Aprendizaje Automático

Departamento de Informática – UC3M

TUTORIAL 1 – Evaluación con árboles

2024

Tutorial 1

Entorno de trabajo

Google colab + GitHub

Otras opciones:

- Spyder
- DataSpell
- Visual Studio

Tutorial 1

Biblioteca sklearn

- Los conjuntos de datos para sklearn son **matrices numpy** (numéricas):
- Esto implica que los atributos/características categóricas deben ser representadas como:
  - Enteros
  - One-hot-encoding / variables dummy
- Sin embargo, hay una tendencia para integrar pandas dataframes con scikit learn
- Los valores inexistentes se representan como **np.nan**

Tutorial 1

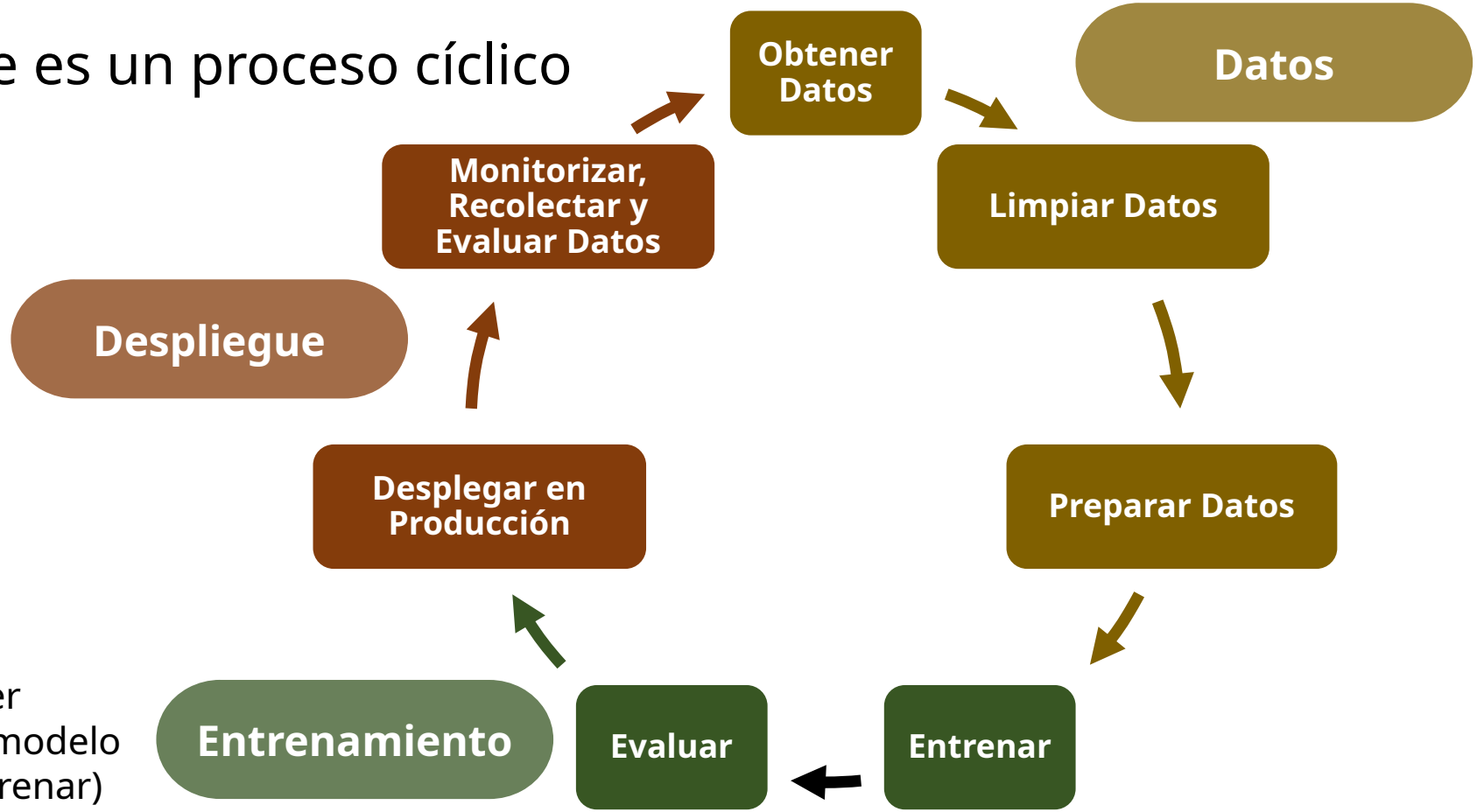
Árbol de decisión. Iris dataset

- Atributos (numéricos):
  - sepal length (cm)
  - sepal width (cm)
  - petal length (cm)
  - petal width (cm)
- Clase (enteros)
  - Iris-Setosa (0)
  - Iris-Versicolour (1)
  - Iris-Virginica (2)

Tutorial 1

Pipeline del Aprendizaje Automático

- Realmente es un proceso cíclico



Obtener
- Un modelo (entrenar)
- Su estimación de comportamiento futuro (evaluar)

Proceso de Aprendizaje Automático

Preprocesamiento de Datos

- Obtención, fusión, transformación, filtrado
- Tratamiento de datos imperfectos **(TRANSFORMER)**
  - Reducción del ruido (p.e. edición de Wilson en k-NN)
  - Eliminación de datos redundantes (p.e. condensación en k-NN)
  - Tratamiento de datos desconocidos
  - Identificación y eliminación de datos anómalos (*outliers*)
- Transformación de datos **(TRANSFORMER)**
  - Discretización
  - Binarización
  - Normalización
  - Selección de instancias
  - Selección de atributos
- Balanceo de datos (*undersampling*, *oversampling*, SMOTE - *Synthetic Minority Over-sampling Technique*, ...)

Proceso de Aprendizaje Automático

Preprocesamiento de Datos (TRANSFORMER)

- Quitar atributos constantes
- Escalado (normalización):
  - range: hacer que todos los atributos estén en un mismo rango, típicamente 0-1
  - estandarización: hacer que todos los atributos sigan una distribución gaussiana de media 0 y de sd 1
- Variables *dummy* / *one-hot encoding*: puede ser conveniente transformar variables categóricas (discretas) con n valores en n (o n-1) variables binarias:
- Imputación (¿qué hacer si hay valores faltantes?)
- Selección de atributos (*feature selection*)
  - Ej: elegir los atributos relevantes (salario) frente a los no relevantes (ej: color de ojos); a la hora de predecir si se va a devolver un préstamo
- Extracción de atributos (*feature extraction*): transformación de atributos
  - PCA, ...
- Creación de atributos / *feature engineering* (ej: velocidad, a partir de v_x y v_y, en un problema de predicción de energía eólica)

Tutorial 1

Obtener datos / limpiar datos / preparar datos

- ¡¡¡Vamos a pintar!!!

