



ARISA Learning Material

Educational Profile and EQF level: DATA SCIENTIST – EQF 6

PLO: 1, 2, 3, 4, 5

Learning Unit (LU): MACHINE LEARNING: SUPERVISED

Topic: MULTICLASS CLASSIFICATION AND REGRESSION TREES



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Copyright © 2024 by the Artificial Intelligence Skills Alliance

All learning materials (including Intellectual Property Rights) generated in the framework of the ARISA project are made freely available to the public under an open license [Creative Commons Attribution–NonCommercial](https://creativecommons.org/licenses/by-nc/4.0/) (CC BY-NC 4.0).

ARISA Learning Material 2024

This material is a draft version and is subject to change after review coordinated by the European Education and Culture Executive Agency (EACEA).

Authors: Universidad Internacional de La Rioja (UNIR)

Disclaimer: This learning material has been developed under the Erasmus+ project ARISA (Artificial Intelligence Skills Alliance) which aims to skill, upskill, and reskill individuals into high-demand software roles across the EU.



This project has been funded with support from the European Commission. The material reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- About ARISA

- The Artificial Intelligence Skills Alliance (ARISA) is a four-year transnational project funded under the EU's Erasmus+ programme. It delivers a strategic approach to sectoral cooperation on the development of Artificial Intelligence (AI) skills in Europe.
- ARISA fast-tracks the upskilling and reskilling of employees, job seekers, business leaders, and policymakers into AI-related professions to open Europe to new business opportunities.
- ARISA regroups leading ICT representative bodies, education and training providers, qualification regulatory bodies, and a broad selection of stakeholders and social partners across the industry.

[ARISA Partners & Associated Partners](#) | [LinkedIn](#) | [Twitter](#)

VALIDACIÓN CRUZADA

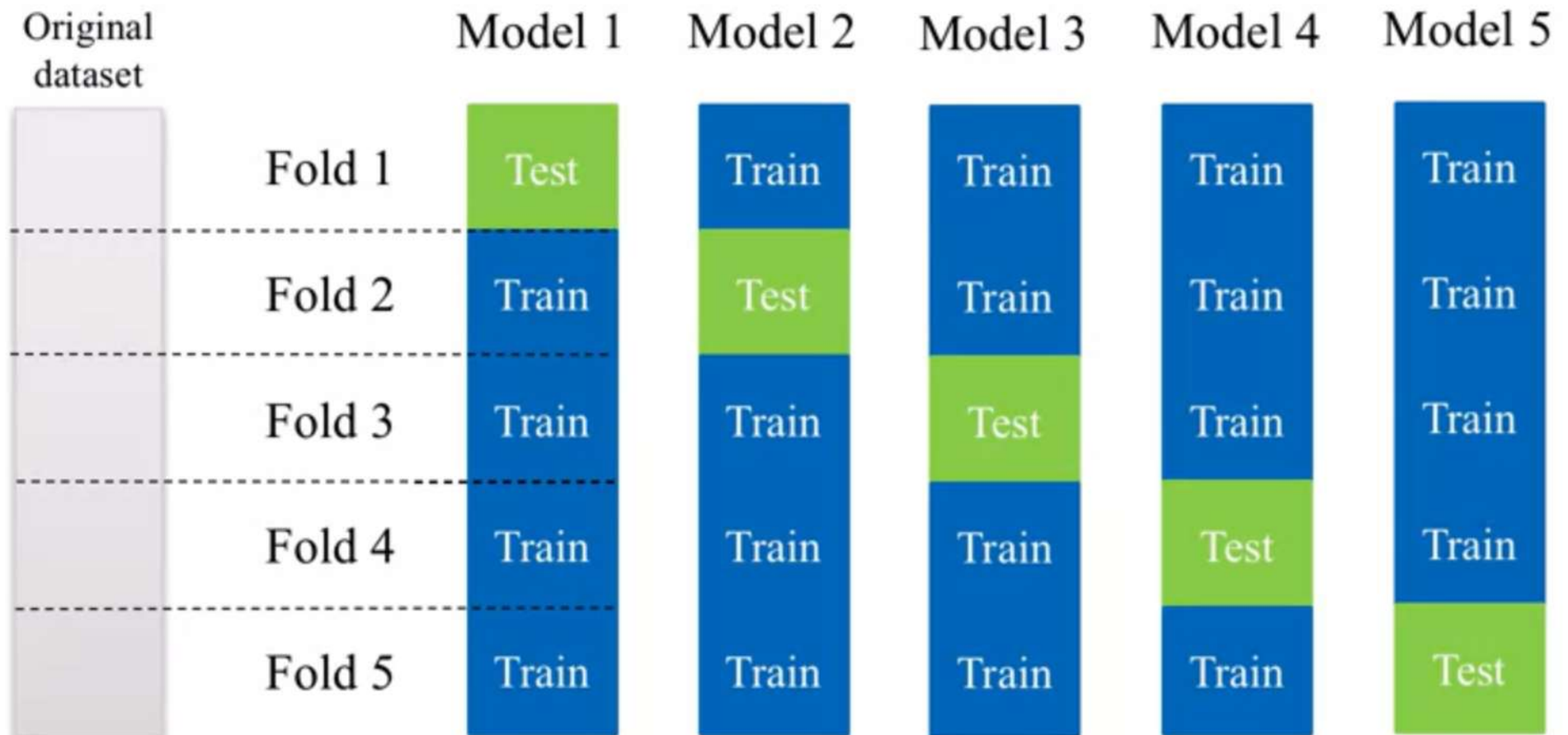
Validación cruzada

- **Utiliza varias divisiones de prueba de entrenamiento, no solo una sola**
- **Cada división se utiliza para entrenar y evaluar un modelo independiente**
- **¿Por qué es mejor?**
 - *La puntuación de precisión de un método de aprendizaje supervisado puede variar, dependiendo de las muestras que terminen en el conjunto de entrenamiento.*
 - *El uso de varias divisiones de prueba de tren proporciona estimaciones más estables y confiables sobre el rendimiento promedio del clasificador.*
 - *Los resultados se promedian en varios conjuntos de entrenamiento diferentes en lugar de basarse en un único modelo entrenado en un conjunto de entrenamiento determinado.*

random_state	Test set accuracy
0	1.00
1	0.93
5	0.93
7	0.67
10	0.87

Accuracy of k-NN classifier (k=5) on fruit data test set for different random_state values in train_test_split.

Ejemplo de validación cruzada (5 veces)



```
from sklearn.model_selection import cross_val_score

clf = KNeighborsClassifier(n_neighbors = 5)
X = X_fruits_2d.values
y = y_fruits_2d.values
cv_scores = cross_val_score(clf, X, y)

print('Cross-validation scores (3-fold):', cv_scores)
print('Mean cross-validation score (3-fold): {:.3f}'
      .format(np.mean(cv_scores)))
```

```
Cross-validation scores (3-fold): [0.75 0.75 0.83 0.83 0.82]
Mean cross-validation score (3-fold): 0.797
```

Validación cruzada estratificada

(Los pliegues y el conjunto de datos se acortan con fines ilustrativos).

