Conjuntos de clasificadores

MIAX-11, noviembre 2023

Contenidos

- ¿Qué es un conjunto de clasificadores? ¿Cómo construirlo?
- Bagging, Boosting, Random forests, class-switching, Gradient Boosting
- Combinar salida
- Stacking
- ¿Por qué funcionan?

Introducción

Teorema del jurado de Condorcet:

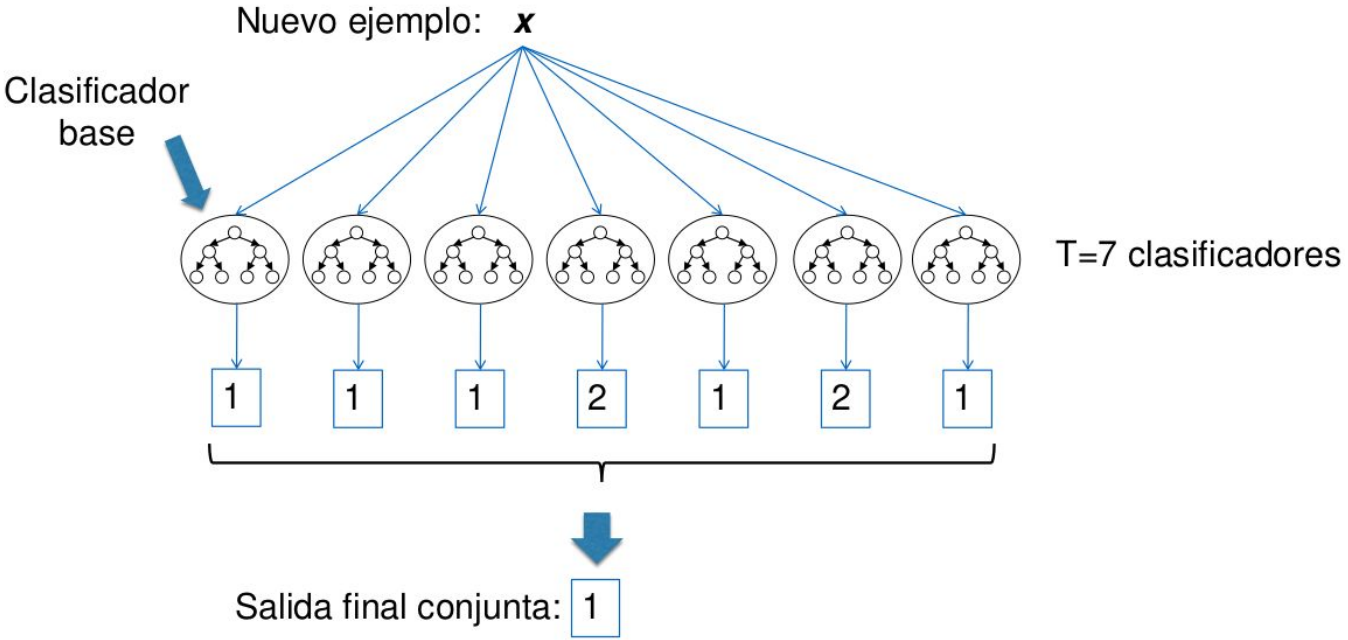
- Dado un jurado (votantes) y suponiendo:
  - Errores independientes
  - La probabilidad de cada miembro del jurado de acertar es superior al 50%

La probabilidad del jurado de acertar tiende a 100% al aumentar el número de miembros del jurado

Nicolas de Condorcet (1743-1794)
Matemático francés

¿Qué es un conjunto (ensemble) de clasificadores?

Una combinación de clasificadores que dan una salida final conjunta. La predicción del conjunto es una combinación de las predicciones individuales.



¿Por qué construir conjuntos de clasificadores?

- Combinar la “opinión” de diferentes expertos puede mejorar la precisión del conjunto
- Divide y vencerás: cada clasificador puede enfocarse en una parte del problema
- No hay modelos universales. Cada modelo es bueno para algunos problemas y malo para otros
- Un conjunto es en general más robusto, funciona mejor que cualquiera de los modelos que lo componen

Requisitos

- Los clasificadores deben ser distintos, no cometer los mismos errores (**diversidad**)
- Los clasificadores deben ser suficientemente precisos, mejores que el azar (**precisión**)

Ejemplo 1

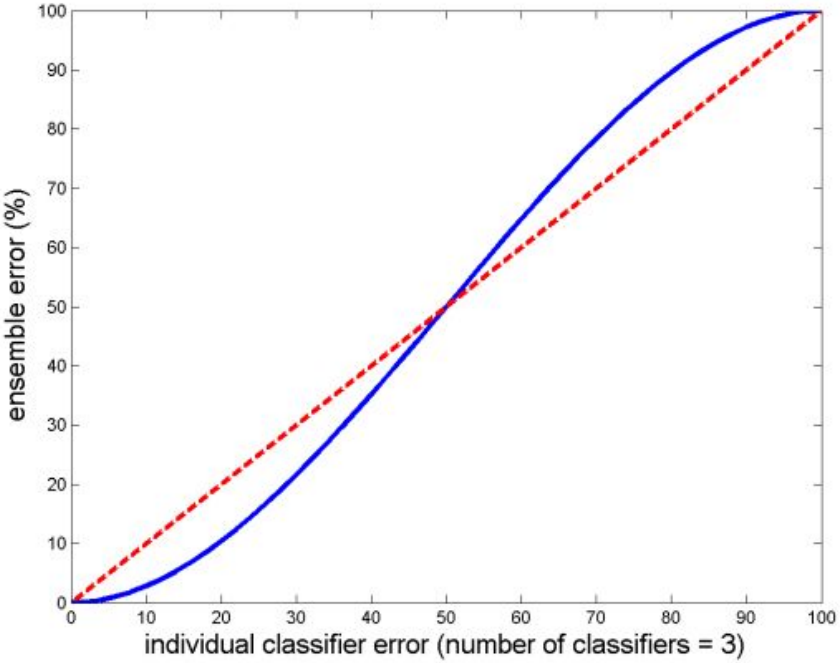
- 3 clasificadores
- Cada uno se equivoca con probabilidad p, de manera independiente de los demás
- El conjunto clasifica por mayoría

Classifier 1	Classifier 2	Classifier 3	p (result)
E	E	H	p·p·(1-p)
E	H	E	p·(1-p)·p
H	E	E	(1-p)·p·p
E	E	E	p·p·p

¿Cuál es la probabilidad de que el conjunto se equivoque?