Tutorial 1

Ejercicio

```
lista=[np.count_nonzero(y == 0), np.count_nonzero(y == 1), np.count_nonzero(y == 2)]
sns.barplot(x = ['setosa','versicolor','virginica'], y = lista)
plt.show()
```

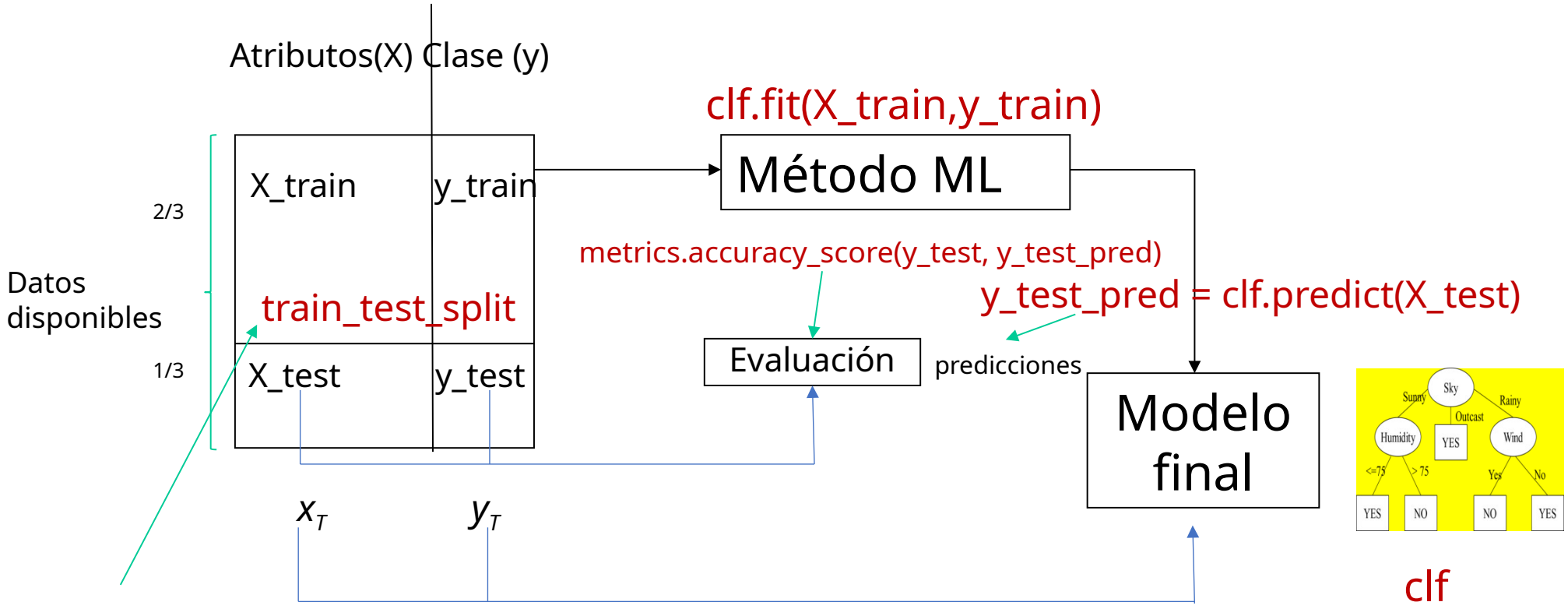
Tutorial 1

Entrenando un árbol de decisión

- Todos los modelos:
  1. Definir el modelo
  2. Entrenar el modelo
- La implementación en scikit-learn es una versión optimizada de los árboles CART (Classification and Regression Trees).
  - Son árboles binarios.
  - No soporta valores categóricos (por ahora...)

Tutorial 1

Entrenando con muestras de entrenamiento/test (holdout)



