Scaling

CSI4106 Introduction to Artificial Intelligence

Marcel Turcotte

2025-10-05

Scenario

We pretend to predict a house price using **k-Nearest Neighbors (KNN) regression** with two features:

- x_1: number of rooms (small scale)
- x_2: square footage (large scale)

We create three examples **a**, **b**, **c** chosen so that:

- **Without scaling**, **a** is closer to **b** (because square footage dominates).
- **With scaling (z-score)**, **a** becomes closer to **c** (rooms difference matters after rescaling).

Data (three houses)

```
import numpy as np
import pandas as pd

# Three examples (rooms, sqft); prices only for b and c (training)
point_names = ["a", "b", "c"]
X = np.array([
    [4, 1500.0],  # a (query)
    [8, 1520.0],  # b (train)
    [4, 1300.0],  # c (train)
], dtype=float)

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

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

	rooms	sqft
a	4.0	1500.0
b	8.0	1520.0
c	4.0	1300.0

	price
a	NaN
b	520000.0
c	390000.0

Note. We'll treat **b** and **c** as the training set, and **a** as the query whose price we want to predict.

Euclidean distances (unscaled)

The (squared) Euclidean distance between u and v is

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

When one feature has a much larger scale (e.g., square footage), it can dominate the sum.

```
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

	a	b	c
c	200.000000	220.036361	0.000000

```
print("Nearest to 'a' (unscaled):", dist_unscaled.loc["a"].drop("a").idxmin())
```

```
Nearest to 'a' (unscaled): b
```

Expectation: **a** is nearest to **b** (similar sqft overwhelms rooms).

Proper scaling for modeling (fit scaler on the training set)

For a fair ML workflow, compute scaling parameters on the training data (**b**, **c**) only, then transform both train and query:

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

```
from sklearn.preprocessing import StandardScaler

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

scaler = StandardScaler()

scaler.fit(df.loc[train_idx])     # fit only on training points

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

Z
```

	rooms	sqft
a	-1.0	0.818182
b	1.0	1.000000
c	-1.0	-1.000000

Euclidean distances (after scaling)

```
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("Nearest to 'a' (scaled):", dist_scaled.loc["a"].drop("a").idxmin())
```

```
Nearest to 'a' (scaled): c
```

Now: **a** is nearest to **c** (rooms difference matters once features are on comparable scales).

KNN regressor: flip in the prediction

We'll run a 1-NN regressor (so the prediction is exactly the nearest neighbor's price) **with and without** scaling.

```
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      # prices for b, c
X_query = df.loc[query_idx].values          # a

# 1) No scaling
knn_plain = KNeighborsRegressor(n_neighbors=1, metric="euclidean")
knn_plain.fit(X_train, y_train)
pred_plain = knn_plain.predict(X_query)[0]

# 2) With scaling (pipeline fits scaler only on training, then KNN on scaled)
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(
    {
        "prediction (no scaling)": [pred_plain],
        "prediction (with scaling)": [pred_scaled],
        "nearest neighbor (no scaling)": [point_names[1] if pred_plain==prices['b'] else poi
        "nearest neighbor (with scaling)": [point_names[1] if pred_scaled==prices['b'] else p
    },
    index=["a"]
)
```

	prediction (no scaling)	prediction (with scaling)	nearest neighbor (no scaling)	nearest neighbor (with
a	520000.0	390000.0	b	c

Takeaway:

- **Unscaled:** a b prediction **$520,000**
- **Scaled:** a c prediction **$390,000**

Same model and data; just **feature scale** changed the neighbor—and the prediction.

Why this happens

- (Squared) Euclidean distance aggregates per-feature squared differences:

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

- A large-scale feature (e.g., **sqft**) can dwarf small-scale features (e.g., **rooms**), so KNN effectively "ignores" the smaller-scale dimensions.
- **Standardization** (z-scores) or **min-max scaling** puts dimensions on comparable footing.
- **Rule of thumb:** For distance-based methods (KNN, k-means, RBF kernels, etc.), always **scale** features.

Show the distance to neighbors only

Distances from **a** to {**b**, **c**} before and after scaling.

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

print("Unscaled distances from a → {b,c}")
display(show_pair("a", ["b", "c"], dist_unscaled))

print("Scaled distances from a → {b,c}")
display(show_pair("a", ["b", "c"], dist_scaled))
```

```
Unscaled distances from a → {b,c}
```

	distance
b	20.396078
c	200.000000

```
Scaled distances from a → {b,c}
```

	distance
b	2.008247
c	1.818182

Switch to Manhattan distance?

Even with L_1 distance, scale still matters:

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

Try replacing `metric="euclidean"` with `metric="manhattan"`—you'll see the same sensitivity to feature scale.

```
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      # prices for b, c
X_query = df.loc[query_idx].values          # a

# 1) No scaling
knn_plain = KNeighborsRegressor(n_neighbors=1, metric="manhattan")
knn_plain.fit(X_train, y_train)
pred_plain = knn_plain.predict(X_query)[0]

# 2) With scaling (pipeline fits scaler only on training, then KNN on scaled)
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(
    {
        "prediction (no scaling)": [pred_plain],
        "prediction (with scaling)": [pred_scaled],
        "nearest neighbor (no scaling)": [point_names[1] if pred_plain==prices['b'] else poi
        "nearest neighbor (with scaling)": [point_names[1] if pred_scaled==prices['b'] else p
    },
    index=["a"]
)
```

	prediction (no scaling)	prediction (with scaling)	nearest neighbor (no scaling)	nearest neighbor (with
a	520000.0	390000.0	b	c

TL;DR

- **Distance-based models** are highly sensitive to **feature scales**.
- **Always scale** your inputs (fit the scaler on the **training** set only).
- Scaling can **change nearest neighbors** and therefore **change predictions**—as seen here with 1-NN regression.