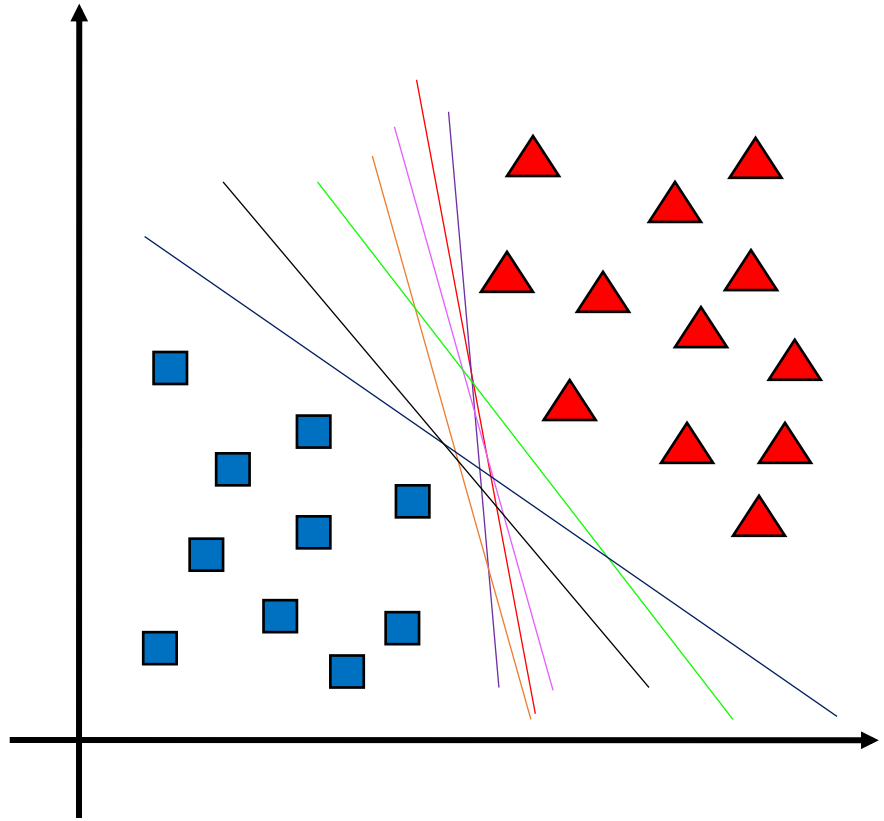
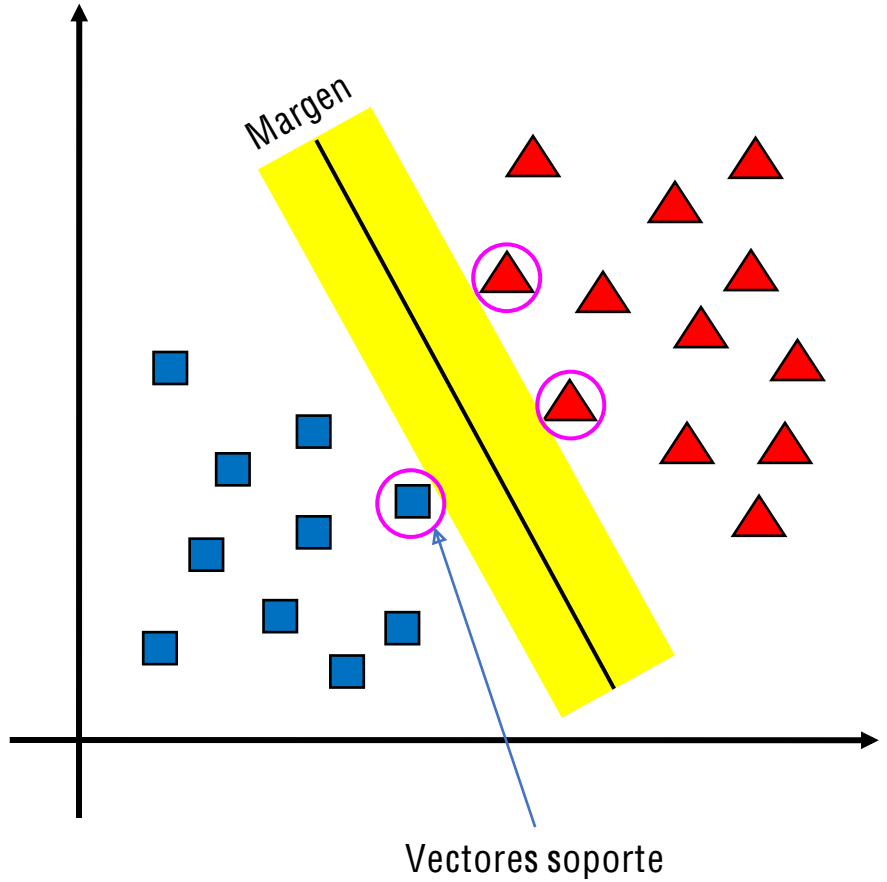
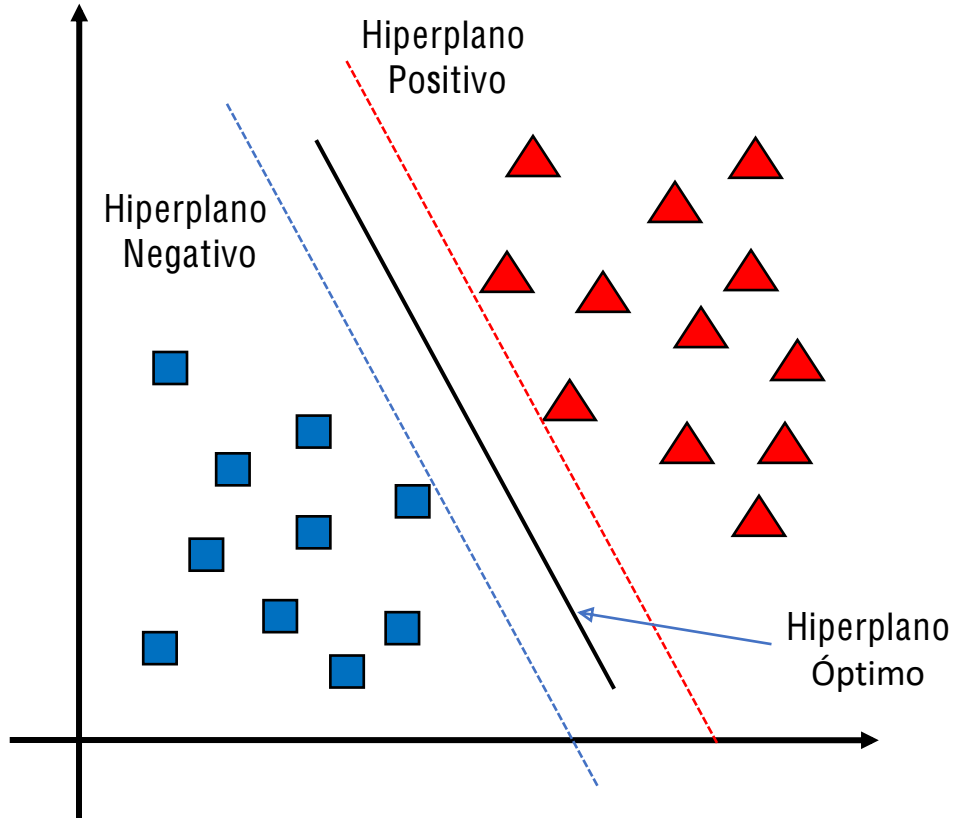
- Las **máquinas de vectores soporte (SVM**) son clasificadores no paramétricos basados en funciones discriminantes linearles.
- Son **clasificadores dicotómicos**, ya que se limitan a **discriminar entre 2 clase distintas.** No obstante, esto **no implica que no se puedan abordar problemas multiclase** utilizando este tipo de algoritmo.
- Los clasificadores SVM consisten **en hallar un hiperplano óptimo** capaz de **separar el espacio muestral en dos regiones**, de manera que cada región pertenezca a una clase.
- Para cada hiperplano se calcula el **margen**, que se define como la distancia entre el hiperplano y los dos puntos más cercanos de cada clase.
- El **hiperplano óptimo** es aquel que maximiza el margen.
- A los dos puntos más cercanos de cada clase (los que tocan con el límite del margen) se les llama **vectores soporte**.
- De esta manera, quedan definidos un **hiperplano positivo** y un **hiperplano negativo.**



