Practicando con el modelo: K vecinos más próximos (k-nearest neighbors K-NN)

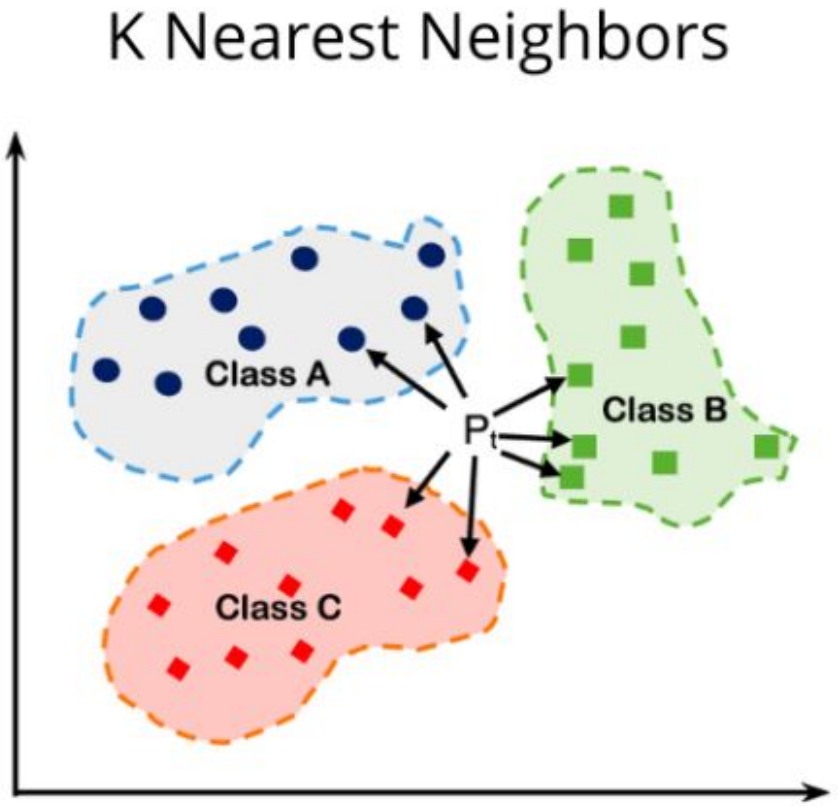
iesperemariaorts

Sistemas de aprendizaje automático

Profesor:
David Campoy Miñarro

¿Qué es K vecinos más próximos?

Algoritmo supervisado no paramétrico empleado para **tareas de clasificación** basado en casos parecidos sin que sea necesario que haya una concordancia exacta con los datos del conjunto de entrenamiento, de forma que el algoritmo clasifica el nuevo elemento en el conjunto que le corresponde buscando en las observaciones más cercanas.



Si bien se puede usar para problemas de regresión o clasificación, **generalmente se usa como un algoritmo de clasificación**, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.

¿Cómo funciona?

El objetivo del algoritmo del vecino más cercano es identificar los vecinos más cercanos de un punto de consulta dado, de modo que podamos asignar una etiqueta de clase a ese punto.