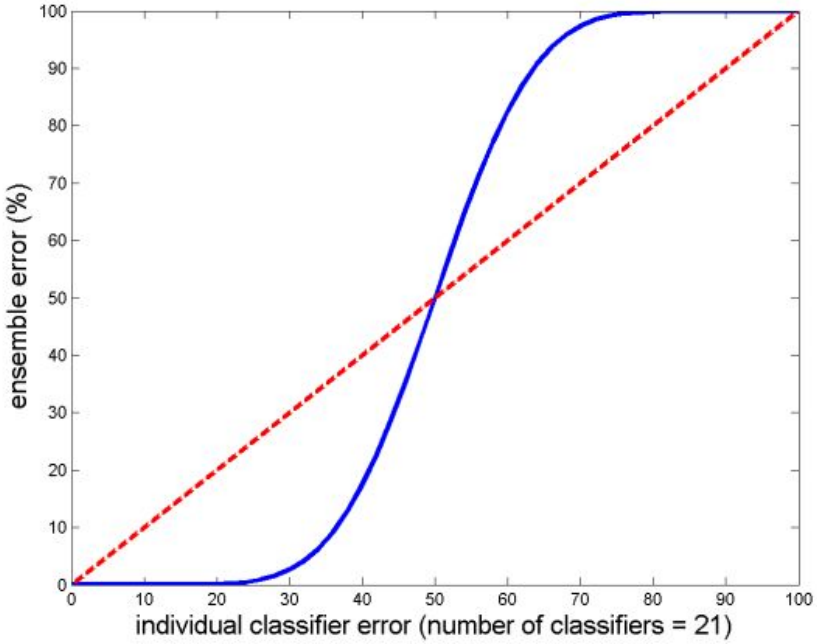
Ejemplo 1

3 clasificadores independientes



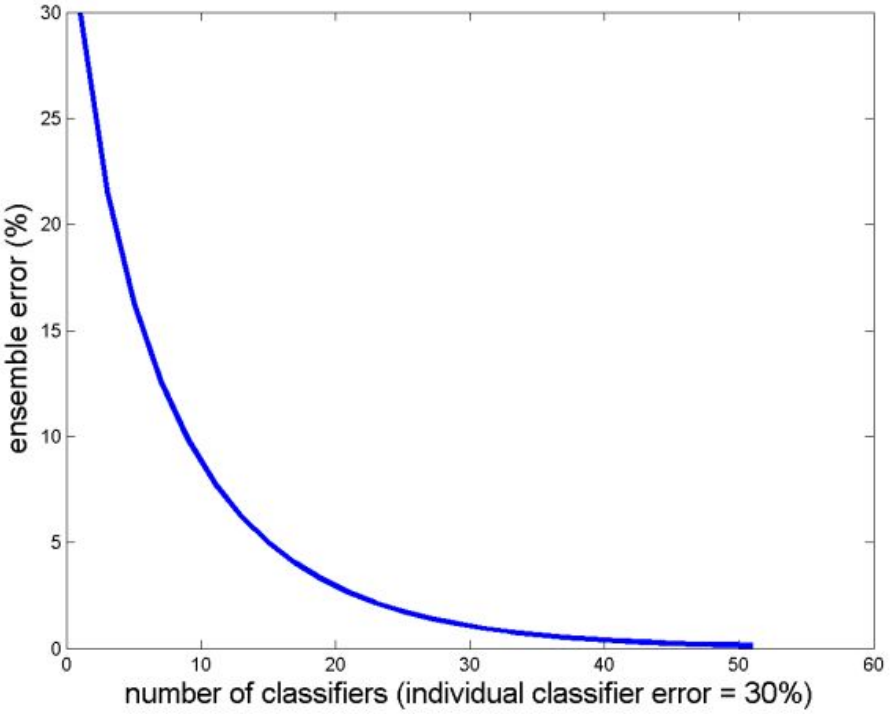
Ejemplo 2

21 clasificadores independientes



Ejemplo 3

Probabilidad de error p = 0.3
Varía el número de clasificadores (siempre impar)

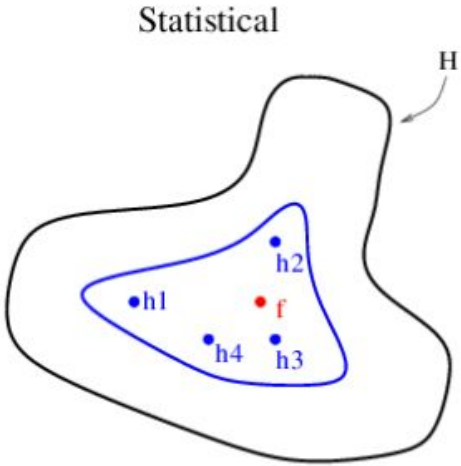


¿Por qué funcionan?

- Motivos estadísticos
- Motivos numéricos/computacionales
- Motivos representacionales

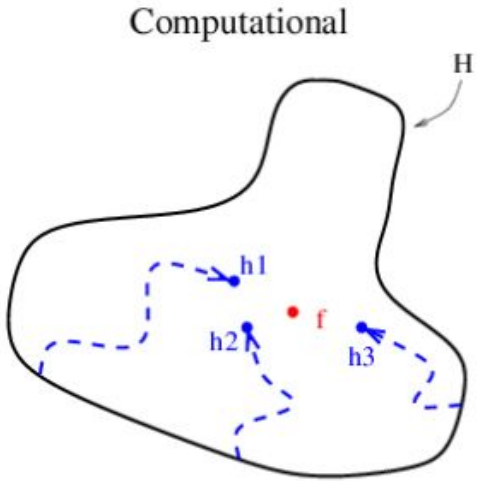
Motivos estadísticos

- No hay suficientes datos (pocos en comparación con el tamaño del espacio de hipótesis)
- Muchas hipótesis son compatibles con los datos
- Combinar varios clasificadores reduce el riesgo de elegir uno particularmente malo