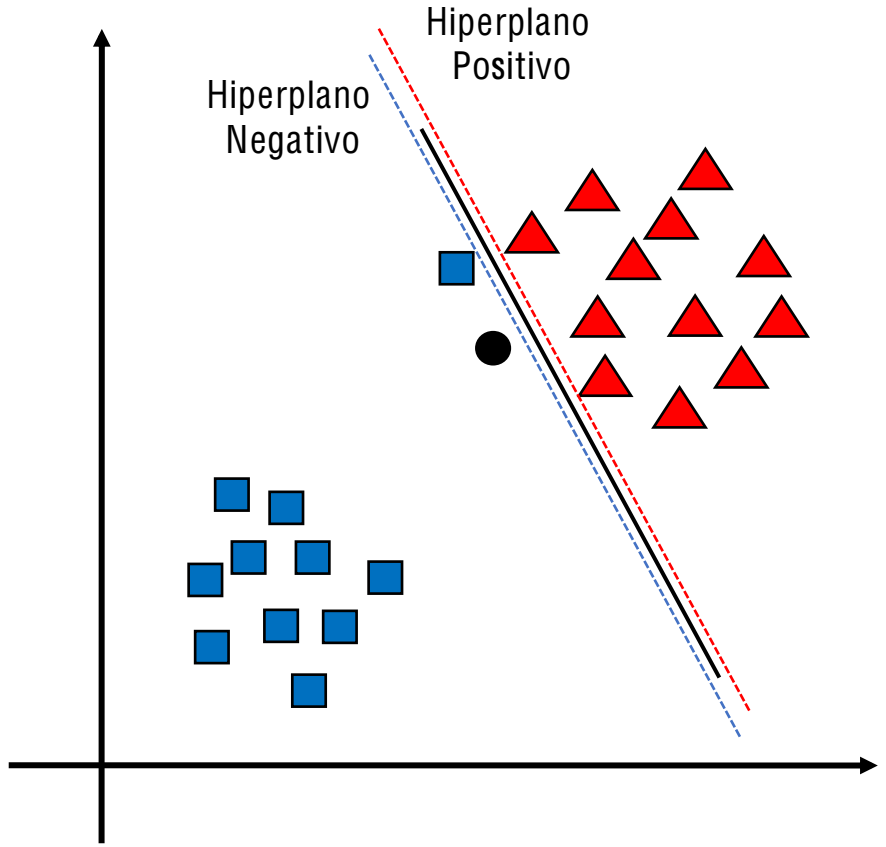
- Las **máquinas de vectores soporte (SVM**) son clasificadores no paramétricos basados en funciones discriminantes linearles.
- Son **clasificadores dicotómicos**, ya que se limitan a **discriminar entre 2 clase distintas.** No obstante, esto **no implica que no se puedan abordar problemas multiclase** utilizando este tipo de algoritmo.
- Los clasificadores SVM consisten **en hallar un hiperplano óptimo** capaz de **separar el espacio muestral en dos regiones**, de manera que cada región pertenezca a una clase.
- Para cada hiperplano se calcula el **margen**, que se define como la distancia entre el hiperplano y los dos puntos más cercanos de cada clase.
- El **hiperplano óptimo** es aquel que maximiza el margen.
- A los dos puntos más cercanos de cada clase (los que tocan con el límite del margen) se les llama **vectores soporte**.
- De esta manera, quedan definidos un **hiperplano positivo** y un **hiperplano negativo.**



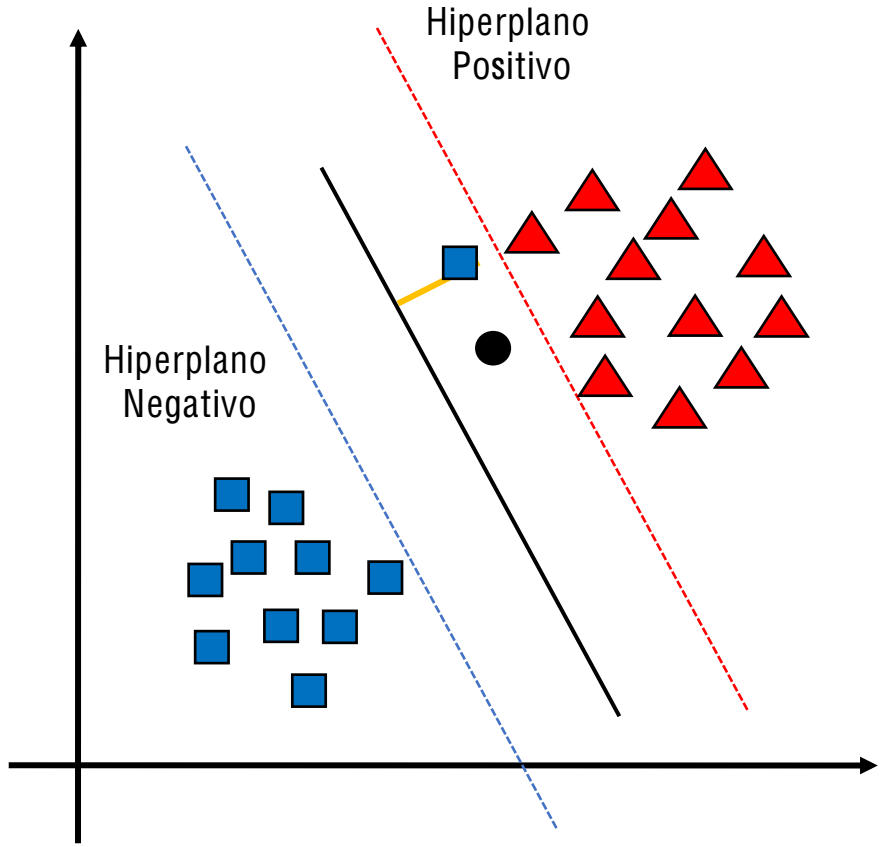
- Las **máquinas de vectores soporte (SVM**) son clasificadores no paramétricos basados en funciones discriminantes linearles.
- Son **clasificadores dicotómicos**, ya que se limitan a **discriminar entre 2 clase distintas.** No obstante, esto **no implica que no se puedan abordar problemas multiclase** utilizando este tipo de algoritmo.
- Los clasificadores SVM consisten **en hallar un hiperplano óptimo** capaz de **separar el espacio muestral en dos regiones**, de manera que cada región pertenezca a una clase.
- Para cada hiperplano se calcula el **margen**, que se define como la distancia entre el hiperplano y los dos puntos más cercanos de cada clase.
- El **hiperplano óptimo** es aquel que maximiza el margen.
- A los dos puntos más cercanos de cada clase (los que tocan con el límite del margen) se les llama **vectores soporte**.
- De esta manera, quedan definidos un **hiperplano positivo** y un **hiperplano negativo.**

