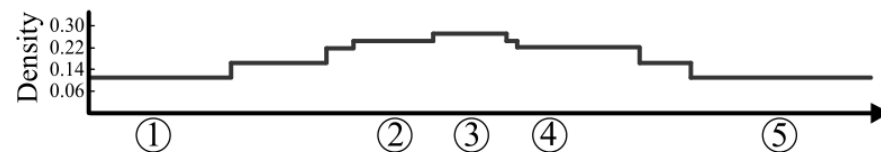


(b) Three completely random trees

Densidad en el punto de un dato  
= media de profundidad de la  
hoja de ese dato en todos los  
árboles del ensemble



(c) The depth of leaves on the one-dimensional data, each corresponding to a random tree in the sub-figure (b)



(d) The density estimation result

# Isolation Forests

**Isolation Forests:** aplicación de Totally Random Trees a detección de anomalías

- Subsampling con reemplazo de  $\psi$  datos,  $\psi \sim 256$
- Se limita la profundidad máxima del árbol a  $\log(\psi)$

Dado un punto  $x$  se le asigna un score de anomalía comparando la **profundidad media del nodo terminal** al que llega en los árboles del ensemble,  $E[h(x)]$ , con la **profundidad media de un árbol** construido con  $n$  datos:

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

- Con  $H(n) = \sum_{k=1}^n \frac{1}{k} \simeq \ln(n) + 0.5772156649$  número armónico

El **score** se calcula como

$$s(x) = 2^{-\frac{E[h(x)]}{c(\psi)}}$$

# Isolation Forests

Dado que los datos más interesantes (anomalías) se encontrarán a poca profundidad, **no es necesario desarrollar árboles muy grandes**

- **Pre-poda:** profundidad máxima del árbol  $\log(\psi)$

A la hora de calcular  $h(x)$ , si se acaba llegando a una hoja pre-podada de profundidad  $p$ , que contiene  $d > 1$  datos, se **estima la profundidad** que se hubiera tenido si no se hubiera hecho poda como:

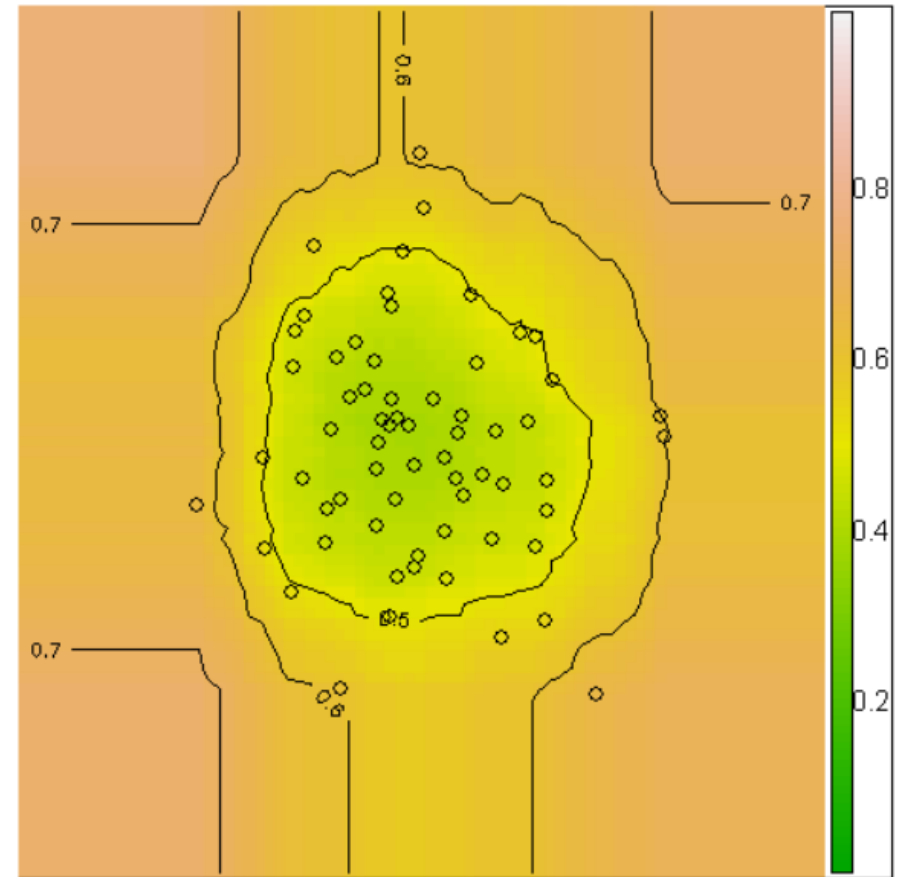
$$h(x) = p + c(d)$$

Con  $c(d)$  la profundidad esperada de la rama que no se ha llegado a construir.