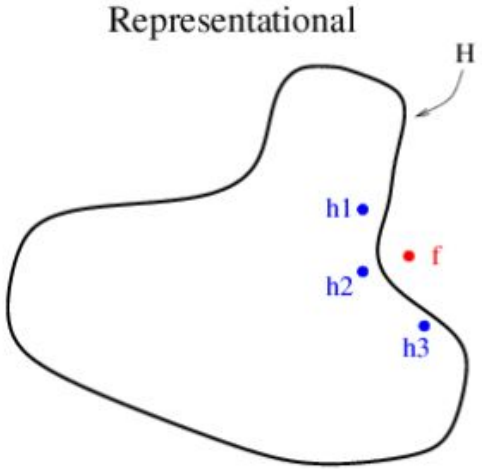
Motivos numéricos/computacionales

- El algoritmo no puede alcanzar la solución óptima (muchos mínimos locales)
- Difícil encontrar la solución óptima usando una búsqueda local
- La combinación de varios clasificadores puede aliviar este problema

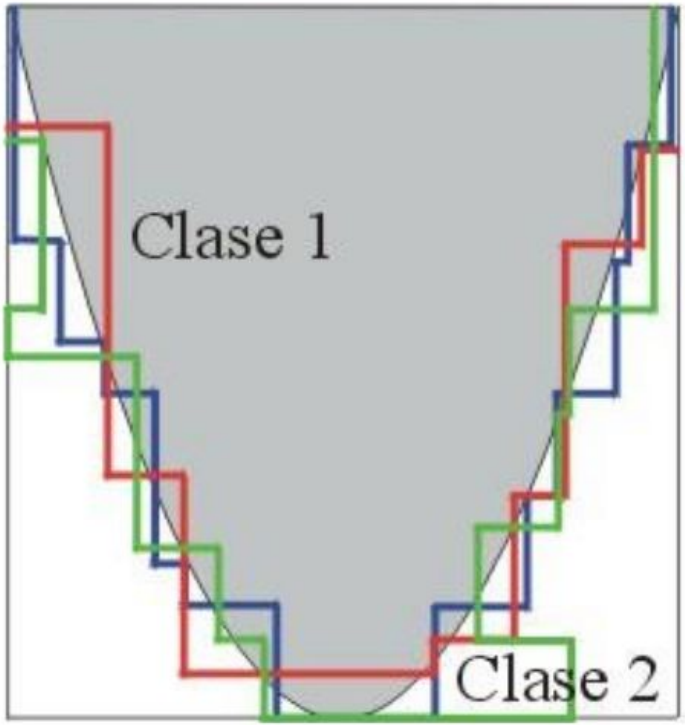


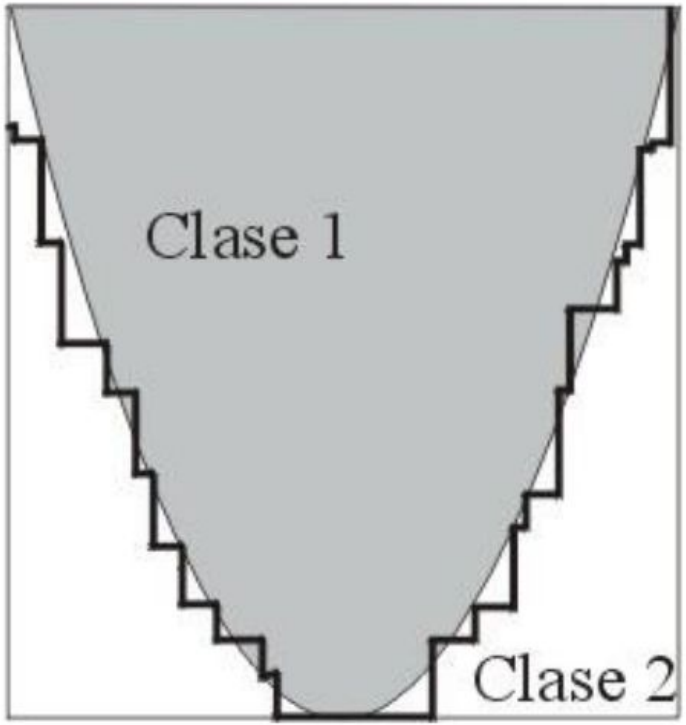
Motivos representacionales

- La solución está fuera del espacio de hipótesis que estamos explorando
- El tipo de clasificador elegido no es capaz de modelar la solución correcta
- Combinar de varios clasificadores puede generar soluciones fuera del espacio de hipótesis de los clasificadores



Ejemplo





Métodos de construcción

La idea fundamental es generar clasificadores con un alto grado de **diversidad**

Diferentes métodos:

- Manipulación de los **datos**
- Manipulación de los **atributos**
- Manipulación de los **objetivos/clases**
- Manipulación de los **hiperparámetros** (cambios en los algoritmos)

Estas estrategias se pueden combinar

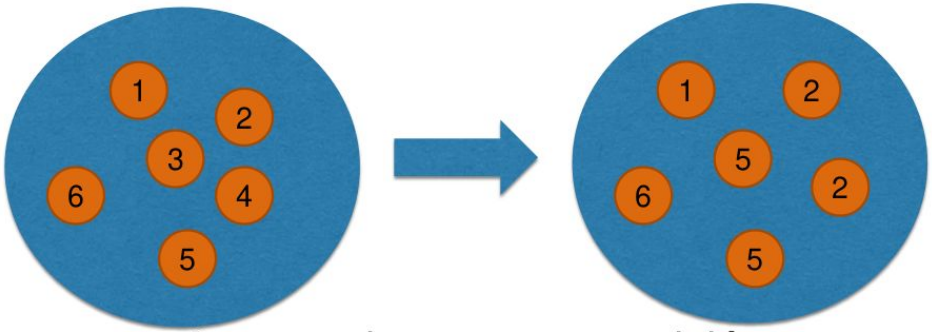
En general, a mayor aleatorización (mayor diversidad) mejores resultados