Random shuffling antes de dividir

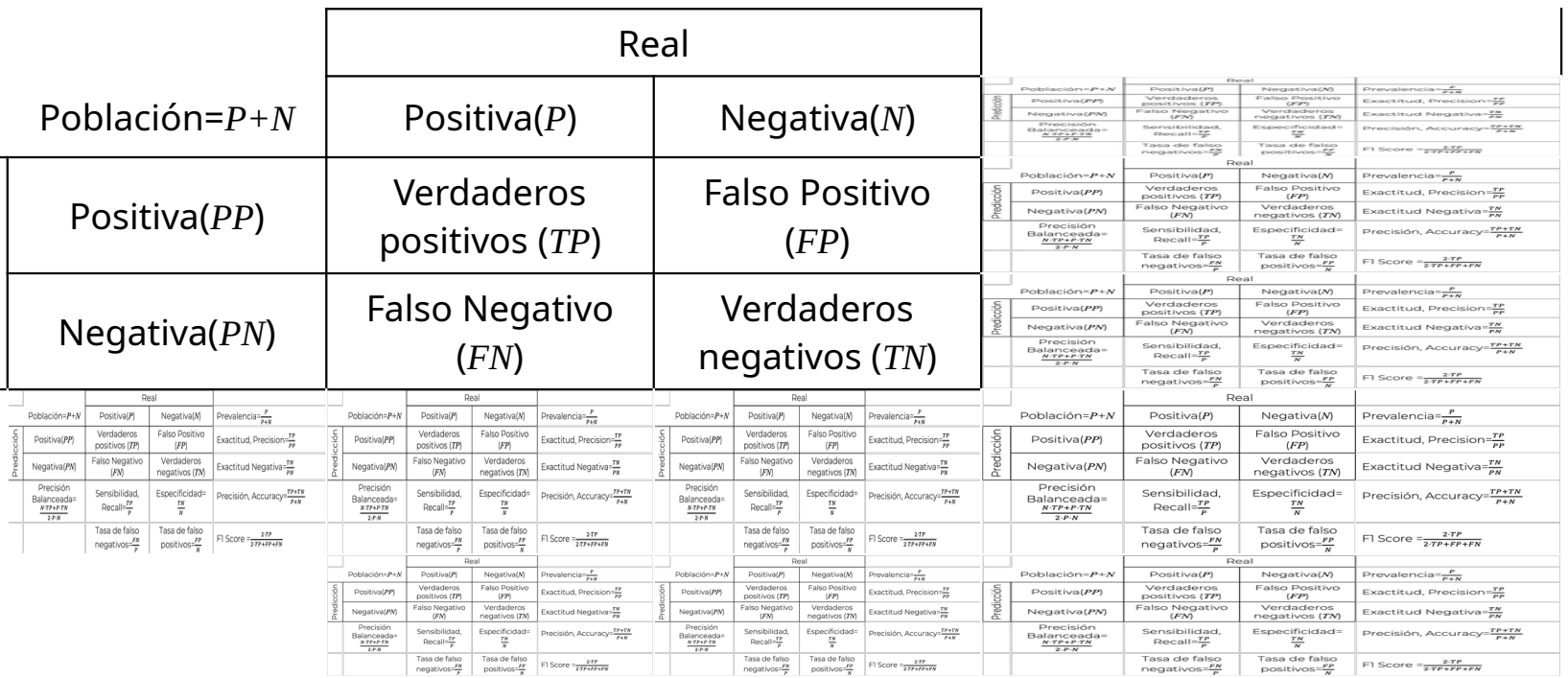
Tutorial 1

Medidas del Error de Clasificación

Matriz de confusión (matriz de error)

- Consideremos una clasificación binaria, con una clase Positiva(P) y otra Negativa(N)

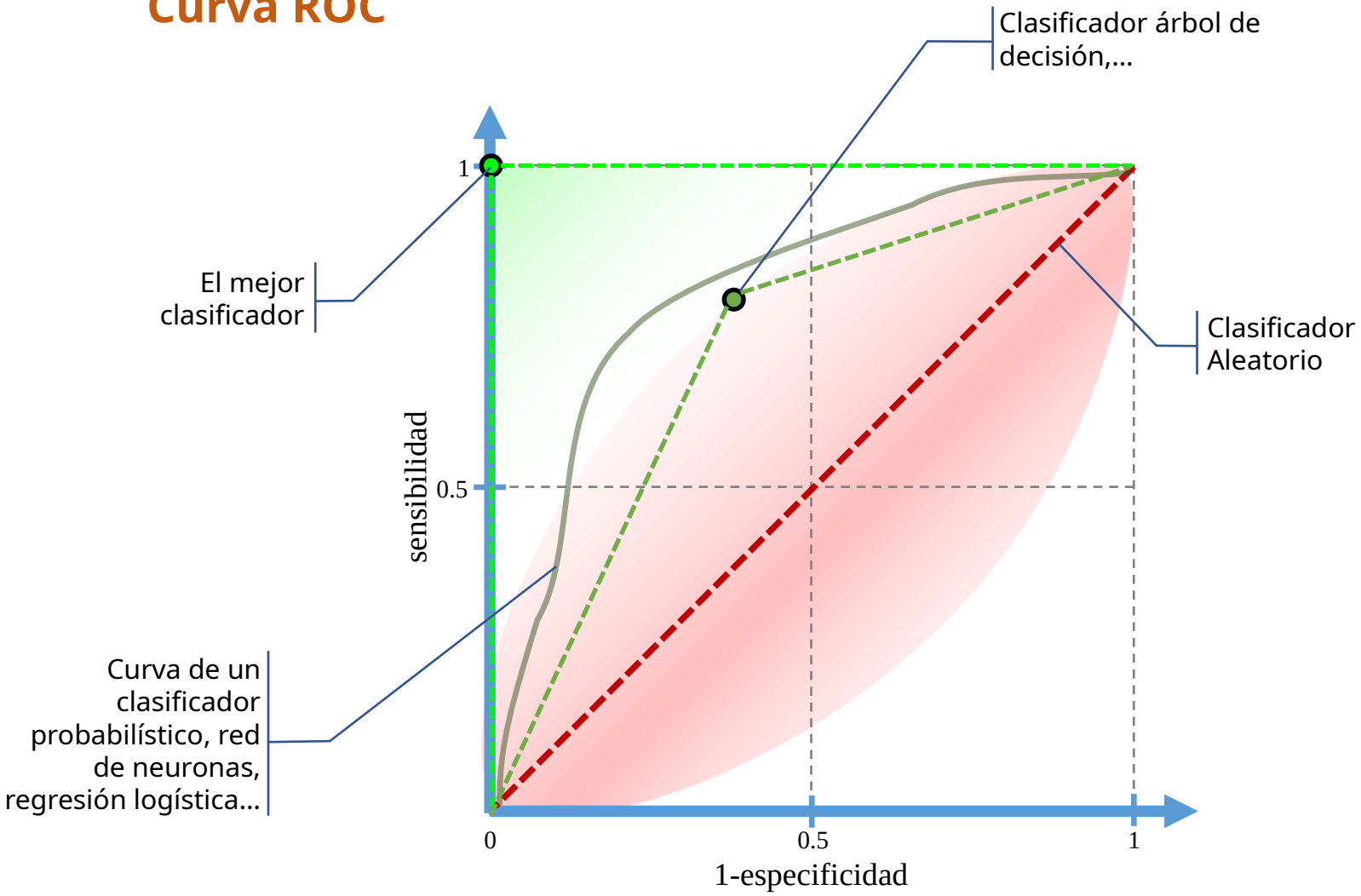
		Real	
	Población=$P+N$	Positiva(P)	Negativa(N)
Predicción	Positiva(PP)	Verdaderos positivos (TP)	Falso Positivo (FP)
	Negativa(PN)	Falso Negativo (FN)	Verdaderos negativos (TN)



Tutorial 1

Medidas del Error de Clasificación

Curva ROC



Cada punto del espacio ROC se corresponde con una matriz de confusión

Una medida muy útil que se obtiene a partir de la curva ROC es el **área bajo la curva ROC** (AUC)