Fórmula de distancia euclidiana

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

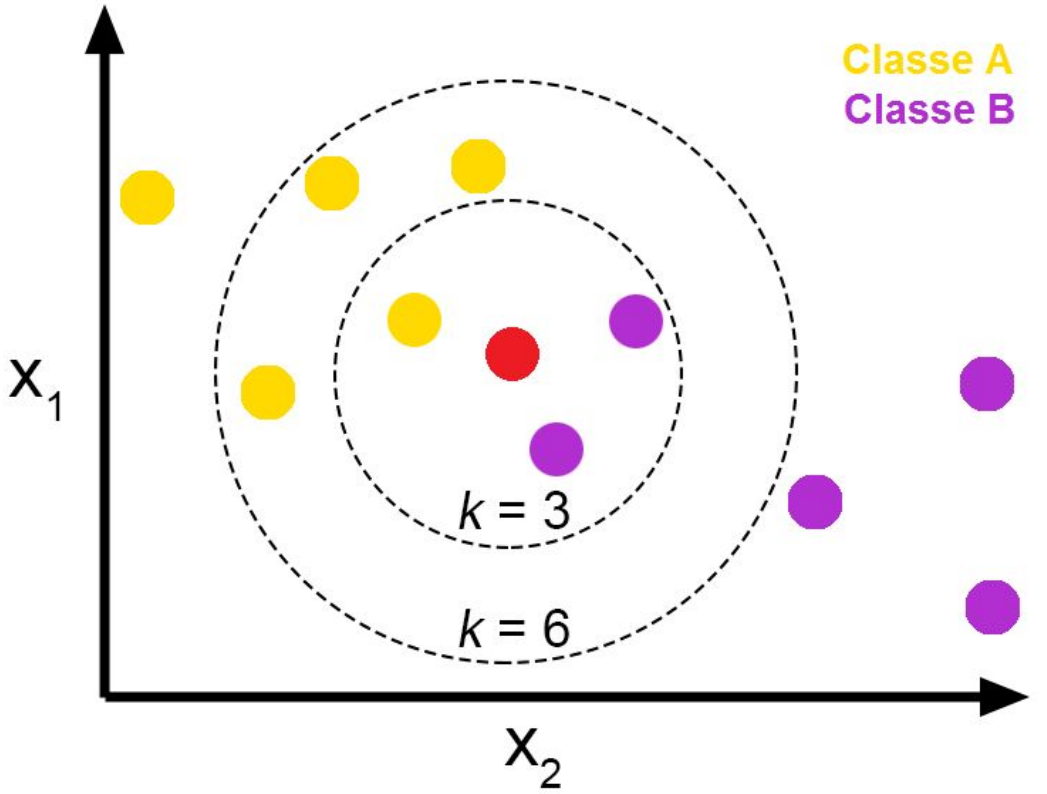
Fórmula de la distancia de Manhattan

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m}\left|x_i - y_i\right|\right)$$

Hiperparámetro K

El valor k en el algoritmo k-NN define cuántos vecinos se verificarán para determinar la clasificación de un punto de consulta específico.

La elección de k dependerá en gran medida de los datos de entrada, ya que los datos con más valores atípicos o ruido probablemente funcionarán mejor con valores más altos de k.



Se recomienda tener un **número impar para k** para evitar empates en la clasificación

```
# Crear y entrenar un modelo de k-NN con k=5
knn = KNeighborsClassifier(n_neighbors=5)
```

Aplicaciones de k-NN en *machine learning*

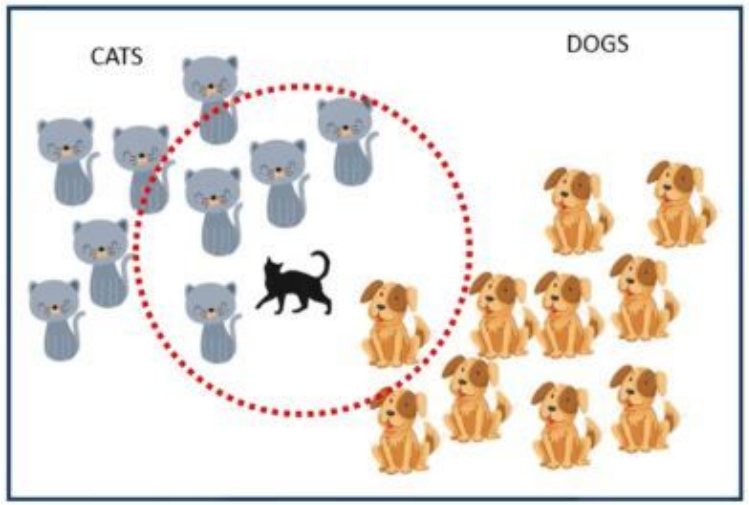
Preprocesamiento de datos: Los conjuntos de datos suelen tener valores faltantes, pero el algoritmo KNN puede estimar esos valores en un proceso conocido como imputación de datos faltantes.

Motores de recomendación: utilizando datos clics de sitios web, el algoritmo KNN se ha utilizado para proporcionar recomendaciones automáticas a los usuarios sobre contenido adicional.

Finanzas: También se ha utilizado en una variedad de casos de uso económico y financiero.

Salud: KNN se ha aplicado dentro de la industria de la salud, haciendo predicciones sobre el riesgo de ataques cardíacos y cáncer de próstata.