Manipulación de los datos

- Cada clasificador se entrena con un subconjunto de datos distinto
- Remuestreo de datos
  - Bootstrap (Bagging)
  - Muestreo ponderado (Boosting)
- Muy útil para clasificadores **inestables**: un pequeño cambio en los datos de entrenamiento genera clasificadores muy diferentes

Bagging

Bootstrap: muestreo aleatorio con reemplazo, mismo tamaño que el conjunto original



6 extracciones con repetición

Bagging (**b**ootstrap **agg**regat**ing**): cada clasificador se entrena con un bootstrap distinto

Bagging

Input:

Dataset D = ($\mathbf{X}_i$, y_i) i=1...N
Ensemble size T

1. **for** t=1 **to** T:
2. sample = BootstrapSample(D)
3. h_t = TrainClassifier(sample)

Bootstrap

+

Output:

$$H(\mathbf{x}) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} I(h_t(\mathbf{x}) = j) \right)$$

Agregación (por voto)

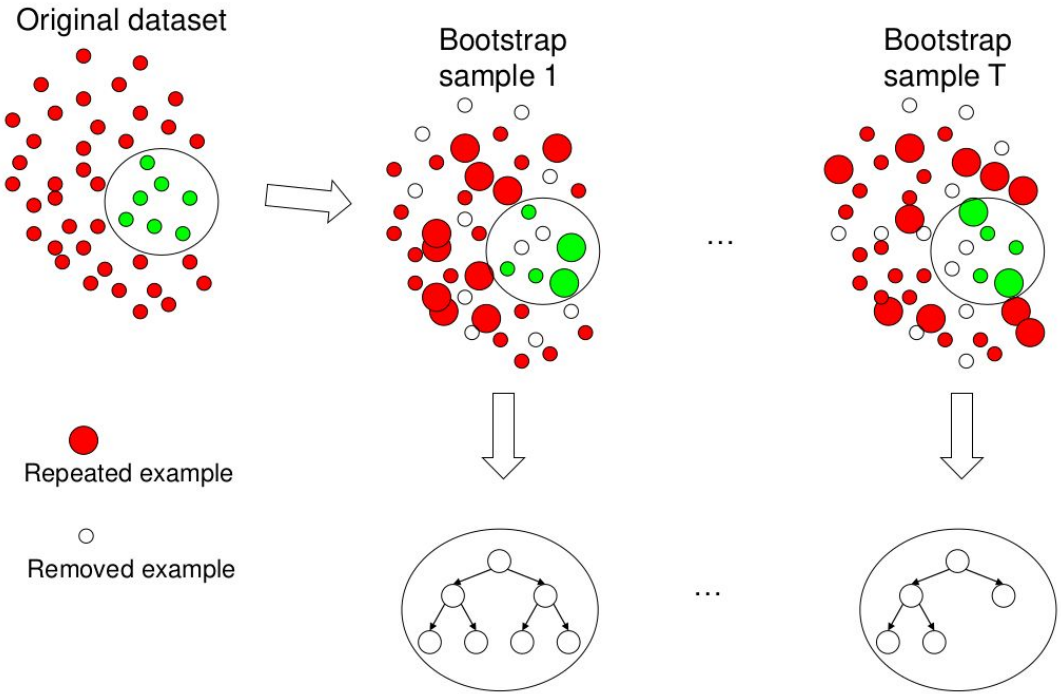
Bagging

Algorithm 1 Bagging

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$
for $t = 1$ to T **do**
  Build a dataset S_t, by sampling N items, randomly *with replacement* from S.
  Train a model h_t using S_t, and add it to the ensemble.
end for
For a new testing point (x', y'),
If model outputs are continuous, combine them by averaging.
If model outputs are class labels, combine them by voting.

Bagging

Cada clasificador se entrena con un bootstrap distinto



Bagging

- Usa un 63,2% de los datos de entrenamiento en media para construir cada clasificador
- Los clasificadores se combinan mediante **voto simple**
- Muy robusto frente a ruido en las etiquetas de clase
- En general, mejora el rendimiento del clasificador base
- Se puede paralelizar fácilmente
- Funciona muy bien para clasificadores **inestables** (por ejemplo árboles de decisión)

Bagging en scikit-learn

sklearn.ensemble.BaggingClassifier

class `sklearn.ensemble.`**`BaggingClassifier`**(*estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0, base_estimator='deprecated'*) [source]

A Bagging classifier.

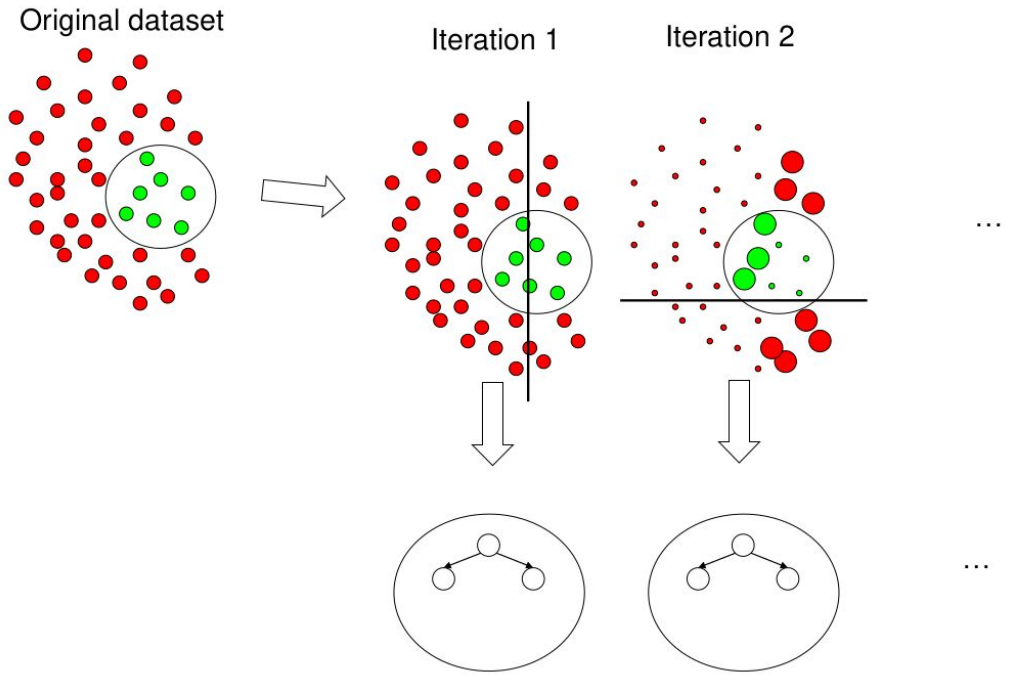
A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn with replacement, then the method is known as Bagging [2]. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces [3]. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches [4].

