

ACTIVIDAD 2.16 - Alvaro Fernandez BECERRA

En esta actividad se realizaban los siguientes estudios a partir del dataset de datos de automóviles utilizando un clasificador k-NN:

1. Predecir el precio del coche en función de sus características técnicas.
2. Predecir el nivel de riesgo (symboling) para una compañía de seguros.

Si nos enfocamos en el segundo, un problema de clasificación, lo podremos luego comparar luego siguiendo el mismo proceso pero usando datos normalizados.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import r2_score, accuracy_score, classification_report, confusion_matrix

# Cargar el dataset:
import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data"

columns = [
    "symboling", "normalized_losses", "make", "fuel_type", "aspiration",
    "num_doors", "body_style", "drive_wheels", "engine_location",
    "wheel_base", "length", "width", "height", "curb_weight",
    "engine_type", "num_cylinders", "engine_size", "fuel_system",
    "bore", "stroke", "compression_ratio", "horsepower",
    "peak_rpm", "city_mpg", "highway_mpg", "price"
]
df = pd.read_csv(url, header=None, names=columns, na_values='?')
```

```
# Podemos comprobar qué columnas contienen valores nulos.
df.isna().sum()
```

```
0
symboling      0
normalized_losses 0
wheel_base      0
length          0
width           0
...
make_saab       0
make_subaru     0
make_toyota     0
make_volkswagen 0
make_volvo      0
76 rows × 1 columns
dtype: int64
```

```
# Ahora vamos a convertir esos valores antes de sustituirlos por la media.
df['normalized_losses'] = pd.to_numeric(df['normalized_losses'], errors='coerce')
df['bore'] = pd.to_numeric(df['bore'], errors='coerce')
df['stroke'] = pd.to_numeric(df['stroke'], errors='coerce')
df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
df['peak_rpm'] = pd.to_numeric(df['peak_rpm'], errors='coerce')
df['price'] = pd.to_numeric(df['price'], errors='coerce')
```

```
# Ahora sí podemos sustituir los valores de estas columnas por un valor representativo, en este caso la media.
df['normalized_losses'] = df['normalized_losses'].fillna(df['normalized_losses'].mean())
df['bore'] = df['bore'].fillna(df['bore'].mean())
df['stroke'] = df['stroke'].fillna(df['stroke'].mean())
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].mean())
df['peak_rpm'] = df['peak_rpm'].fillna(df['peak_rpm'].mean())
df['price'] = df['price'].fillna(df['price'].mean())
```

```
# También liminamos filas que aún tengan nulos en columnas categóricas
df.dropna(inplace=True)

# Convertimos variables categóricas en variables dummy binarias
df = pd.get_dummies(df, columns=[
    'fuel_type', 'aspiration', 'num_doors', 'body_style',
    'drive_wheels', 'engine_location', 'engine_type',
    'num_cylinders', 'fuel_system', 'make'
])
```

2. Predecir el nivel de riesgo (symboling) para una compañía de seguros.

```

# Preparar datos para symboling
X_clf = df.drop('symboling', axis=1)
y_clf = df['symboling']

# Clasificador k-NN
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_clf, y_clf)
y_clf_pred = clf.predict(X_clf)

exactitud = accuracy_score(y_clf, y_clf_pred)
print(f"Exactitud: {exactitud:.4f}\n")