Reconocimiento de patrones: KNN también ha ayudado a identificar patrones, como en texto y clasificación de dígitos. Esto ha sido particularmente útil para identificar números escritos a mano que puede encontrar en formularios o sobres de correo.



Ventajas y desventajas del algoritmo KNN

Fácil de implementar: Dada la simplicidad y precisión del algoritmo.

Se adapta fácilmente: A medida que se añaden nuevas muestras de entrenamiento, el algoritmo se ajusta para tener en cuenta cualquier dato nuevo, ya que todos los datos de entrenamiento se almacenan en la memoria.

Pocos hiperparámetros: KNN solo requiere un valor k y una métrica de distancia, que es baja en comparación con otros algoritmos de *machine learning*.

No escala bien: Ocupa más memoria y almacenamiento de datos en comparación con otros clasificadores. Esto puede ser costoso desde una perspectiva de tiempo y dinero.

La maldición de la dimensionalidad: No funciona bien con entradas de datos de alta dimensión.

Propenso al sobreajuste: Los valores más bajos de k pueden sobreajustar los datos, mientras que los valores más altos de k tienden a "suavizar" los valores de predicción, ya que están promediando los valores en un área o vecindario más grande. Sin embargo, si el valor de k es demasiado alto, entonces puede ajustarse mal a los datos.

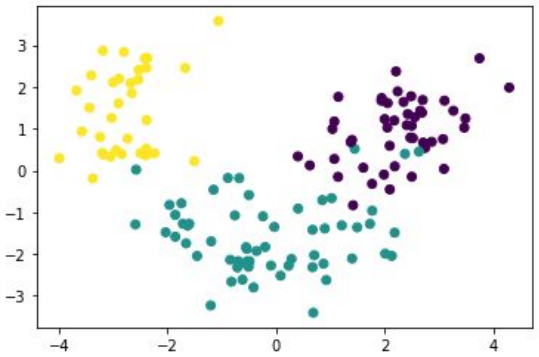
LDA versus K-NN

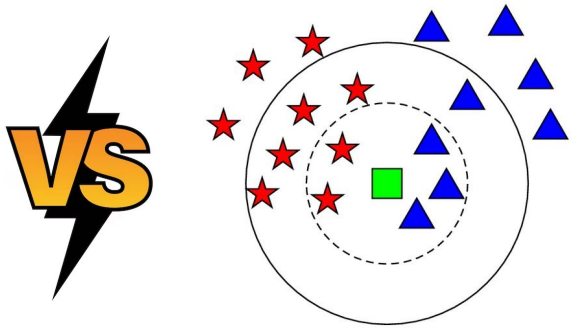
Tamaño del conjunto de datos:
- K-NN: más lento con muchos datos.
- LDA: puede ser más eficiente con grandes cantidades de datos.

Complejidad del modelo y suposiciones:
- K-NN: Funciona bien en datos no lineales y complejos.
- LDA: Asume distribuciones específicas y puede funcionar mejor cuando esas suposiciones son ciertas.

Preprocesamiento y dimensionalidad:
- K-NN: sensible a la escala y la dimensionalidad de datos.
- LDA: proyecta los datos en un espacio de dimensionalidad reducida manteniendo la estructura discriminativa.

Interpretación y eficiencia:
- K-NN: puede ser más complicado de interpretar debido a su naturaleza basada en la vecindad.
- LDA: facilitar la interpretación mediante la reducción de la dimensionalidad.