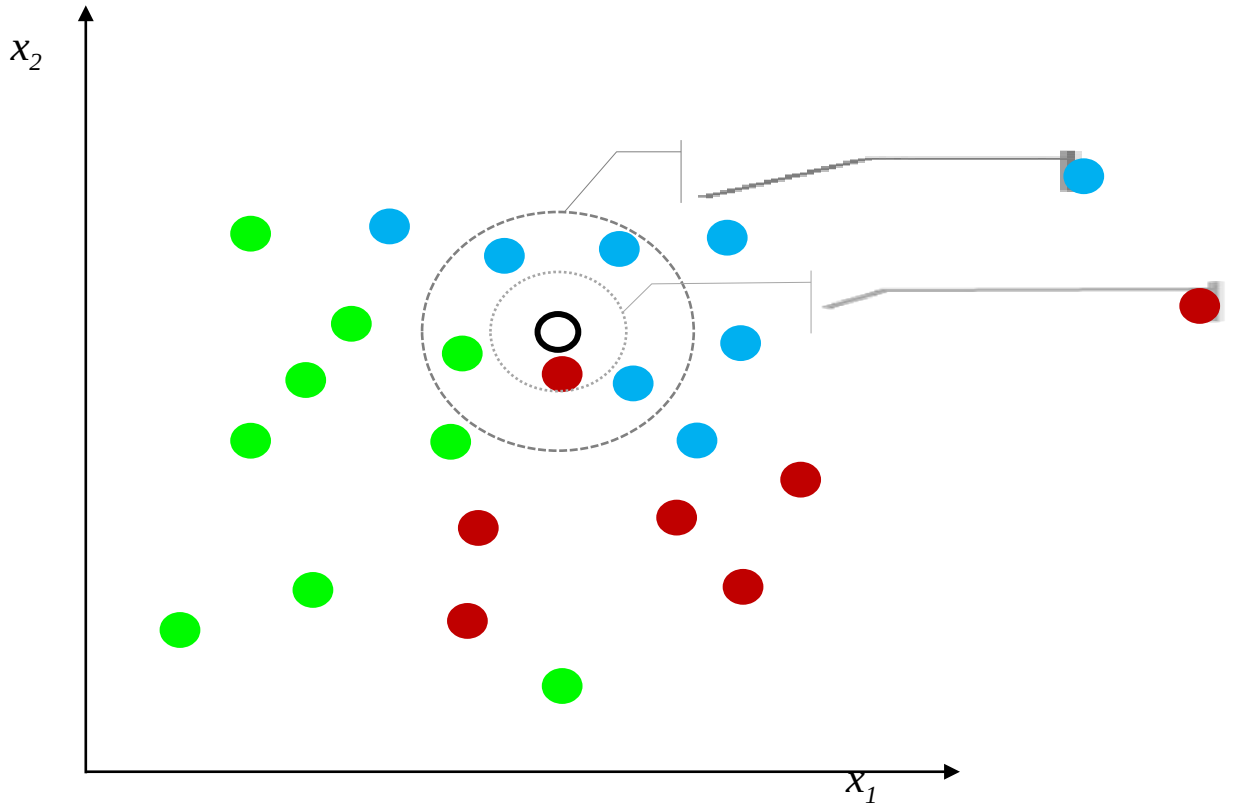
El clasificador aleatorio tiene AUC=0.5 y el mejor clasificador AUC=1

Dos clasificadores pueden tener el mismo ACC pero diferente AUC, será preferible escoger al de mayor AUC

Un clasificador puede ser considerado bueno a partir de un AUC>0.75

Tutorial 1

Aprendizaje Basado en Instancias. *K-NN*



Tutorial 1

Entrenamiento , predicción, evaluación y construcción del modelos final (KNN)

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')


# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)


# Here, we set our model to classification tree
clf = KNeighborsClassifier()
np.random.seed(42) # reproducibility
# We train it
clf.fit(X_train, y_train)
# We obtain predictions on the test set
y_test_pred = clf.predict(X_test)
```

Tutorial 1

Entrenamiento , predicción, evaluación y construcción del modelos final (KNN)

```
# We compute accuracy
accuracy_knn = metrics.accuracy_score(y_test, y_test_pred)
print(f"Accuracy of KNN: {accuracy_knn} ")
result1 = metrics.classification_report(y_test, y_test_pred)
print("Classification Report:",)
print (result1)
# Creates a confusion matrix
cm = metrics.confusion_matrix(y_test, y_test_pred)
# Transform to df for easier plotting
cm_df = pd.DataFrame(cm,
                     index = ['setosa','versicolor','virginica'],
                     columns = ['setosa','versicolor','virginica’])
```

Tutorial 1

Entrenamiento , predicción, evaluación y construcción del modelos final (KNN)

```
# Transform to df for easier plotting
cm_df = pd.DataFrame(cm,
                     index = ['setosa','versicolor','virginica'],
                     columns = ['setosa','versicolor','virginica'])


plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Accuracy:{0:.3f}'.format(accuracy_tree))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# We finally compute the final model with all available data

final_clf = KNeighborsClassifier()
np.random.seed(42)  # reproducibility
final_clf.fit(X, y)
```

Tutorial 1

Entrenando un árbol de decisión (K-fold)

Tutorial 1

Entrenando un árbol de decisión (StratifiedKFold). Ejercicio

```
from sklearn.model_selection import cross_val_score,StratifiedKFold

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

clf = tree.DecisionTreeClassifier()

# Making model training reproducible
np.random.seed(42)

scores = cross_val_score(clf, X, y, scoring='accuracy', cv = cv)