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

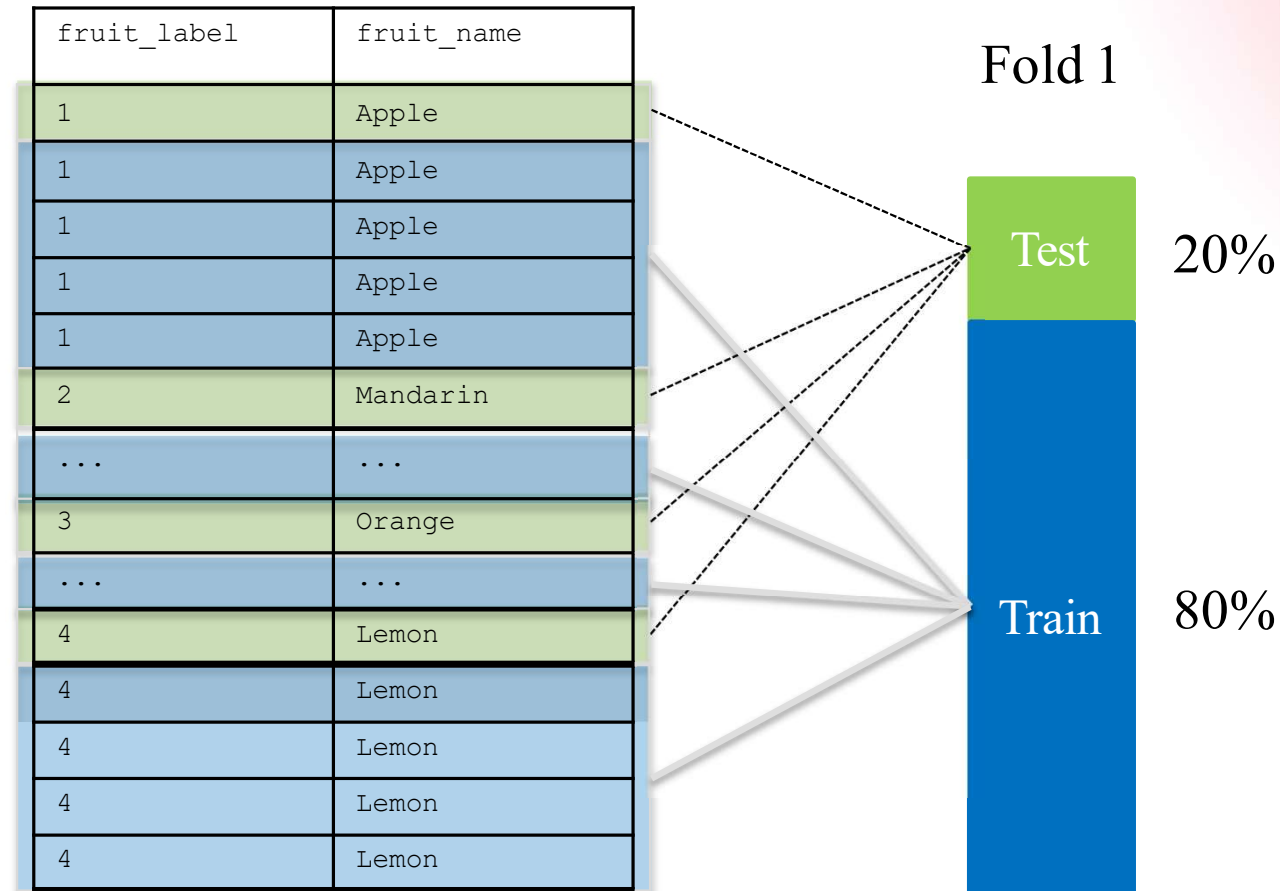
El ejemplo tiene 20 muestras de datos
= 4 clases con 5 muestras cada una.

CV de 5 pliegues: 5 pliegues de 4 muestras cada uno.

El pliegue 1 utiliza el primer 20% del conjunto de datos como conjunto de pruebas, que solo contiene muestras de la clase 1.

Las clases 2, 3 y 4 faltan por completo en el conjunto de pruebas y, por lo tanto, faltarán en la evaluación.

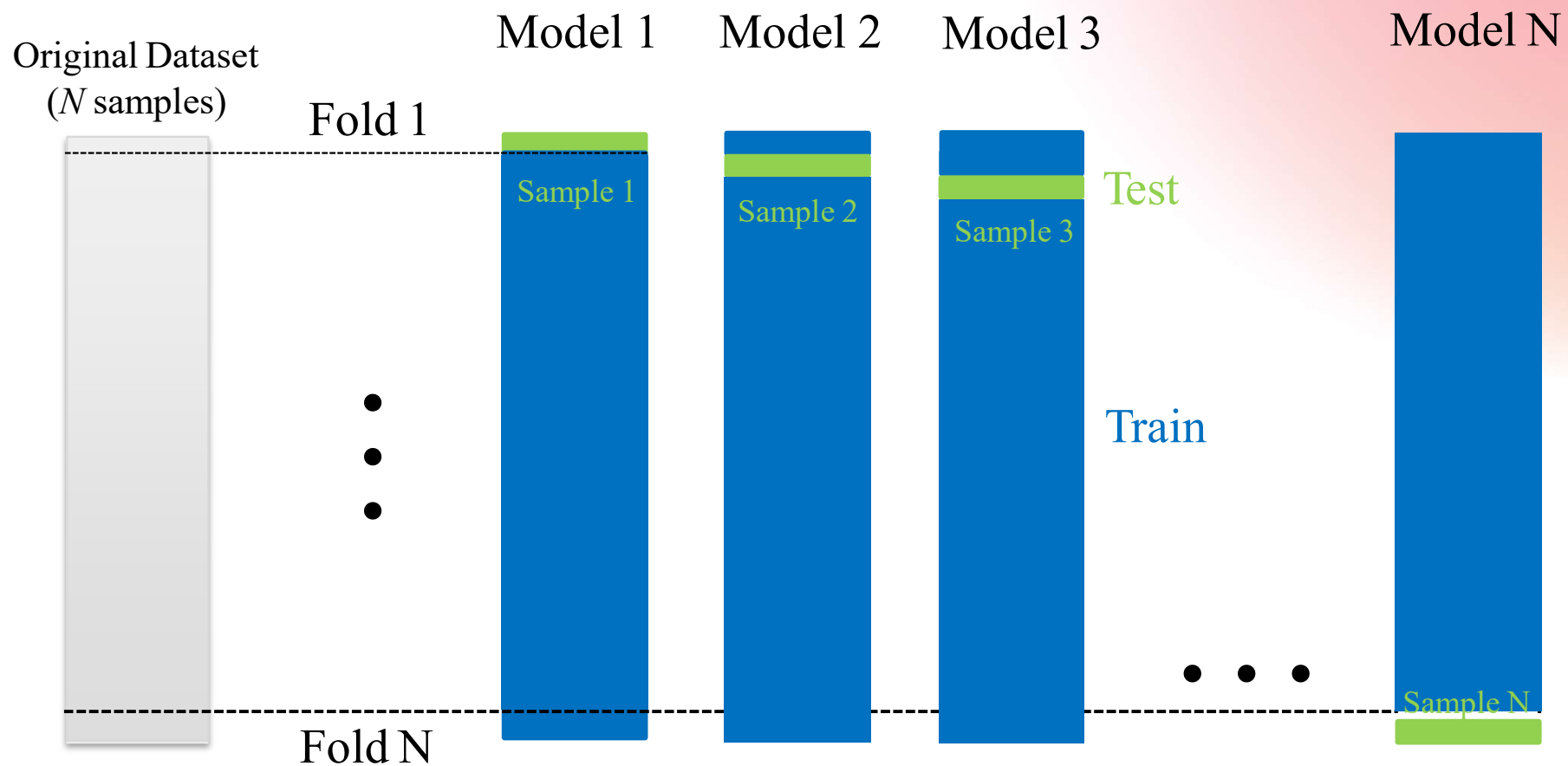
Validación cruzada estratificada



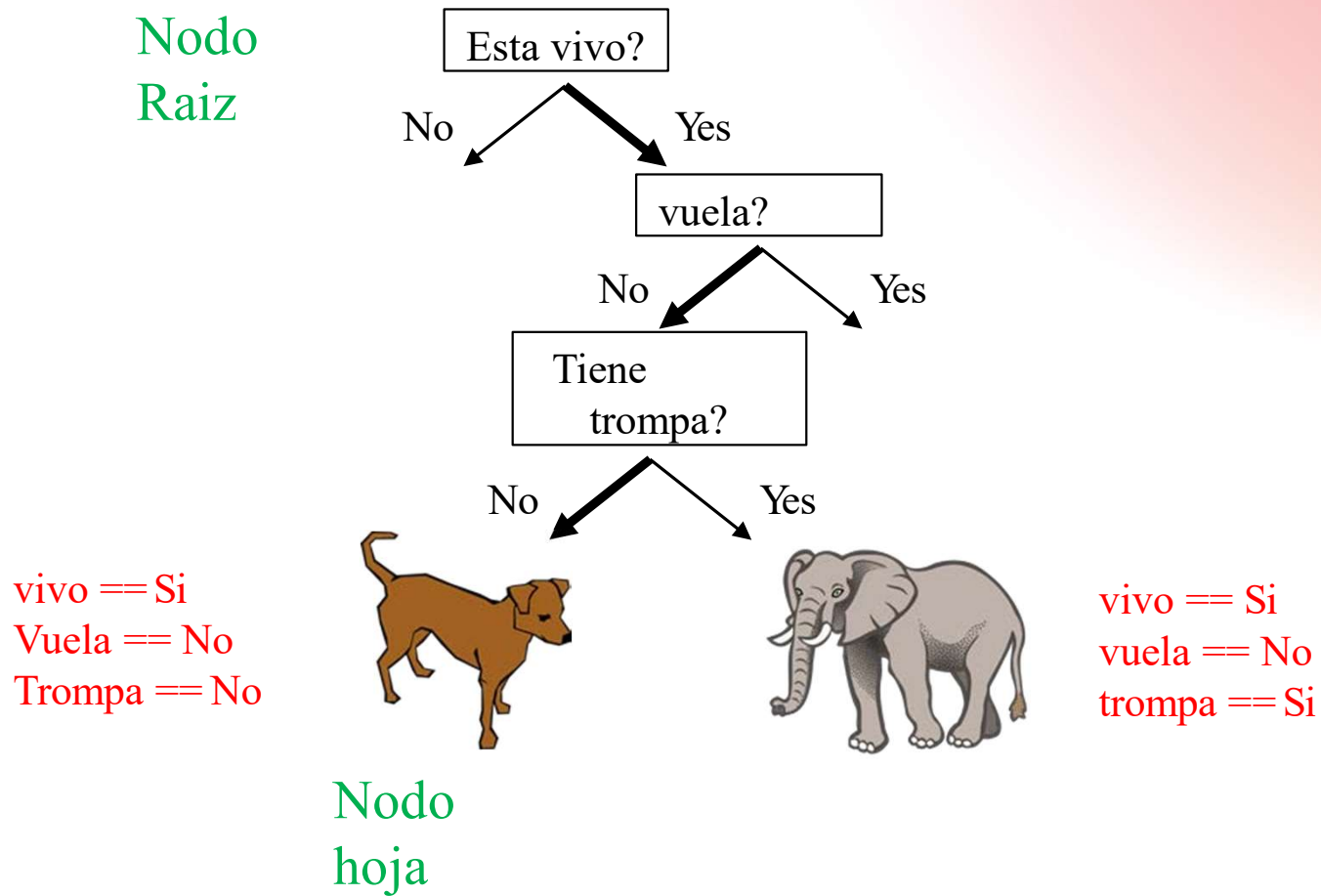
Cada uno de los pliegues estratificados contiene una proporción de clases que coincide con el conjunto de datos general.