# Isolation Forests

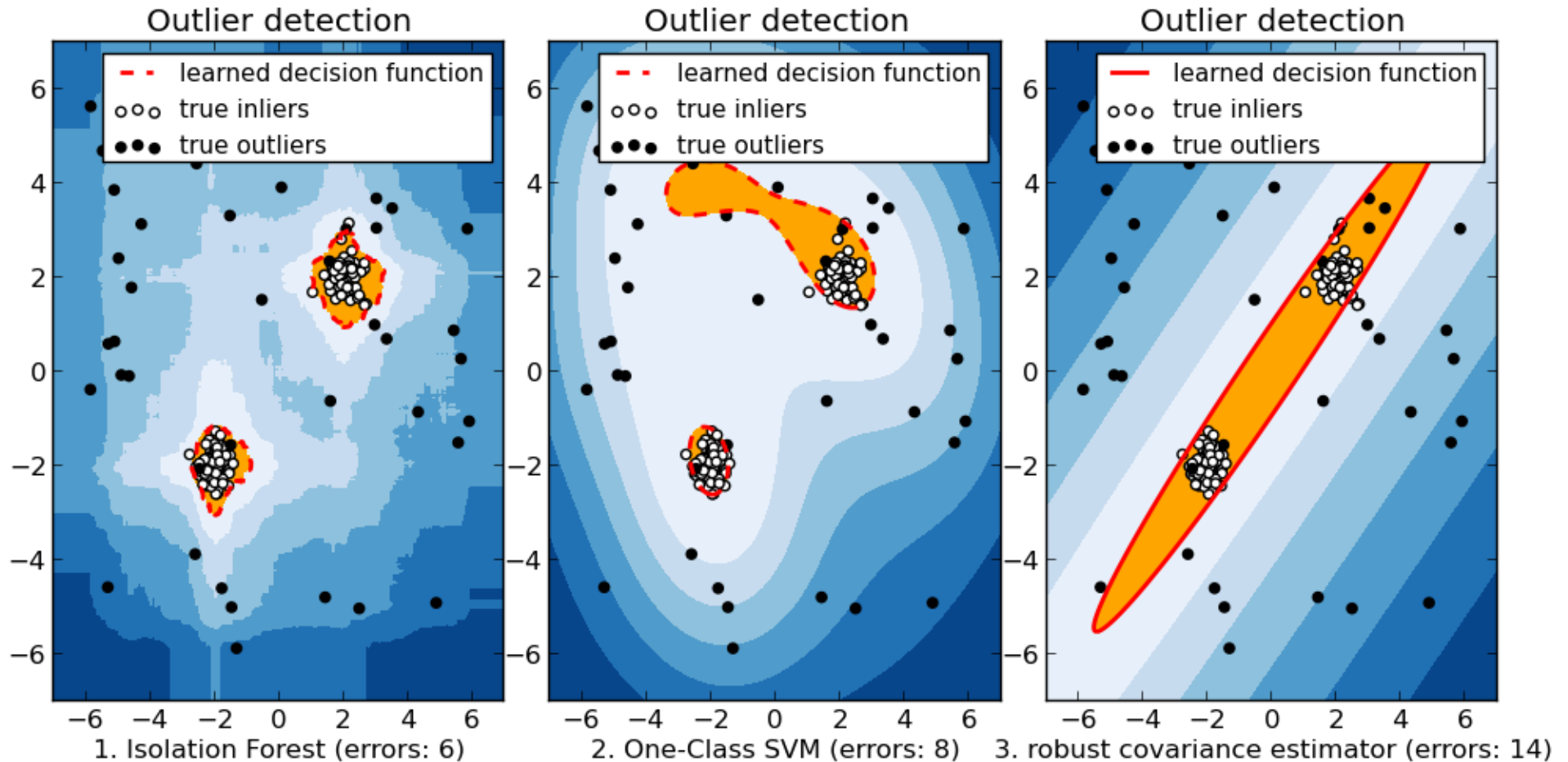
$$s(x) = 2^{-\frac{E[h(x)]}{c(\phi)}}$$

- Si  $E[h(x)] \rightarrow n - 1, s \rightarrow 0$   
 $\rightarrow x$  es normal
- Si  $E[h(x)] \rightarrow 0, s \rightarrow 1$   
 $\rightarrow x$  es una anomalía



**Figure 3. Anomaly score contour of iForest for a Gaussian distribution of sixty-four points. Contour lines for  $s = 0.5, 0.6, 0.7$  are illustrated. Potential anomalies can be identified as points where  $s \geq 0.6$ .**

# Isolation Forests



scikit-learn: [http://scikit-learn.org/dev/auto\\_examples/covariance/plot\\_outlier\\_detection.html](http://scikit-learn.org/dev/auto_examples/covariance/plot_outlier_detection.html)



# Aprendizaje con costes

Si para un problema de clasificación los costes de error por clases son conocidos, pueden utilizarse para construir un ensemble que minimize estos costes en lugar del error de clasificación.