print(f"All the accuracies are: {scores}")
print(f"And the average crossvalidation accuracy is: {scores.mean():.2f} +-
{scores.std():.2f}")
```

Tutorial 1

Modificación de parámetros de un árbol

- Se han discutido anteriormente técnicas de pre y post-poda para evitar tener árboles muy profundos con pocas instancias
- La mayor parte de métodos de árboles permiten controlar la profundidad del árbol mediante dos hiperparámetros:
  - **max_depth**

    máxima profundidad del árbol
  - **min_samples_split**

    número mínimo de instancias que debe de tener un nodo interno para ser subdividido

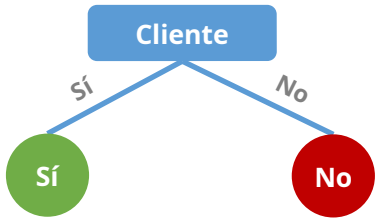
Tutorial 1.

Árboles de decisión. Hiper-parámetros

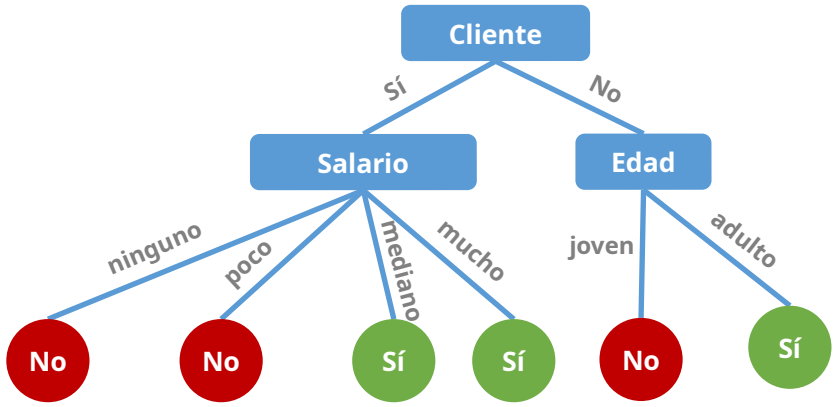
max_depth

- máxima profundidad del árbol

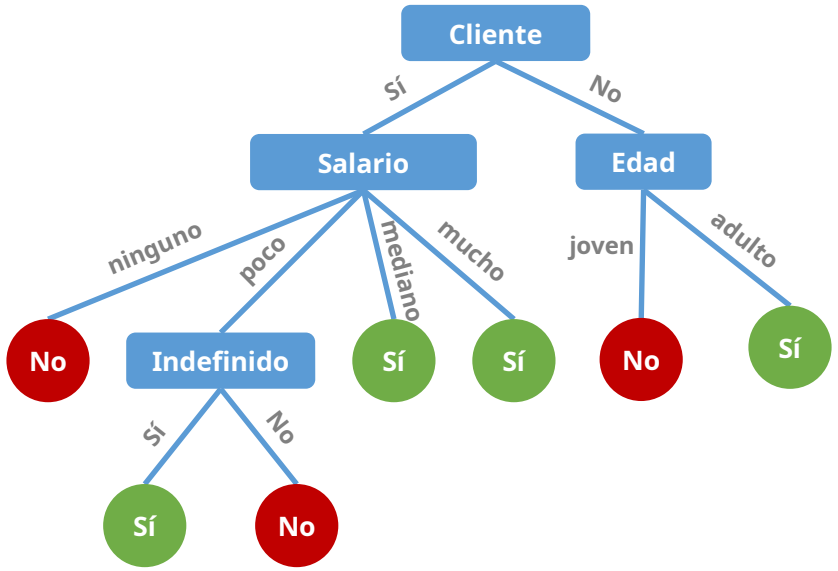
max_depth = 1