Ahora, todas las clases estarán representadas de manera justa en el conjunto de pruebas.

Validación cruzada de una sola salida (con N muestras en el conjunto de datos)



Ejemplo de árbol de decisión



The Iris Dataset (Lirios)



Iris setosa

Iris versicolor

Iris virginica

150 flores
3 especies
50 ejemplares/especie

The Iris Dataset (Lirios)

iris setosa



petal sepal

iris versicolor



petal sepal

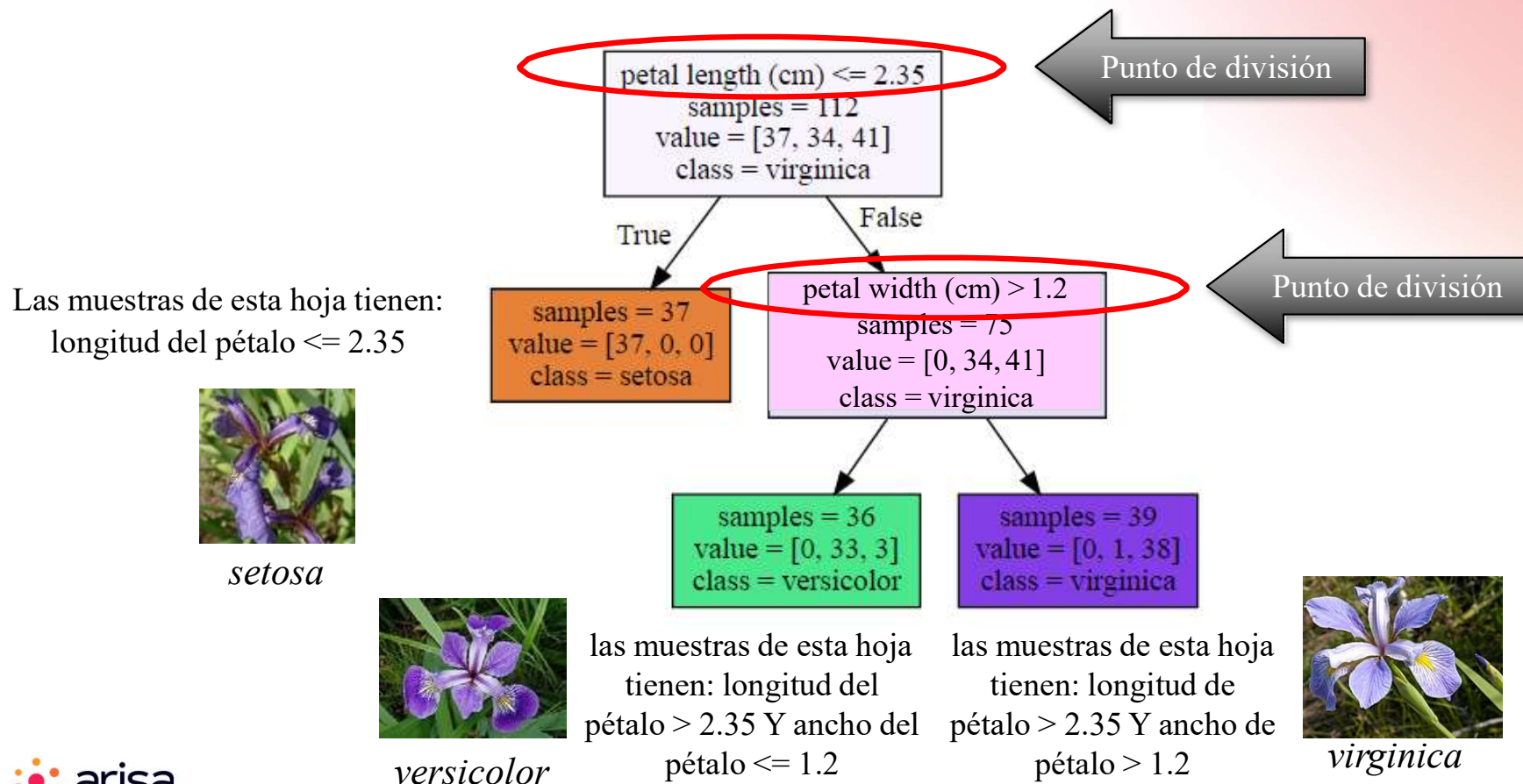
iris virginica



petal sepal

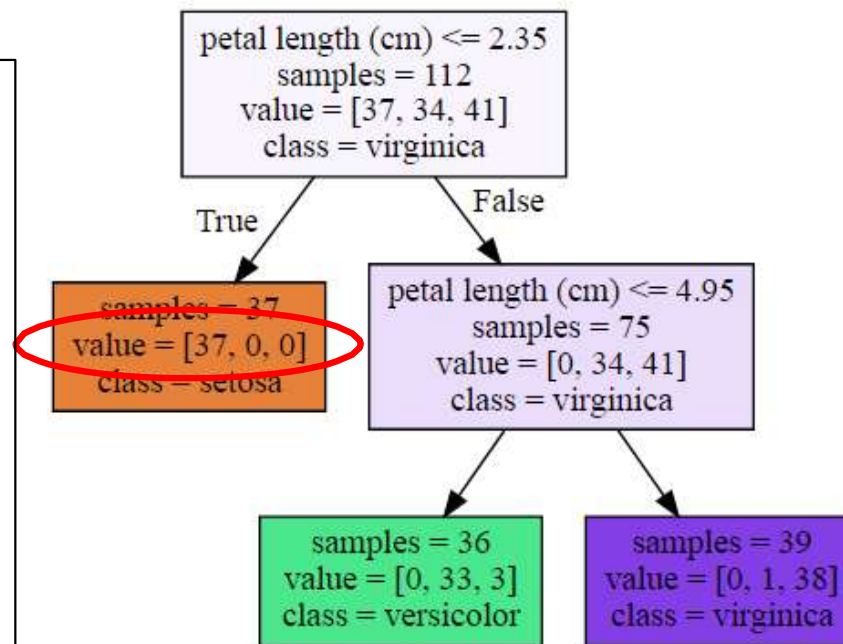
150 flores
3 especies
50 ejemplares/especie

Divisiones del árbol de decisión



Informativo de las divisiones

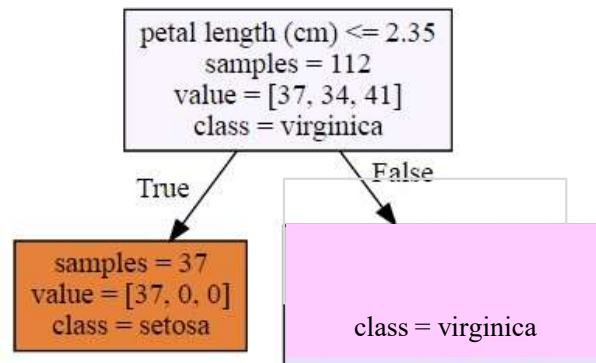
La lista de valores proporciona el número de muestras de cada clase que terminan en este nodo hoja durante el entrenamiento. El conjunto de datos de iris tiene 3 clases, por lo que hay tres recuentos. Esta hoja tiene 37 muestras de setosa, cero versicolor y cero muestras de virginica.



