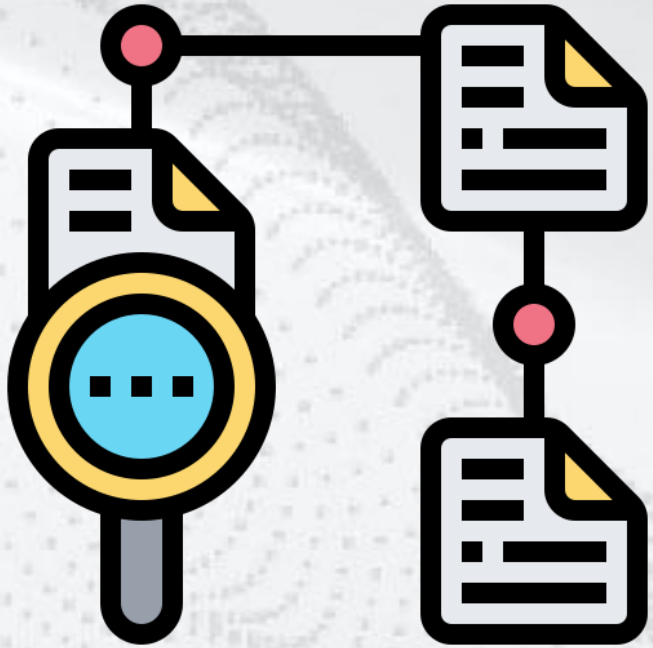
KIBERNUM
IT Academy

Módulo 5 – Aprendizaje de Máquina Supervisado

Classification Tree

Especialización en Ciencia de Datos

Classification Tree
(Árbol de clasificación)



KIBERNUM

Decision Trees & Random Forest

Los algoritmos CART (Classification and Regression Trees) son una clase de algoritmos de aprendizaje automático basados en árboles de decisión que se utilizan tanto para problemas de clasificación como de regresión.

Classification
And
Regression
Tree

(Árboles de Clasificación y Regresión)

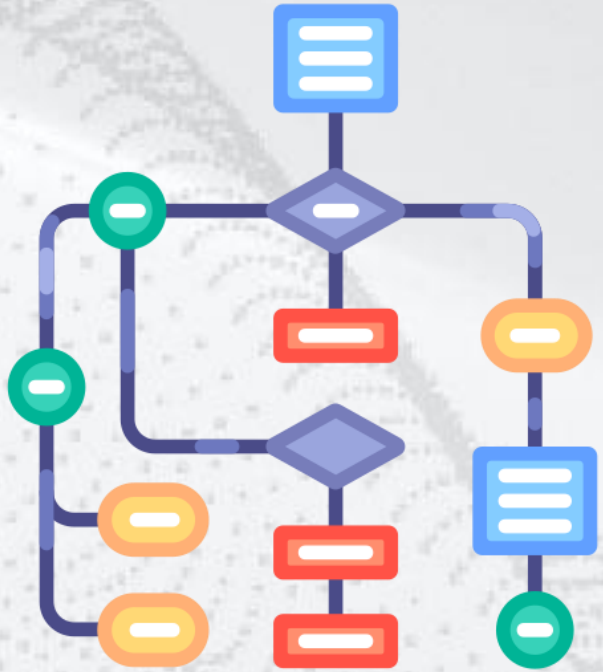
KIBERNUM

Decision Trees & Random Forest

Una de las ventajas de los algoritmos CART es que son fáciles de interpretar y visualizar, lo que los hace útiles para explicar y comunicar los resultados del modelo. Además, los árboles de decisión son relativamente rápidos de entrenar y de usar para predecir nuevos datos. Sin embargo, como con cualquier algoritmo de aprendizaje automático, hay que tener en cuenta que los resultados del modelo dependen en gran medida de la calidad y cantidad de los datos utilizados para entrenarlo.

