MINERÍA DE DATOS

Maximiliano Ojeda

muojeda@uc.cl

IIC-2433

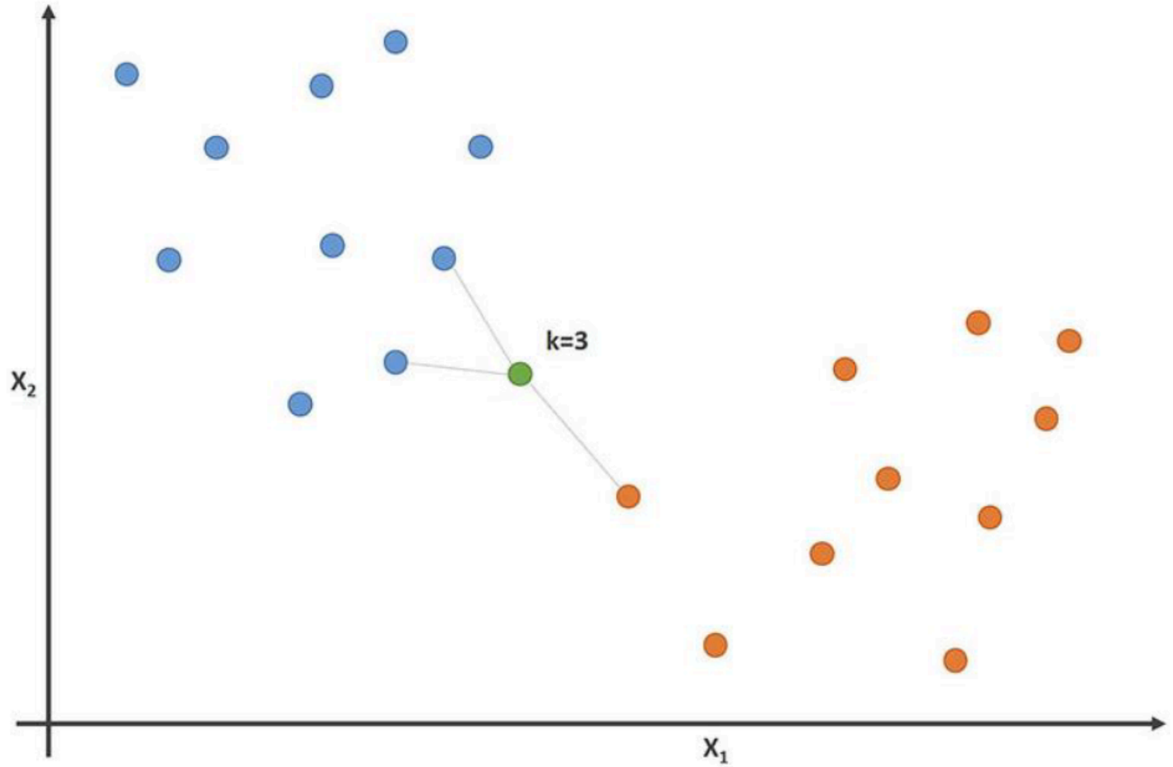
Vecinos Cercanos

Vecinos Cercanos

Los métodos de vecinos cercanos son la base de muchos otros métodos de aprendizaje. **Consiste en encontrar un número predefinido de muestras que están más próximas en distancia a un nuevo punto.**

El número de muestras puede ser:

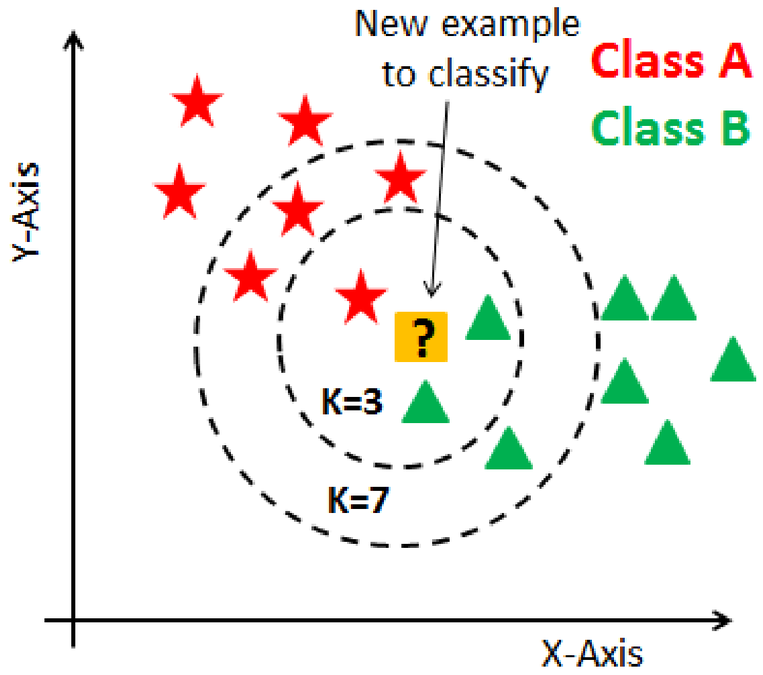
- Una constante definida por el usuario (k-vecinos más cercanos)
- Variar según la densidad local de puntos (aprendizaje basado en un radio determinado).



Vecinos Cercanos

Para disponer de una estructura de vecinos cercana que podamos consultar, se usa alguno de los siguientes tres algoritmos:

- **Pairwise metric** (fuerza bruta, pocos datos)
- **BallTree** (alta dimensionalidad)
- **kd-trees** (baja dimensionalidad)

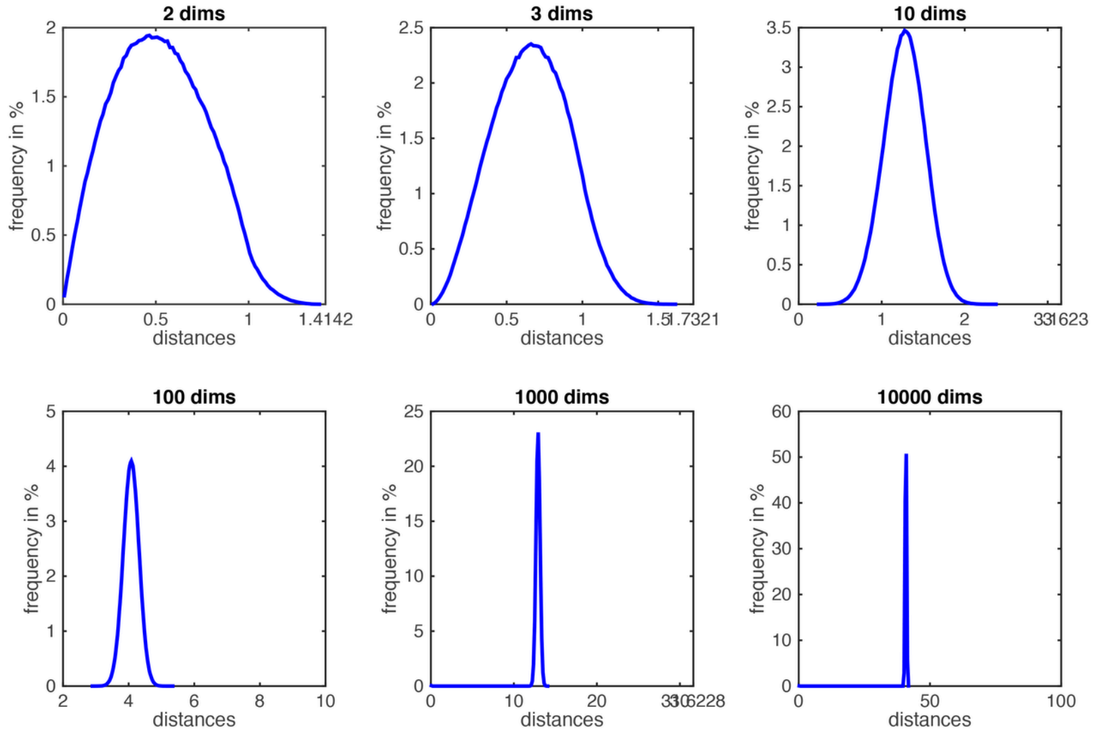


Vecinos Cercanos

Pairwise distance

La cantidad de puntos es una de las grandes limitaciones de K-NN. Cuando hay muchos datos o muchas dimensiones, **calcular la distancia euclídea contra todos los puntos se vuelve muy costoso.**