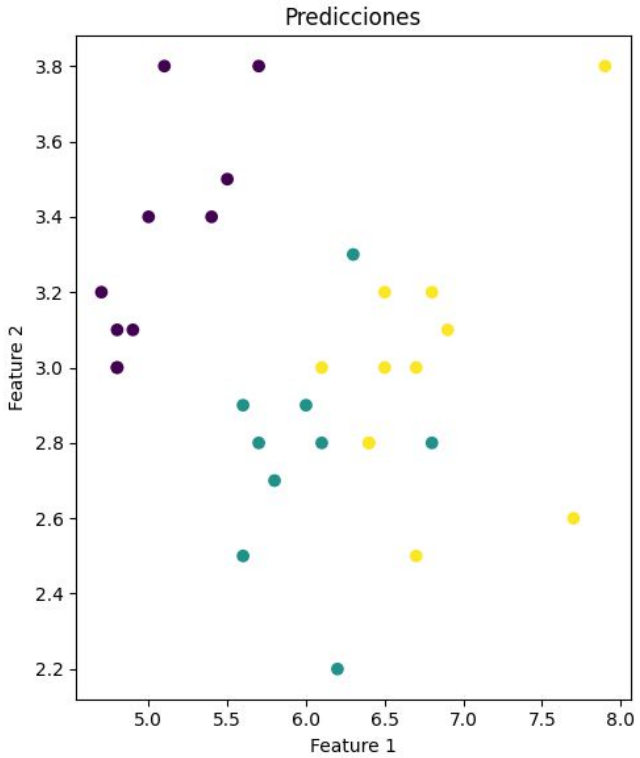


Actividad 1.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar el dataset Iris
iris = load_iris()
X = iris.data  # Características
y = iris.target  # Etiquetas

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Crear el clasificador KNN (K=1, por ejemplo)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Entrenar el clasificador con los datos de entrenamiento
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = knn_classifier.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo KNN con K={k}: {accuracy:.2f}")
```

Este programa utiliza el modelo K-NN para clasificar las flores del dataset Iris.

Hiperparámetro: K=3

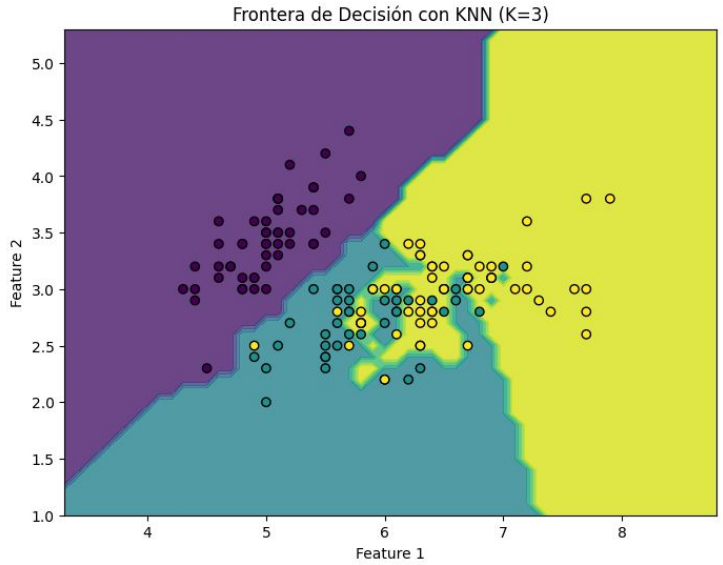


Actividad 2.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier

# Cargar el dataset Iris
iris = load_iris()
X = iris.data[:, :2]  # Utilizamos solo las dos primeras
características para visualización
y = iris.target

# Crear el clasificador KNN (K=3, por ejemplo)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Entrenar el clasificador con todos los datos
knn_classifier.fit(X, y)

# Crear una malla para el gráfico de la frontera de decisión
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
Z = knn_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Graficar la frontera de decisión y los puntos de datos
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Frontera de Decisión con KNN (K=3)')
plt.show()
```

Este código muestra cómo funciona el modelo K-NN a nivel visual.

Prueba con diferentes valores de K.



Actividad 3. K-NN en finanzas

El objetivo de este programa es aplicar el algoritmo de clasificación K-NN a un conjunto de datos de crédito alemán para predecir si un cliente será aprobado o no para un crédito.