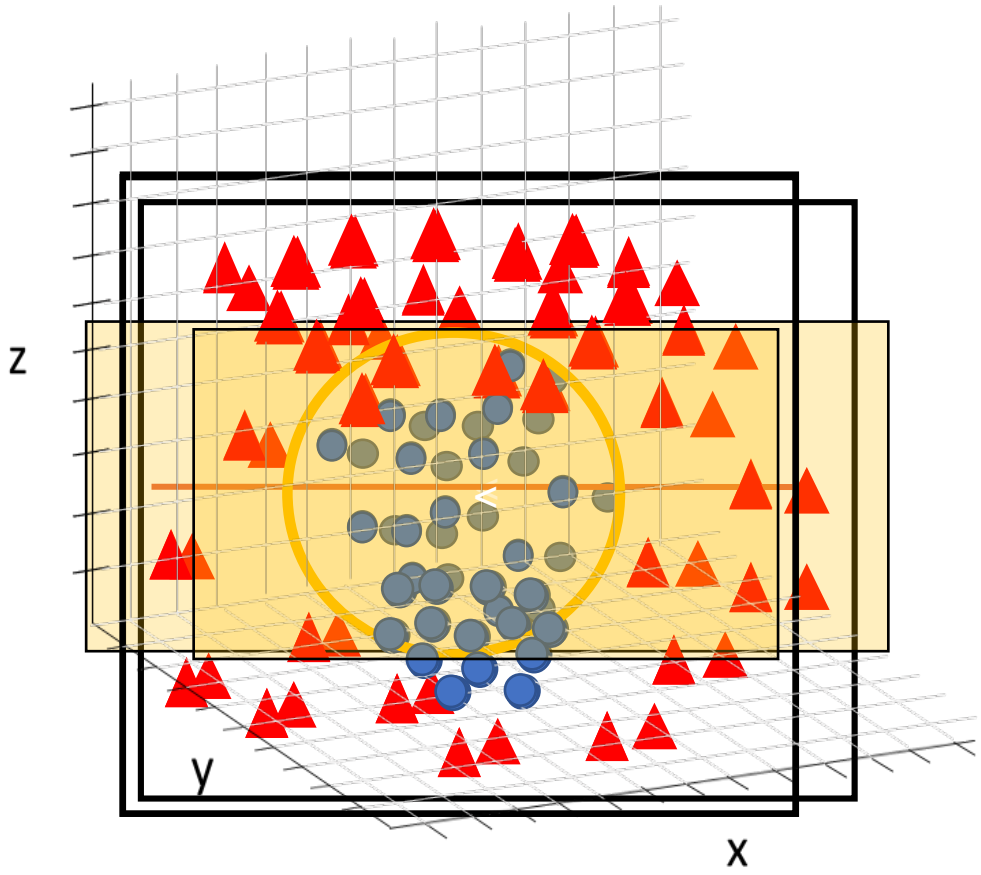


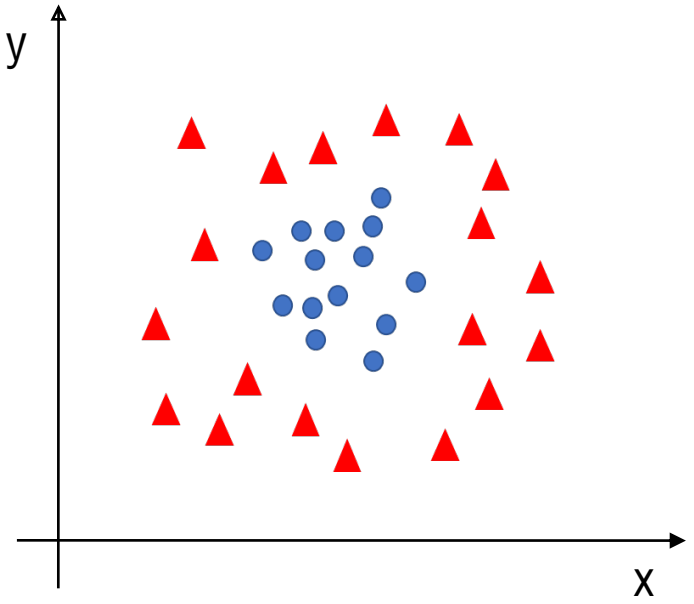
- **¿Cómo calcular el hiperplano óptimo?**
  - ***Support Vector Machines Succinctly by Alexandre Kowalczyk***
    Páginas 39-43
  - *Colgado en el aula del campus virtual:*
    Recursos y Materiales / 03. Materiales del professor / Recursos adicionales / Algoritmos / Support Vector Machines Succinctly

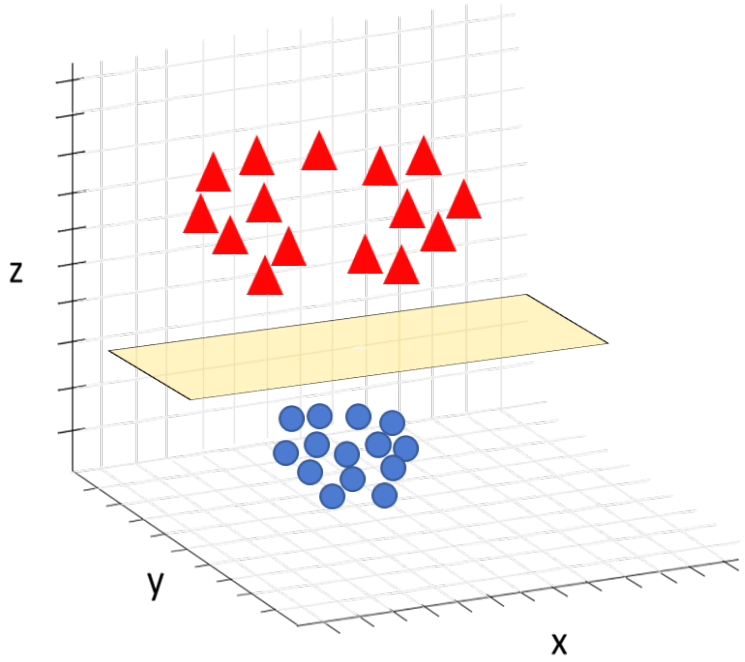


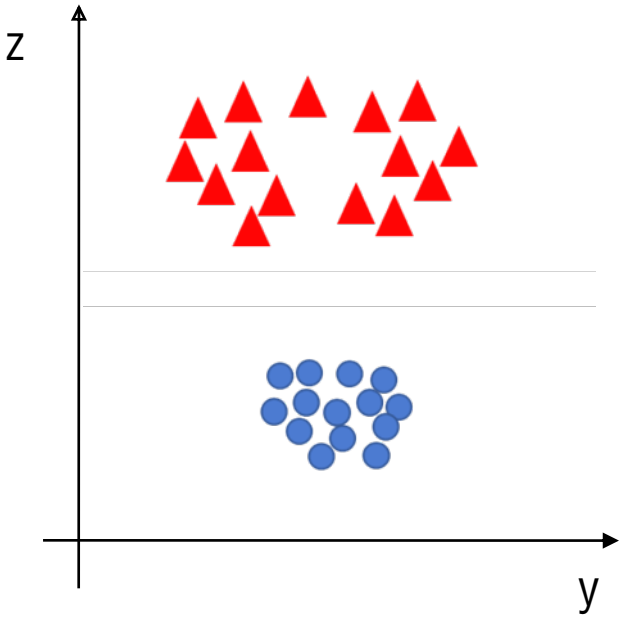
- **¿Cómo calcular el hiperplano óptimo?**
  - *Support Vector Machines Succinctly by Alexandre Kowalczyk*
    Páginas 39-43
  - *Colgado en el aula del campus virtual:*
    Recursos y Materiales / 03. Materiales del professor / Recursos adicionales / Algoritmos / Support Vector Machines Succinctly