Recordar que en espacios de alta dimensión, la distancia euclídea pierde sentido porque **todos los puntos tienden a estar casi igual de lejos.**



Vecinos Cercanos

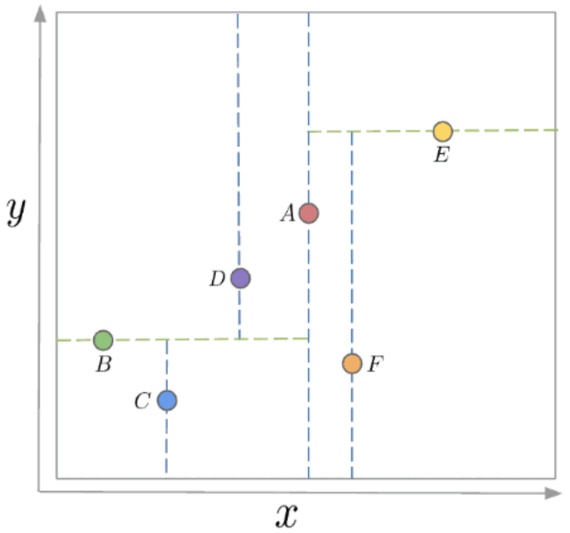
KD-Tree

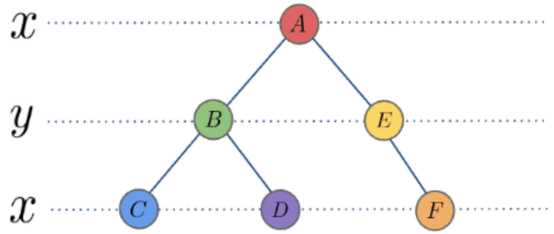
Es una estructura de datos diseñada para **organizar puntos en un espacio de varias dimensiones y así acelerar la búsqueda de vecinos cercanos**.

Es un árbol binario que en cada nodo interno:

1. Se escoge una dimensión (x, y, z, ...).
2. Se selecciona un punto pivote.
3. Se divide el espacio en dos:
   - **Subárbol izq:** puntos con coordenada menor al pivote en esa dimensión.
   - **Subárbol der:** puntos con coordenada mayor.

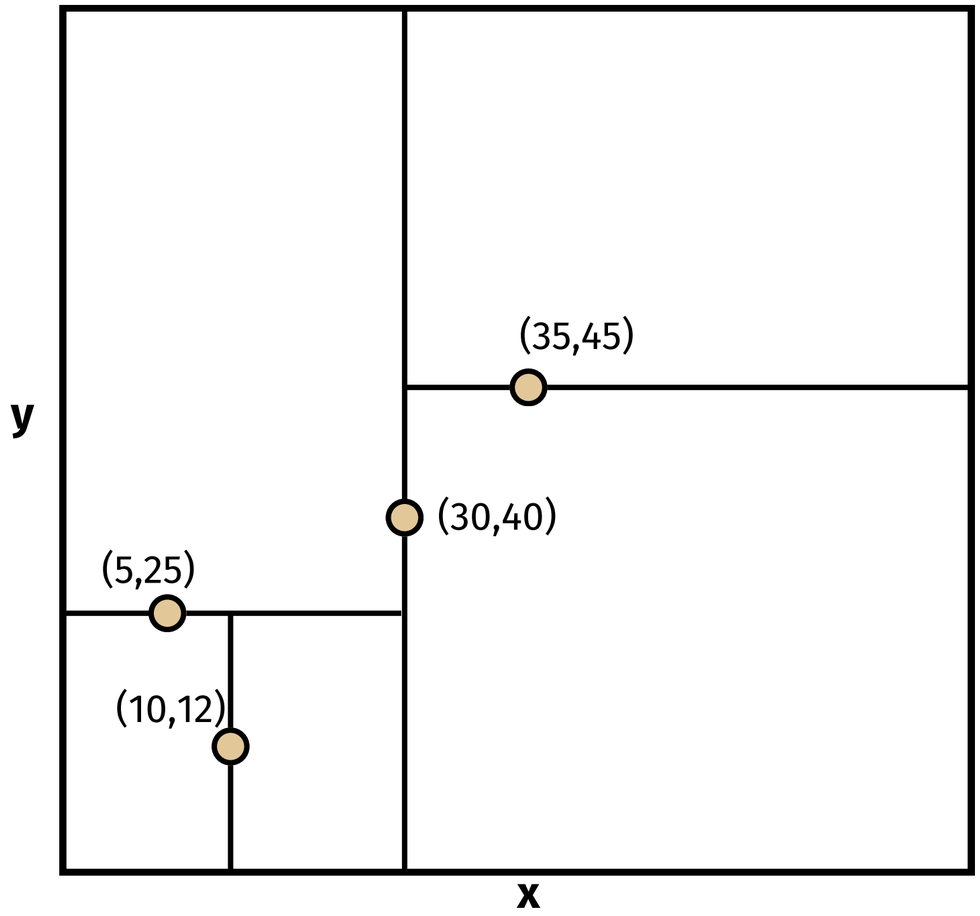
Esto se repite de manera recursiva alternando las dimensiones (ejemplo: primero x, luego y, etc).

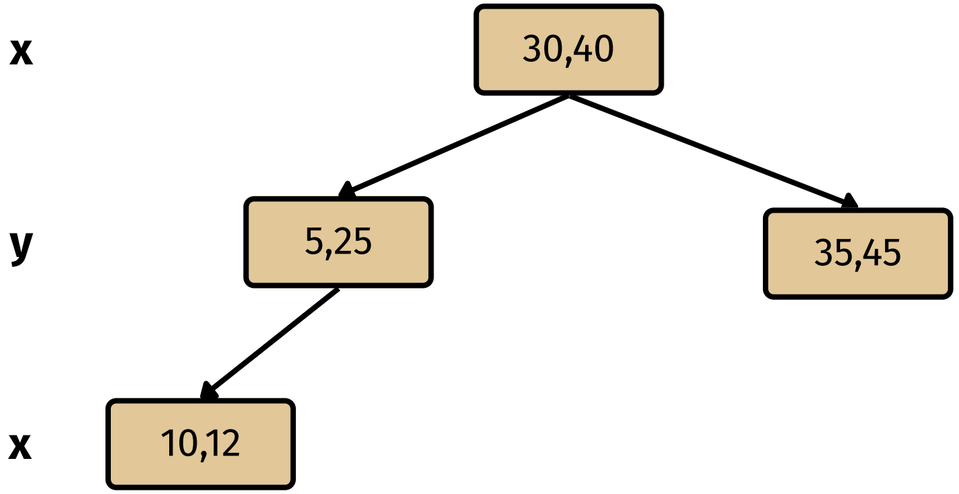




Vecinos Cercanos

KD-Tree





Puntos = [(30,40), (5,25), (10,12), (35,45)]