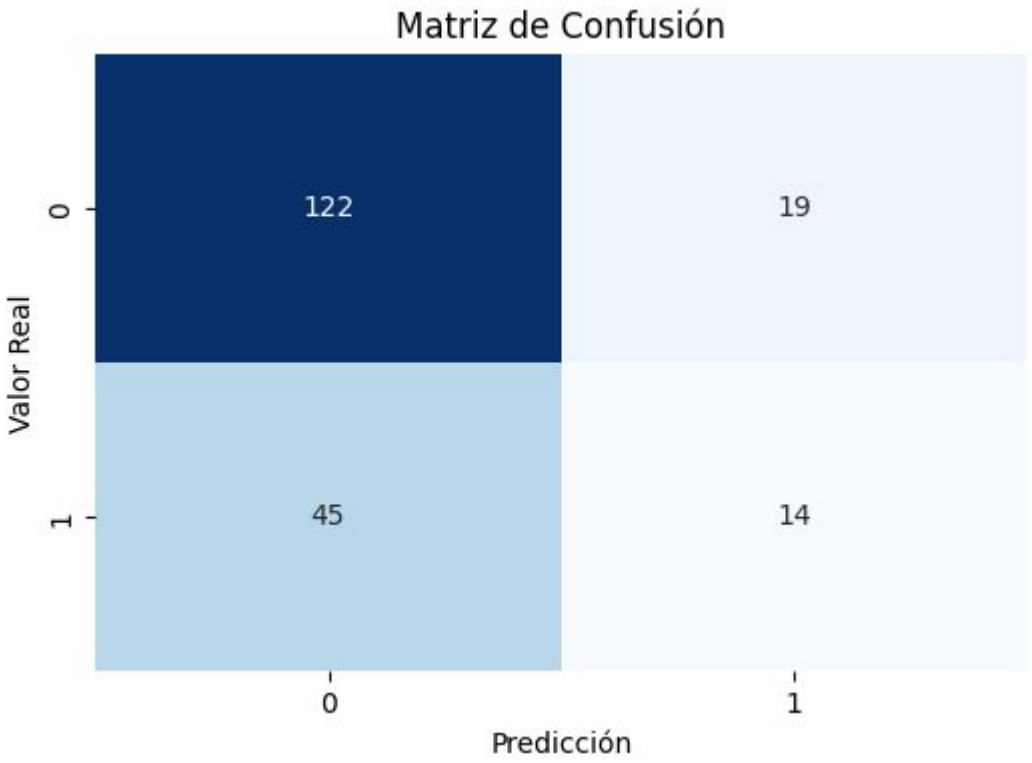
Datos de crédito alemán

`https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data`

```
A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 A121 67 A143 A152 2 A173 1 A192 A201 1
A12 48 A32 A43 5951 A61 A73 2 A92 A101 2 A121 22 A143 A152 1 A173 1 A191 A201 2
A14 12 A34 A46 2096 A61 A74 2 A93 A101 3 A121 49 A143 A152 1 A172 2 A191 A201 1
A11 42 A32 A42 7882 A61 A74 2 A93 A103 4 A122 45 A143 A153 1 A173 2 A191 A201 1
A11 24 A33 A40 4870 A61 A73 3 A93 A101 4 A124 53 A143 A153 2 A173 2 A191 A201 2
A14 36 A32 A46 9055 A65 A73 2 A93 A101 4 A124 35 A143 A153 1 A172 2 A192 A201 1
A14 24 A32 A42 2835 A63 A75 3 A93 A101 4 A122 53 A143 A152 1 A173 1 A191 A201 1
A12 36 A32 A41 6948 A61 A73 2 A93 A101 2 A123 35 A143 A151 1 A174 1 A192 A201 1
A14 12 A32 A43 3059 A64 A74 2 A91 A101 4 A121 61 A143 A152 1 A172 1 A191 A201 1
A12 30 A34 A40 5234 A61 A71 4 A94 A101 2 A123 28 A143 A152 2 A174 1 A191 A201 2
A12 12 A32 A40 1295 A61 A72 3 A92 A101 1 A123 25 A143 A151 1 A173 1 A191 A201 2
A11 48 A32 A49 4308 A61 A72 3 A92 A101 4 A122 24 A143 A151 1 A173 1 A191 A201 2
A12 12 A32 A43 1567 A61 A73 1 A92 A101 1 A123 22 A143 A152 1 A173 1 A192 A201 1
A11 24 A34 A40 1199 A61 A75 4 A93 A101 4 A123 60 A143 A152 2 A172 1 A191 A201 2
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Descargar el conjunto de datos de crédito alemán desde una URL
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data'
column_names = [
    "Status_of_existing_checking_account", "Duration_in_month", "Credit_history", "Purpose",
    "Credit_amount", "Savings_account", "Present_employment_since", "Installment_rate",
    "Personal_status_and_sex", "Other_debtors", "Present_residence_since", "Property",
    "Age", "Other_installment_plans", "Housing", "Number_of_existing_credits", "Job",
    "Number_of_people_being_liable", "Telephone", "Foreign_worker", "Credit_approval"
]

# Cargar el conjunto de datos en un DataFrame de Pandas
data = pd.read_csv(url, delimiter=' ', header=None, names=column_names)

# Convertir variables categóricas usando One-Hot Encoding
categorical_cols = [
    "Status_of_existing_checking_account", "Credit_history", "Purpose", "Savings_account",
    "Present_employment_since", "Personal_status_and_sex", "Other_debtors", "Property",
    "Other_installment_plans", "Housing", "Job", "Telephone", "Foreign_worker"
]
numeric_cols = [col for col in data.columns if col not in categorical_cols and col != "Credit_approval"]

# Aplicar One-Hot Encoding a las variables categóricas
ct = ColumnTransformer(
    [('one_hot_encoder', OneHotEncoder(), categorical_cols)],
    remainder='passthrough'
)
X = ct.fit_transform(data.drop('Credit_approval', axis=1))

# Separar características (X) y etiquetas (y)
y = data['Credit_approval']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Crear el clasificador KNN (K=5, por ejemplo)
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Entrenar el clasificador con los datos de entrenamiento
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = knn_classifier.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo KNN con K={k}: {accuracy:.2f}")

# Matriz de confusión para evaluar el rendimiento del modelo
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()
```