KIBERNUM

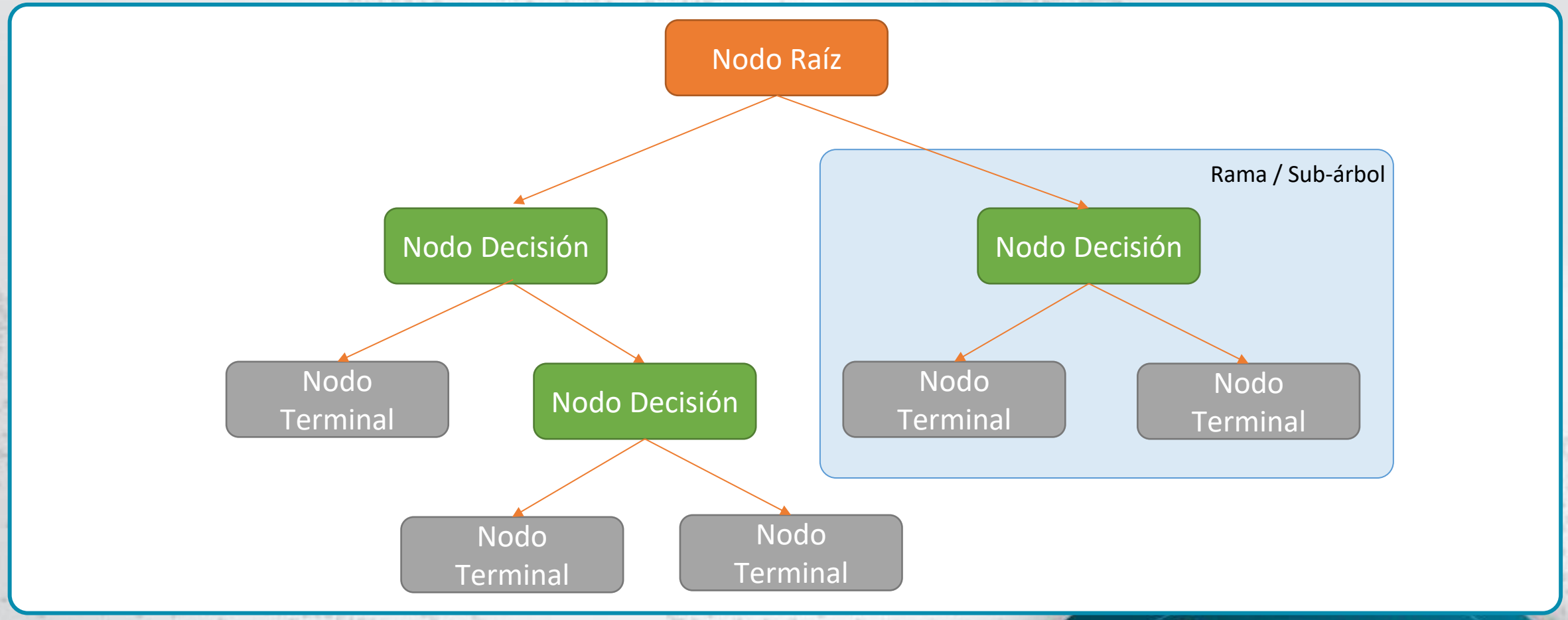
Árboles de Decisión



KIBERNUM

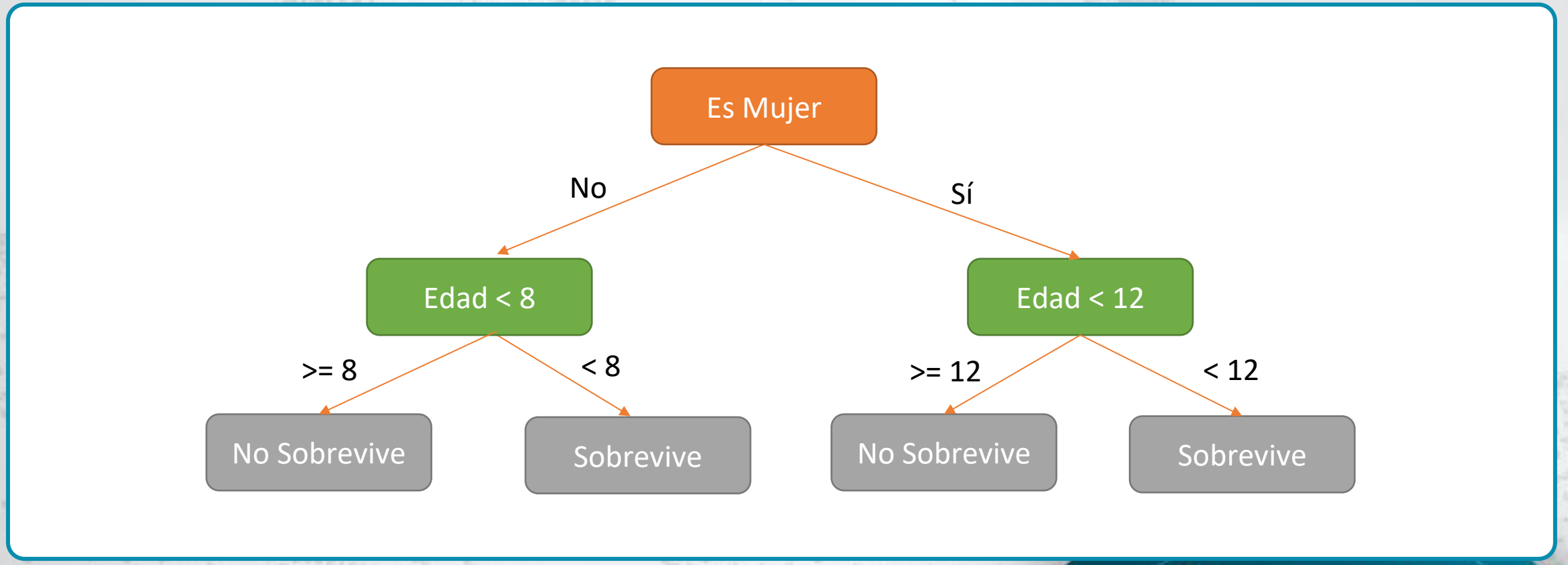
Elementos de un Árbol de Decisión

La estructura de árbol permite representar una lógica de decisión. De ahí su nombre.