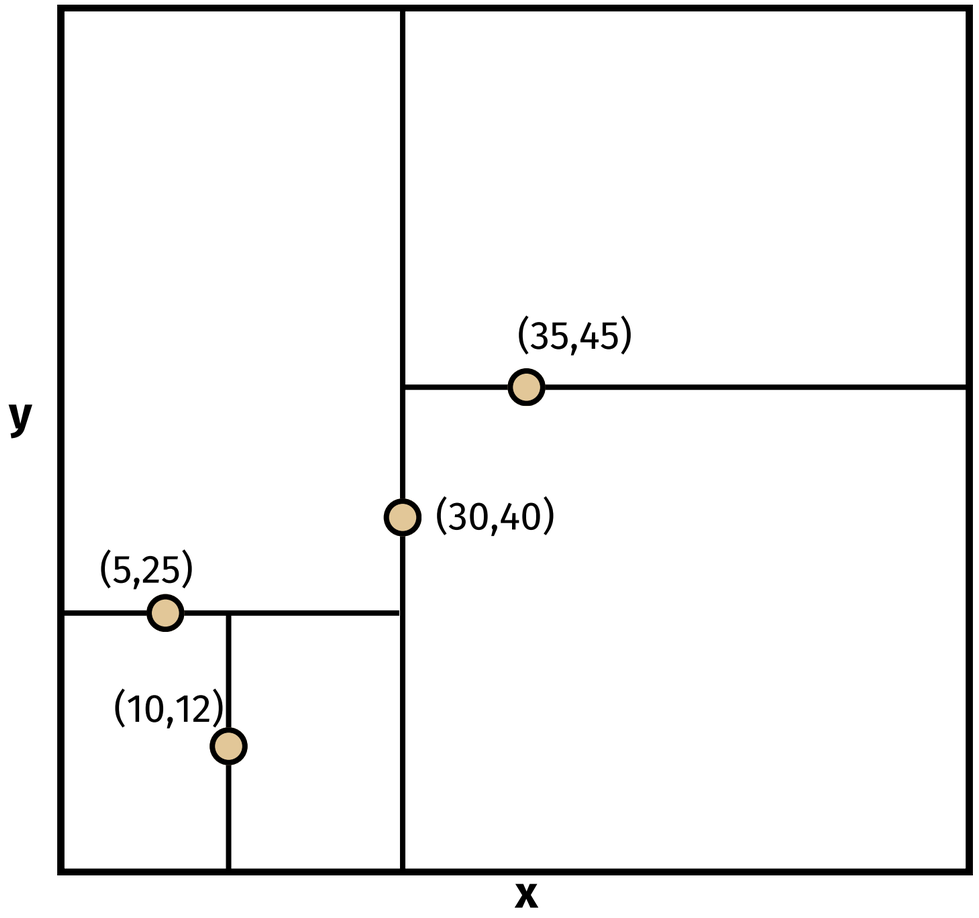
Vecinos Cercanos

KD-Tree

Siempre se divide usando la mediana del sub-espacio

Puntos = {(30,40), (5,25), (10,12), (35,45), (50,30), (70,70)}

1. Eje x: Mediana {5, 10, **30**, 35, 50, 70} → (30, 40)
2. Eje y: Mediana {12, **25**, 40}→ (5, 25)
3. Eje x: Mediana {5, **10**, 30}→ (10, 12)
4. Eje y: Mediana {30, **45**, 70}→ (35, 45)



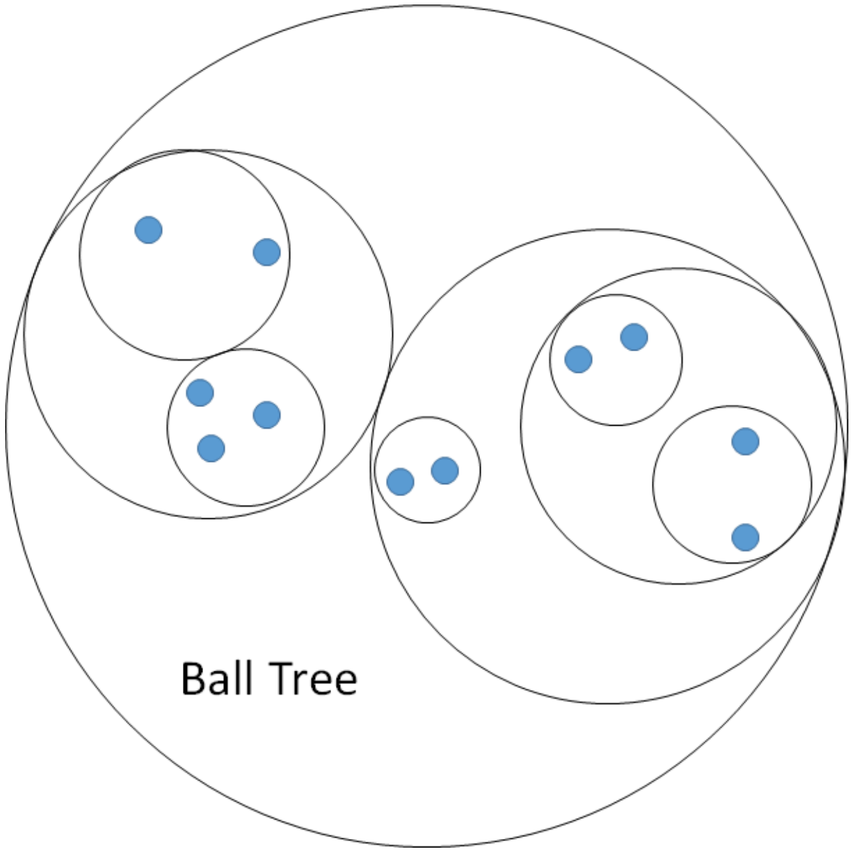
Ball-Tree

Un **Ball-Tree es otra estructura de datos** para vecinos cercanos, pensada para superar algunas limitaciones de los KD-tree.

Se crea un árbol binario. **Cada nodo define la esfera más pequeña** que contiene los puntos de su subárbol.

El criterio de construcción da lugar a un invariante que usaremos en búsqueda:

Dado un punto externo *t* a una esfera *B*, su distancia a cualquier punto de *B* será mayor o igual que la distancia a la superficie de *B*.