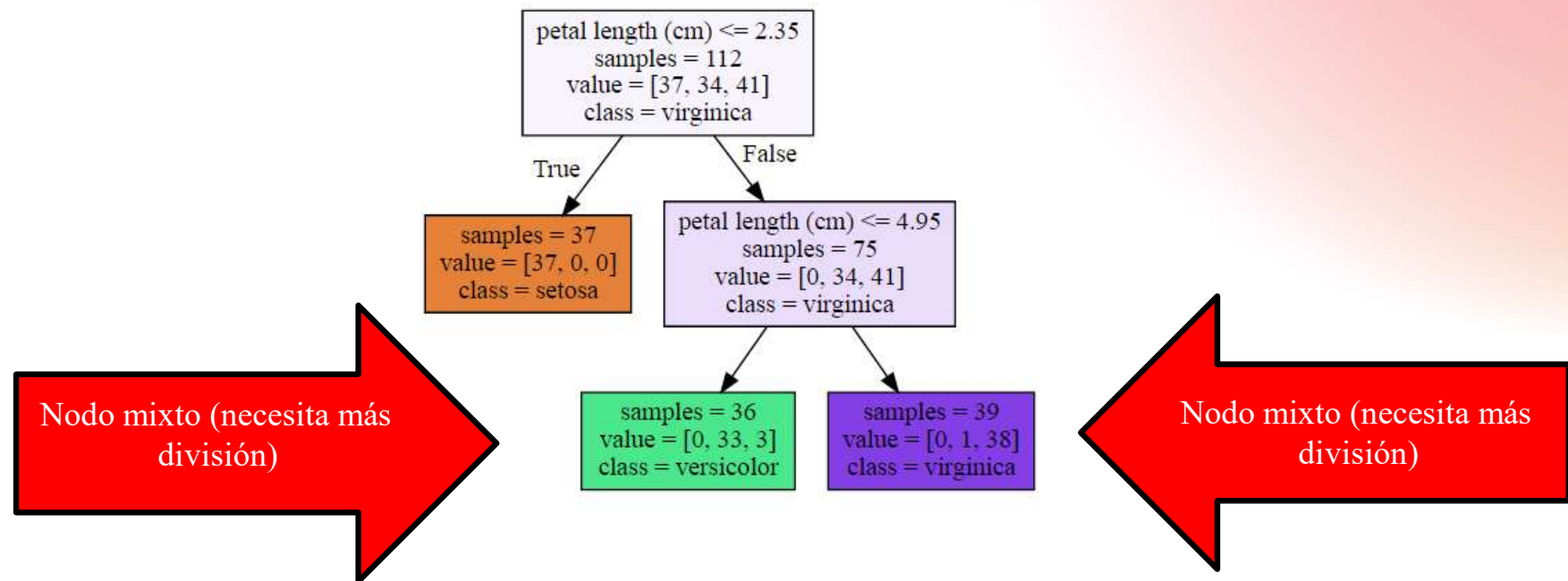
Esta hoja tiene 0 ejemplares de setosa, 33 de versicolor y 3 de virginica.

Esta hoja tiene 0 muestras de setosa, 1 versicolor y 38 virginica.

Nodo puro (todos una
clase: clasificación
perfecta)



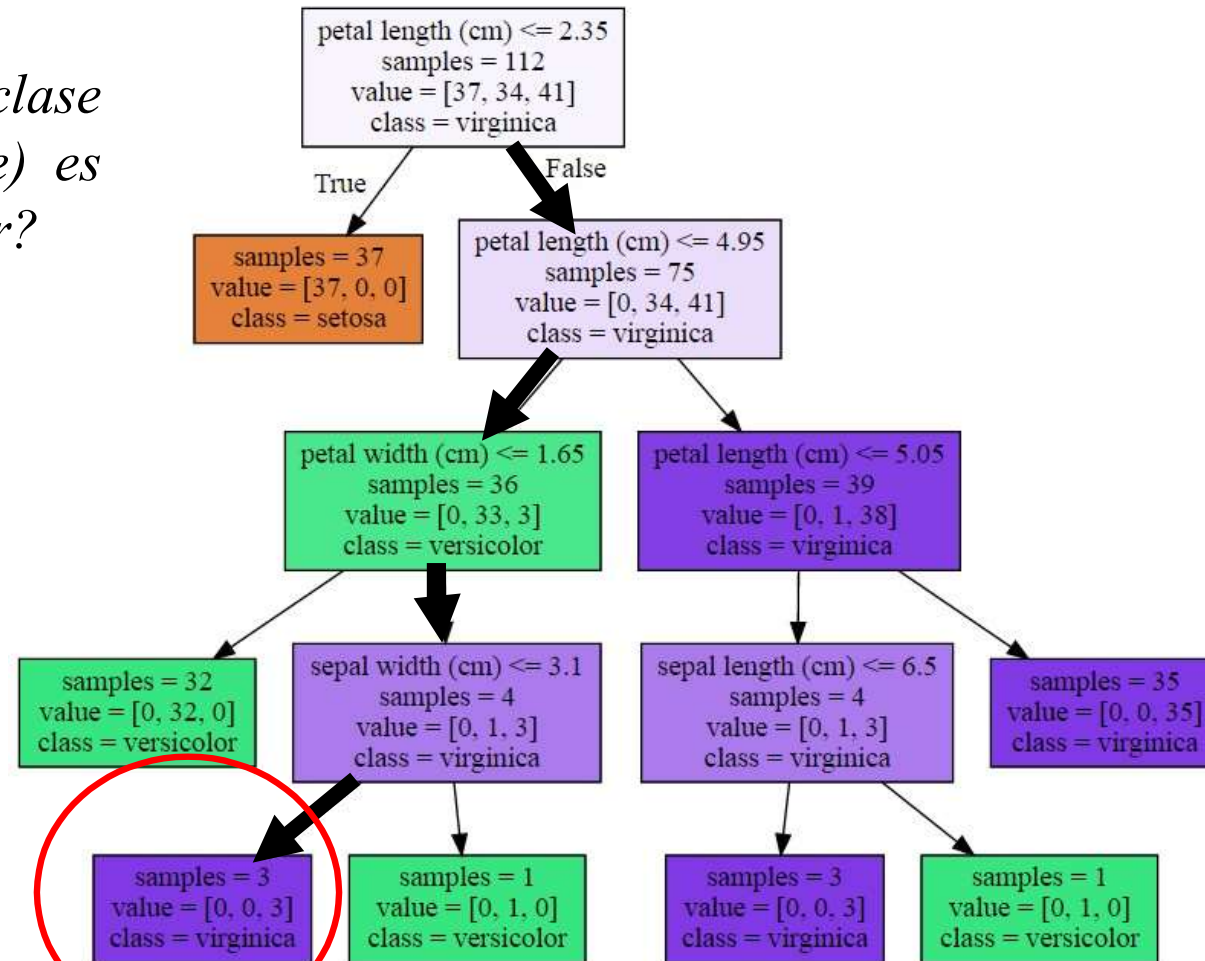
Nodo mixto (mezcla de
clases, aún necesita más
división)





¿Qué clase (especie) es esta flor?

petal length: 3.0
petal width: 2.0
sepal width: 2.0
sepal length: 4.2



Los recuentos de hojas son: setosa = 0, versicolor = 0, virginica = 3 La clase predicha es la clase mayoritaria en esta hoja: virginica