**Estrategias de árbol:** construir un ensemble en el que cada árbol tiene en cuenta esos costes durante su construcción.

**Re-muestreo:** al hacer bootstrap asignar más probabilidad de ser seleccionados a los patrones de clases con mayor coste


- Para problemas de 2 clases:


$$P(x_i \in \mathcal{S} | y_i = +1) \propto \frac{\mathcal{C}(-1 | +1)}{\mathcal{C}(+1 | -1)}$$

# Aprendizaje con costes

**Ejercicio:** calcular las probabilidades de muestreo de bootstrap de un dataset de 20 patrones de la clase  $y = +1$  y 80 patrones de la clase  $y = -1$ , considerando los siguientes costes:

$$\mathcal{C} = \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}$$

 Real ( $k$ )

 Predicho ( $c$ )

$$P(x_i \in \mathcal{S} | y_i = +1) \propto \frac{\mathcal{C}(-1 | +1)}{\mathcal{C}(+1 | -1)}$$

$$P(x_i \in \mathcal{S} | y_i = -1) \propto 1$$

# Estimación de las incertidumbres

**Clasificación:** podemos obtener la probabilidad de que un patrón  $x$  pertenezca a una cierta clase  $y$ :

- Encontrar el nodo terminal al que llega  $x$  en cada árbol
- Para cada árbol  $t$  en el forest,  $P(y|x; t)$  es proporcional al número de muestras de entrenamiento de la clase  $y$  que llegan al mismo nodo terminal que  $x$
- Probabilidad en el forest: 
$$P(y|x) = \frac{1}{|T|} \sum_{t \in T} P(y|x; t)$$