Ball-Tree

Representación 1 Nodo

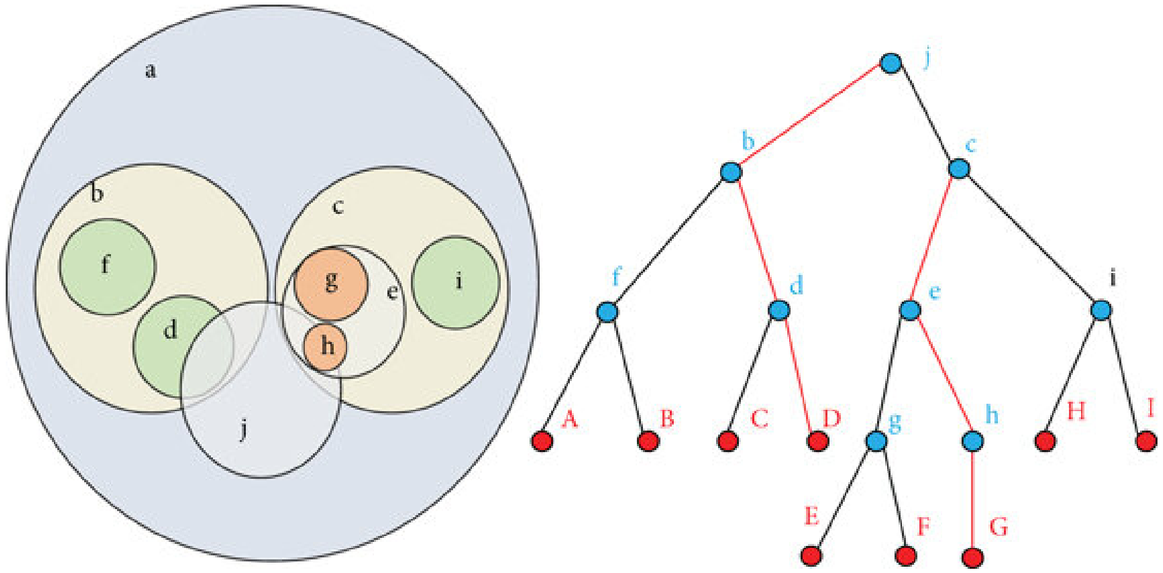
Un nodo del Ball-tree está definido por dos cosas:

$$B = \{c, r\}$$

$c \in \mathbb{R}^d$ centro del ball (centroide de los puntos que contiene).

$r \in \mathbb{R}$ radio mínimo que cubre todos los puntos:

$$r = \max_{x \in S} \|x - c\|$$



Ball-Tree

Construcción del árbol

Dado un conjunto de puntos $S \subset \mathbb{R}^d$

$$c = \frac{1}{|S|} \sum_{x \in S} x, \quad r = \max_{x \in S} \|x - c\|$$

Si $|S|$ es mayor que el tamaño mínimo de hoja (***leaf size***), dividir en dos subconjuntos:

- Escoger dos puntos alejados $a, b \in S$ (ej. el par más distante)
- Asignar cada punto al subconjunto más cercano:

$$S_a = \{x \in S : \|x - a\| \leq \|x - b\|\}, \quad S_b = \{x \in S : \|x - b\| < \|x - a\|\}$$

Ball-Tree

Búsqueda de vecino cercano

Supongamos que queremos encontrar el vecino más cercano de una consulta $q \in \mathbb{R}^d$

$$c = \frac{1}{|S|} \sum_{x \in S} x, \quad r = \max_{x \in S} \|x - c\|$$

Paso 1: Distancia mínima a un ball

$$d(q, B) = \max\left(0, \|q - c\| - r\right)$$

Si q está dentro de la esfera → $d(q, B) = 0$
Si está afuera → es la distancia del punto al borde de la esfera.

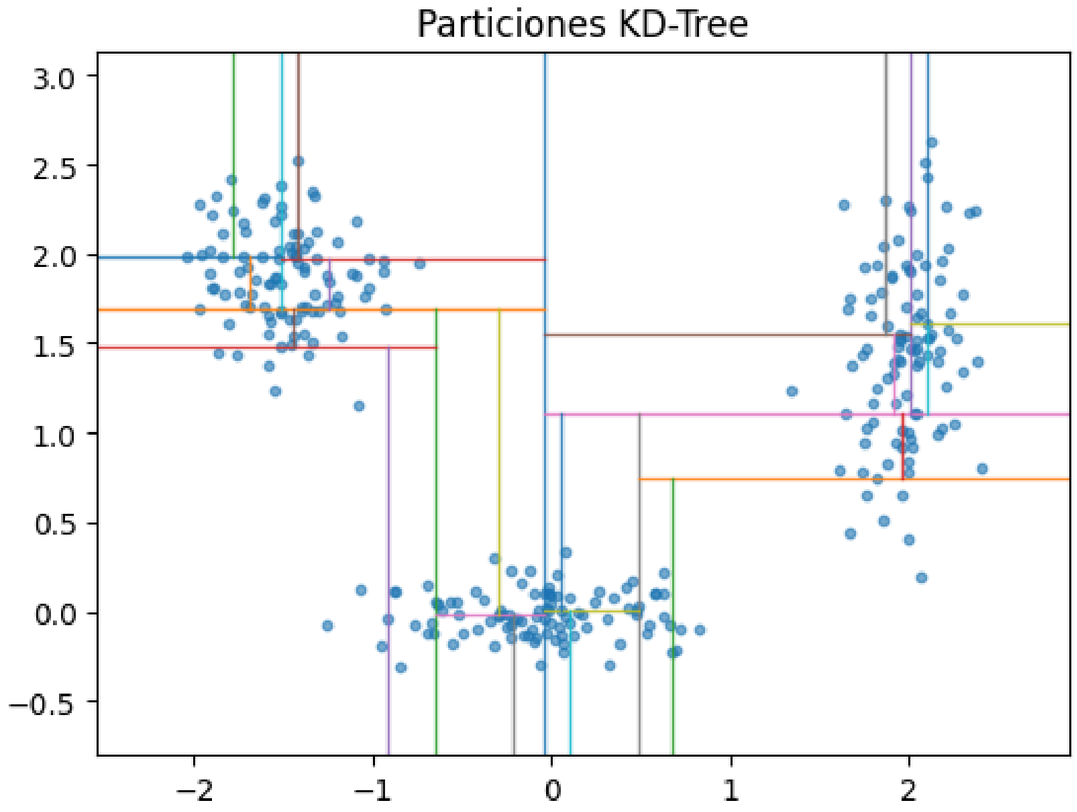
Paso 2: Poda de subárboles

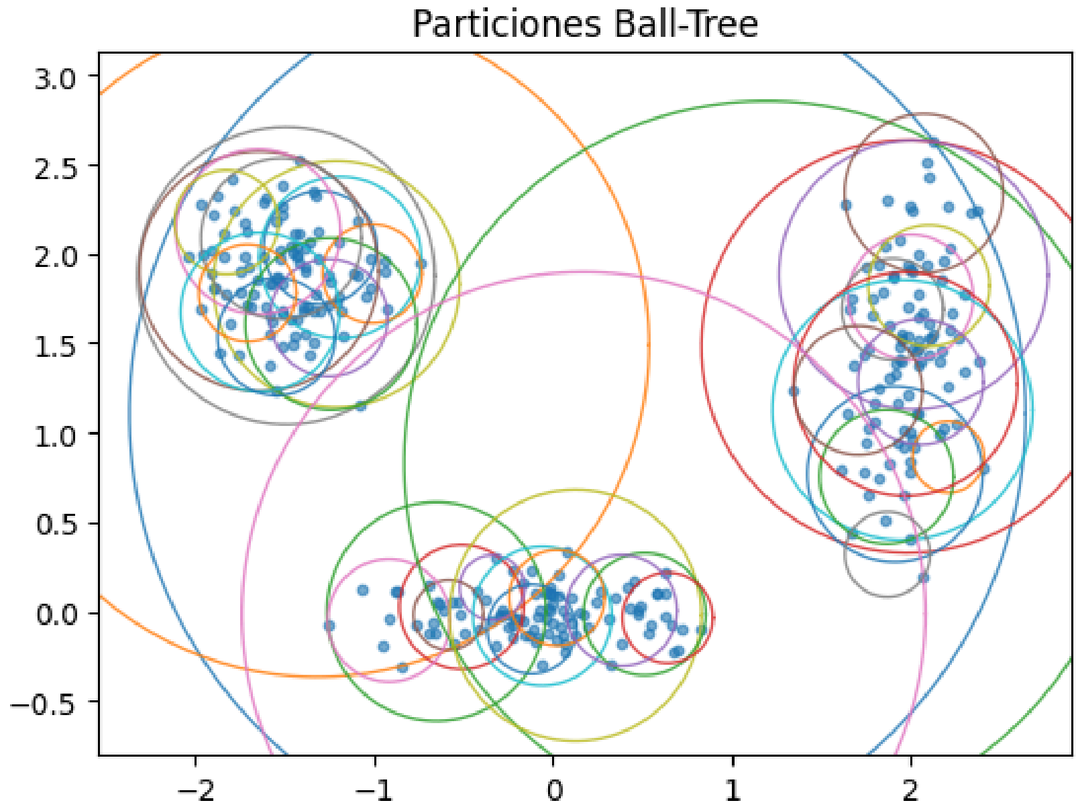
Si ya tenemos un mejor candidato con distancia d^*, y para un ball:

$$d(q, B) \geq d^*$$

podemos descartar todo el ball, porque ningún punto en él puede estar más cerca que d^*

Particiones

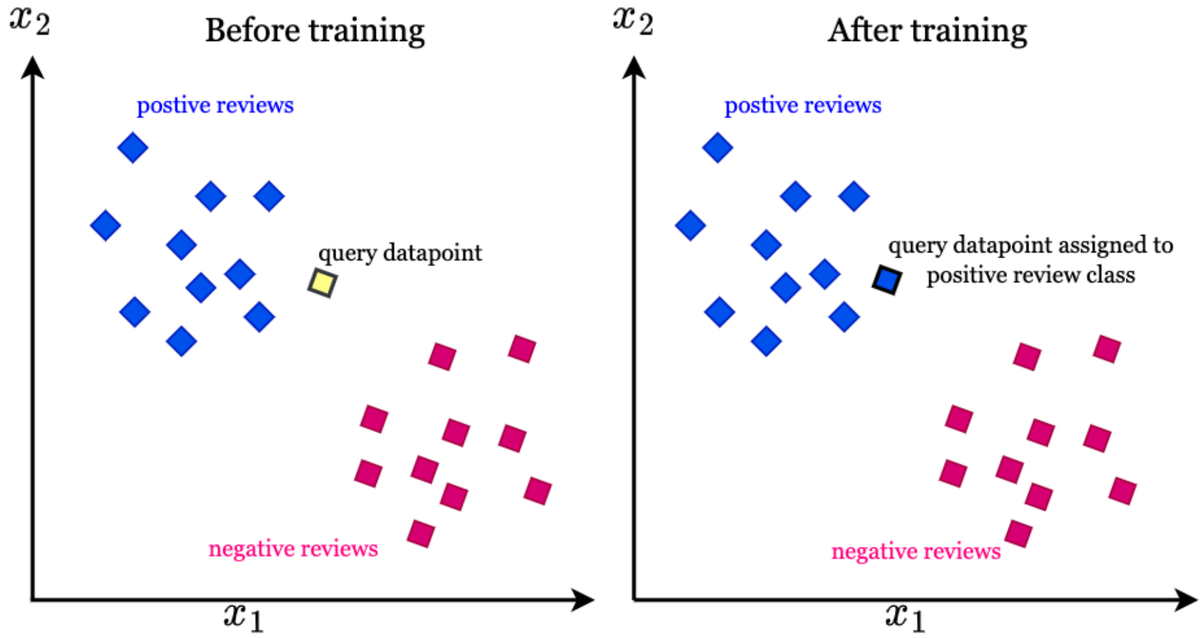




K-NN: Clasificación

Cuando se quiere **clasificar o predecir un valor** para un nuevo punto:

1. Se consideran los datos de entrenamiento (ya etiquetados).
2. Para el punto nuevo, se calcula la distancia a todos los puntos de entrenamiento (euclídea, Manhattan, coseno, etc.).
3. Seleccionar los **k** más cercanos.
4. Decidir según esos vecinos:
    - Clasificación: **votación por mayoría** → la clase más repetida entre los k vecinos.



K-NN: Clasificación

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

# 1) Dataset
X, y = make_moons(n_samples=1500, noise=0.35, random_state=42)

# 2) Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y)

# 3) Estandarización
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std  = scaler.transform(X_test)

# 5) Entrenar k=3
knn = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
knn.fit(X_train_std, y_train)

# 6) Evaluación
y_pred = knn.predict(X_test_std)
y_proba = knn.predict_proba(X_test_std)[:,1]

print("Reporte de clasificación:")
print(classification_report(y_test, y_pred, digits=2))

print("Matriz de confusión:")
print(confusion_matrix(y_test, y_pred))
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       150
           1       0.86      0.85      0.86       150

    accuracy                           0.86       300
   macro avg       0.86      0.86      0.86       300
weighted avg       0.86      0.86      0.86       300