ARBOLES DE DECISION

```
In [24]: from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from adspy_shared_utilities import plot_decision_tree
from sklearn.model_selection import train_test_split

iris = load_iris()

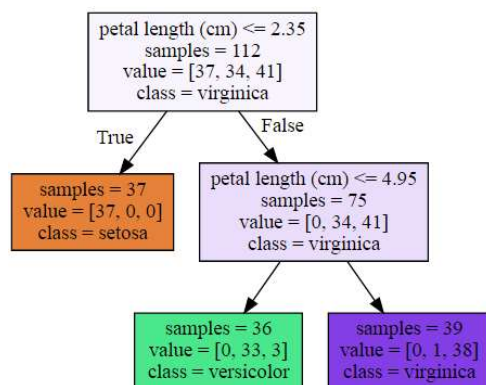
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=42)
clf = DecisionTreeClassifier().fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

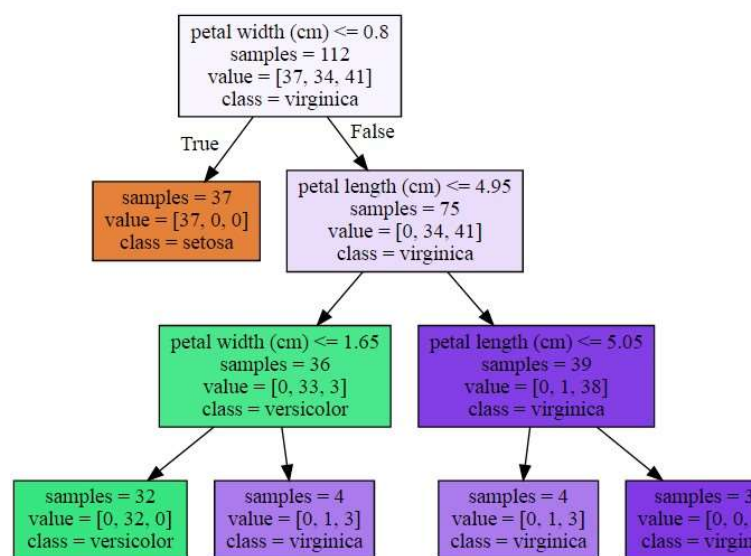
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.97

Control de la complejidad del modelo de los árboles de decisión

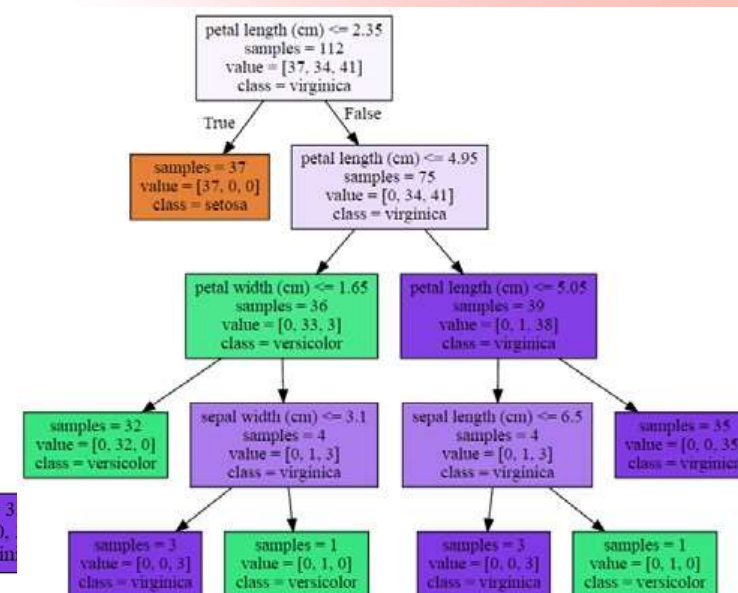
`max_depth = 2`



`max_depth = 3`



`max_depth = 4`



Otros parámetros: Max. # of leaf nodes: `max_leaf_nodes` Min. samples to consider splitting: `min_samples_leaf`

Setting max decision tree depth to help avoid overfitting

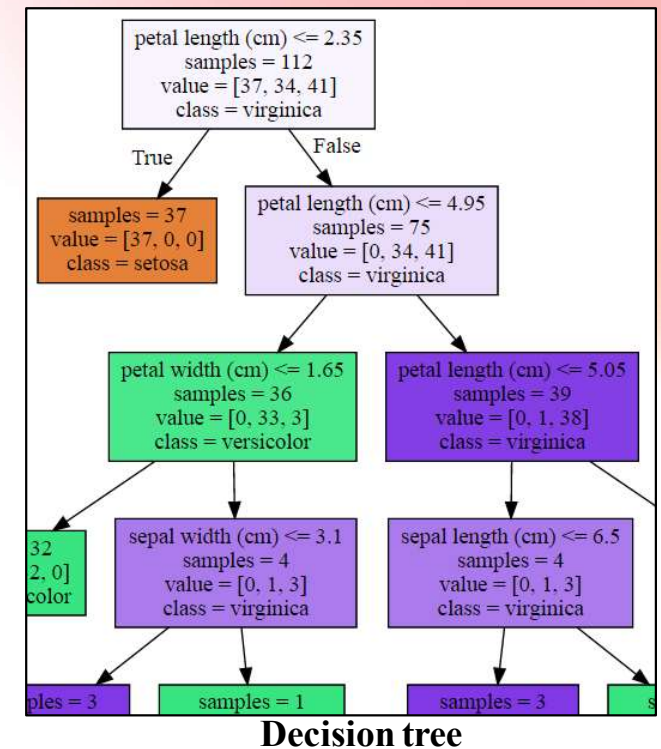
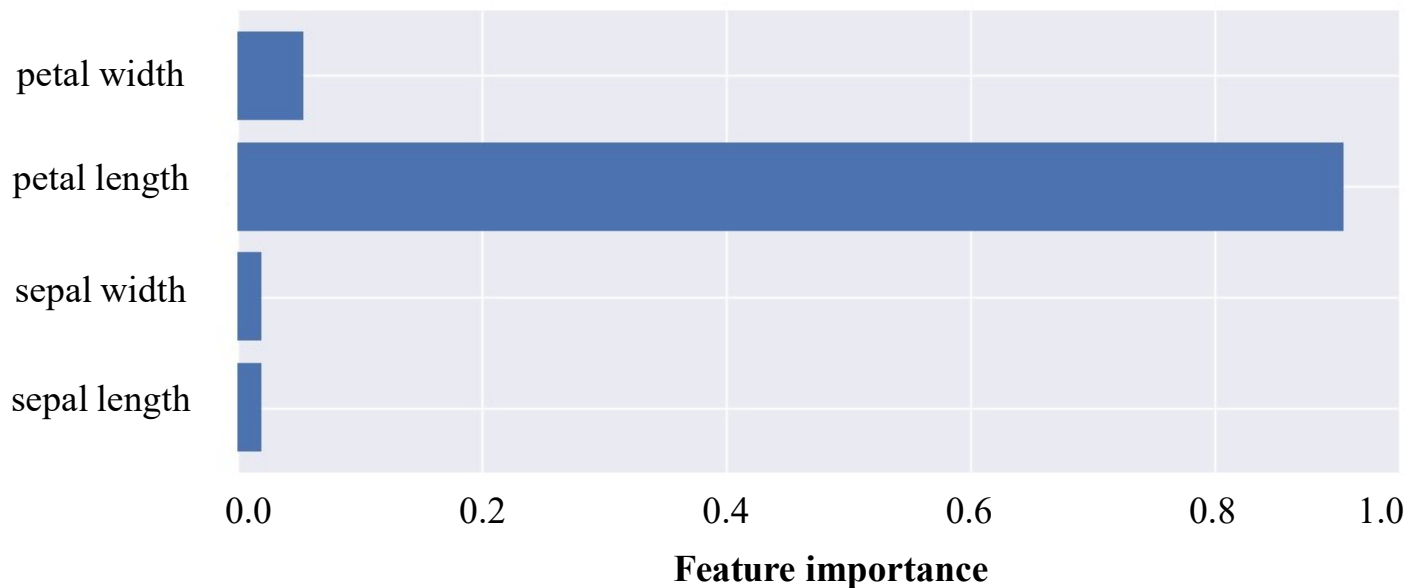
```
] clf2 = DecisionTreeClassifier(max_depth = 3).fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
      .format(clf2.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
      .format(clf2.score(X_test, y_test)))
```

Accuracy of Decision Tree classifier on training set: 0.98
Accuracy of Decision Tree classifier on test set: 0.97

Tabla de importancia de las características

Se normalizan para que la suma de 1



See: `plot_feature_importances()` function in `adspy_shared_utilities.py` code

Parámetros

- `max_depth`: controla la profundidad máxima (número de puntos de división). La forma más común de reducir la complejidad y el sobreajuste de los árboles.
- `min_samples_leaf`: umbral para el # mínimo de instancias de datos que puede tener una hoja para evitar una mayor división.
- `max_leaf_nodes`: limita el número total de hojas en el árbol.
- En la práctica, ajustar solo uno de ellos (e.g. `max_depth`) es suficiente para reducir el sobreajuste.

Decision Trees: Pros and Cons

- **Pros:**

- **visualizable e interpretable.**
- **No necesita normalización ni escalado.**
- **Trabajar bien con conjuntos de datos que utilizan una combinación de tipos de características (continuos, categóricos, binarios)**

- **Cons:**

- **Incluso después del ajuste, los árboles de decisión a menudo pueden sobreajustarse.**
- **Por lo general, necesitan un conjunto de árboles para un mejor rendimiento de generalización.**

Bosques aleatorios (Random Forest)

- **Un conjunto de árboles, no solo un árbol.**
- **Clasificación: RandomForestClassifier**
- **Regresión: RandomForestRegressor**
- **Un árbol de decisión → Propenso al sobreajuste.**
- **Muchos árboles de decisión → Más estable, mejor generalización**
- **El conjunto de árboles debe ser diverso: introducir variaciones aleatorias en la construcción de árboles.**

Conjunto de
datos original

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

bootstrap

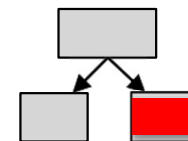
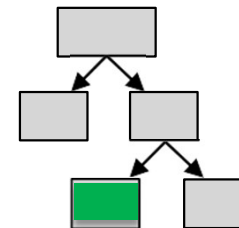
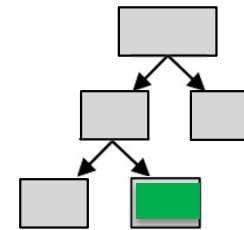
n_estimator = numero arboles