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Boosting

Cada clasificador añadido al conjunto intenta mejorar los errores de los anteriores



Boosting

Adaboost (**ada**ptive **boost**ing):

- Cada dato tiene un **peso** asociado, que determina la probabilidad de ser elegido para construir el siguiente clasificador
- El peso aumenta para los datos mal clasificados
- El peso disminuye para los datos bien clasificados
- El voto de los clasificadores es **ponderado** (en base a su error)

Boosting

Entrada:

Datos D =($\mathbf{X_i}$, y_i) i=1...N

Tamaño del conjunto T

1. w_i = *1/N para i=1...N*
2. **for** t=1 **to** T:
3. h_t = BuildClassifier(D, **w**)
4. e_t = WeightedError(D, **w**)
5. **if** e_t==*0* **or** $e_t \geq 0.5$ **break**
6. Modificar pesos de los ejemplos a

* $w_i = \frac{w_i}{2e_t}$ $\quad \forall i \mid h_t(\boldsymbol{X_i}) \neq y_i$

* $w_i = \frac{w_i}{2(1-e_t)}$ $\quad \forall i \mid h_t(\boldsymbol{X_i}) = y_i$

Boosting

Algorithm 2 Adaboost

Input: Required ensemble size T

Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

Define a uniform distribution $D_1(i)$ over elements of S.

for $t = 1$ to T **do**

Train a model h_t using distribution D_t.

Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$

If $\epsilon_t \geq 0.5$ break

Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

end for

For a new testing point (x', y'),

$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x') \right)$

Boosting

- Muy buen error de generalización (en promedio)
- No es robusto frente al ruido en las etiquetas (clases)
- Muy sensible a outliers
- Puede incrementar el error del clasificador de base
- No es fácil de paralelizar

Adaboost en scikit-learn

sklearn.ensemble.AdaBoostClassifier

class `sklearn.ensemble.`**`AdaBoostClassifier`**(*estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None, base_estimator='deprecated'*) [source]

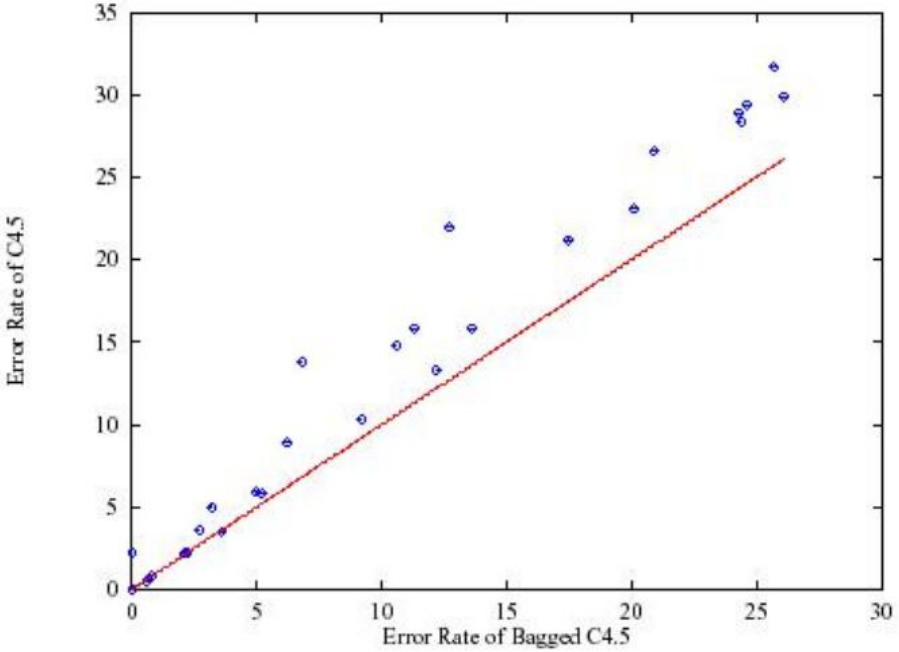
An AdaBoost classifier.

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

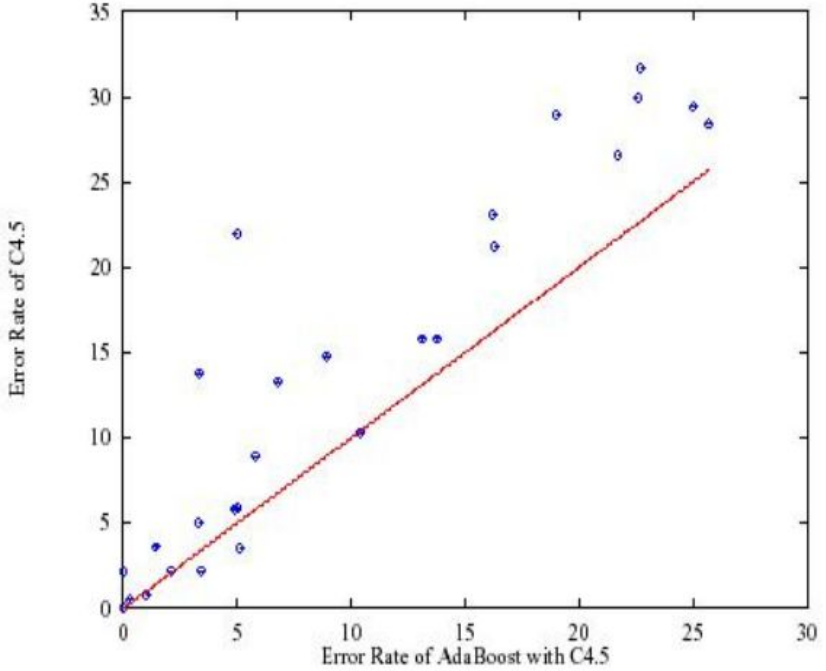
This class implements the algorithm known as AdaBoost-SAMME [2].

