k-Nearest Neighbors: Theory and Practice

September 10, 2025

1 Introduction

k-Nearest Neighbors (k-NN) is a non-parametric, instance-based learner: predictions are made by looking up the closest training samples. It is simple and often competitive on low-dimensional, well-scaled data, but can be sensitive to feature scaling and suffers in high dimensions.

2 Theory and Formulas

Given a query $\mathbf{x}$, find its k nearest neighbors under a distance metric $d(\cdot, \cdot)$ (e.g., Euclidean, Manhattan). For classification, predict by majority vote among the labels of these neighbors; with distance weighting (`weights=distance`), closer neighbors have higher influence. For regression, predict the average (or distance-weighted average) of neighbor targets.

Computationally, naive search costs $\mathcal{O}(nd)$ per query with n samples and d features. Tree-based indices (KDTree/BallTree) can accelerate queries in moderate dimensions. k-NN is affected by the curse of dimensionality; proper feature scaling and metric choice are critical.

3 Applications and Tips

- **Choose k:** tune via cross-validation; odd k helps avoid ties in binary classification.
- **Scaling:** standardize features or use pipelines; scale-sensitive.
- **Metric:** try Euclidean vs Manhattan; consider domain-specific distances.
- **Weights:** `uniform` vs `distance`; weighting can help with class overlap.
- **Complexity:** prediction cost grows with data size; consider approximate neighbors for large datasets.

4 Python Practice

Run the script in this chapter directory to generate figures into `figures/`.

Listing 1: Generate k-NN figures

```
python gen_knn_figures.py
```

Listing 2: gen_knn_figures.py

```python
"""
Figure generator for the k-NN chapter.

Generates illustrative figures and saves them into the chapter's 'figures/'
folder next to this script, regardless of current working directory.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn

Install (if needed):
  pip install numpy matplotlib scikit-learn

This script avoids newer or experimental APIs for broader compatibility.
"""
from __future__ import annotations

import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    from sklearn.datasets import make_moons, make_regression,
        make_classification
    from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
    from sklearn.preprocessing import StandardScaler
    from sklearn.pipeline import make_pipeline
    from sklearn.model_selection import cross_val_score
except Exception:
    raise SystemExit(
        "Missing scikit-learn. Please install with: pip install scikit-learn"
    )


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD."""
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400)
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
```

```python
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).size)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k", s=20)
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")


def fig_knn_k_compare(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=0)
    models = [
        (KNeighborsClassifier(n_neighbors=1), "k=1 (high variance)"),
        (KNeighborsClassifier(n_neighbors=15), "k=15 (smoother)")
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"k-NN: {title}")
    fig.suptitle("Effect of k on decision boundary")
    out_path = os.path.join(out_dir, "knn_k_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_knn_metric_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=500, noise=0.28, random_state=1)
    models = [
        (KNeighborsClassifier(n_neighbors=11, metric="euclidean"), "metric=euclidean"),
        (KNeighborsClassifier(n_neighbors=11, metric="manhattan"), "metric=manhattan"),
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"k-NN: {title}")
    fig.suptitle("Effect of distance metric")
    out_path = os.path.join(out_dir, "knn_metric_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_knn_scaling_effect(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_classification(
```

```
        n_samples=600,
        n_features=2,
        n_informative=2,
        n_redundant=0,
        n_clusters_per_class=1,
        class_sep=1.0,
        random_state=2,
    )
    # Impose different scales on features
    X_scaled_variance = X.copy()
    X_scaled_variance[:, 0] *= 8.0  # make feature 0 dominate distances

    knn_raw = KNeighborsClassifier(n_neighbors=11)
    knn_std = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=11))

    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    knn_raw.fit(X_scaled_variance, y)
    _plot_decision_boundary(axes[0], knn_raw, X_scaled_variance, y, "Without scaling")
    knn_std.fit(X_scaled_variance, y)
    _plot_decision_boundary(axes[1], knn_std, X_scaled_variance, y, "With StandardScaler")
    fig.suptitle("Feature scaling impact on k-NN")
    out_path = os.path.join(out_dir, "knn_scaling_effect.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_knn_weight_compare(out_dir: str) -> str:
    np.random.seed(3)
    X, y = make_moons(n_samples=500, noise=0.32, random_state=3)
    models = [
        (KNeighborsClassifier(n_neighbors=11, weights="uniform"), "weights=uniform"),
        (KNeighborsClassifier(n_neighbors=11, weights="distance"), "weights=distance"),
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        _plot_decision_boundary(ax, m, X, y, f"k-NN: {title}")
    fig.suptitle("Uniform vs distance weighting")
    out_path = os.path.join(out_dir, "knn_weight_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path
```

```python
def fig_knn_regression_curve(out_dir: str) -> str:
    rng = np.random.RandomState(4)
    # 1D regression: y = sin(x) + noise
    X = np.sort(rng.uniform(-3.0, 3.0, size=150)).reshape(-1, 1)
    y = np.sin(X).ravel() + rng.normal(scale=0.25, size=X.shape[0])

    grid = np.linspace(-3.5, 3.5, 600).reshape(-1, 1)
    models = [
        (KNeighborsRegressor(n_neighbors=1), "k=1"),
        (KNeighborsRegressor(n_neighbors=15), "k=15"),
        (KNeighborsRegressor(n_neighbors=45), "k=45"),
    ]
    fig, ax = plt.subplots(figsize=(7.5, 4.2), dpi=160)
    ax.scatter(X[:, 0], y, s=18, c="#555", alpha=0.7, label="data")
    colors = ["#E74C3C", "#3498DB", "#2ECC71"]
    for (m, title), col in zip(models, colors):
        m.fit(X, y)
        y_pred = m.predict(grid)
        ax.plot(grid[:, 0], y_pred, color=col, lw=2, label=title)
    ax.set_title("k-NN regression: smoothing vs k")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend()
    ax.grid(True, linestyle=":", alpha=0.4)
    out_path = os.path.join(out_dir, "knn_regression_curve.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_knn_k_compare,
        fig_knn_metric_compare,
        fig_knn_scaling_effect,
        fig_knn_weight_compare,
        fig_knn_regression_curve,
    ]
    print("Generating figures into:", os.path.abspath(out_dir))
    for gen in generators:
        try:
            p = gen(out_dir)
            print("Saved:", p)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)


if __name__ == "__main__":
    main()
```

