Mise à l'échelle

CSI4506 Introduction à l'intelligence artificielle

Marcel Turcotte

2025-10-05

Scénario

Nous prétendons prédire le prix d'une maison en utilisant la **régression des k-plus proches voisins (KNN)** avec deux attributs :

- x_1 : nombre de chambres (petite échelle)
- x_2 : superficie en pieds carrés (grande échelle)

Nous créons trois exemples **a**, **b**, **c** choisis de manière à ce que :

- **Sans mise à l'échelle**, **a** soit plus proche de **b** (car la superficie en pieds carrés domine).
- **Avec mise à l'échelle (score-z)**, **a** devienne plus proche de **c** (la différence de chambres importe après la remise à l'échelle).

Données (trois maisons)

```
import numpy as np
import pandas as pd

# Trois exemples (pièces, sqft); prix uniquement pour b et c (entraînement)
point_names = ["a", "b", "c"]
X = np.array([
    [4, 1500.0],  # a (requête)
    [8, 1520.0],  # b (entraînement)
    [4, 1300.0],  # c (entraînement)
], dtype=float)

prices = pd.Series([np.nan, 520_000, 390_000], index=point_names, name="prix")
```

```
df = pd.DataFrame(X, columns=["pièces", "sqft"], index=point_names)
display(df)
display(prices.to_frame())
```

	pièces	sqft
a	4.0	1500.0
b	8.0	1520.0
c	4.0	1300.0

	prix
a	NaN
b	520000.0
c	390000.0

Remarque. Nous traiterons **b** et **c** comme l'ensemble d'entraînement, et **a** comme la requête dont nous voulons prédire le prix.

Distances euclidiennes (non normalisées)

La distance euclidienne (au carré) entre u et v est

$$\|u - v\|_2^2 = \sum_j (u_j - v_j)^2.$$

Lorsqu'un attribut a une échelle beaucoup plus grande (par exemple, la superficie en pieds carrés), elle peut dominer la somme.

```
from sklearn.metrics import pairwise_distances

dist_unscaled = pd.DataFrame(
    pairwise_distances(df.values, metric="euclidean"),
    index=df.index, columns=df.index
)
dist_unscaled
```

	a	b	c
a	0.000000	20.396078	200.000000
b	20.396078	0.000000	220.036361
c	200.000000	220.036361	0.000000

```
print("Le plus proche de 'a' (non normalisé) :", dist_unscaled.loc["a"].drop("a").idxmin())
```

```
Le plus proche de 'a' (non normalisé) : b
```

Attente : **a** est le plus proche de **b** (des pieds carrés similaires l'emportent sur les pièces).

Mise à l'échelle appropriée pour la modélisation (ajuster le scaler sur l'ensemble d'entraînement)

Pour un flux de travail ML équitable, calculez les paramètres d'échelle sur les données d'entraînement (**b**, **c**) uniquement, puis transformez à la fois l'entraînement et la requête :

$$z(x) = \frac{x - \mu_{\text{train}}}{\sigma_{\text{train}}}.$$

```
from sklearn.preprocessing import StandardScaler

train_idx = ["b", "c"]
query_idx = ["a"]

scaler = StandardScaler()

scaler.fit(df.loc[train_idx])     # ajuster uniquement sur les points d'entraînement

Z = pd.DataFrame(
    scaler.transform(df),
    columns=df.columns, index=df.index
)

Z
```

	pièces	sqft
a	-1.0	0.818182

	pièces	sqft
b	1.0	1.000000
c	-1.0	-1.000000

Distances euclidiennes (après mise à l'échelle)

```
dist_scaled = pd.DataFrame(
    pairwise_distances(Z.values, metric="euclidean"),
    index=Z.index, columns=Z.index
)
dist_scaled
```

	a	b	c
a	0.000000	2.008247	1.818182
b	2.008247	0.000000	2.828427
c	1.818182	2.828427	0.000000

```
print("Le plus proche de 'a' (échelle ajustée) :", dist_scaled.loc["a"].drop("a").idxmin())
```

```
Le plus proche de 'a' (échelle ajustée) : c
```

Maintenant : **a** est le plus proche de **c** (la différence de pièces est importante lorsque les attributs sont sur des échelles comparables).