**Regresión:** media de las varianzas de las muestras de entrenamiento en los nodos terminales.

# \*. Poda de bosques

# Mejor muchos que todos

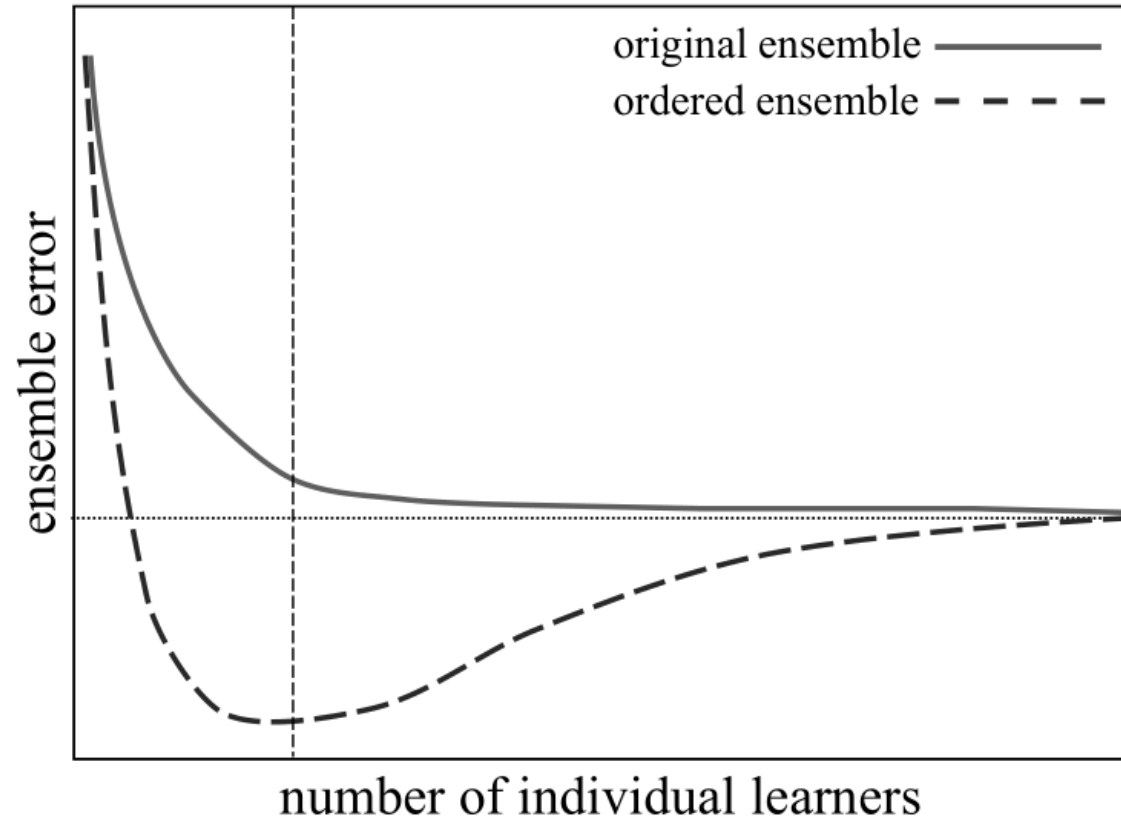


FIGURE 6.1: Illustration of error curves of the original ensemble (aggregated in random order) and ordered ensemble.

# Estilos de poda

**Ordering-based:** hacer un ranking de los  $N$  árboles del conjunto, ordenados según su calidad, y quedarse con los  $n$  mejores ( $n < N$ )

→ descartar árboles de baja calidad

**Clustering-based:** encontrar grupos de árboles que son muy similares entre sí, y quedarse solo con un representante de cada grupo

→ descartar árboles que no aumentan la diversidad

**Optimization-based:** formular el problema de selección de árboles como un problema de optimización

**Stacking:** utilizar otro modelo predictivo para encontrar la mejor poda

# Ordering-based

## Reduce-Error Pruning

- Escoger el **primer árbol** como el que tenga **menor error de validación**
- Escoger los **siguientes árboles** uno a uno, tomando en cada paso el árbol que **más reduce el error de validación**, hasta llegar a  $n$

Una vez ejecutado el proceso, puede mejorarse mediante **backfitting**

- Para cada árbol de los  $n$  seleccionados, comprobar si **intercambiarlo** por alguno de los no seleccionados reduce el error de validación

Se realizan rondas de backfitting hasta que ningún árbol puede ser intercambiado

# Ordering-based

## Kappa Pruning

- Escoger los **dos primeros árboles** como la pareja con **menor índice de acuerdo kappa**

$$\kappa = 1 - \frac{\frac{1}{T} \sum_x \rho(x)(T - \rho(x))}{|X_{train}|(T - 1)\bar{p}(1 - \bar{p})}$$

donde  $\rho(x)$  es el número de clasificadores del conjunto que clasifican  $x$  correctamente,  $T$  es el tamaño del conjunto,  $|X_{train}|$  es el tamaño del dataset de entrenamiento y  $\bar{p}$  es la media de precisión de los clasificadores.

- Escoger los **siguientes árboles** uno a uno, tomando en cada paso el árbol que **menor kappa produzca al unirlo** al conjunto de árboles escogidos hasta ahora



# Ordering-based

## Complementariness Pruning

- Escoger el **primer árbol** como el que tenga **menor error de validación**
- Escoger los **siguientes árboles** uno a uno, tomando en cada paso el árbol que **genere el mayor número de predicciones correctas en las que el conjunto se equivoca**, en los datos de validación:

$$\arg \max_t \sum_{(x,y) \in \mathcal{V}} \delta_{h_t(x)=y \text{ and } H_{t-1} \neq y}$$

# Optimization-based

El problema de poda de un bosque se puede escribir como el **problema de optimización**

$$\begin{aligned} \arg \max_w \quad & \sum_{(x,y) \in \mathcal{V}} y \cdot \text{sign} \left( \sum_{t \in T} w_t h_t(x) \right) \\ \text{s.t.} \quad & w \in \{0, 1\} \end{aligned}$$

☹ Se trata de un problema de tipo **Integer Linear Program**, que es una subclase de problemas **NP**.

→ En la práctica se intentan encontrar soluciones aproximadas

# Optimization-based

## Optimización heurística



El método **GASEN** intenta encontrar la combinación de árboles óptima usando un **algoritmo genético**

- **Genotipo**: array de pesos  $w$
- **Función de fitness**: precisión del ensemble con pesos  $w$
- Creación de una **población inicial aleatoria** de genotipos
- Repetir  $k$  veces:
  - **Seleccionar** subconjunto de población con **mayor fitness**
  - Generar nuevos genotipos mediante **recombinación**
  - Introducir **mutaciones aleatorias** en los genotipos

Otras aproximaciones: greedy hill-climbing, immune algorithms



**\*. Extras**

# VR-Trees

Variable Random (VR) Trees es un método que permite ajustar de forma paramétrica la cantidad de aleatoriedad en el ensemble.