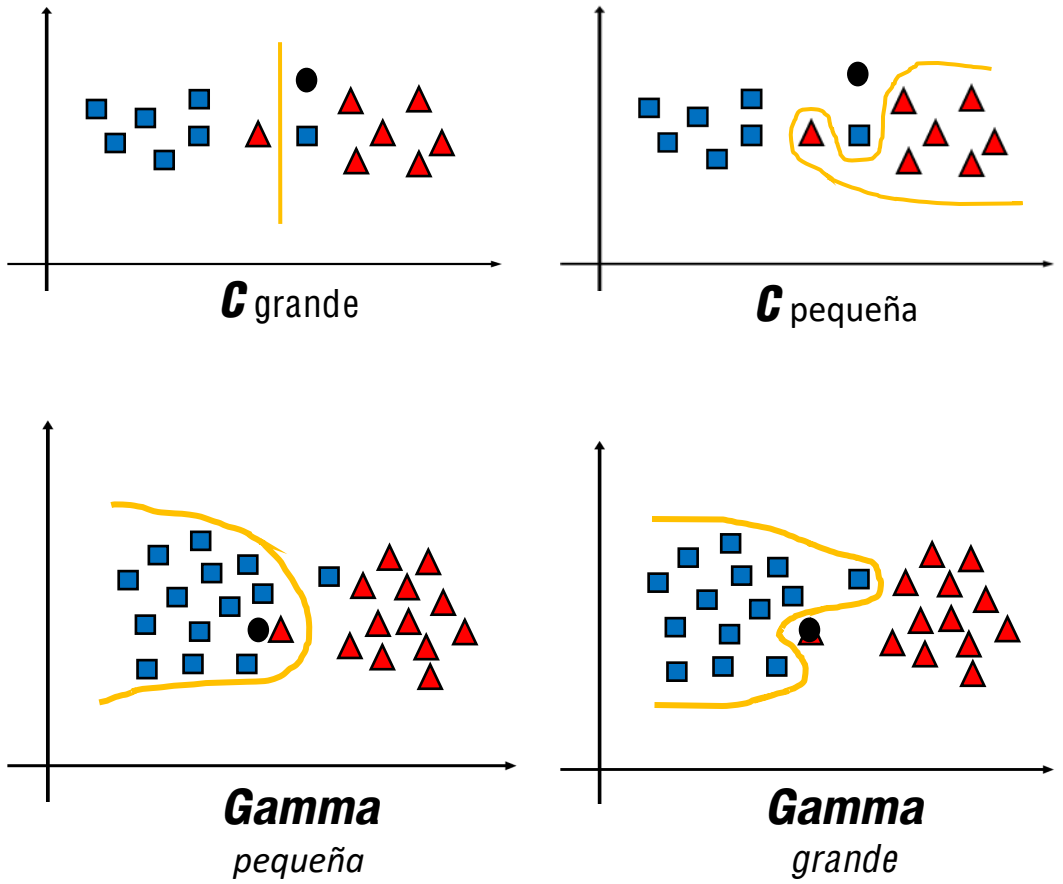

z
y
x


y
x

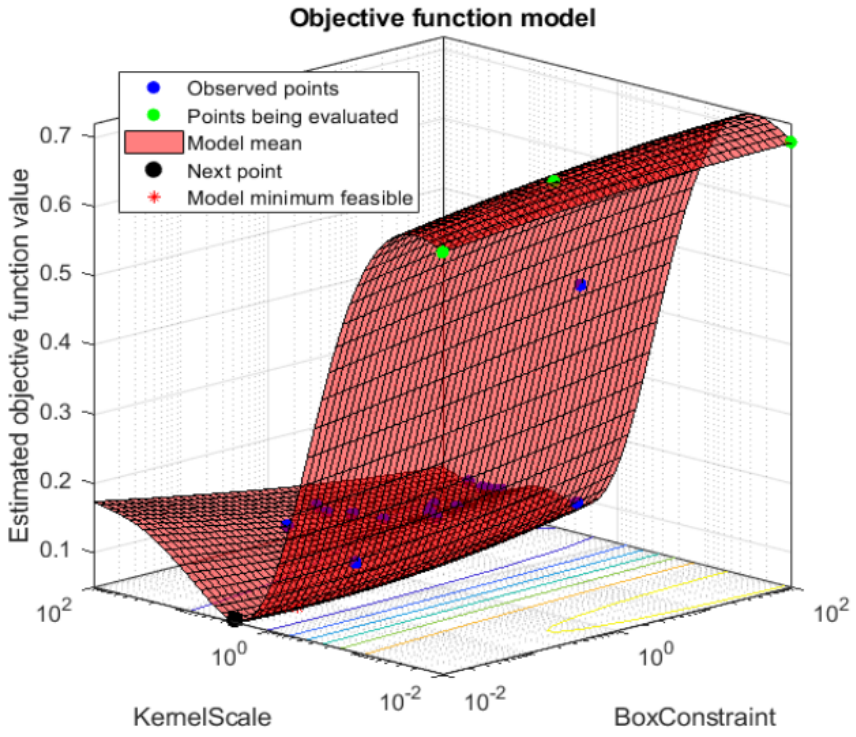

z
y
x

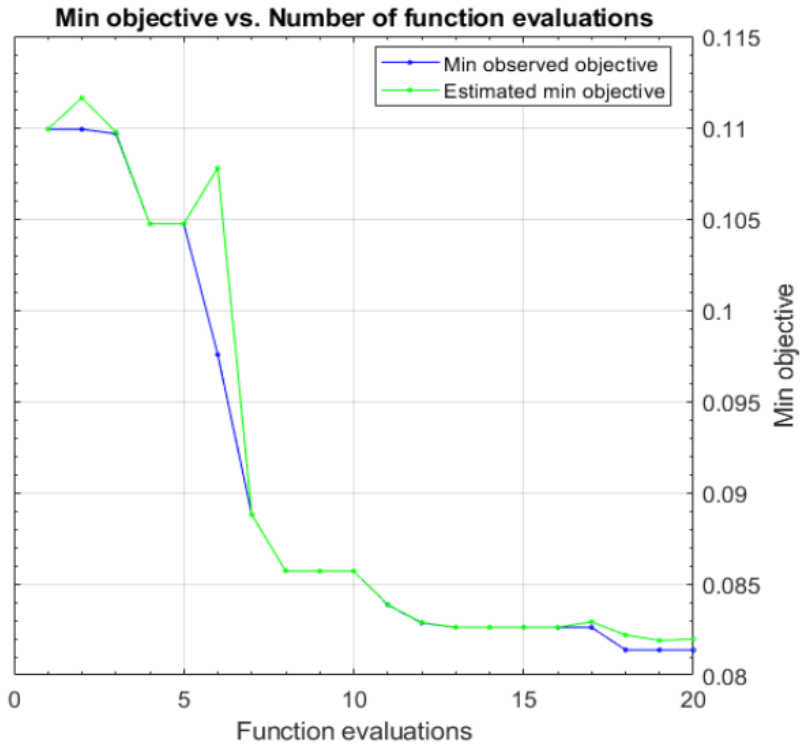

z
y

HIPERPARÁMETROS



HIPERPARÁMETROS





https://www.mdpi.com/1099-4300/21/4/356

VENTAJAS

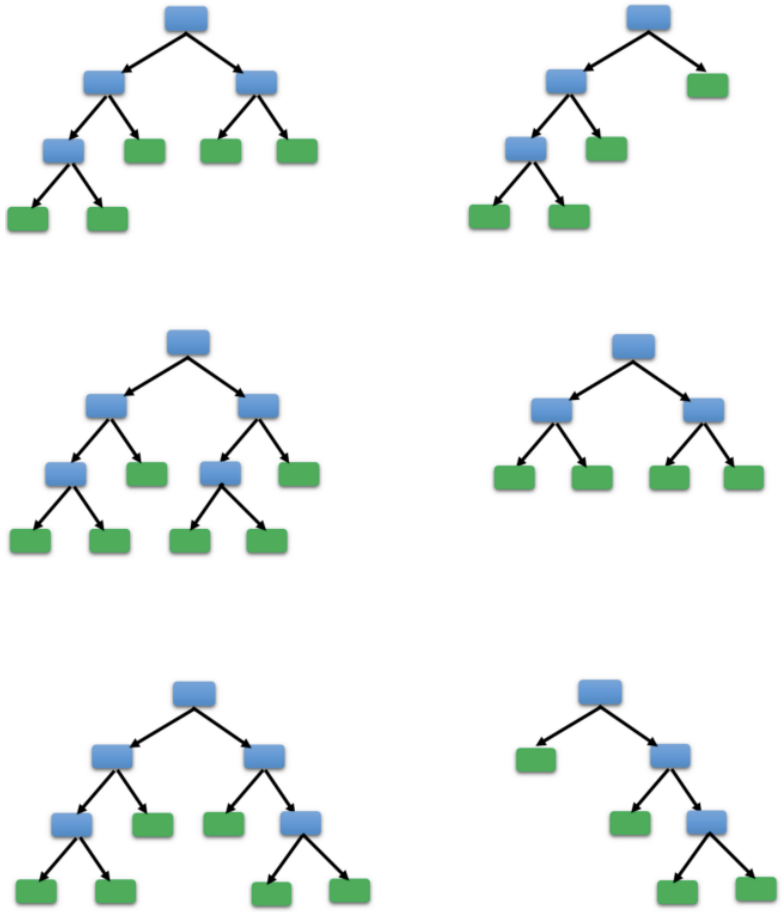
- El **clasificador SVM** utiliza un **método de optimización convexa** (*convex optimization*) que **garantiza una convergencia a un mínimo global** y no a un mínimo local.
- El **algoritmo SVM** es **adecuado tanto para problemas lineales como no lineales** usando el **kernel trick.**
- Modelos basados en **SVM funcionan bien independientemente de la dimensionalidad del espacio de características**.
- **SVM funciona de manera eficaz en conjuntos de datos pequeños**, ya que su rendimiento no está supeditado al uso de todos los datos.

INCONVENIENTES

- **SVM no es adecuado para conjuntos de datos muy grandes** ya que presentan un coste **computacional elevado.**
- **SVM es menos efectivo en conjuntos de datos más ruidosos**, cuando existen **clases superpuestas**.

BAGGING – *Bootstrap aggregating*

- Es una técnica de clasificación que consiste en **dividir el conjunto de entrenamiento en varios subconjuntos** con repetición (*boostrapped datasets).*