Régression KNN : inversion dans la prédiction

Nous exécuterons un régressseur 1-NN (donc la prédiction est exactement le prix du voisin le plus proche) **avec et sans** mise à l'échelle.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

X_train = df.loc[train_idx].values          # b, c
y_train = prices.loc[train_idx].values      # prix pour b, c
X_query = df.loc[query_idx].values          # a
```

```
# 1) Sans mise à l'échelle
knn_plain = KNeighborsRegressor(n_neighbors=1, metric="euclidean")
knn_plain.fit(X_train, y_train)
pred_plain = knn_plain.predict(X_query)[0]

# 2) Avec mise à l'échelle (pipeline ajuste le scaler uniquement sur l'entrainement, puis KN
knn_scaled = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsRegressor(n_neighbors=1, metric="euclidean"))
])
knn_scaled.fit(X_train, y_train)
pred_scaled = knn_scaled.predict(X_query)[0]

pd.DataFrame(
    {
        "prédiction (sans mise à l'échelle)": [pred_plain],
        "prédiction (avec mise à l'échelle)": [pred_scaled],
        "voisin le plus proche (sans mise à l'échelle)": [point_names[1] if pred_plain==price
        "voisin le plus proche (avec mise à l'échelle)": [point_names[1] if pred_scaled==pric
    },
    index=["a"]
)
```

	prédiction (sans mise à l'échelle)	prédiction (avec mise à l'échelle)	voisin le plus proche (sans mise à l'éc
a	520000.0	390000.0	b

À retenir :

- **Non échelonné :** a b prédiction **520 000 $**
- **Échelonné :** a c prédiction **390 000 $**

Même modèle et données ; simplement **l'échelle des attributs** a changé le voisin—et la prédiction.

Pourquoi cela se produit

- La distance euclidienne (au carré) agrège les différences au carré par attribut :

$$|u - v|_2^2 = \sum_j (u_j - v_j)^2.$$

— Un attribut à grande échelle (par exemple, **sqft**) peut éclipser des attributs à petite échelle (par exemple, **chambres**), de sorte que KNN "ignore" effectivement les dimensions à plus petite échelle.
— La **standardisation** (scores z) ou la **normalisation min-max** met les dimensions sur un pied d'égalité comparable.
— **Règle empirique :** Pour les méthodes basées sur la distance (KNN, k-means, noyaux RBF, etc.), toujours **normaliser** les attributs.

Montrer uniquement la distance aux voisins

Distances de **a** à {**b**, **c**} avant et après normalisation.

```
def show_pair(name_from, names_to, D):
    return D.loc[name_from, names_to].to_frame("distance")

print("Distances non normalisées de a → {b,c}")
display(show_pair("a", ["b", "c"], dist_unscaled))

print("Distances normalisées de a → {b,c}")
display(show_pair("a", ["b", "c"], dist_scaled))
```

```
Distances non normalisées de a → {b,c}
```

	distance
b	20.396078
c	200.000000

```
Distances normalisées de a → {b,c}
```

	distance
b	2.008247
c	1.818182

Passer à la distance de Manhattan ?

Même avec la distance L_1, l'échelle reste importante :

$$|u - v|_1 = \sum_j |u_j - v_j|.$$

Essayez de remplacer `metric="euclidean"` par `metric="manhattan"`—vous verrez la même sensibilité à l'échelle des attributs.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

X_train = df.loc[train_idx].values          # b, c
y_train = prices.loc[train_idx].values      # prix pour b, c
X_query = df.loc[query_idx].values          # a

# 1) Sans normalisation
knn_plain = KNeighborsRegressor(n_neighbors=1, metric="manhattan")
knn_plain.fit(X_train, y_train)
pred_plain = knn_plain.predict(X_query)[0]

# 2) Avec normalisation (le pipeline ajuste le scaler uniquement sur l'entraînement, puis KNN
knn_scaled = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsRegressor(n_neighbors=1, metric="manhattan"))
])
knn_scaled.fit(X_train, y_train)
pred_scaled = knn_scaled.predict(X_query)[0]

pd.DataFrame(
    {
        "prédiction (sans normalisation)": [pred_plain],
        "prédiction (avec normalisation)": [pred_scaled],
        "voisin le plus proche (sans normalisation)": [point_names[1] if pred_plain==prices[
        "voisin le plus proche (avec normalisation)": [point_names[1] if pred_scaled==prices
    },
    index=["a"]
)
```

	prédiction (sans normalisation)	prédiction (avec normalisation)	voisin le plus proche (sans normalisation
a	520000.0	390000.0	b

TL ;DR

— Les **modèles basés sur la distance** sont très sensibles aux **échelles des attributs**.
— **Toujours normaliser** vos entrées (ajuster le scaler uniquement sur l'ensemble **d'entraînement**).
— La normalisation peut **changer les voisins les plus proches** et donc **changer les prédictions**—comme vu ici avec la régression 1-NN.