Divisiones de
características

fruit_label	fruit_name
1	Apple
1	Apple
2	Mandarin
3	Orange
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
3	Orange
3	Orange
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
3	Orange
3	Orange
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon



Predicción
de
conjunto



Bootstrap (Bagging)

¿Cómo se genera una muestra bootstrap en Random Forest?

Dado un conjunto de datos con N observaciones:

Se seleccionan N observaciones de manera aleatoria con reemplazo. Como hay reemplazo, cada selección es independiente, y una misma observación puede ser elegida más de una vez.

Aproximadamente el **63.2%** de las observaciones aparecerán al menos una vez en la muestra bootstrap, mientras que el **36.8%** quedará fuera.

Se usa la muestra bootstrap para entrenar un árbol de decisión. Esta muestra actúa como si fuera un conjunto de entrenamiento exclusivo para ese árbol.

♦ Ejemplo Práctico

Supongamos que tenemos el siguiente dataset de entrenamiento con 5 observaciones:

ID	Característica 1	Característica 2	Clase
1	3.2	5.1	A
2	4.1	6.3	B
3	5.5	3.8	A
4	2.9	4.7	B
5	6.3	5.0	A

Si tomamos una muestra bootstrap de tamaño 5, un posible resultado (recuerda que hay reemplazo) podría ser:

ID	Característica 1	Característica 2	Clase
3	5.5	3.8	A
1	3.2	5.1	A
5	6.3	5.0	A
1	3.2	5.1	A
2	4.1	6.3	B

Aquí vemos que:

- La observación 1 se repitió dos veces.
- La observación 4 no fue seleccionada (queda en el conjunto OOB).

Bootstrap Muestras

Bootstrap sample 1

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

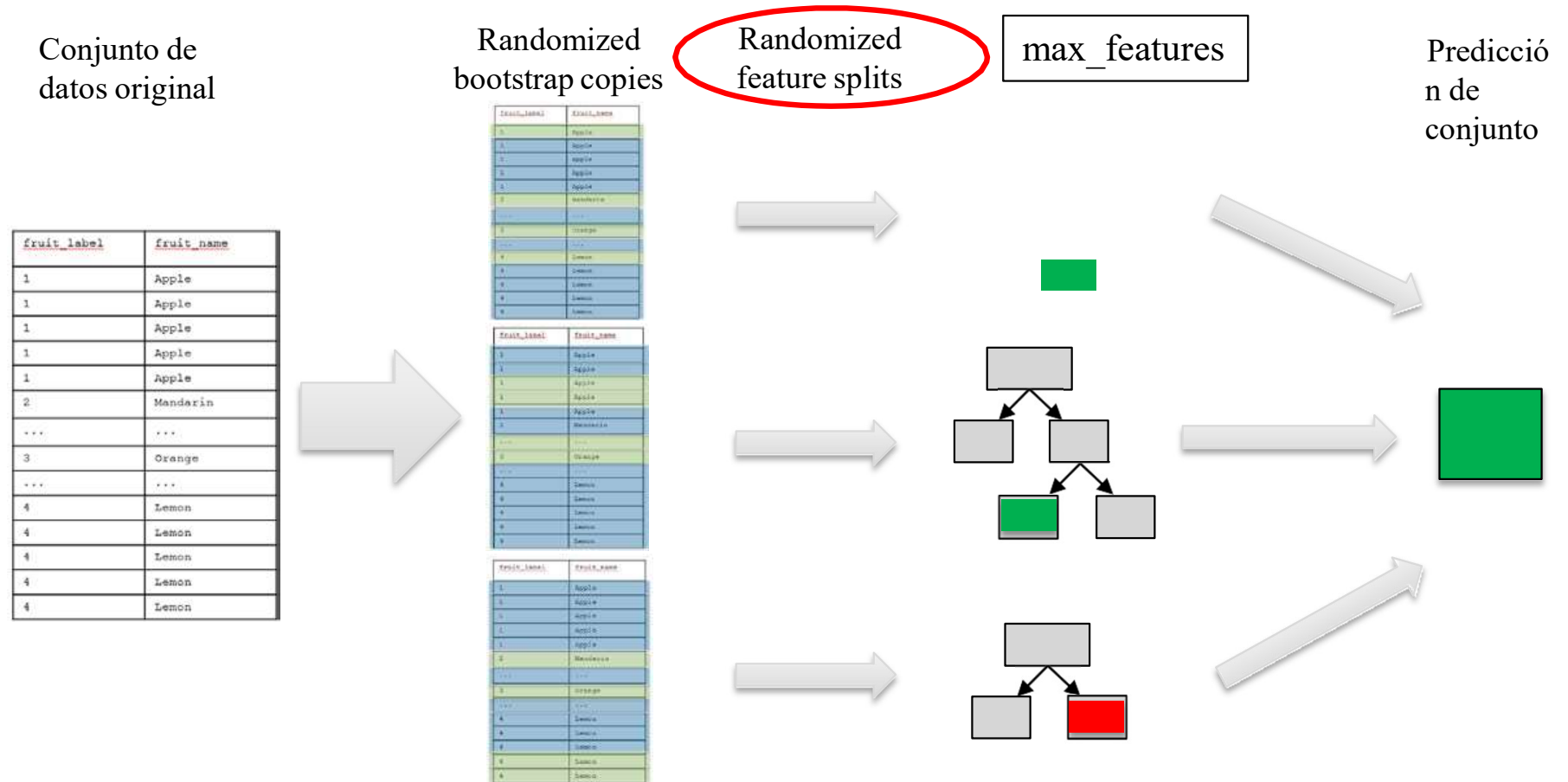
Bootstrap sample 2

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Bootstrap sample 3

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Random Forest



`max_features`

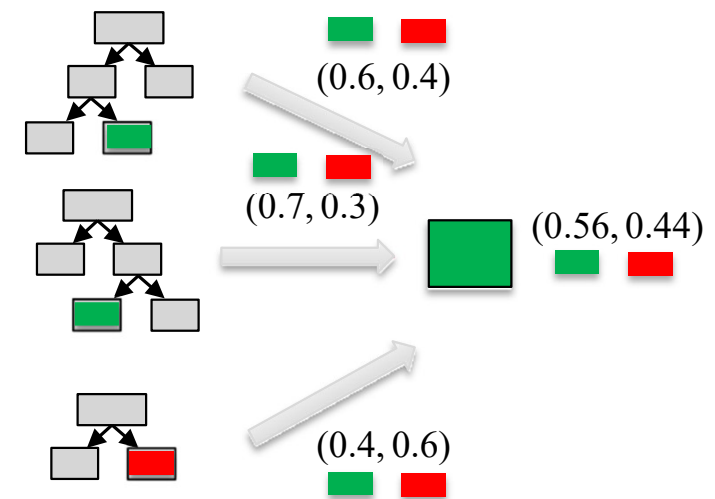
- El aprendizaje es bastante sensible a `max_features`.
- `max_features = 1` -> muy complejo
- Si se establece `max_features = <cerca del número de entidades>` se producirán bosques similares con árboles más sencillos.

Predicción mediante bosques aleatorios

1. Haz una predicción para cada árbol

2. Combinar predicciones individuales.

- *Regresión: media de las predicciones de árboles individuales.*
- *Clasificación:*
 - *Cada árbol da probabilidades para cada clase.*
 - *Promediado de probabilidades entre árboles.*
 - *Predecir la clase con mayor probabilidad.*



Random forest: Fruit dataset

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot

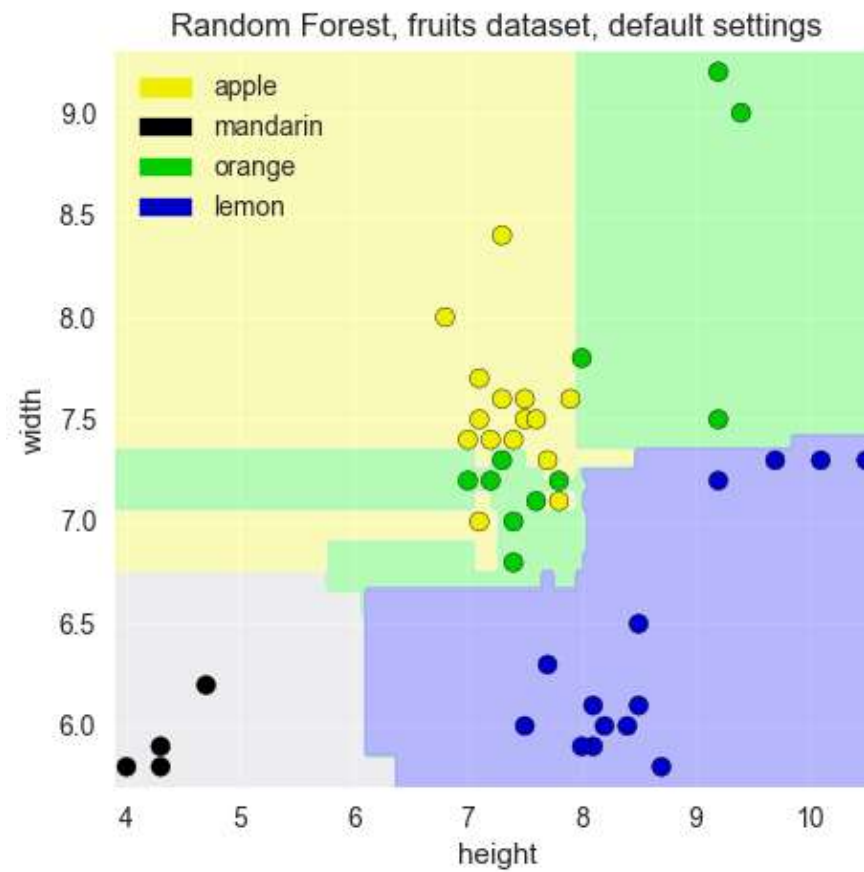
X_train, X_test, y_train, y_test = train_test_split(X_fruits.values,
                                                    y_fruits.values,
                                                    random_state = 0)

clf = RandomForestClassifier(n_estimators=10, random_state=0).fit(X_train, y_train)

print('Random Forest, Fruit dataset, default settings')
print('Accuracy of RF classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of RF classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