- Se entrenan **tantos clasificadores como subconjuntos** haya.
- La **clasificación final** se obtiene **promediando** los resultados de todos los clasificadores.
- Esta técnica permite **reducir la varianza** y **minimizar el *overfitting***.
- Se puede usar **cualquier clasificador**, pero los **más utilizados** son los **árboles de decisión.**
- El **método más conocido** dentro de las **técnicas de *bagging*** es el llamado ***random forest .***



BAGGING – *Bootstrap aggregating*

- El algoritmo ***RANDOM FOREST*** no solo utiliza el concepto de *bagging* al utilizar **distintos subconjuntos de datos**, sino que también **utiliza distinto número de características** en cada clasificador. De esta manera, se realiza un ***feature selection*** de forma **automática**.

- El método ***random forest*** se basa en el uso de ***weak learners*** (clasificadores con al menos un 51% de éxito) para construir el ***strong learner.***

Paso 1 – Crear un *bootstrapped dataset*

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa

Bootstrapped dataset

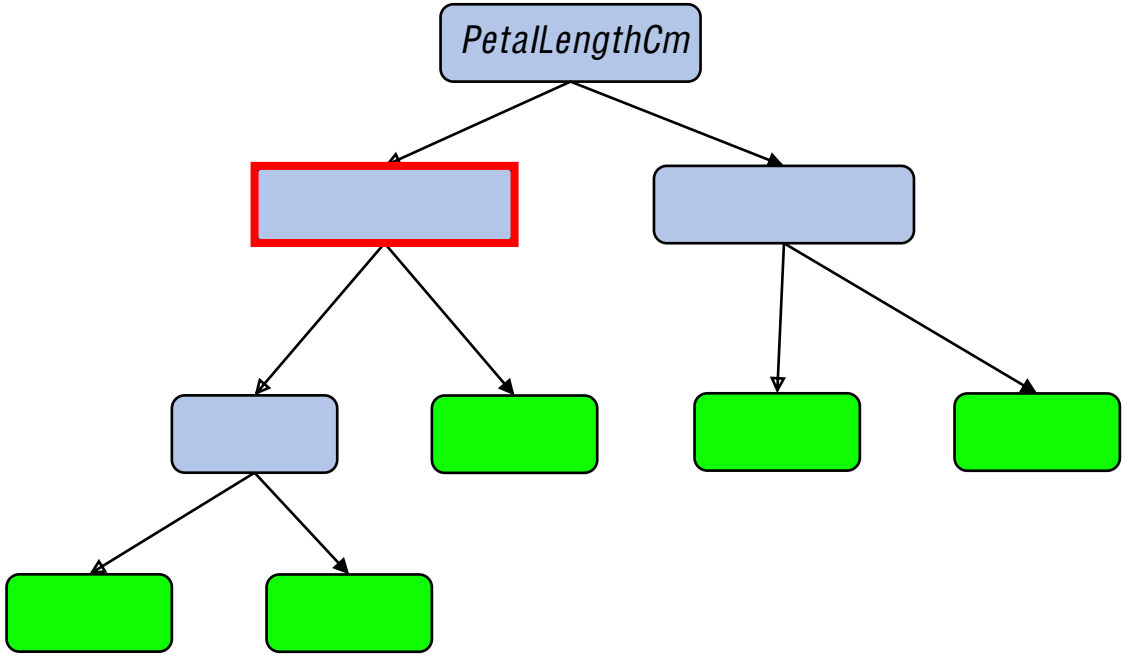
Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	4.9	3	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa

BAGGING – *Bootstrap aggregating*

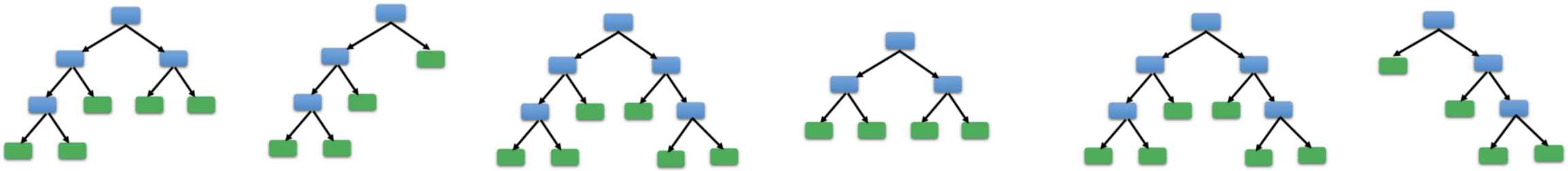
Paso 2 – Considerar solo un subconjunto de variables en cada etapa

Bootstrapped dataset

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
2	4.9	3	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa



Paso 3 – Repetir pasos 1 y 2 muchas veces



BAGGING – *Bootstrap aggregating*

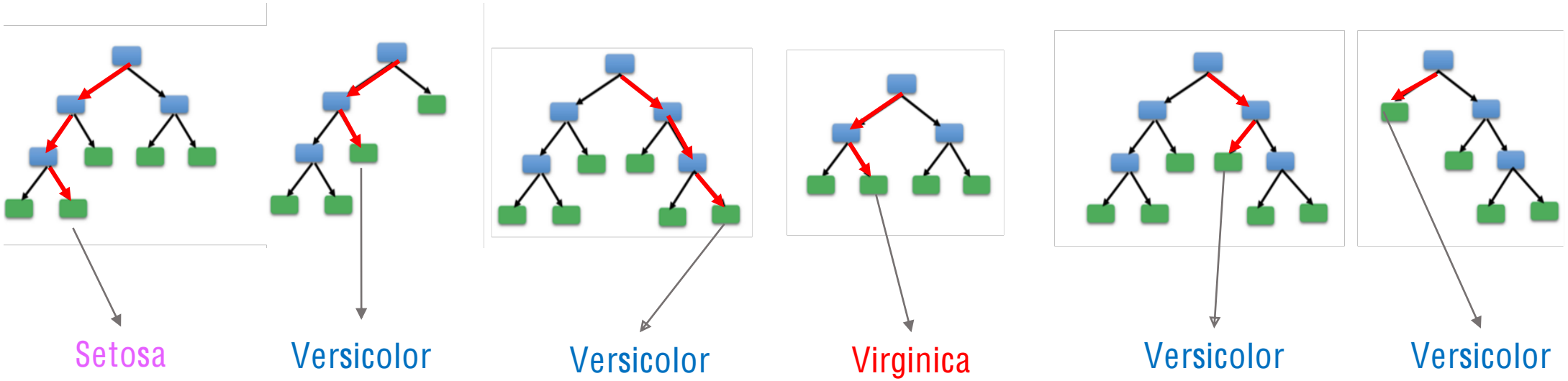
Setosa	Versicolor	Virginica
1	4	1

Predicción final ⇨ **Versicolor**

Paso 4 – Predicción de un nuevo registro

Se aplica una técnica de *majority voting* teniendo en cuenta todos los árboles construidos para determinar la predicción final.

SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5.4	3.9	1.7	0.4	?????



BAGGING – *Bootstrap aggregating*

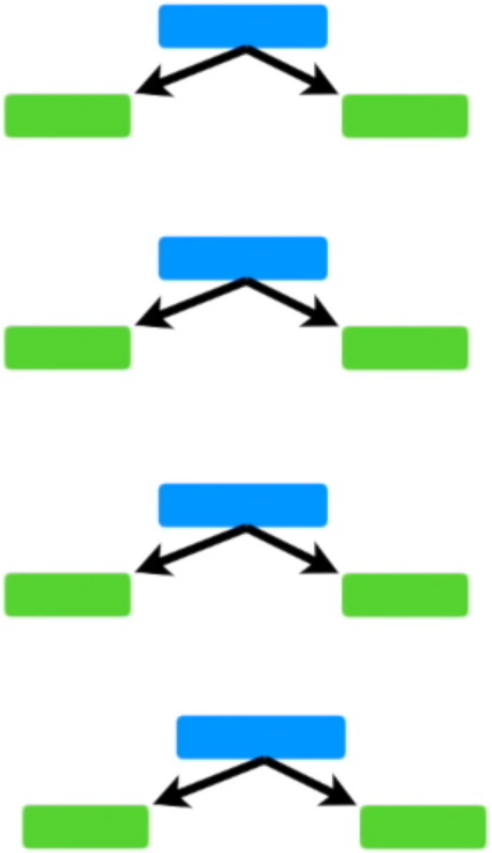
- ***VENTAJAS:***
  - Eficiente sin ajustar hiperparámetros en problemas de clasificación y regresión
  - Estabilidad y robustez en la predicción, ya que al utilizar muchos árboles prevalece el promedio de las votaciones.
  - Posibilidad de utilizar gran cantidad de características
  - Tiende a reducir el riesgo de *overfitting*.

- **INCONVENIENTES**
  - Coste computacional más elevado que construir un solo árbol de decisión.
  - Inconsistencia cuando se utilizan datasets pequeños.
  - Difícil interpretabilidad.

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

BOOSTING

- Es una **técnica de clasificación** similar al método *bagging*, pero el **entrenamiento** se lleva a cabo **en serie**, en lugar de en paralelo.
- El algoritmo **se entrena de manera iterativa** dando **más peso en cada iteración al error cometido durante la iteración anterior.**
- La **clasificación final** se obtiene **promediando** los resultados de todos los clasificadores **de manera ponderada**, a diferencia del método *bagging*.
- Esta técnica permite **reducir la varianza** y el **sesgo** (bias).
- Se puede usar **cualquier clasificador**, pero los **más utilizados** son los **árboles de decisión.**
- El **método más conocido** dentro de las **técnicas de *boosting*** es el llamado ***Adaboost*.**

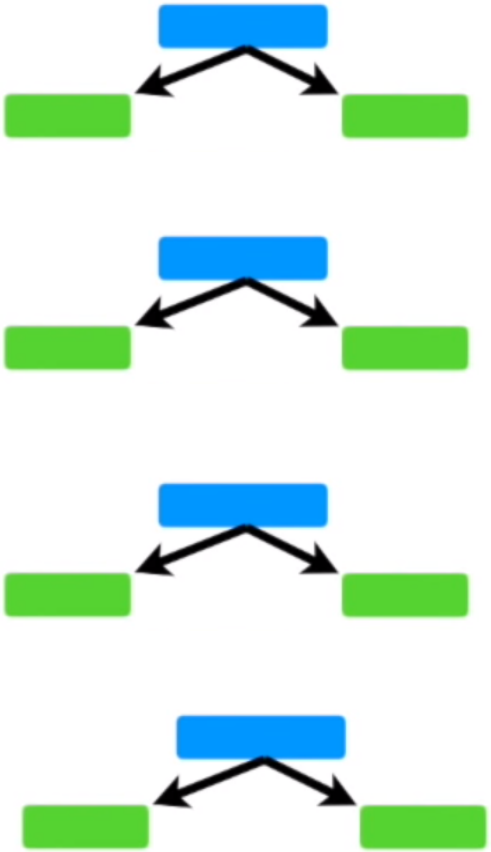


BOOSTING - ADABOOST

1. A diferencia del algoritmo *RandomForest*, que crea árboles de diferentes tamaños, en el algoritmo ***Adaboost*** solo se crean **árboles compuestos por un solo nodo** (cuadro azul) **y dos hojas** (cuadros verdes), lo que se denomina ***stump.*** De esta manera, se crea lo que se conoce como ***Forest of Stumps.*** Los ***stumps*** no son buenos clasificadores por sí solos, por eso **son *weak learners*.**

2. En el algoritmo *RandomForest* todos los árboles contribuían equitativamente en la predicción final, mientras que ***Adaboost*** aplica la técnica de ***majority voting*** de manera **ponderada,** por lo que algunos árboles contribuyen más que otros en la decisión final.