Visualizando un Árbol

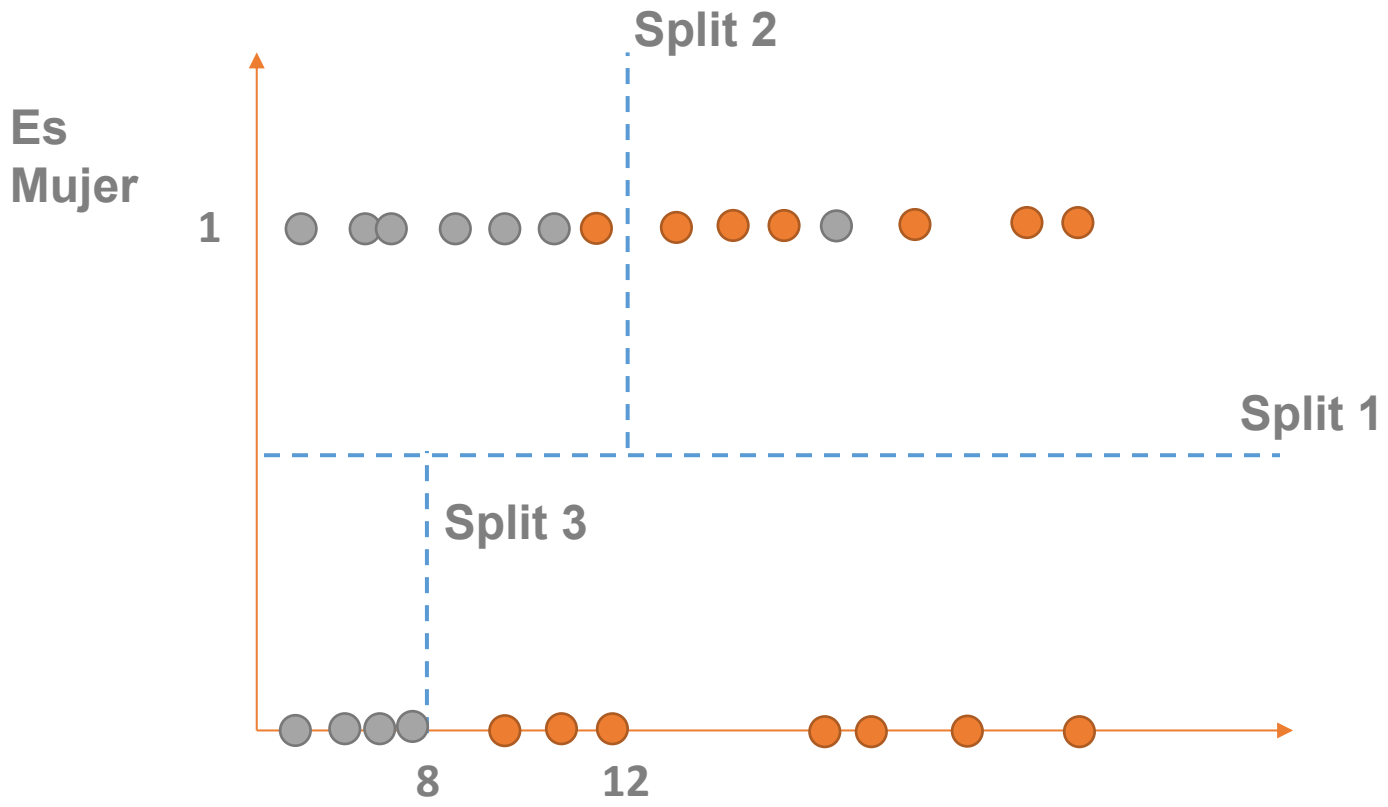
El siguiente ejemplo representa una decisión que podría representar de buena manera las personas que sobrevivieron al desastre del Titanic.