```
Random Forest, Fruit dataset, default settings
Accuracy of RF classifier on training set: 1.00
Accuracy of RF classifier on test set: 0.80
```


Random Forest: Fruit Dataset



Random Forests on a real-world dataset

```
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_state = 0)

clf = RandomForestClassifier(max_features = 8, random_state = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset')
print('Accuracy of RF classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of RF classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

Breast cancer dataset

Accuracy of RF classifier on training set: 1.00

Accuracy of RF classifier on test set: 0.97

Random Forest: pros y contras

Pros:

- **Ampliamente utilizado, excelente rendimiento de predicción en muchos problemas.**
- **No requiere una normalización cuidadosa de las características ni un ajuste exhaustivo de los parámetros.**
- **Puede manejar una mezcla de tipos de características.**
- **Se puede paralelizar fácilmente en varias CPU.**
- **Binario/Multiclase**

Cons:

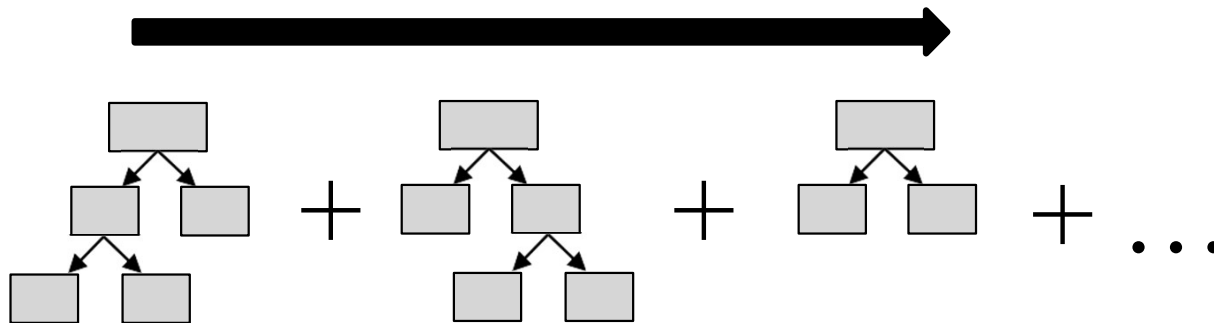
- **Los modelos resultantes son a menudo difíciles de interpretar para los humanos.**
- **Pueden no ser una buena opción para tareas de muy alta dimensión (por ejemplo, clasificadores de texto) en comparación con los modelos lineales rápidos y precisos.**

Bosques aleatorios: parámetros

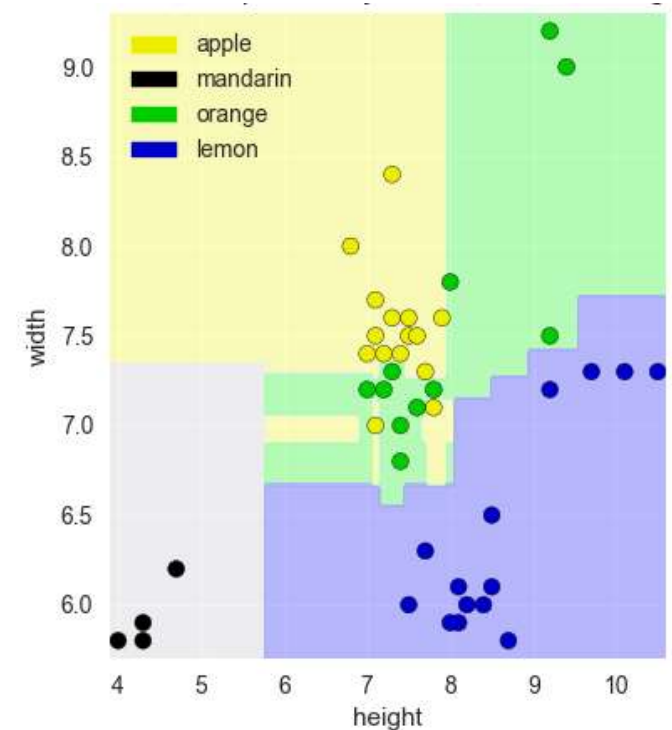
- **n_estimators: number of trees to use in ensemble (default: 10).**
 - *Should be larger for larger datasets to reduce overfitting (but uses more computation).*
- **max_features: has a strong effect on performance. Influences the diversity of trees in the forest.**
 - *Default works well in practice, but adjusting may lead to some further gains.*
- **max_depth: controls the depth of each tree (default: None. Splits until all leaves are pure).**
- **n_jobs: How many cores to use in parallel during training.**
- **Choose a fixed setting for the random_state parameter if you need reproducible results.**

Gradient Boosted Decision Trees (GBDT)

- El entrenamiento construye una serie de pequeños árboles de decisión.
- Cada árbol intenta corregir los errores de la etapa anterior.