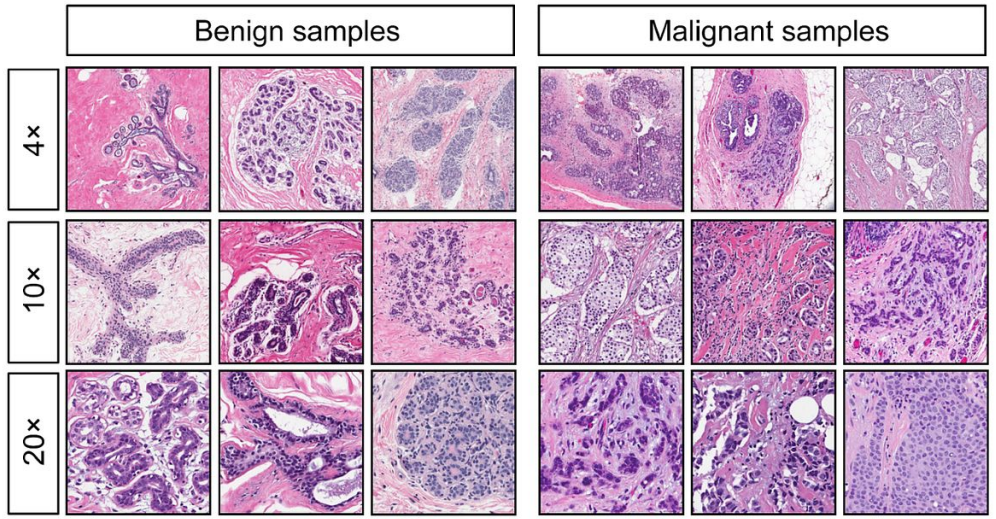
¿Qué te parecen los resultados?

Actividad 4. K-NN en salud

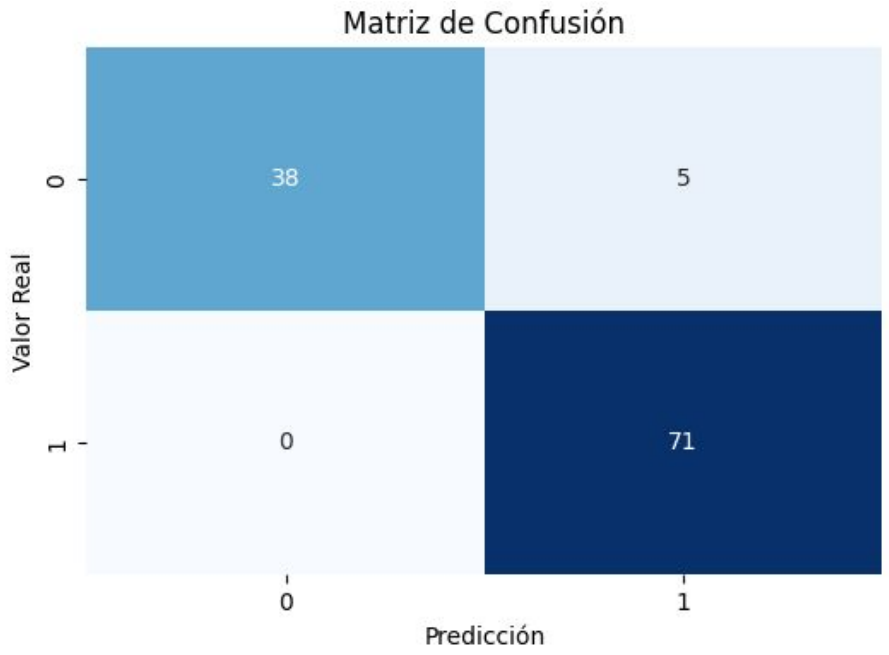
Averiguar si un paciente tiene cáncer basándonos en características médicas. Utilicemos el conjunto de datos "Breast Cancer Wisconsin (Diagnostic)" que contiene características extraídas de imágenes de tumores de mama.

```
https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic
```