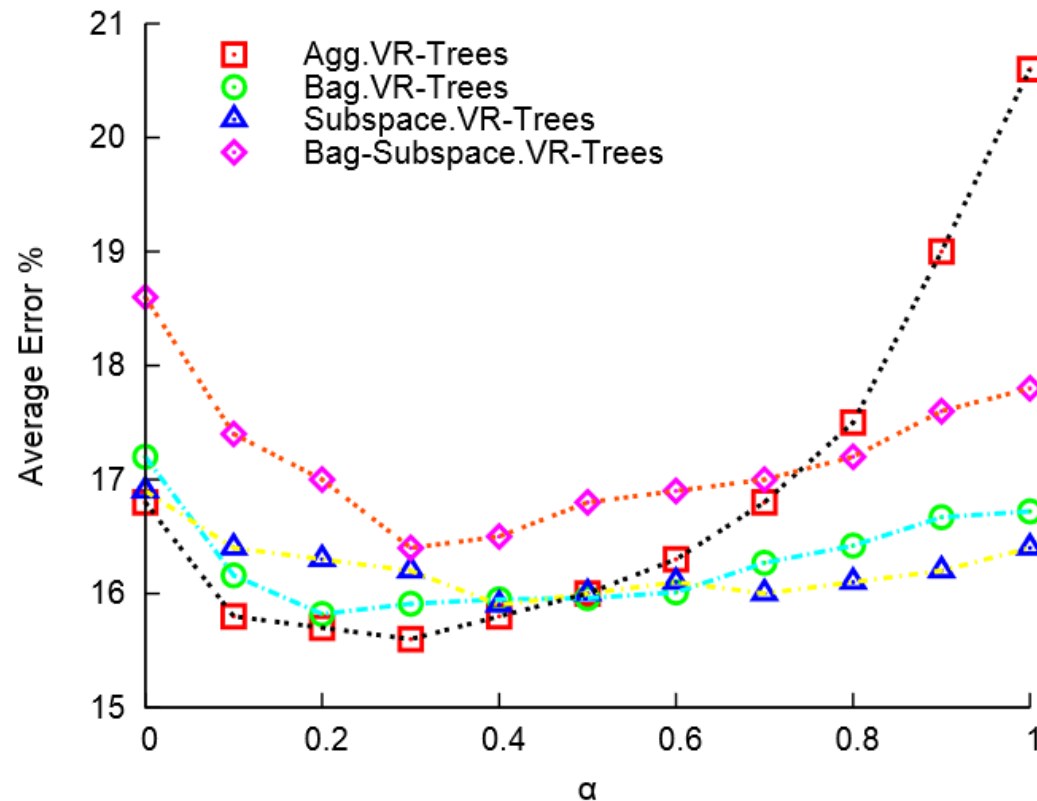
Cada vez que se va a realizar un corte en un nodo de un VR-Tree:

- Generar un número binario  $d \sim b(\alpha)$  con  $b(\alpha)$  ensayo de Bernoulli de probabilidad de éxito  $\alpha$
- Si  $d = 1$ , realizar un corte de árbol estándar
- Si  $d = 0$ , elegir la variable y el punto de corte aleatoriamente

El valor de  $\alpha$  determina el nivel de aleatoriedad del ensemble.

# VR-Trees

Figure 1: The spectrum of predictive performance for Aggregating, Bagging and Subspacing, as well as Bagging plus Subspacing: error rates against  $\alpha$ , average over forty-five data sets.



# VR-Trees

El punto idóneo de aleatoriedad  $\alpha$  es muy diferente para cada dataset.

Como aproximación para encontrar este punto óptimo se ha propuesto el método ***coalesce***:

- Agregación de árboles VR (sin bootstrap) en los que para cada uno de ellos se escoge  $\alpha \in [0, 0.5]$  aleatoriamente
- ~ En general el método *coalesce* produce resultados similares a Random Forest, por lo que no suele usarse en la práctica.
- ✓ Esta técnica ha resultado útil para ilustrar por qué la aleatoriedad ayuda en la generación de ensembles.