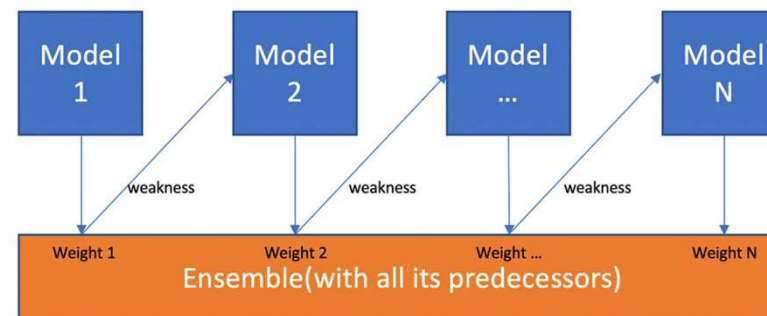
- La tasa de aprendizaje controla la intensidad con la que cada nuevo árbol intenta corregir los errores restantes de la ronda anterior.
 - Alta tasa de aprendizaje: árboles más complejos
 - Baja tasa de aprendizaje: árboles más simples



Regiones de decisión GBDT en un conjunto de datos de frutas de dos características

Boosting

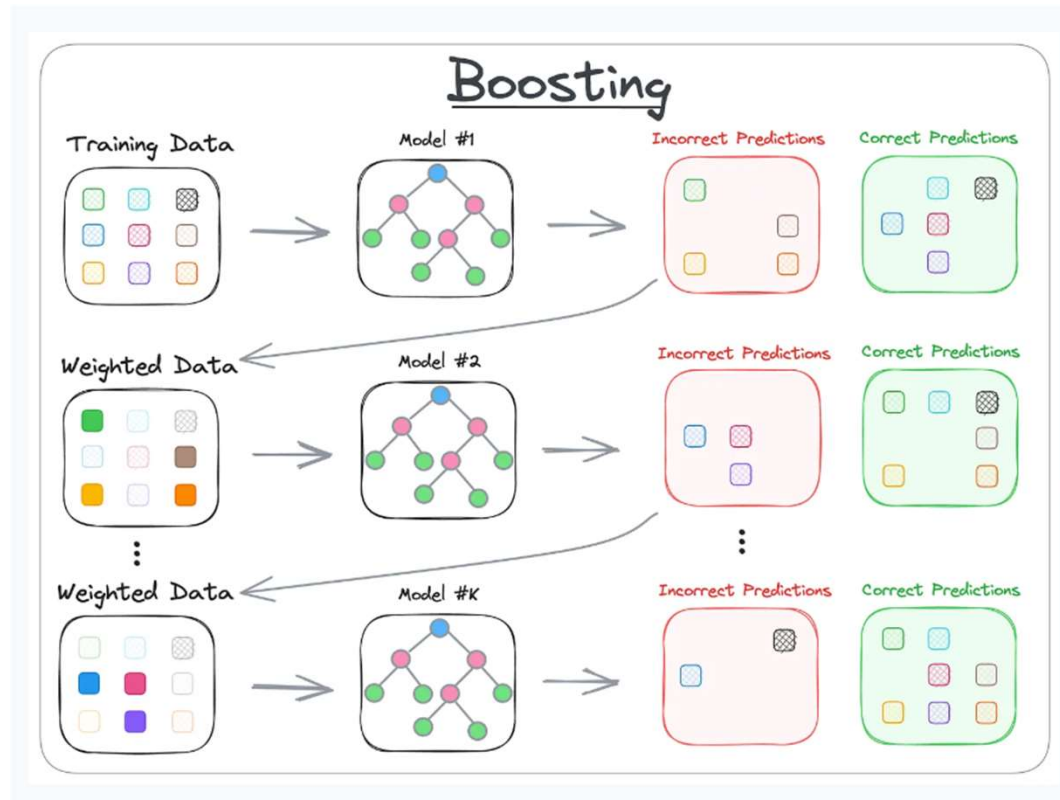
- Es una técnica de aprendizaje supervisado que combina múltiples modelos débiles (generalmente árboles de decisión simples) para formar un modelo fuerte.
- Funciona de manera secuencial: cada modelo se entrena para corregir los errores cometidos por los anteriores.
- Objetivo: reducir el sesgo y la varianza del modelo final, mejorando así su capacidad predictiva



Características clave del Boosting

- Modelos base débiles: modelos simples que por sí solos tienen un rendimiento limitado.
- Ponderación adaptativa: los ejemplos incorrectamente clasificados en cada iteración reciben mayor peso en la siguiente.
- Agregación ponderada: el resultado final se obtiene ponderando la predicción de cada modelo individual según su precisión.
- Algoritmos populares: AdaBoost, Gradient Boosting, XGBoost, LightGBM, CatBoost.

Características clave del Boosting



<https://blog.dailydoseofds.com/p/a-visual-guide-to-adaboost>

¿Qué es la búsqueda de gradiente?

- Es un algoritmo de optimización matemática usado para minimizar funciones de costo (error)
- Se utiliza para encontrar los valores óptimos de los parámetros en modelos de aprendizaje automático, especialmente redes neuronales.
- Se basa en iterar pasos pequeños en la dirección de máximo descenso del error hasta alcanzar un mínimo local o global.

¿Cómo funciona?

- Inicialización: Comienza con valores iniciales aleatorios de los parámetros.
- Cálculo del gradiente: Calcula la derivada parcial (gradiente) de la función de costo con respecto a cada parámetro.
- Actualización: Ajusta cada parámetro dando un paso proporcional al gradiente calculado:
$$\text{nuevo_parametro} = \text{parametro_actual} - \alpha \cdot \text{gradiente}$$

donde α es la tasa de aprendizaje.
- Iteración: Repite hasta alcanzar un criterio de parada (convergencia).

Variantes principales

- Batch Gradient Descent: Calcula el gradiente usando todo el conjunto de entrenamiento en cada paso. Lento para conjuntos grandes.
- Stochastic Gradient Descent (SGD): Usa un solo ejemplo de entrenamiento por actualización. Es más rápido pero menos estable.
- Mini-batch Gradient Descent: Utiliza pequeños lotes (mini-batches) del conjunto de datos para calcular el gradiente, equilibrando eficiencia y estabilidad.
- <https://uclaacm.github.io/gradient-descent-visualiser/#playground>
- <https://medium.com/data-science/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

```
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.model_selection import train_test_split  
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot  
  
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state = 0)  
fig, subaxes = plt.subplots(1, 1, figsize=(6, 6))  
  
clf = GradientBoostingClassifier().fit(X_train, y_train)  
title = 'GBDT, complex binary dataset, default settings'  
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, X_test,  
                                         y_test, title, subaxes)  
  
plt.show()
```

```
] from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_state = 0)

clf = GradientBoostingClassifier(random_state = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset (learning_rate=0.1, max_depth=3)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}\n'
      .format(clf.score(X_test, y_test)))

clf = GradientBoostingClassifier(learning_rate = 0.01, max_depth = 2, random_state = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset (learning_rate=0.01, max_depth=2)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset (learning_rate=0.1, max_depth=3)
Accuracy of GBDT classifier on training set: 1.00
Accuracy of GBDT classifier on test set: 0.97
```

```
Breast cancer dataset (learning_rate=0.01, max_depth=2)
Accuracy of GBDT classifier on training set: 0.97
Accuracy of GBDT classifier on test set: 0.97
```

GBDT: Pros y contras

Pros:

- **A menudo, la mejor solución para usar en muchos problemas.**
- **El uso del modelo para la predicción solo requiere una memoria pequeña y es rápido.**
- **No requiere una normalización.**
- **Al igual que los árboles de decisión, maneja una mezcla de tipos de características.**

Cons:

- **Al igual que los bosques aleatorios, los modelos suelen ser difíciles de interpretar**
- **Requiere un ajuste cuidadoso de la tasa de aprendizaje y otros parámetros.**
- **El entrenamiento puede requerir una computación significativa.**
- **Al igual que los árboles de decisión, no se recomienda para la clasificación de texto y otros problemas de dimensiones muy altas, por razones de precisión y costo computacional.**



Parámetros

- **n_estimators:** Establece # de pequeños árboles de decisión para usar (aprendices débiles) en el conjunto.
- **learning_rate:** Controla Énfasis en la corrección de errores de la iteración anterior.
- Los dos anteriores suelen estar afinados juntos.
- **max_depth** Por lo general, se establece en un valor pequeño (por ejemplo, 3-5) para la mayoría de las aplicaciones.

XGBoost

- XGBoost ('eXtreme Gradient Boosting') y GradientBoost de sklearn son fundamentalmente lo mismo, ya que ambas son implementaciones de aumento de gradiente.
- Sin embargo, hay diferencias muy significativas en un sentido práctico. XGBoost es mucho más rápido(<http://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>) que Sklearn. XGBoost es bastante eficiente en memoria y se puede paralelizar.



#AIskills4all |  @AIskillsEU |  [linkedin.com/AIskillsEU](https://www.linkedin.com/AIskillsEU)



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.