El objetivo es predecir si un tumor es benigno o maligno basándonos en características extraídas de imágenes.



```python
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el conjunto de datos de cáncer de mama desde sklearn
data = load_breast_cancer()
X = data.data  # Características
y = data.target  # Etiquetas (0 para tumor benigno, 1 para tumor maligno)

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear el clasificador KNN (K=5, por ejemplo)
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Entrenar el clasificador con los datos de entrenamiento
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = knn_classifier.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo KNN con K={k}: {accuracy:.2f}")

# Matriz de confusión para evaluar el rendimiento del modelo
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()
```

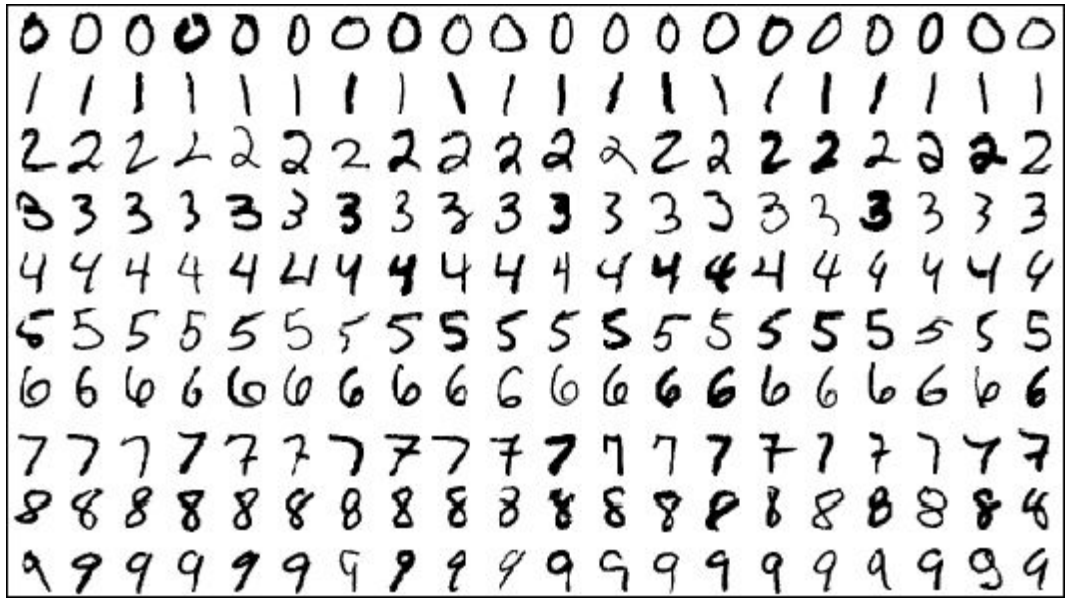


Actividad 5. K-NN en reconocimiento de patrones

Consideremos un conjunto de datos de dígitos escritos a mano. Usaremos el conjunto de datos MNIST, que contiene imágenes de dígitos escritos a mano del 0 al 9.