max_depth = 2



max_depth = 3



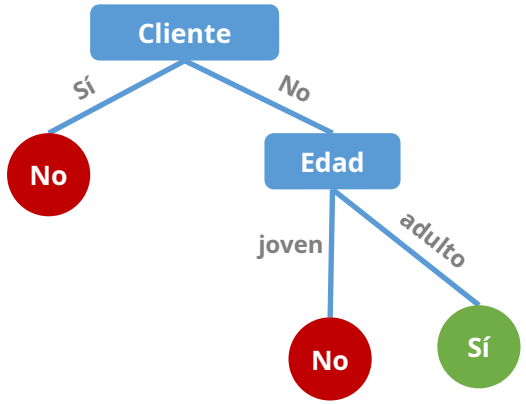
Tutorial 1

Árboles de decisión. Hiper-parámetros

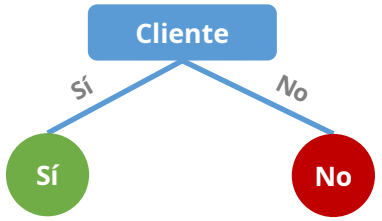
min_samples_split

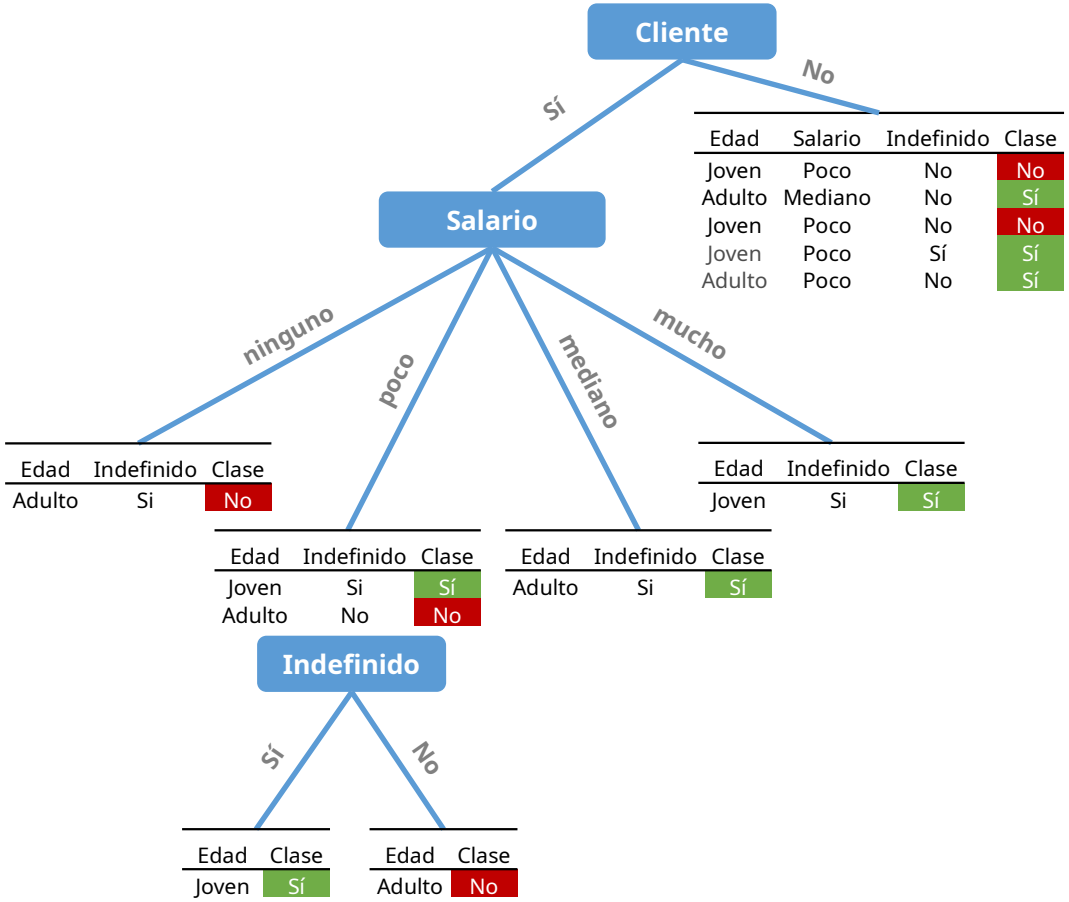
- número mínimo de instancias que debe de tener un nodo para ser subdividido

min_samples_split = 2



min_samples_split = 3





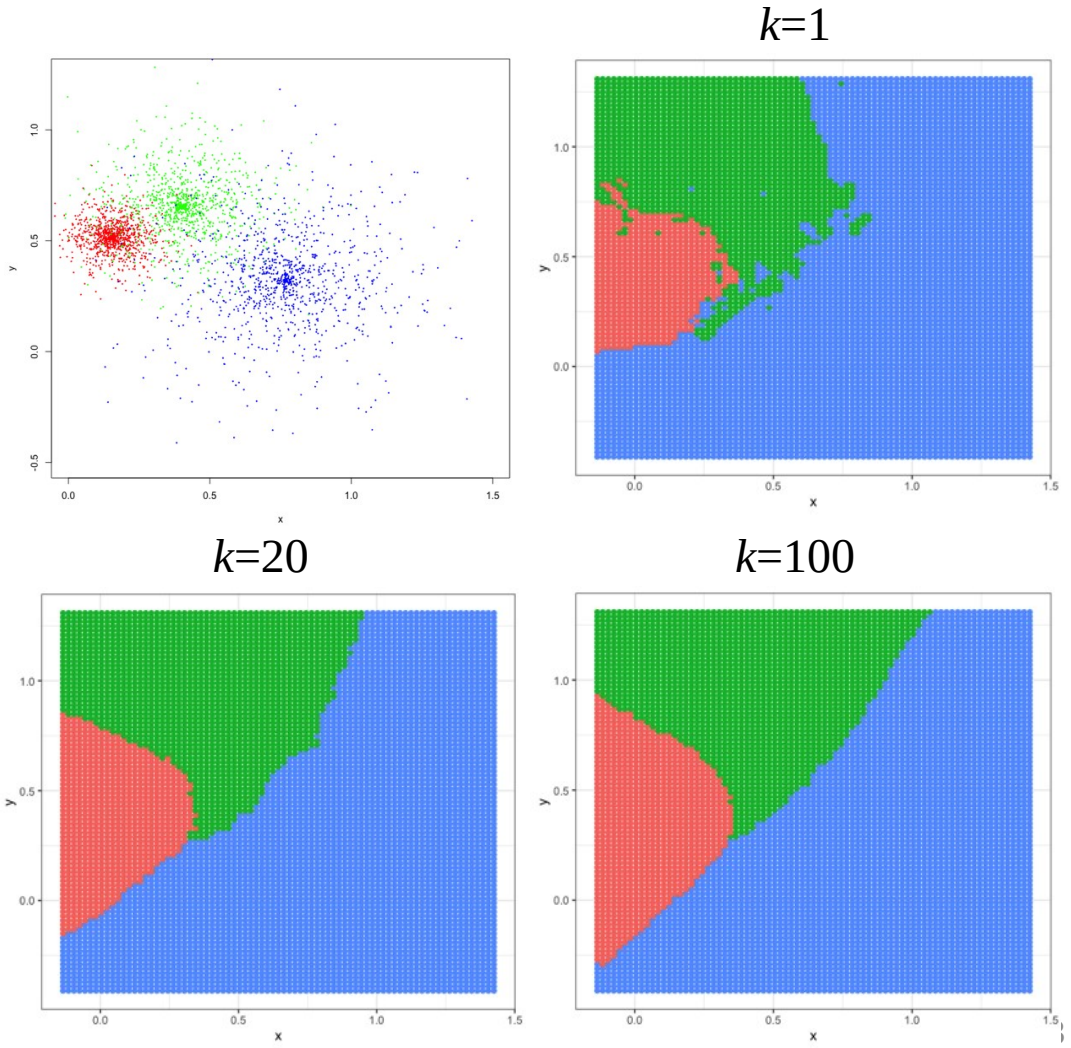
Tutorial 1

KNN

k es el hiperparámetro principal

Efecto de k

- Si k es muy pequeño entonces sobre-aprende y es sensible al ruido. Frontera de decisión ruidosa
- Si k es muy grande entonces se incluyen instancias de otras clases perdiendo la idea de localidad. Frontera de decisión muy suave
- Con el k adecuado, se consideran suficientes vecinos y el ruido pierde influencia (se realiza un promedio local)



Tutorial 1

KNN