KIBERNUM

Superficie de Decisión

En el siguiente gráfico se puede visualizar la superficie de decisión que el árbol representa.



(Split: división)

KIBERNUM

Árbol de Decisión en Python

Tomemos el dataset de Titanic y formulemos un modelo con dos variables en la matriz de features: Age y Sex.

```
X = df[['Age','Sex']]
y = df['Survived']
```

```
X = pd.get_dummies(X, drop_first=True)
```

```
X.head(2)
```

	Age	Sex_male
0	22.0	1
1	38.0	0

Validación Cruzada

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=41)
```

KIBERNUM

Árbol de decisión en Python

Realizamos la importación de la clase DecisionTreeClassifier, del módulo tree, de la librería sklearn. Posteriormente, creamos un objeto DecisionTreeClassifier y ajustamos el modelo con la matriz de mediciones.

```
from sklearn.tree import DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier(max_depth = 2, random_state = 0)
```

```
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

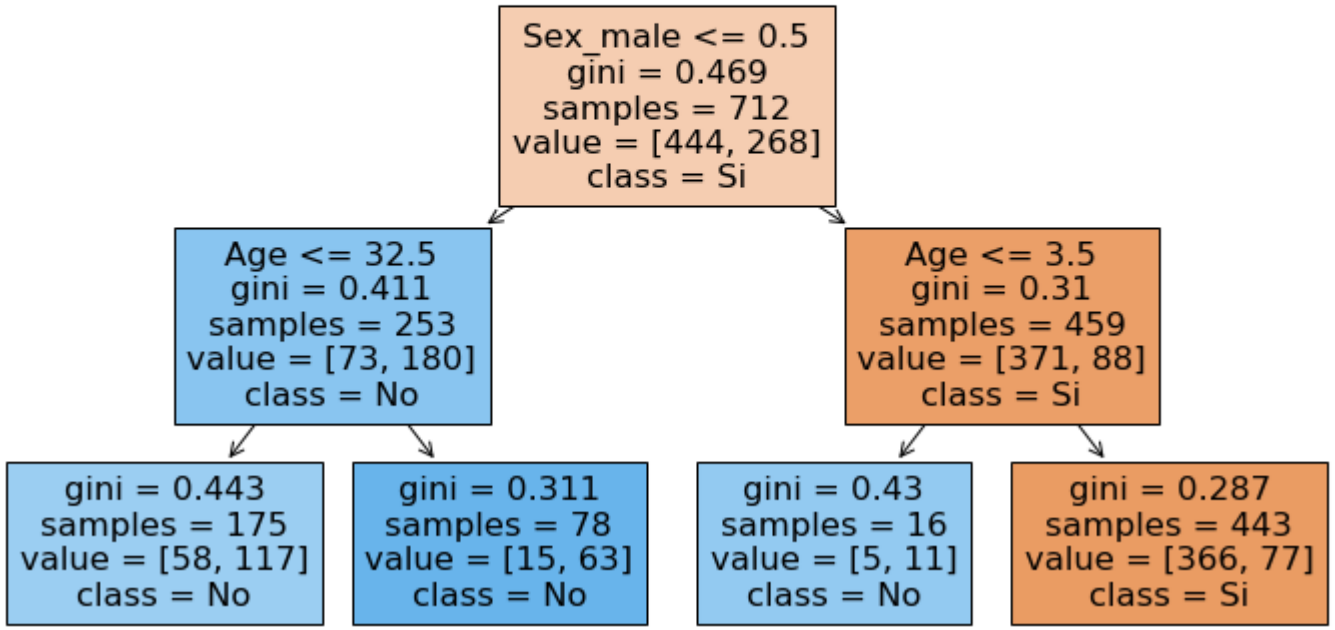
KIBERNUM

Árbol de decisión en Python

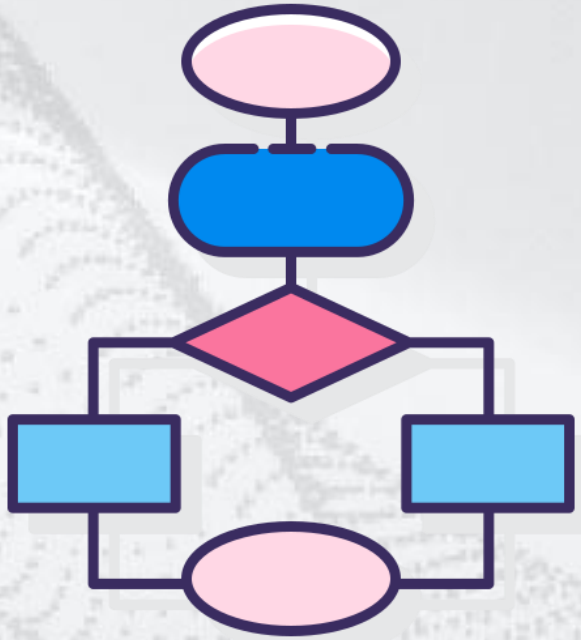
A continuación, se presenta una forma simple para visualizar el árbol de decisión que se ajustó a partir de los datos de entrenamiento.

```