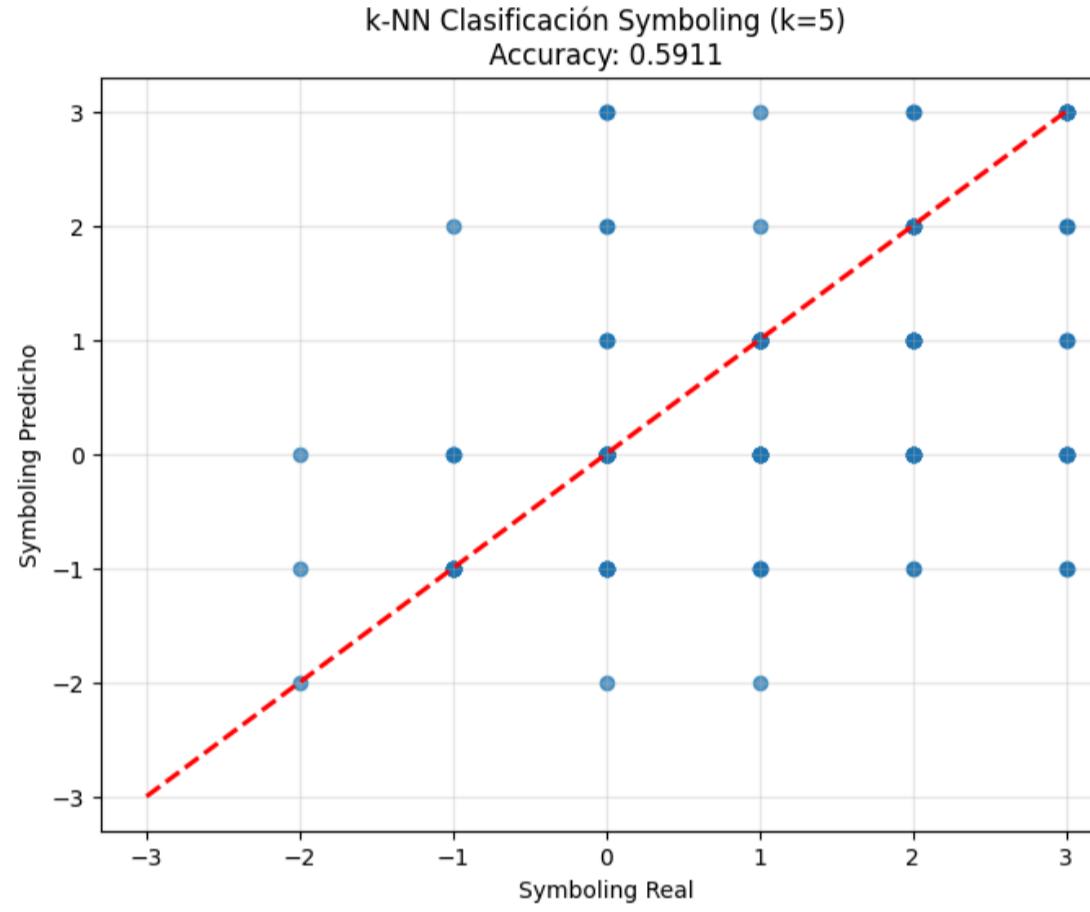
```

Exactitud: 0.5911

```

# Gráfica: Predicciones vs Reales
plt.figure(figsize=(8,6))
plt.scatter(y_clf, y_clf_pred, alpha=0.7)
plt.plot([-3,3], [-3,3], 'r--', lw=2)
plt.xlabel('Symboling Real')
plt.ylabel('Symboling Predicho')
plt.title(f'k-NN Clasificación Symboling (k=5)\nAccuracy: {exactitud:.4f}')
plt.grid(True, alpha=0.3)
plt.show()

```



ACTIVIDAD 2.16 - Álvaro Fernández Becerra

Hasta ahora en los ejemplos hemos utilizado un clasificador k-NN sin normalizar los datos numéricos. ¿Crees que esto puede haber afectado negativamente a los resultados obtenidos con el clasificador?

- La respuesta es que sí, ya que como hemos visto, si nos basamos en distancias entre observaciones, si algunos atributos se mueven en rangos de valores mucho mayores frente a otros más pequeños, estos atributos tendrán mucha prioridad en los cálculos haciendo que unos atributos tengan mucha importancia y otros casi ninguna.

Comprueba si tu respuesta es la correcta usando cualquiera de los ejemplos que hemos realizado anteriormente que utilizan clasificadores k-NN para el dataset de las flores iris y compara los resultados que se obtuvieron con los que se obtienen normalizando previamente los atributos. (automóviles en este caso)

Haz doble clic (o pulsa Intro) para editar

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# ---SYMBOLING SIN NORMALIZAR

X_clf = df.drop('symboling', axis=1)
y_clf = df['symboling']

clasificador_sin_norm = KNeighborsClassifier(n_neighbors=5)
clasificador_sin_norm.fit(X_clf, y_clf)
y_clf_pred_sin = clasificador_sin_norm.predict(X_clf)

accuracy_sin_norm = accuracy_score(y_clf, y_clf_pred_sin)

# ---SYMBOLING CON NORMALIZACION

# Seleccion de columnas numéricas
X_clf_num = X_clf.select_dtypes(include=['number'])

# Normalización
X_clf_num_norm = (X_clf_num - X_clf_num.min()) / (X_clf_num.max() - X_clf_num.min())

clasificador_con_norm = KNeighborsClassifier(n_neighbors=5)
clasificador_con_norm.fit(X_clf_num_norm, y_clf)

```

```

y_clf_pred_con = clasificador_con_norm.predict(X_clf_num_norm)

accuracy_con_norm = accuracy_score(y_clf, y_clf_pred_con)

# RESULTADOS
print(f"Accuracy SIN normalizar      : {accuracy_sin_norm:.4f}")
print(f"Accuracy CON normalización   : {accuracy_con_norm:.4f}")

```

```

Accuracy SIN normalizar      : 0.5911
Accuracy CON normalización   : 0.7685

```

- Despues de hacer la comprobación podemos ver como claramente la exactitud del modelo cambia al escalar los datos. Obteniendo datos más precisos y menos dispersos

```

# Gráfico SIN normalizar
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.scatter(y_clf, y_clf_pred_sin, alpha=0.6, edgecolors='k')
plt.plot([y_clf.min(), y_clf.max()], [y_clf.min(), y_clf.max()], 'r--', lw=2)
plt.xlabel('Symboling real')
plt.ylabel('Symboling predicho')
plt.title(f'SIN normalizar\nAccuracy = {accuracy_sin_norm:.4f}')
plt.grid(alpha=0.3)

# Gráfico CON normalización
plt.subplot(1,2,2)
plt.scatter(y_clf, y_clf_pred_con, alpha=0.6, edgecolors='k')
plt.plot([y_clf.min(), y_clf.max()], [y_clf.min(), y_clf.max()], 'r--', lw=2)
plt.xlabel('Symboling real')
plt.ylabel('Symboling predicho')
plt.title(f'CON normalización\nAccuracy = {accuracy_con_norm:.4f}')
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()

```