3. A diferencia del algoritmo *RandomForest*, donde cada árbol proporcionaba una predicción independiente, en ***Adaboost*** la metodología es secuencial. Es decir, el **error cometido en uno de los stumps repercute en el comportamiento del siguiente.**

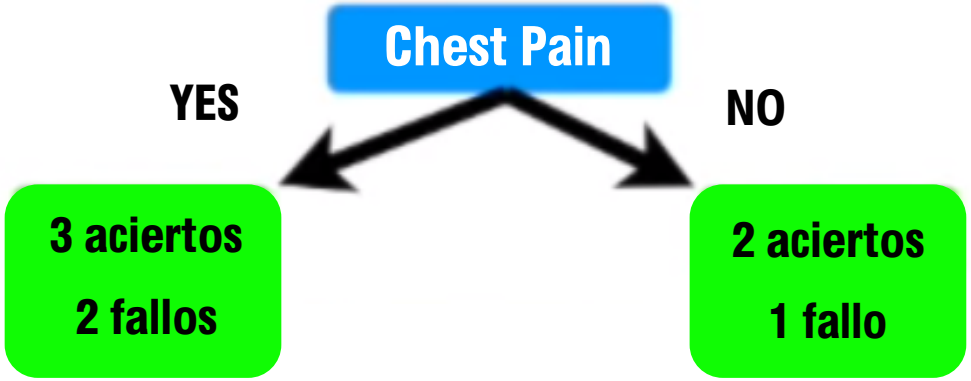


BOOSTING - ADABOOST

Intentamos **predecir si el paciente tendrá una enfermedad cardiaca o no**, es decir, un **problema de clasificación binario**. Para eso, se tienen en cuenta **3 atributos** (dolor de pecho, bloqueo de arterias y peso del paciente).

En una **primera iteración**, **todas las instancias tienen la misma ponderación**. Como hay 8 instancias en nuestro *dataset*, cada instancia tiene una importancia de 1/8.

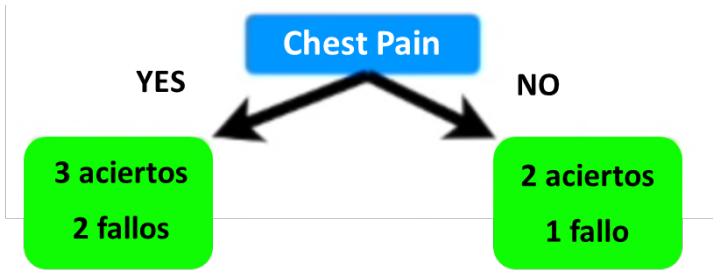
El **primer stump se constuiría analizando el valor que toma el primer atributo**, en este caso *Chest Pain* (dolor de pecho).



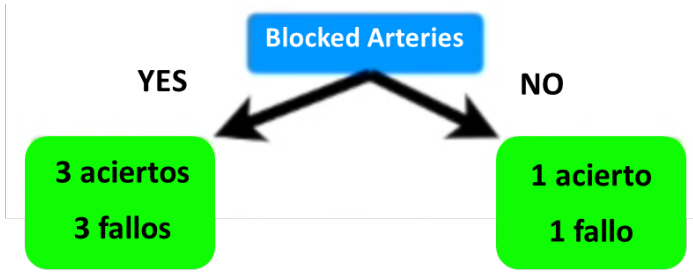
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

BOOSTING - ADABOOST

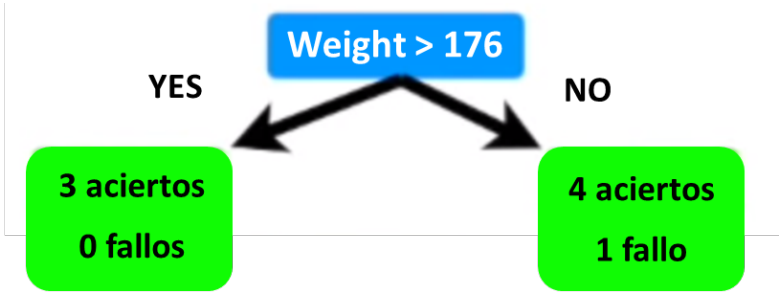
Se repite el **mismo procedimiento para todas las variables** y se calcula el **índice de Gini para cada uno** de los ***stumps***.



Gini index = 0.47



Gini index = 0.5



Gini index = 0.2

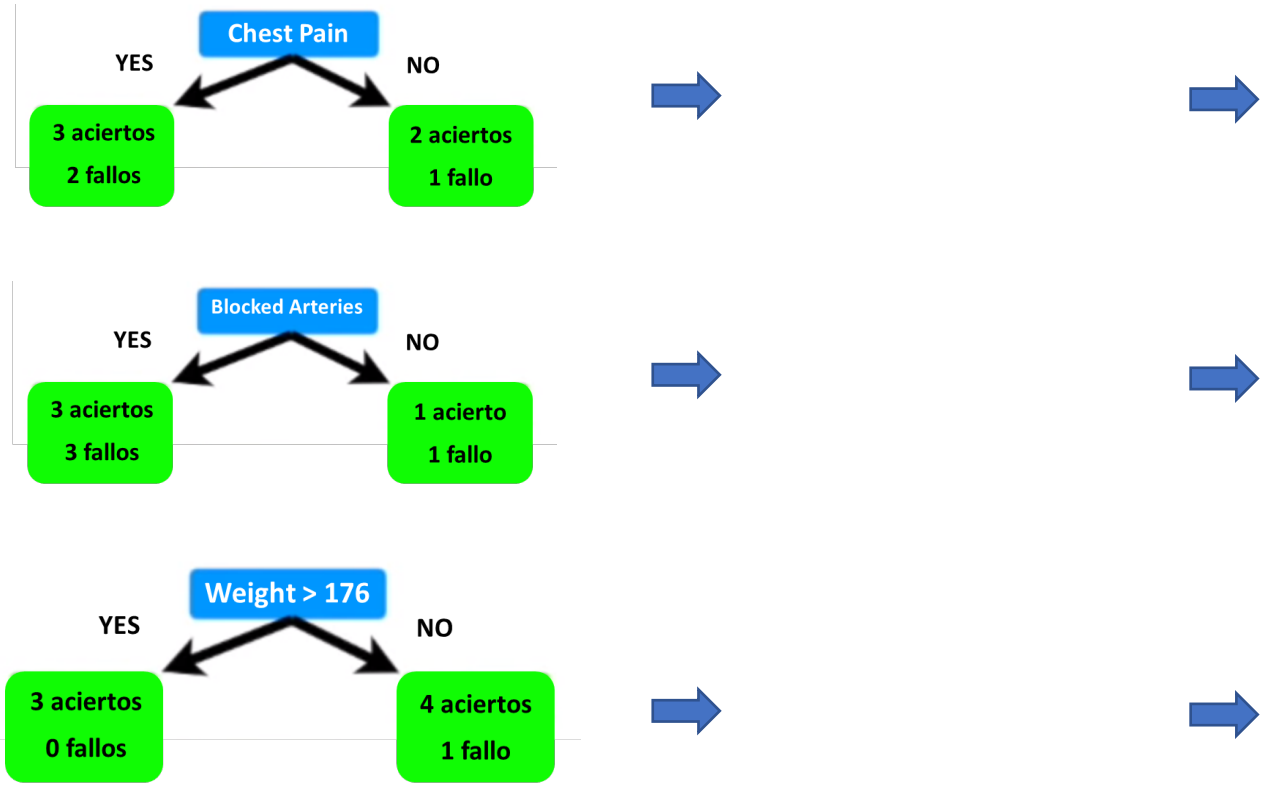
El ***stump*** que tenga un **menor índice de Gini** será el **primer *stump* del *Forest of Stumps.*** En este caso, el *stump "weight>176".*