Matriz de confusión:
[[130  20]
 [ 22 128]]
```

Métricas de Clasificación

Métricas Clasificación

Accuracy: mide la proporción de predicciones correctas sobre el total.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Qué tanto acerté en general

Precision: indica qué proporción de los predichos como positivos realmente lo son.

$$\text{Precision} = \frac{TP}{TP + FP}$$

de lo que predije como X, ¿cuánto realmente era X?

Recall: indica qué proporción de los positivos reales fueron correctamente detectados.

$$\text{Recall} = \frac{TP}{TP + FN}$$

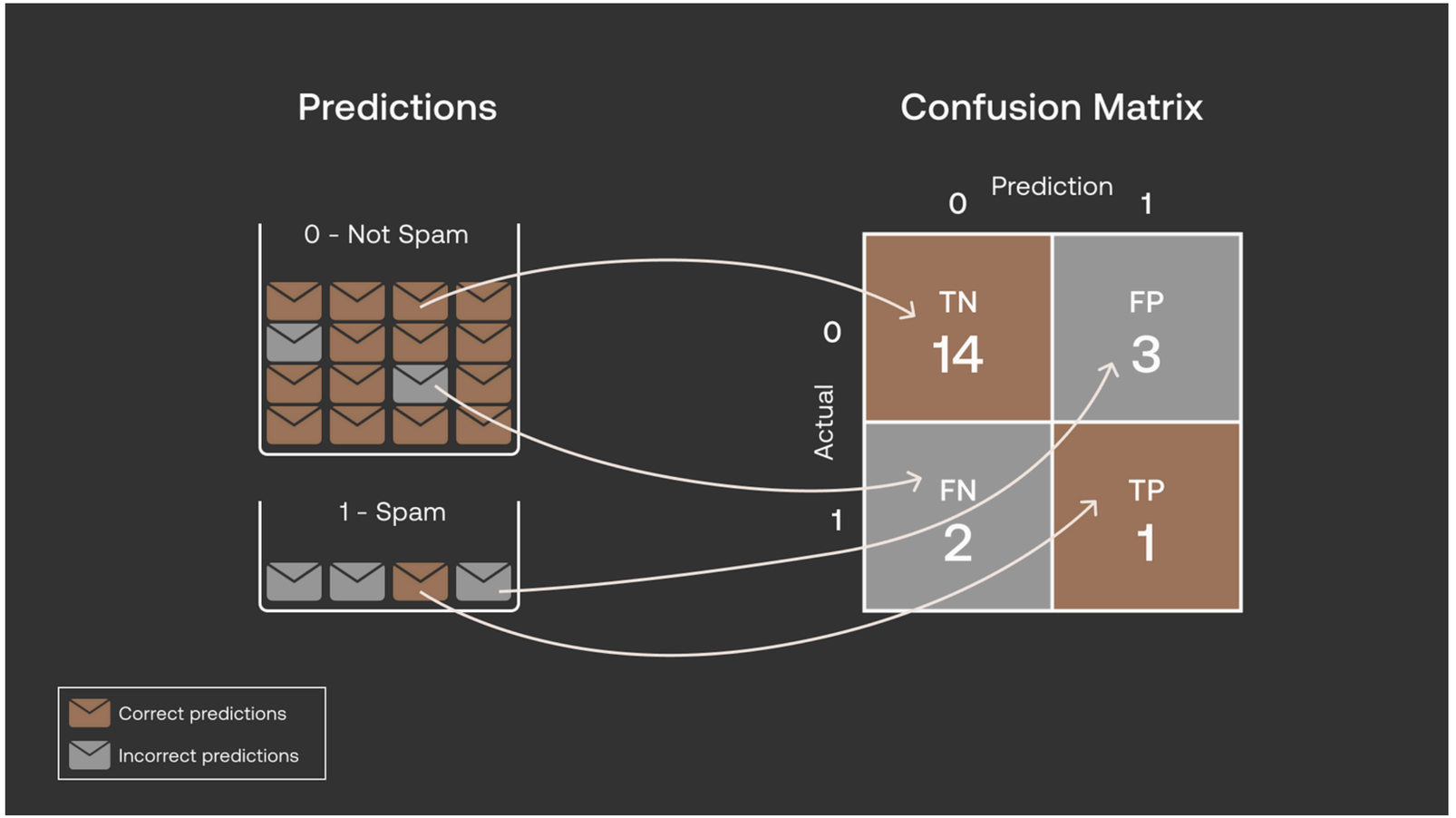
de lo que realmente era X, ¿cuánto detecté?

F1-score: es la media armónica entre precisión y recall.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

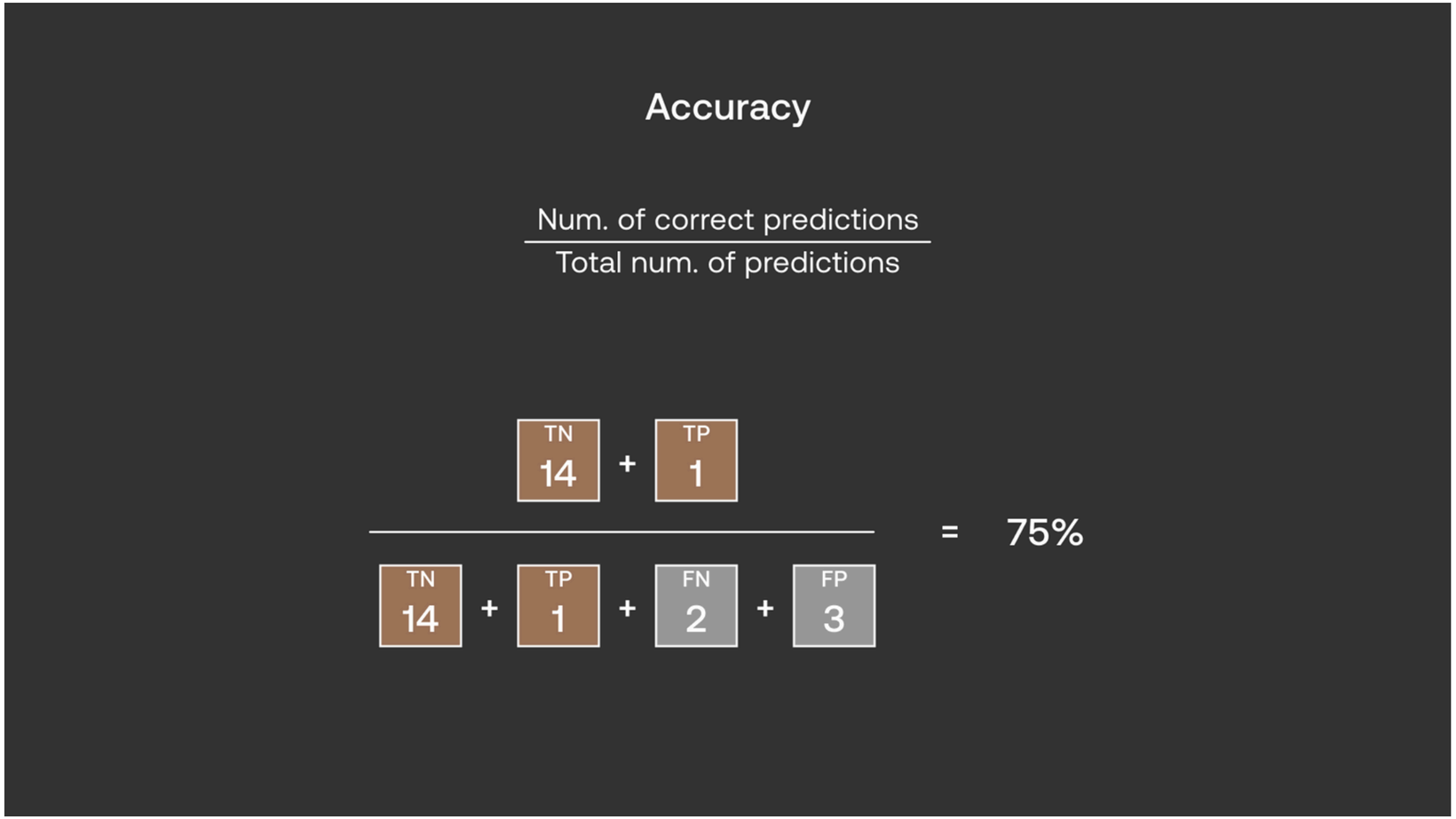
equilibrio entre ambos.

Métricas Clasificación

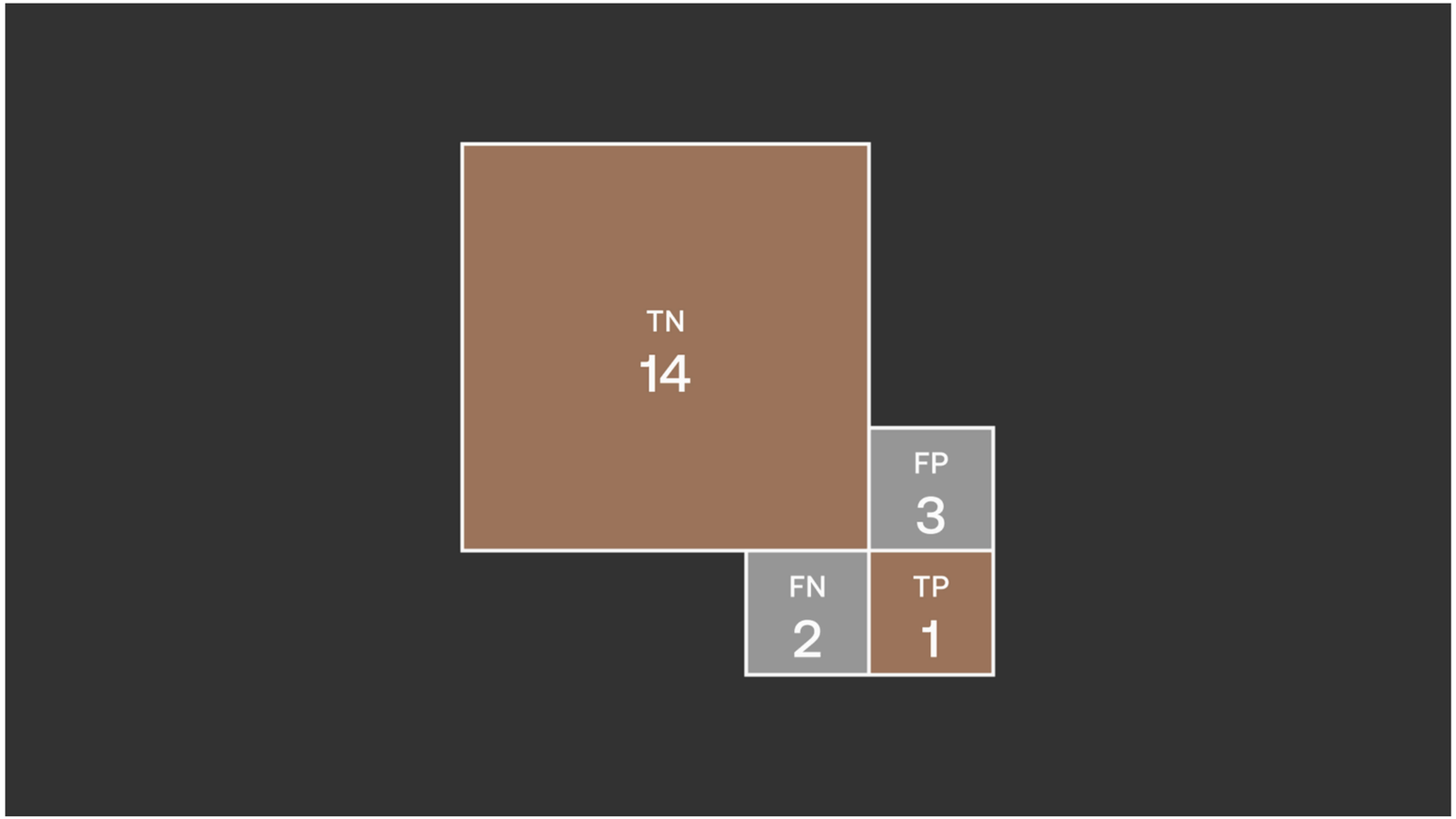


Las siguientes imágenes fueron sacadas de este link: Classification Metrics Explained: Accuracy, Precision, Recall & F1 Score

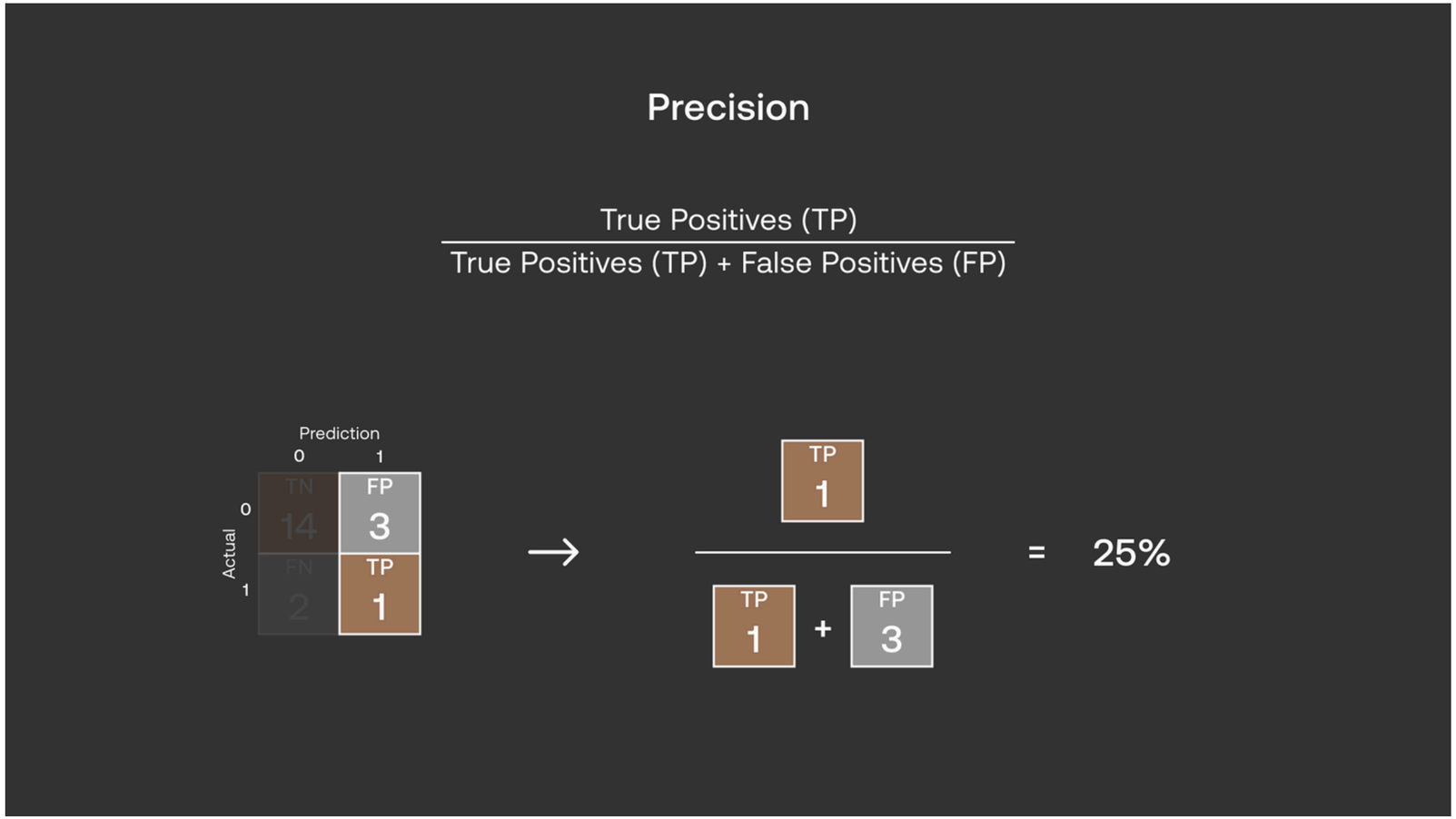