from sklearn.tree import plot_tree
```

```
plt.figure(figsize=(12,6))
plot_tree(clf,feature_names = X.columns,
                class_names=['Si','No'],
                filled = True);
```



Construcción de un árbol de decisión

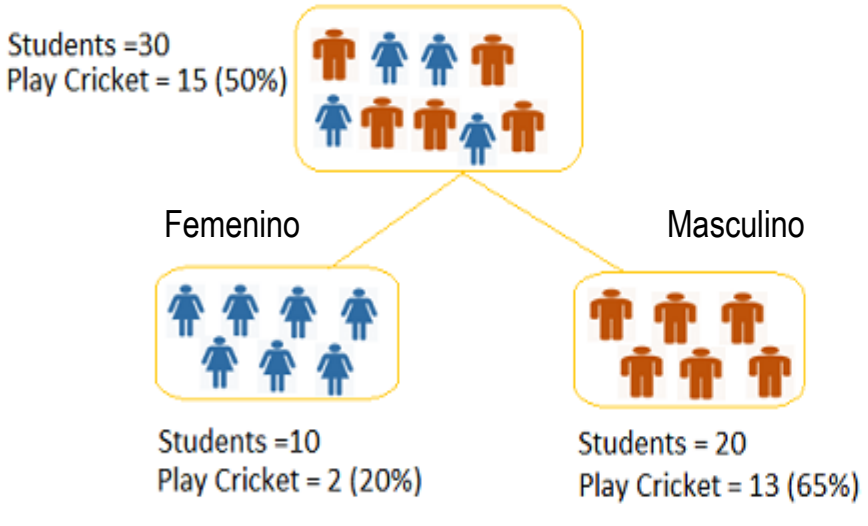


KIBERNUM

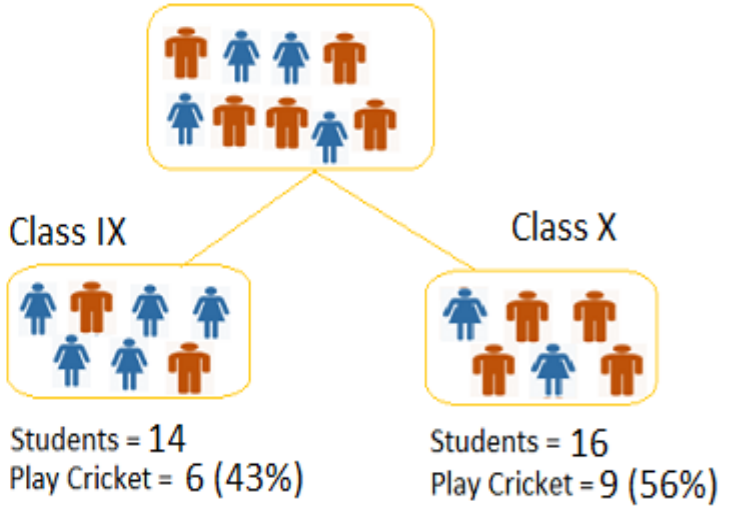
Construcción de un árbol de decisión

El algoritmo CART se basa en la idea de que la mejor forma de dividir un conjunto de datos en subconjuntos más pequeños y homogéneos es encontrar la variable y el valor que maximicen la ganancia de información o reduzcan la impureza en los subconjuntos. Para ello, se realizan iterativamente divisiones binarias recursivas en cada subconjunto de datos, creando así un árbol de decisión.

División por Género



División por Clase



KIBERNUM

Índice de Gini

En índice de Gini, o medida de impureza, mide el grado o probabilidad de una variable de ser incorrectamente clasificada cuando es aleatoriamente elegida. Si todos los elementos corresponden a una clase, entonces se le denomina “puro”.

$$Gini = 1 - \sum_{i=1}^{n} (p_i\ )^2$$

En donde p_i es la probabilidad de un objeto de ser clasificado en una clase particular y n es la cantidad de clases posibles.