```
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

for k in range(1,100,5):
        clf = KNeighborsClassifier(n_neighbors=k)
        np.random.seed(42)
        clf.fit(X_train,y_train)
        y_test_pred = clf.predict(X_test)
        accuracy_knn = metrics.accuracy_score(y_test, y_test_pred)
        print(f"With k neighbors {k}: {accuracy_knn:.2f}")


print()

for weights in ["uniform", "distance"]:
        clf = KNeighborsClassifier(n_neighbors=5,weights=weights)
        np.random.seed(42)
        clf.fit(X_train,y_train)
        y_test_pred = clf.predict(X_test)
        accuracy_knn = metrics.accuracy_score(y_test, y_test_pred)
        print(f"With {weights}: {accuracy_knn:.2f}")
```

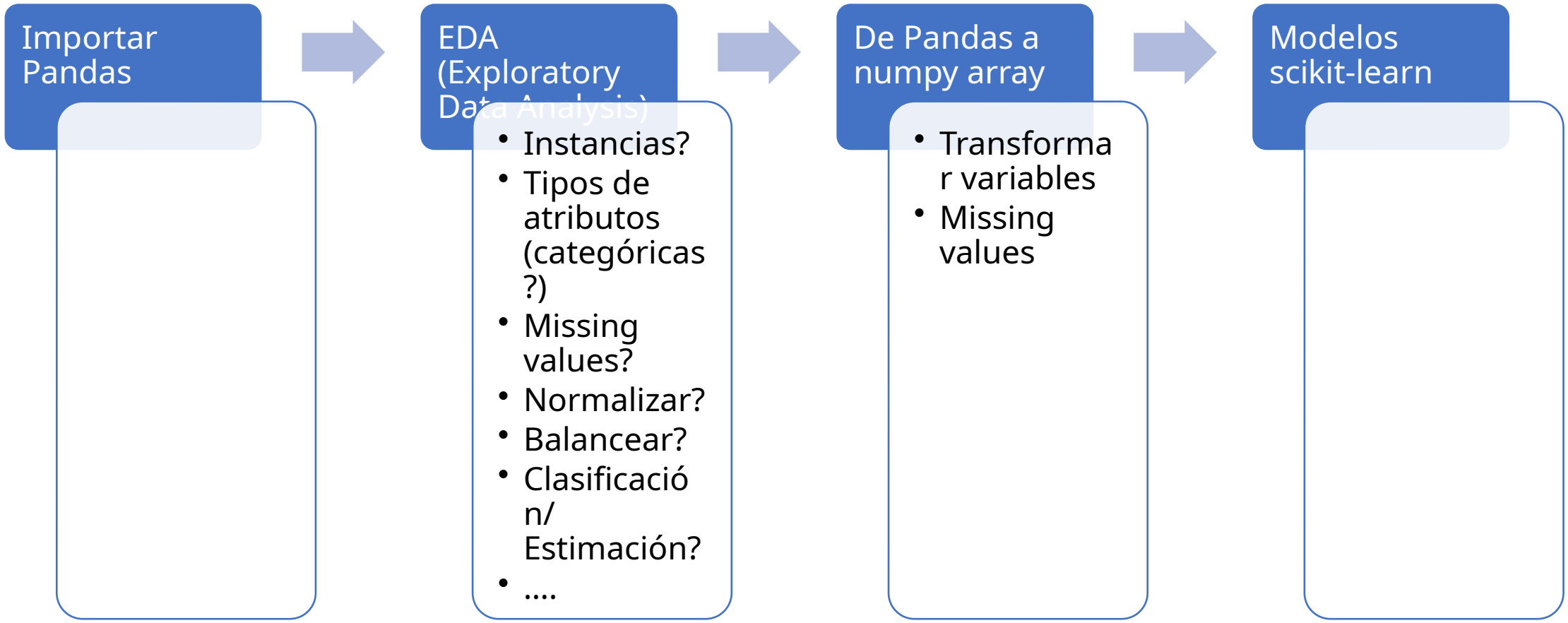
Tutorial 1

Tratamiento de variables categóricas en DecisionTreeClassifier

- La implementación de árboles de Sklearn **NO PUEDE** tratar con variables categóricas (en la mayoría de los casos).
- Deben convertirse en variables ficticias (one-hot-encoding).
- Los árboles Sklearn tampoco pueden tratar con valores faltantes (missing values)

Tutorial 1

Tratamiento de variables categóricas en DecisionTreeClassifier



Tutorial 1

Preprocesamiento de Datos

- Quitar atributos constantes
- Escalado (normalización):
  - range: hacer que todos los atributos estén en un mismo rango, típicamente 0-1
  - estandarización: hacer que todos los atributos sigan una distribución gaussiana de media 0 y de sd 1
- **Variables *dummy / one-hot encoding***: puede ser conveniente transformar variables categóricas (discretas) con n valores en n (o n-1) variables binarias:
- Imputación (¿qué hacer si hay valores faltantes?)
- Selección de atributos (*feature selection*)
  - Ej: elegir los atributos relevantes (salario) frente a los no relevantes (ej: color de ojos); a la hora de predecir si se va a devolver un préstamo
- Extracción de atributos (*feature extraction*): transformación de atributos
  - PCA, ...
- Creación de atributos / *feature engineering* (ej: velocidad, a partir de v_x y v_y, en un problema de predicción de energía eólica)

Tutorial 1

Preprocesamiento de Datos. Tennis.txt

- Variables de entrada:
  - Sky: sunny, rainy, overcast,.
  - Temperature: Integer
  - Humidity: Integer
  - Windy: False, True
- Clase:
  - Play: Yes, No

Tutorial 1