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Bagging vs boosting



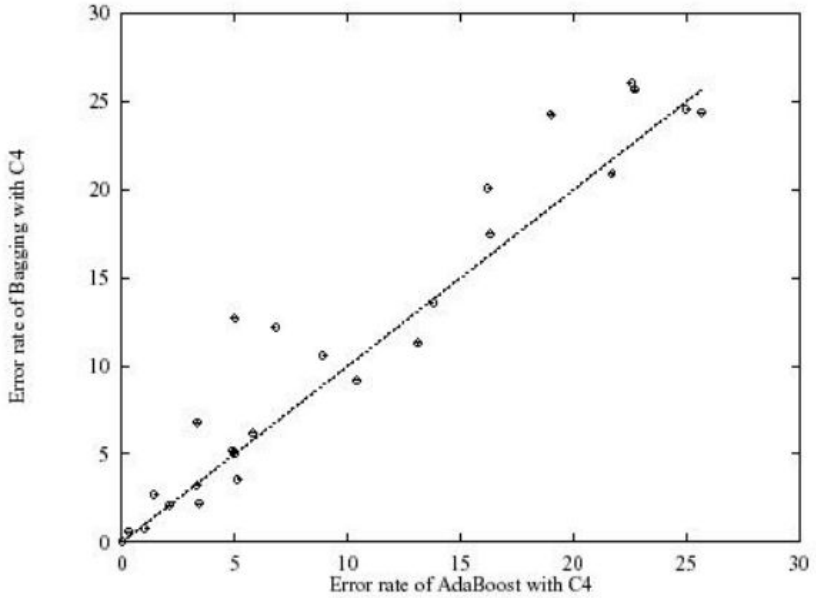
Bagging



Adaboost

Bagging vs boosting

- Adaboost normalmente tiene mejor rendimiento que bagging
- Adaboost funciona peor cuando hay ruido, pues tiende a sobreajustar más

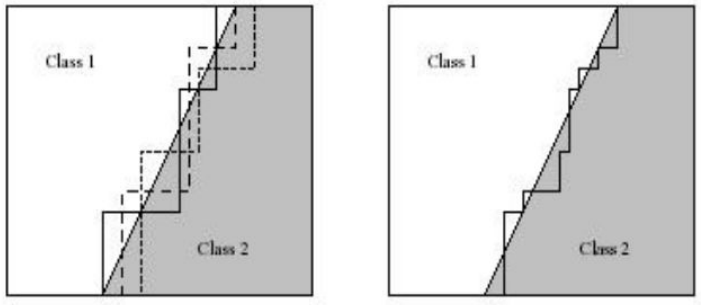


Conjuntos de árboles de decisión

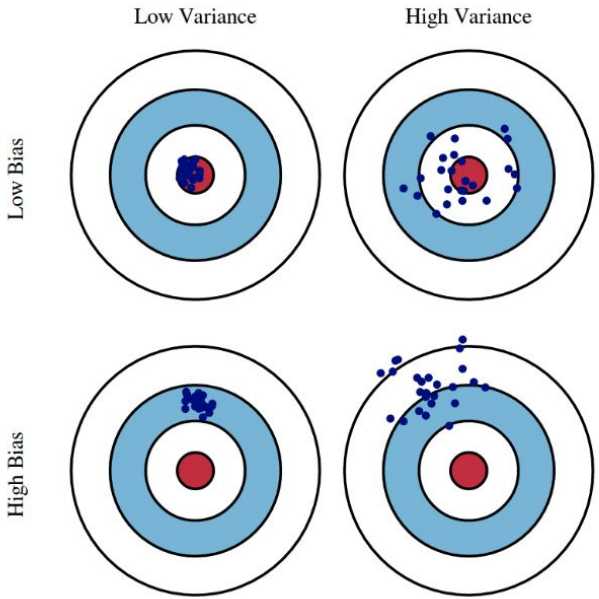
- Los árboles de decisión son clasificadores **débiles**
- Los árboles de decisión son clasificadores **inestables**

Además:

- Problemas **estadísticos**: muchos árboles distintos compatibles con los datos
- Problemas **numéricos/computacionales**: algoritmo greedy
- Problemas **representacionales**: particiones paralelas a los ejes



Sesgo y varianza



Bagging

- Ataca los problemas estadístico y numérico
- Tiende a reducir la varianza

Boosting

- Ataca el problema representacional
- Tiende a reducir el sesgo
- Mayor sobreajuste a los datos

Métodos de construcción

La idea fundamental es generar clasificadores con un alto grado de **diversidad**

Diferentes métodos:

- Manipulación de los **datos**
- Manipulación de los **atributos**
- Manipulación de los **objetivos/clases**
- Manipulación de los **hiperparámetros** (cambios en los algoritmos)

Estas estrategias se pueden combinar

En general, a mayor aleatorización (mayor diversidad) mejores resultados

Manipulación de los atributos

Cada clasificador se entrena con un subconjunto distinto de los atributos

Random forest

- Igual que en bagging, cada clasificador se entrena con un bootstrap distinto
- El árbol se construye realizando cada corte con un subconjunto aleatorio de los atributos
- Normalmente no se poda

Random forest

- Introduce mayor grado de aleatoriedad que bagging (más diversidad)
- En general mejor rendimiento que boosting
- Muy robusto frente a ruido
- Poco sobreajuste
- Fácilmente paralelizable
- Los árboles aleatorios se entrenan muy rápidamente

Random forest en scikit learn

sklearn.ensemble.RandomForestClassifier

class `sklearn.ensemble.`**`RandomForestClassifier`**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

For a comparison between tree-based ensemble models see the example Comparing Random Forests and Histogram Gradient Boosting models.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Manipulación de los objetivos/clases

Diferentes técnicas para alterar las etiquetas:

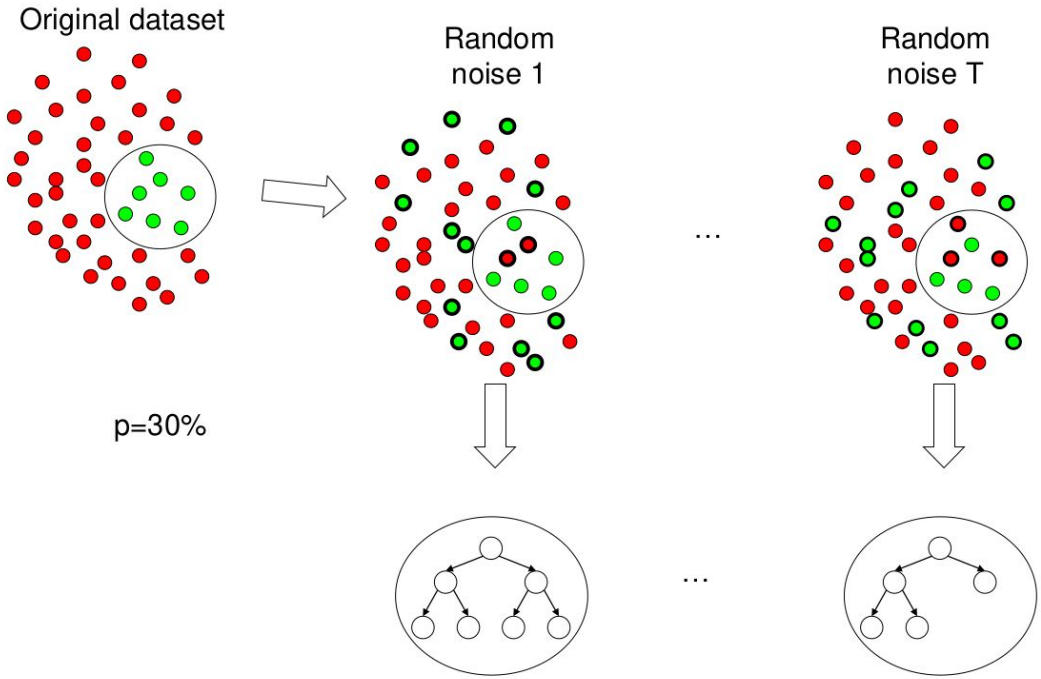
- Agrupación de clases
- Nuevas clases aleatorias
- Modificación aleatoria de las clases (**class switching**)

Class switching

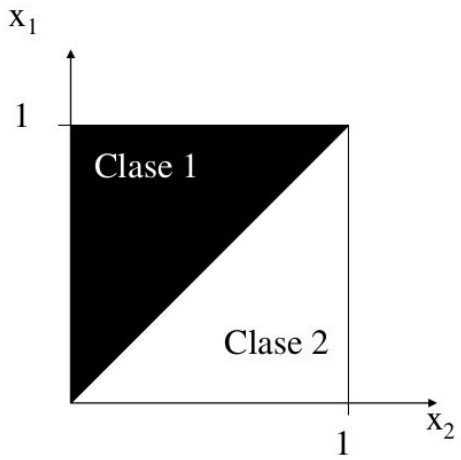
- La diversidad se consigue modificando los datos de entrenamiento con ruido aleatorio en las clases
- Para cada clasificador, se alteran los datos modificando las clases aleatoriamente con probabilidad p

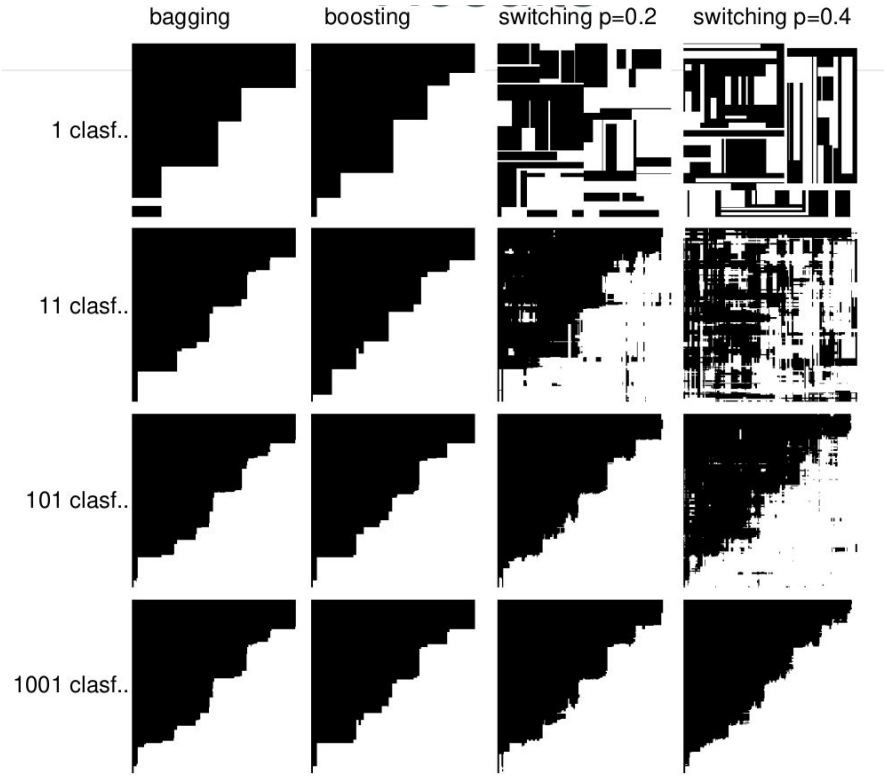
Class switching

Para cada clasificador se alteran las clases del problema



Ejemplo


x1
1
Clase 1
Clase 2
1
x2


bagging
boosting
switching p=0.2
switching p=0.4
1 clasf..
11 clasf..
101 clasf..
1001 clasf..

Combinación de clasificadores

Estrategias basadas en voto:

- Voto simple (bagging)
- Voto ponderado (adaboost)

Otras estrategias:

- Valor máximo, mínimo, promedio o mediana
- Stacking

Stacking

- La fase de combinación se incluye en el proceso de aprendizaje
- Primero se entrenan los clasificadores
- A continuación se utilizan las predicciones de estos clasificadores como atributos para un **meta-clasificador**

Stacking en scikit learn

sklearn.ensemble.StackingClassifier

class `sklearn.ensemble.`**`StackingClassifier`**(*estimators, final_estimator=None, *, cv=None, stack_method='auto', n_jobs=None, passthrough=False, verbose=0*) [source]

Stack of estimators with a final classifier.