El índice de Gini va de 0 a 1, **donde 0 denota que todos los elementos pertenecen a una clase (puro)** y 1 denota que los elementos están aleatoriamente distribuidos entre varias clases. Un índice de 0.5 denota que los elementos están igualmente distribuidos en las distintas clases.

KIBERNUM

Construcción de un árbol de decisión

Past Trend	Open Interest	Trading Volume	Return
Positivo	Low	High	Up
Negativo	High	Low	Down
Positivo	Low	High	Up
Positivo	High	High	Up
Negativo	Low	High	Down
Positivo	Low	Low	Down
Negativo	High	High	Down
Negativo	Low	High	Down
Positivo	Low	Low	Down
Positivo	High	High	Up

Variables independiente

Outcome

Sea el siguiente caso en donde se busca modelar el retorno de un activo financiero a partir de 3 predictores (Past Trend, Open Interes, Trading Volume).

KIBERNUM

Índice de Gini

Calculamos el índice de Gini para la variable PasTrend:

```
P(PastTrend =“Positive”) = 6/10
P(PastTrend =“Negative”) = 4/10


P(PastTrend=“Positive” & Return=“Up”) = 4/6
P(PastTrend=“Positive” & Return=“Down”) = 2/6

Gini(‘Positive’) = 1 – (P(‘Up’)**2 + P(‘Down’)**2) = 1 – ((4/6)**2 + (2/6)**2) = 0.45


P(PastTrend=“Negative” & Return=“Up”) = 0
P(PastTrend=“Negative” & Return=“Down”) = 4/4

Gini(‘Negative’) = 1 – (P(‘Up’)**2 + P(‘Down’)**2) = 1 – ((0/2)**2 + (4/4)**2) = 0
```

El índice de Gini para la variable **PastTrend** es la ponderación de ambos casos:

```
Gini(‘Past Trend’) = (6/10)*0.45 + (4/10)*0 = 0.27
```

KIBERNUM

Índice de Gini

Repetimos el procedimiento y calculamos el índice de Gini para las demás variables:

Features	Indice Gini
PastTrend	0.27
OpenInterest	0.47
TradingVolume	0.34

Como se puede observar, PastTrend tiene el menor índice de Gini, por lo cual es elegido como el nodo raíz.
Se repite el procedimiento con cada nodo y rama para formar el árbol.

KIBERNUM

Ventajas

1. **Fácil interpretación:** Esto se debe a que el árbol se puede visualizar como un diagrama de flujo con decisiones binarias en cada nodo.

1. **Manejo de datos faltantes:** lo que los hace útiles para conjuntos de datos incompletos o de mala calidad.

1. **No paramétricos:** no se hacen suposiciones sobre la distribución de los datos subyacentes. Esto los hace adecuados para conjuntos de datos complejos o no lineales.

1. **Flexibilidad:** se pueden utilizar para resolver problemas de clasificación o de regresión.

1. **Velocidad de entrenamiento y predicción:** son rápidos en el entrenamiento y la predicción, especialmente en comparación con algunos otros algoritmos de aprendizaje automático.

Desventajas

1. **Propensos al sobreajuste:** especialmente cuando se utilizan con conjuntos de datos pequeños o complejos.

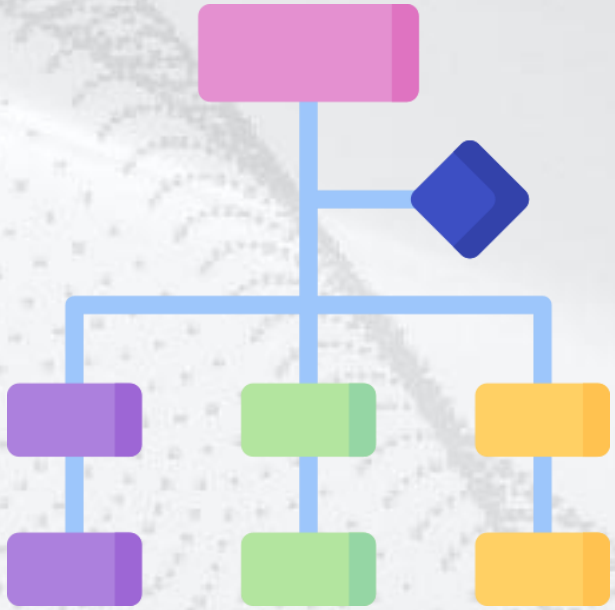
1. **Sensibles a cambios en los datos:** pequeños cambios en los datos de entrada, podría provocar cambios significativos en la estructura del árbol y, por lo tanto, en las predicciones.