```
probs = clf.predict_proba(X_test)[:, 1]
auc = metrics.roc_auc_score(y_test, probs)
fpr, tpr, thresholds = metrics.roc_curve(y_test, probs)
plt.figure(figsize=(8, 5))
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], color='blue', linestyle='--',
label='Baseline')
plt.title('Curva ROC', size=20)
plt.xlabel('Falsos Positivos', size=14)
plt.ylabel('Verdaderos Positivos', size=14)
plt.legend();
```

Tutorial 1

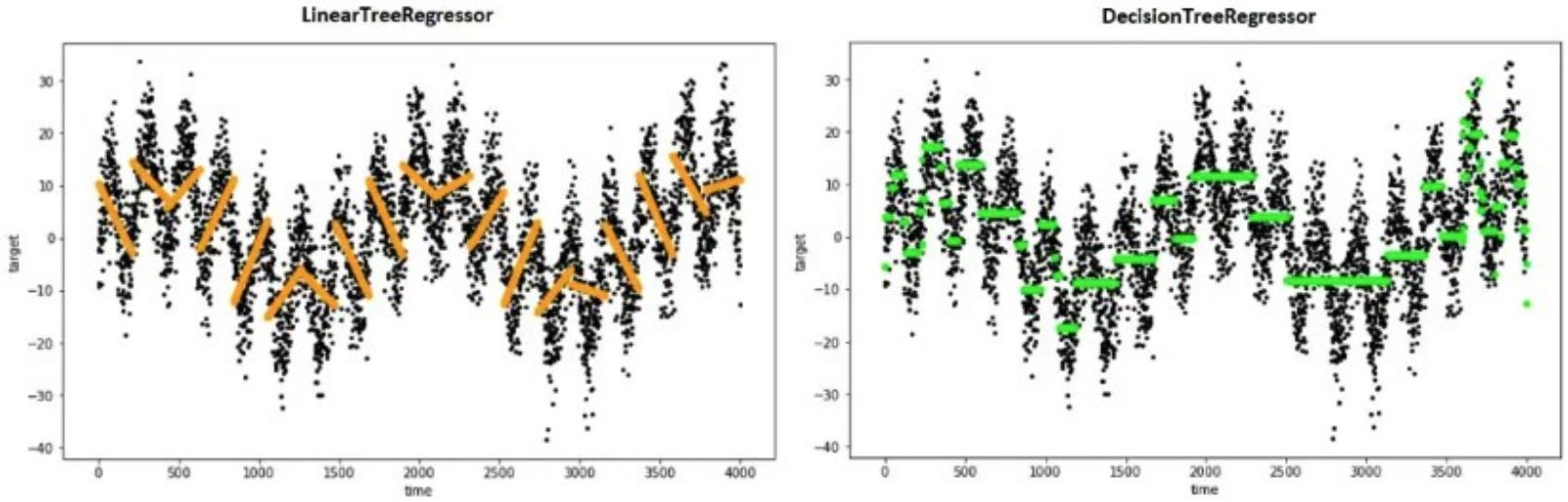
Árboles de regresión

- California dataset:
  - `MedInc - median income in block group`
  - `HouseAge - median house age in block group`
  - `AveRooms - average number of rooms per household`
  - `AveBedrms - average number of bedrooms per household`
  - `Population -  block group population`
  - `AveOccup - average number of household members`
  - `Latitude - block group latitude`
  - `Longitude - block group longitude`

Tutorial 1

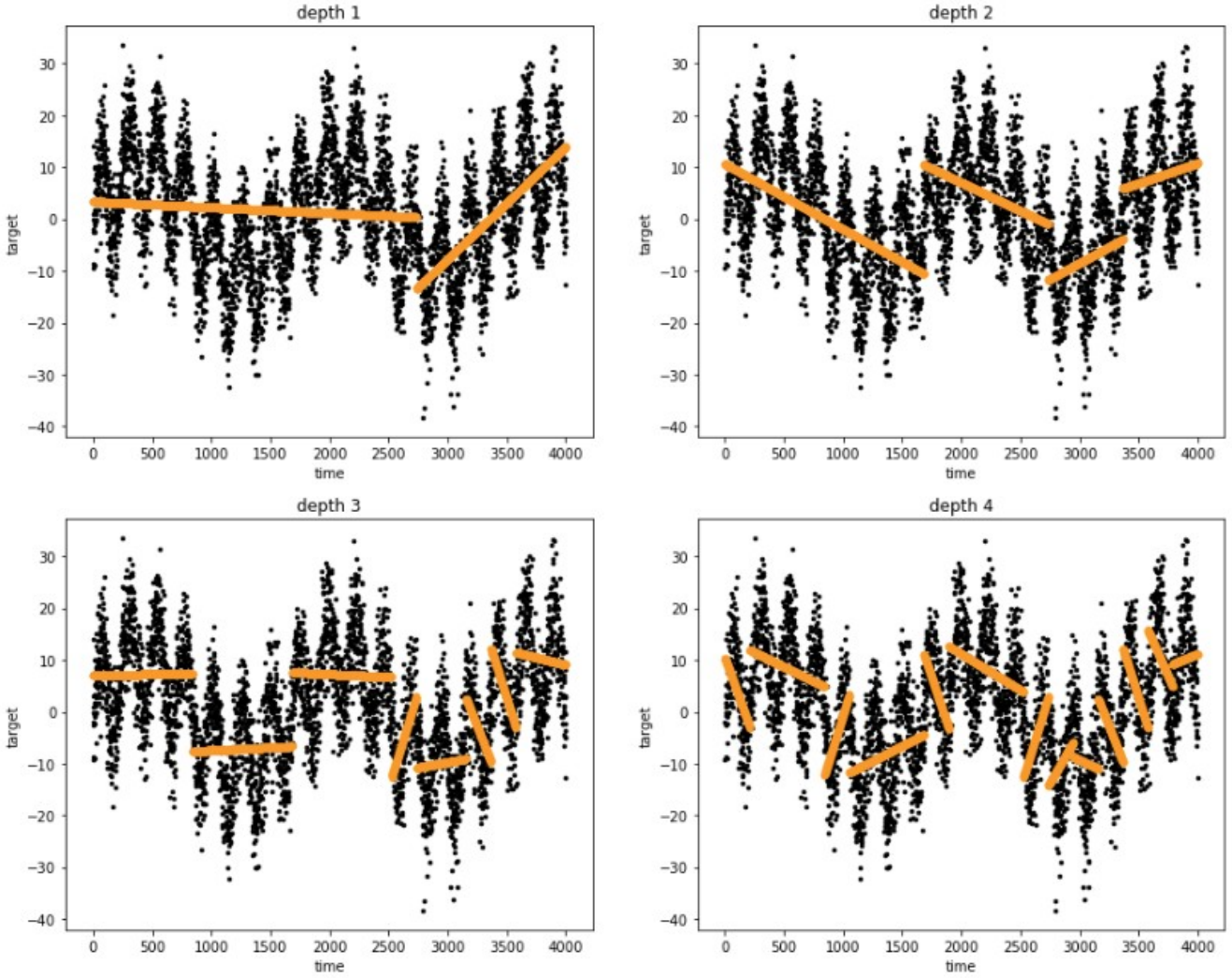
Árboles de regresión

- Dos posibilidades:
  - DecisionTreeRegressor (sklearn)
  - LinearTreeRegressor (lineartree)



Tutorial 1

Árboles de regresión



Tutorial 1

```
regr = LinearTreeRegressor(base_estimator=LinearRegression())
np.random.seed(42) # reproducibility
# We train it
regr.fit(X_train, y_train)
# We obtain predictions on the test set
y_test_pred = regr.predict(X_test)
# We compute accuracy
rmse_tree = np.sqrt(metrics.mean_squared_error(y_test,
y_test_pred))
print(f"RMSE of the tree: {rmse_tree}")
```

Tutorial 1

```
# obtenemos información de los nodos hoja del árbol

leaves = regr.summary(feature_names=['MedInc'], only_leaves=True,
max_depth=None)
leaves


# Obtenemos los coeficientes para el nodo 12 (u otro cualquiera)
model_12_coefs = leaves[12]['models'].coef_
model_12_intercept = leaves[12]['models'].intercept_
pprint(list(zip(['MedInc'], model_12_coefs)))
pprint(f'intercept: {model_12_intercept}')
```