Stacked generalization consists in stacking the output of individual estimator and use a classifier to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

Note that `estimators_` are fitted on the full `X` while `final_estimator_` is trained using cross-validated predictions of the base estimators using `cross_val_predict`.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

Conclusiones

Aspectos **positivos** de la clasificación mediante conjuntos:

- Muy buen rendimiento
- Pocos parámetros que ajustar
- La combinación de muchos clasificadores reduce el riesgo de error
- Mejora la generalización y reduce el sobreajuste
- Si se usan árboles de decisión el entrenamiento es muy rápido

Conclusiones

Aspectos **negativos** de la clasificación mediante conjuntos:

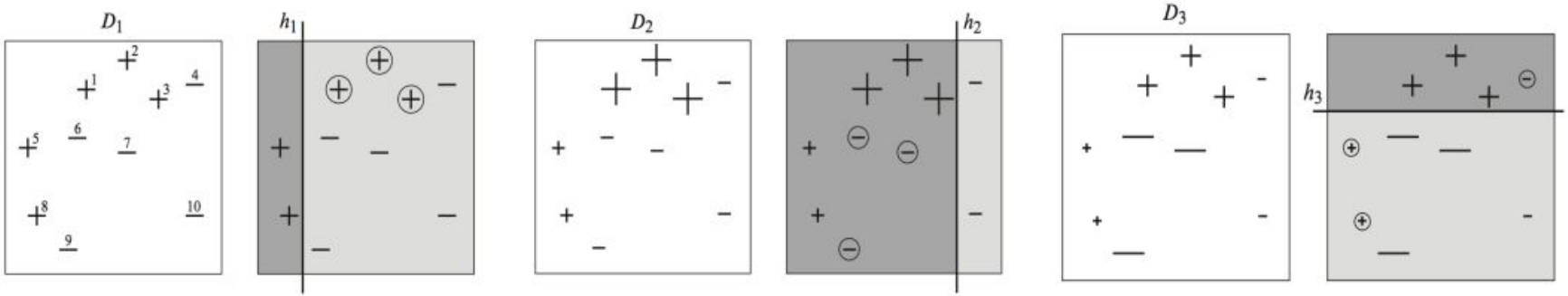
- Son más lentos que un clasificador simple
- Puede no ser una buena opción para clasificadores complejos
- Se pierde la interpretabilidad de un árbol de decisión

Extra - Gradient Boosting

Gradient boosting

Descenso por gradiente + boosting

Adaboost:



- Cada clasificador intenta compensar las debilidades de los anteriores
- Esas debilidades se identifican con un vector de pesos
- En gradient boosting las debilidades se identifican con el gradiente

Ejemplo

- Sea el problema $(\boldsymbol{x}_1, t_1)$, $(\boldsymbol{x}_2, t_2)$, …, $(\boldsymbol{x}_N, t_N)$, $t \in \mathbf{R}$ (regresión)
- Supongamos que tenemos un modelo previo $y = F(\boldsymbol{x})$
- El modelo no es perfecto, en general $y_i \neq t_i$
- Queremos mejorar el modelo, pero:
  - No podemos cambiar nada del modelo previo *F*
  - Sólo podemos añadir un modelo adicional a *F* (supondremos un árbol de regresión $h(\boldsymbol{x})$)
  - La nueva predicción será, por tanto, $y' = F(\boldsymbol{x}) + h(\boldsymbol{x})$

Ejemplo

Idealmente:

$$y_1' = F(\mathbf{x}_1) + h(\mathbf{x}_1) = t_1$$

$$y_2' = F(\mathbf{x}_2) + h(\mathbf{x}_2) = t_2$$

$$\dots$$

$$y_N' = F(\mathbf{x}_N) + h(\mathbf{x}_N) = t_N$$

O bien:

$$h(\mathbf{x}_1) = t_1 - F(\mathbf{x}_1) = t_1 - y_1$$

$$h(\mathbf{x}_2) = t_2 - F(\mathbf{x}_2) = t_2 - y_2$$

$$\dots$$

$$h(\mathbf{x}_N) = t_N - F(\mathbf{x}_N) = t_N - y_N$$

Ajustemos el árbol de regresión $h(\mathbf{x})$ al nuevo problema:

$$(\mathbf{x}_1, t_1 - y_1), (\mathbf{x}_2, t_2 - y_2), \dots, (\mathbf{x}_N, t_N - y_N)$$

Ejemplo

Estamos construyendo un nuevo modelo para aproximar los **residuos**

- $r_i = t_i - y_i$
- La parte del problema que el modelo *F* no resuelve bien

Si el nuevo modelo *F* + *h* todavía no funciona bien, podemos repetir el proceso

Relación con el gradiente

Para un problema de regresión, usando como función de coste el MSE, el residuo coincide con menos el gradiente de la función de coste:

$$L_i = (t_i - y_i)^2 / 2$$

$$L = \Sigma_i L_i$$

$$- (\partial L / \partial y_i) = y_i - t_i$$

Podemos interpretar los residuos como gradientes negativos

Y todo el proceso como un descenso por gradiente (*F* se ha actualizado sumando *h*, que se ha ajustado a los residuos, que son a su vez equivalentes al gradiente)

Relación con el gradiente

¿Podemos generalizar esta idea a otros problemas / funciones de coste?

1. Partimos de modelo inicial $F(\mathbf{x})$
2. Calculamos los gradientes $-g(\mathbf{x}_i)$
3. Ajustamos el nuevo modelo $h(\mathbf{x})$ a los gradientes
4. Actualizamos $F(\mathbf{x}) = F(\mathbf{x}) + \rho h(\mathbf{x})$

Gradient boosting en scikit-learn

sklearn.ensemble.GradientBoostingClassifier

class `sklearn.ensemble.`**`GradientBoostingClassifier`**(**, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0*) [source]

Gradient Boosting for classification.

This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced.

`sklearn.ensemble.HistGradientBoostingClassifier` is a much faster variant of this algorithm for intermediate datasets (`n_samples >= 10_000`).

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html