1. **Limitados en la capacidad de generalización:** pueden tener un rendimiento pobre en conjuntos de datos nuevos o diferentes.

1. **Sesgo hacia atributos con alta cardinalidad:** lo que significa que pueden enfocarse en atributos con muchos valores diferentes y pasar por alto otros atributos importantes.

1. **No adecuados para ciertas estructuras de datos:** no son adecuados para todos los tipos de estructuras de datos, como datos de series temporales o imágenes, donde se requieren otros tipos de algoritmos de aprendizaje automático.

Árboles Aleatorios
(Random Forest)



KIBERNUM

Random Forest

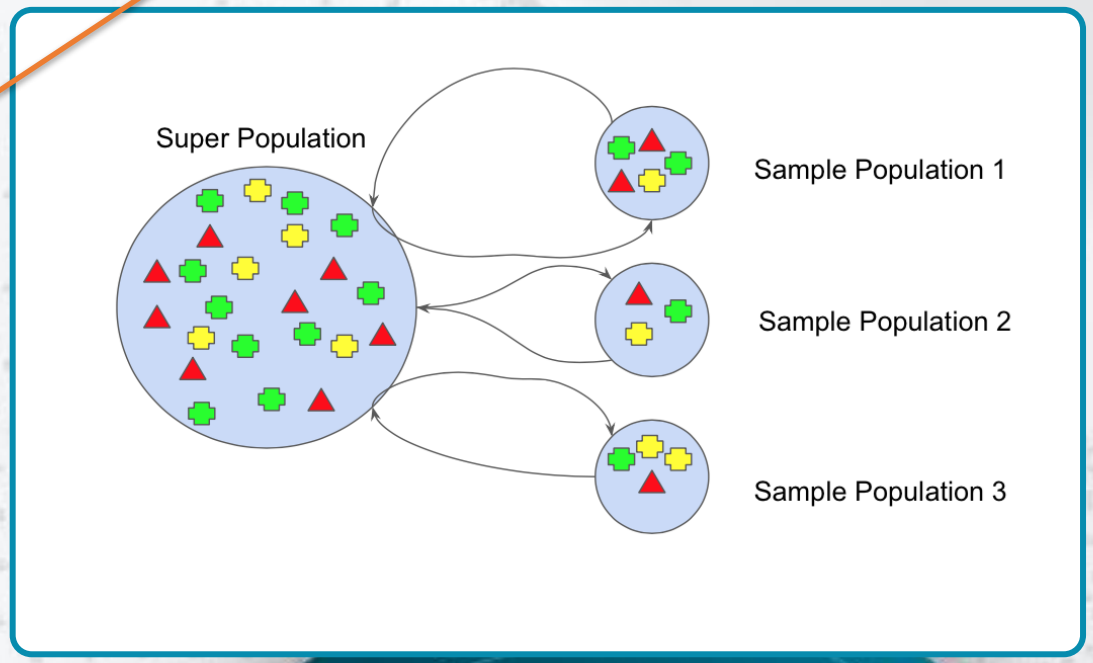
- Un algoritmo Random Forest es un método de aprendizaje automático que utiliza la técnica de ensamblado de árboles de decisión. En particular, consiste en construir múltiples árboles de decisión durante el entrenamiento y luego, combinar sus predicciones para obtener una predicción final.

- Cada árbol en el bosque se construye de forma independiente utilizando un subconjunto aleatorio del conjunto de datos original y un subconjunto aleatorio de las características disponibles. Esto se conoce como "muestreo de bolsa" o ”bagging", y tiene como objetivo reducir la correlación entre los árboles y mejorar la precisión general del modelo.

- El algoritmo Random Forest es conocido por su capacidad para manejar conjuntos de datos grandes y complejos, y por su resistencia al sobreajuste. Además, puede proporcionar información sobre la importancia relativa de cada característica en el conjunto de datos, lo que puede ser útil para la selección de características y la comprensión del problema.