Para **ponderar cada *stump*** y determinar **cómo de importante es cada uno en la predicción de la clase**, se utiliza la siguiente ecuación:

$$\textbf{Importancia} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

siendo el error total igual al número de fallos cometidos entre el total de los registros

BOOSTING - ADABOOST



BOOSTING - ADABOOST

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

BOOSTING - ADABOOST

Para **normalizar** los pesos, **se divide cada valor entre el sumatorio de los *"new weights"*** para dar lugar a la nueva columna ***"norm weight".*** De esta manera, los **pesos normalizados constituirán el nuevo *"sample weight"*** que se **utilizará para construir el siguiente *stump.***

Para **construir el siguiente *stump***, se define un **conjunto de datos vacío** de las **mismas dimensiones que el original**. Después, se **selecciona aleatoriamente** un **valor entre 0 y 1**, y se **añade el registro correspondiente** en función de la **suma ponderada** de la **columna de sample weight**. Por ejemplo, si el valor aleatorio es 0.13, se añadirá al nuevo *dataset* el registro 2.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

0 - 0.07
0.07 - 0.14
0.14 – 0.21
0.21 – 0.70
0.70 - 0.77
0.77 - 0.84
0.84 – 0.91
0.91 - 1

Finalmente, **se obtiene un nuevo *dataset*** donde aparece, con una **frecuencia mayor**, **los registros predichos de manera errónea** durante el paso anterior.

A partir de aquí, se repiten todos los pasos y, de esta manera, **el error cometido por un *stump* repercute en la construcción del siguiente *stump***.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

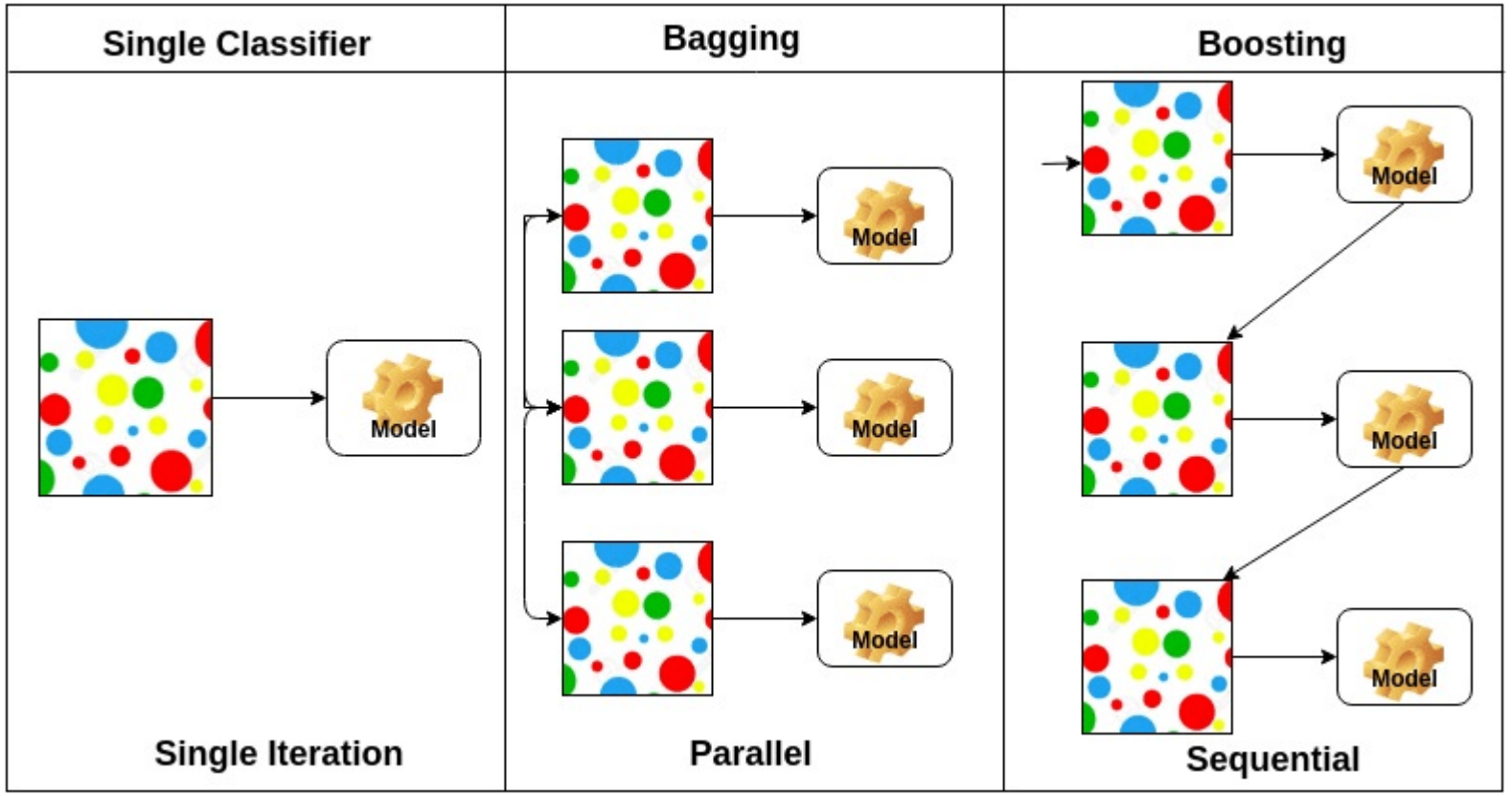
New Weight	Norm. Weight
0.05	0.07
0.05	0.07
0.05	0.07
0.33	0.49
0.05	0.07
0.05	0.07
0.05	0.07
0.05	0.07

Boosting – *Adaboost*

- ***VENTAJAS:***
  - Fácil implementación.
  - Permite corregir errores de manera iterativa usando *weak classifiers.*
  - Mejora el rendimiento combinando *weak learners.*
  - No produce *overfitting.*

- **INCONVENIENTES**
  - Es sensible a datos ruidosos.
  - Es poco robusto frente a outliers
  - Es computacionalmente menos eficiente que otros algoritmos debido a su entrenamiento iterativo.

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


Single Classifier
Bagging
Boosting
Model
Model
Model
Model
Model
Model
Model
Model
Single Iteration
Parallel
Sequential