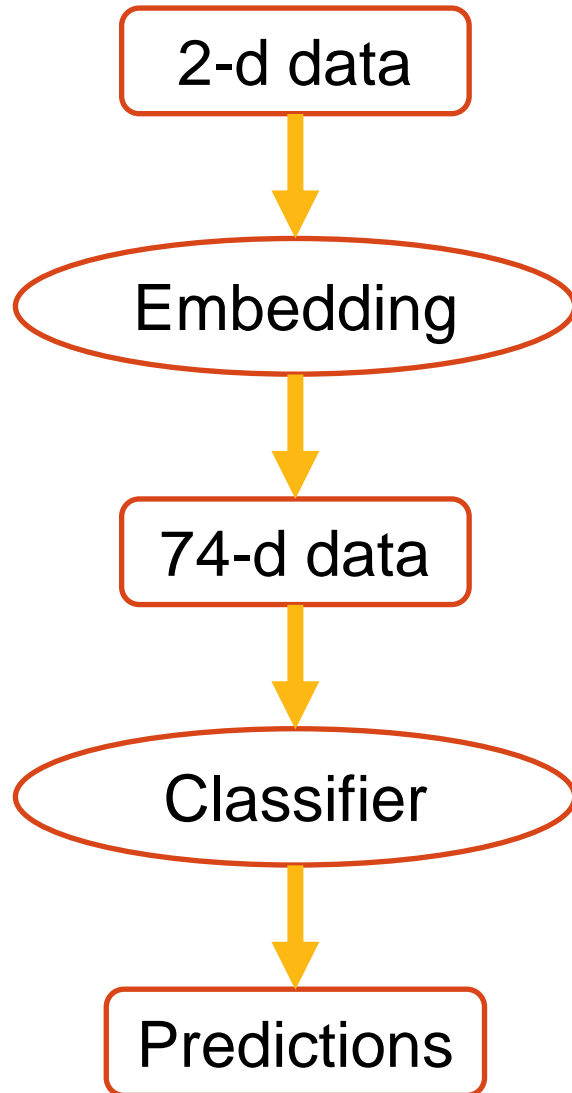
# Totally Random Trees Embedding

- ✓ Otra utilidad de los árboles totalmente aleatorios es la de **generar nuevas variables explicativas** a partir de las originales
1. Construir un ensemble de Totally Random Trees usando los datos de entrenamiento
  2. Dado un patrón  $x = [x_1, x_2, \dots, x_d]$ , se hace pasar este por todos los árboles del ensemble y se anota qué hojas alcanza
  3. Se representa el patrón como un vector binario de tantas entradas como hojas en el ensemble, con 0s en las hojas no alcanzadas y 1s en las alcanzadas

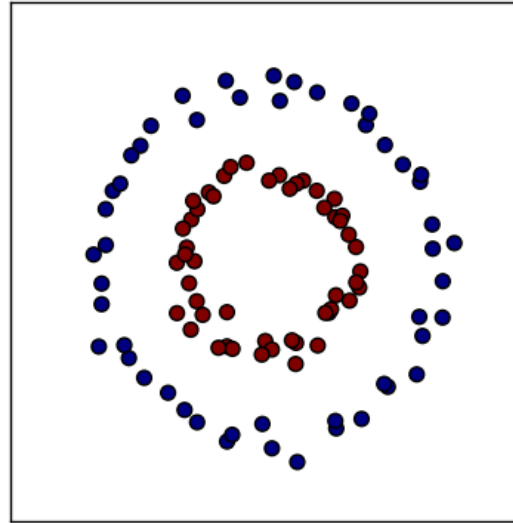
$$\tilde{x} = \begin{array}{c} \text{Árbol 1} \quad \text{Árbol 2} \quad \text{Árbol 3} \\ \boxed{0, 0, 1, 0, 0}, \boxed{0, 0, 1, 0, 0}, \boxed{0, \dots, 0, 1} \end{array}$$