Bagging

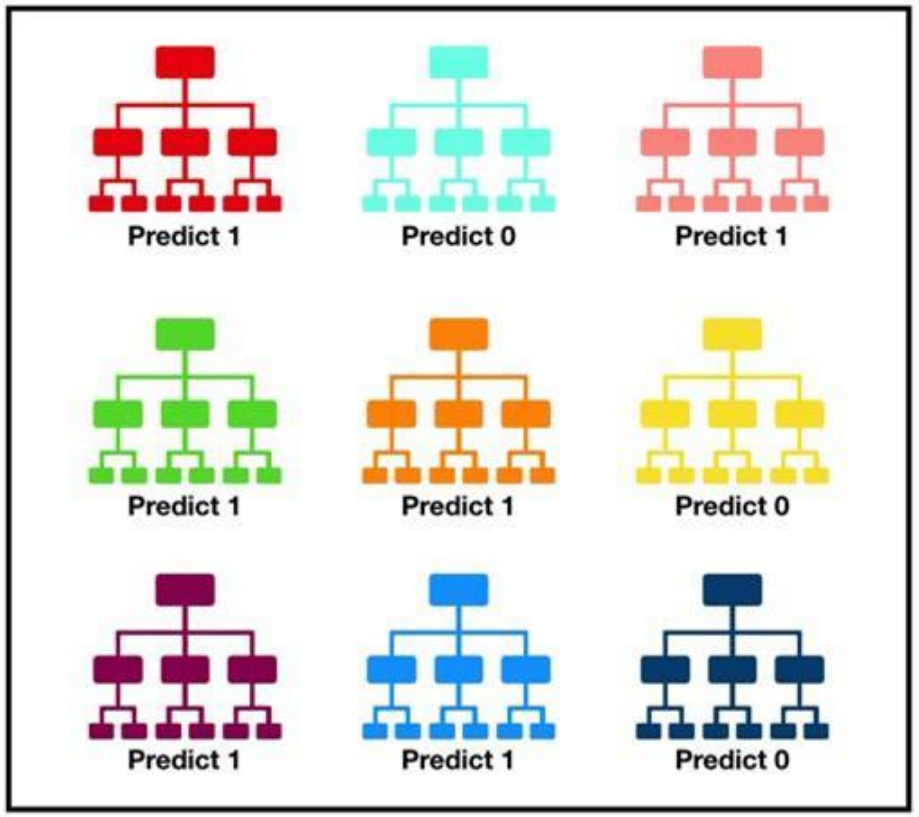
Bootstrapping + Aggregation



KIBERNUM

Random Forest

Una vez que se han construido los árboles, se combinan las predicciones individuales de cada árbol para producir una predicción final. En la mayoría de los casos, esto se hace utilizando el promedio (en problemas de regresión) o la mayoría de votos (en problemas de clasificación) de las predicciones de cada árbol.



(Predict: predicción)

La lógica del concepto de árboles aleatorios es simple pero poderosa: "***La sabiduría de la multitud***".

KIBERNUM

Entrenando Random Forest

Ahora repetimos el ejercicio entrenando esta vez el algoritmo Random Forest, con un valor de 200 árboles. Posteriormente, realizamos algunas predicciones

```
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(n_estimators=200)
```

```
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
rfc_predictions = rfc.predict(X_test)
```

KIBERNUM

Ventajas

1. Alta precisión: Random Forest es conocido por su alta precisión en la predicción de la variable de interés. Este modelo es capaz de manejar tanto problemas de clasificación como de regresión con una alta tasa de acierto.

1. Robustez: Random Forest es un modelo muy robusto que puede manejar conjuntos de datos grandes y complejos con múltiples características y clases.

1. Resistencia al sobreajuste: La técnica de muestreo de bolsa utilizada por Random Forest ayuda a reducir la correlación entre los árboles y, por lo tanto, reduce la probabilidad de sobreajuste.

1. Selección de características: El modelo proporciona información sobre la importancia relativa de cada característica en el conjunto de datos, lo que puede ser útil para la selección de características y la comprensión del problema.

1. Fácil de implementar: Random Forest es fácil de implementar y de usar, incluso para usuarios con poca experiencia en aprendizaje automático.

Desventajas

1. Dificultad de interpretación: Aunque Random Forest es un modelo muy preciso, puede ser difícil de interpretar debido a la complejidad de los árboles de decisión.

1. Tiempo de entrenamiento: El entrenamiento de un modelo Random Forest puede llevar más tiempo que otros algoritmos de aprendizaje automático debido a la construcción de múltiples árboles.

1. Espacio de almacenamiento: Como Random Forest utiliza múltiples árboles, puede requerir más espacio de almacenamiento que otros modelos más simples.

1. Sensible a datos desequilibrados: Si el conjunto de datos está desequilibrado, es decir, si hay más muestras de una clase que de otra, el modelo puede tener un sesgo hacia la clase dominante.

Dudas y consultas

KIBERNUM

Fin de la Presentación

KIBERNUM