Métricas Clasificación



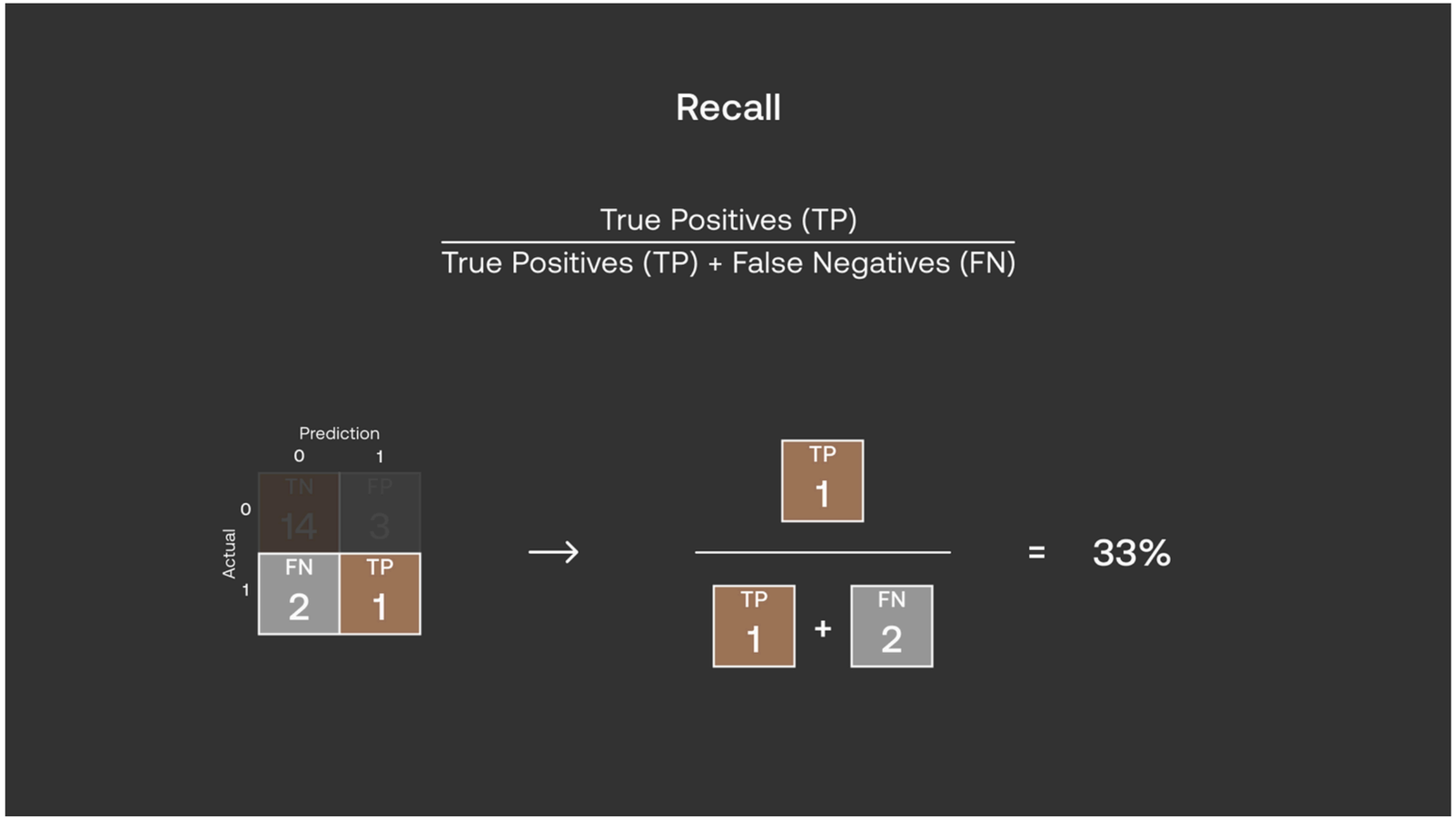
Métricas Clasificación



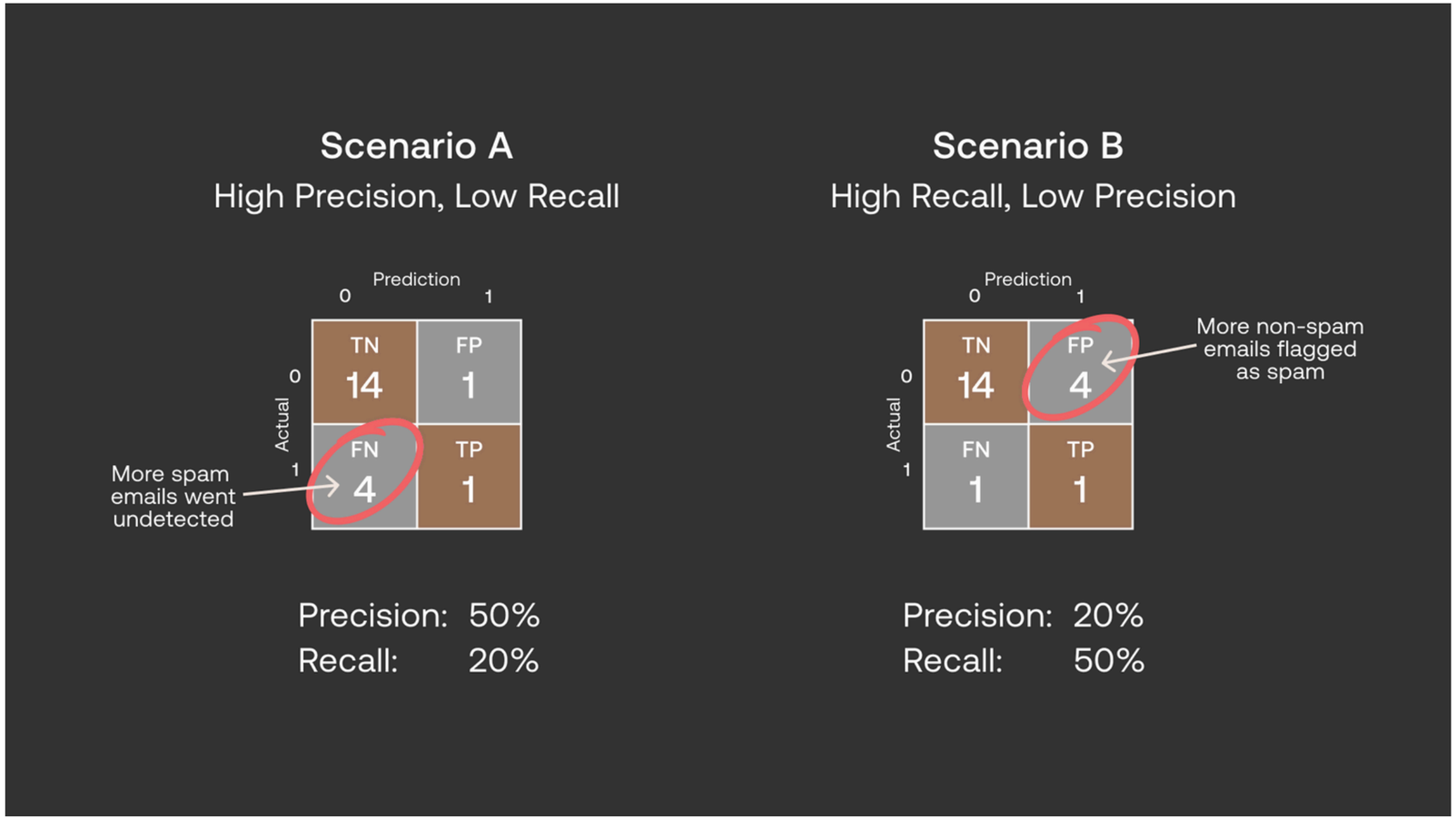
Métricas Clasificación



Métricas Clasificación



Métricas Clasificación



Métricas Clasificación

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 25\% * 33\%}{25\% + 33\%} = 28\%$$

Cross-Validation

Es una técnica para **evaluar el desempeño real** de un modelo de machine learning, reduciendo el riesgo de que los resultados dependan demasiado de la **partición de los datos** en train/test.