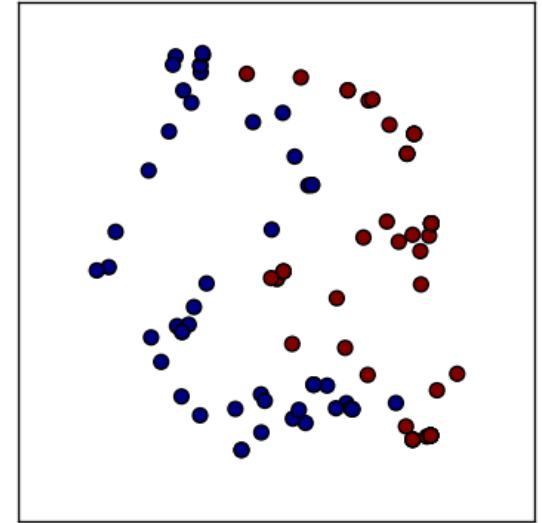
# Totally Random Trees Embedding



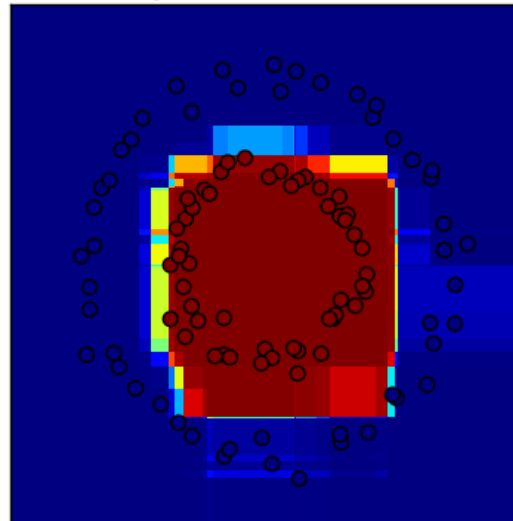
Original Data (2d)



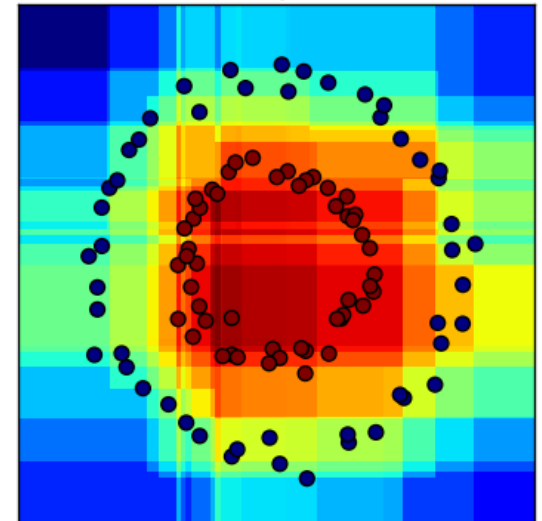
PCA reduction (2d) of transformed data (74d)



Naive Bayes on Transformed data



ExtraTrees predictions



[http://scikit-learn.org/stable/auto\\_examples/ensemble/plot\\_random\\_forest\\_embedding.html](http://scikit-learn.org/stable/auto_examples/ensemble/plot_random_forest_embedding.html)

# Otros mecanismos de diversidad

**Manipulación de parámetros del aprendizaje:** utilizar diferentes hiper-parámetros para cada árbol: criterios de corte, funciones de impureza, criterios de poda...

**Flipping Output:** intercambiar aleatoriamente la clase de parejas de patrones, manteniendo la proporción de clases

[Breiman – Randomizing Outputs to Increase Prediction Accuracy]

**Class switching:** cambiar la clase de algunos patrones por otra clase aleatoria, sin tener que mantener la proporción de clases

[Martínez-Muñoz & Suárez – Switching Class Labels to Generate Classification Ensembles]