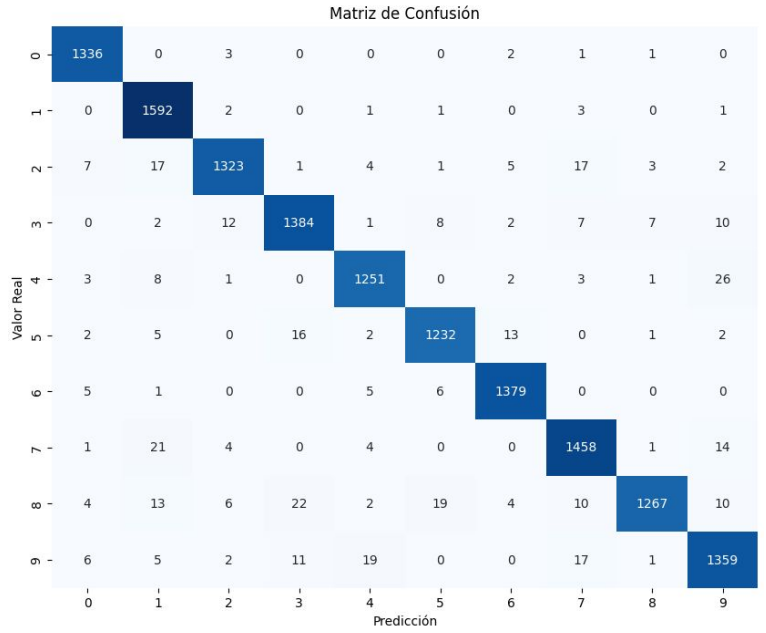
```
https://www.openml.org/search?type=data&sort=runs&id=554&status=active
```

El objetivo es reconocer dígitos escritos a mano correctamente basándonos en las imágenes proporcionadas en el conjunto de datos.



```python
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el conjunto de datos MNIST
mnist = fetch_openml('mnist_784')

# Separar características (X) y etiquetas (y)
X = mnist.data
y = mnist.target

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Crear el clasificador KNN (K=5, por ejemplo)
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Entrenar el clasificador con los datos de entrenamiento
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = knn_classifier.predict(X_test)
```

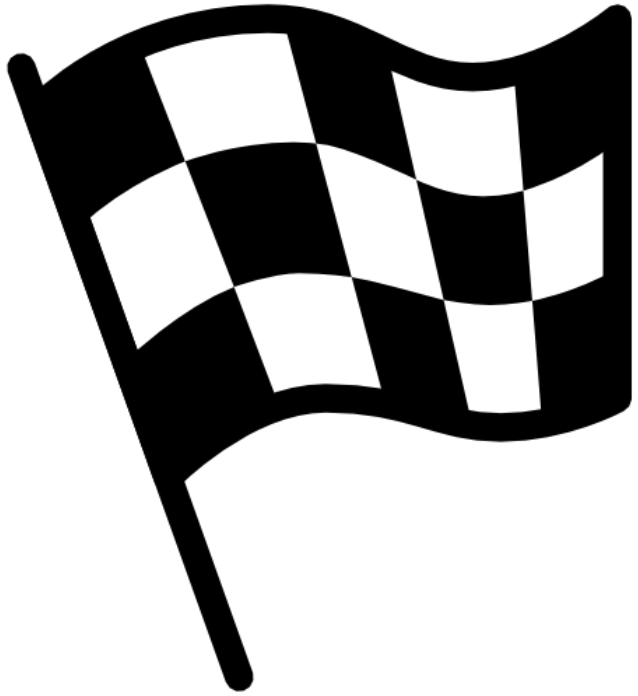
```python
# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo KNN con K={k}: {accuracy:.2f}")

# Matriz de confusión para evaluar el rendimiento del modelo
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicción')
plt.ylabel('Valor Real')
plt.title('Matriz de Confusión')
plt.show()
```



Comenta los resultados.

¿Qué has aprendido?

- Utilizar el modelo K vecinos más próximos.
- Diferencias entre LDA y K-NN.
- Utilizar K-NN para clasificar.
- Utilizar K-NN para buscar patrones.
- Utilizar K-NN en la medicina.
- ¿te parece poco?



"Nadie lo dice así, pero creo que la inteligencia artificial es casi una disciplina de humanidades. Es realmente un intento de entender la inteligencia humana y la cognición humana."

Sebastian Thrun - profesor de Inteligencia artificial en la Universidad de Stanford