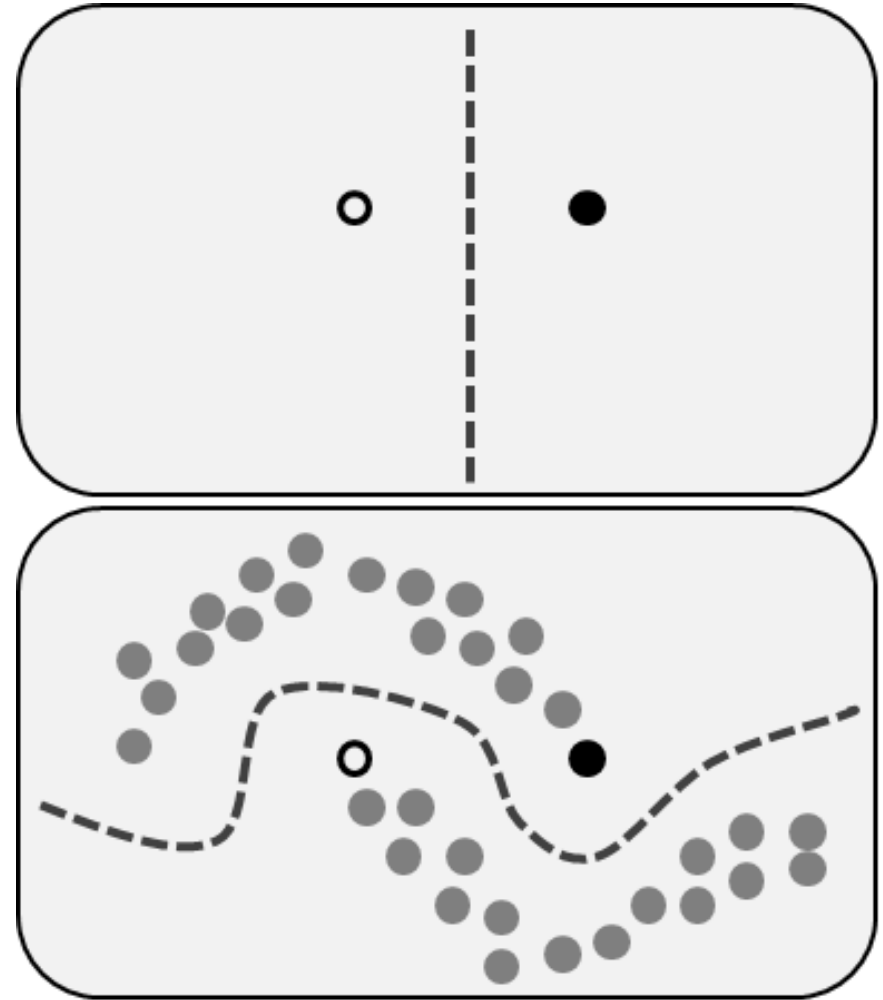
# Aprendizaje semi-supervisado

- Datos  $D = \{(x_1, y_1), \dots (x_n, y_n)\}$
- **Datos adicionales sin etiqueta**  $U = \{\tilde{x}_1, \dots, \tilde{x}_N\}$ 
  - $N \gg n$

Habitual en tareas de análisis de textos, imágenes, ...

- Mucha información bruta disponible
- **Etiquetado manual y costoso**

**Los datos no etiquetados pueden utilizarse para mejorar el clasificador**



# Aprendizaje semi-supervisado

**Co-Forest** es una modificación de Random Forest que permite explotar los datos no etiquetados

$R$  rondas de entrenamiento

- Construir un Random Forest con los datos etiquetados
- Para cada árbol, escoger un subconjunto aleatorio de datos sin etiqueta, y **etiquetarlos con la predicción del resto** de árboles del ensemble
- Añadir los datos con alta certidumbre al conjunto de entrenamiento de ese árbol, para la siguiente ronda

# Aprendizaje semi-supervisado

AVERAGE ERROR RATES OF THE COMPARED ALGORITHMS UNDER THE UNLABEL RATE OF 80%

Data set	RTree	Forest	SVM	AdaBoost	Self-Training			Co-Training			Co-Forest		
					initial	final	improv.	initial	final	improv.	initial	final	improv.
<i>bupa</i>	.396	.395	.420	.387	.396	.424	-7.1%*	.427	.443	-3.6%	.395	.384	<b>2.9%*</b>
<i>colic</i>	.272	.208	.233	.230	.272	.278	-2.3%	.255	.285	-11.7%*	.208	.178	<b>14.5%*</b>
<i>diabetes</i>	.321	.278	.261	.263	.321	.318	0.8%	.374	.356	4.8%*	.278	.261	<b>6.2%*</b>
<i>hepatitis</i>	.231	.203	.186	.206	.231	.240	-4.2%	.246	.240	2.8%	.203	.180	<b>11.5%*</b>
<i>hypothyroid</i>	.023	.018	.035	.014	.023	.023	-2.6%	.032	.038	-18.4%*	.018	.017	<b>6.6%</b>
<i>ionosphere</i>	.159	.129	.155	.156	.159	.191	-20.4%*	.179	.194	-8.6%	.129	.092	<b>28.7%*</b>
<i>kr-vs-kp</i>	.100	.051	.055	.080	.100	.122	-22.2%*	.112	.123	-9.4%*	.051	.035	<b>32.2%*</b>
<i>sonar</i>	.367	.312	.273	.306	.367	.388	-5.7%	.366	.398	-8.7%*	.312	.282	<b>9.7%*</b>
<i>vote</i>	.088	.066	.056	.053	.088	.096	-9.2%	.104	.135	-30.1%*	.066	.056	<b>15.0%*</b>
<i>wpbc</i>	.333	.328	.244	.304	.333	.373	-12.3%*	.353	.341	3.5%	.328	.279	<b>15.0%*</b>
avg.	.229	.199	.192	.200	.229	.245	-8.5%	.245	.255	-7.9%	.199	.176	<b>14.2%</b>



Afi Escuela

---

© 2021 Afi Escuela. Todos los derechos reservados.