5 Result

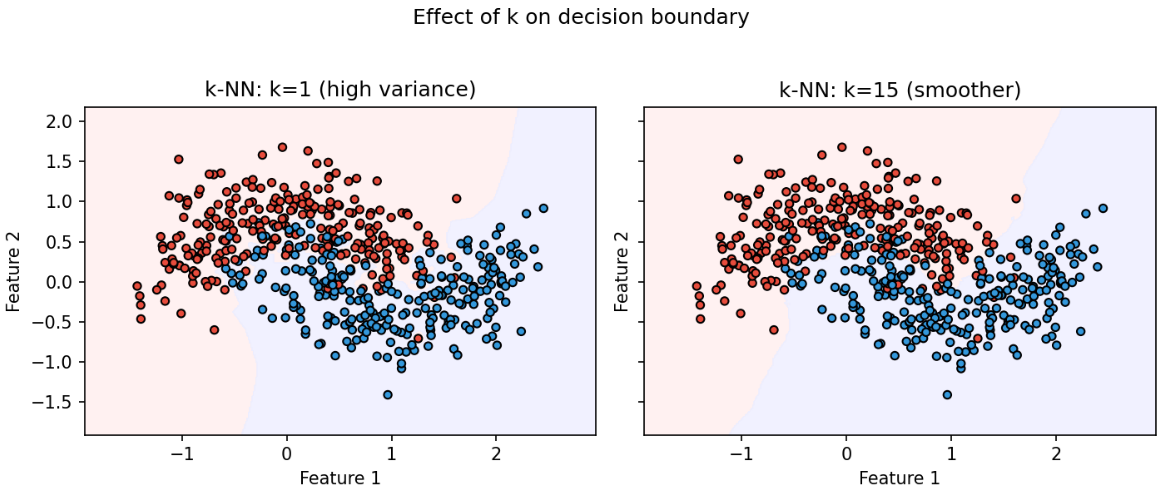

Figure 1: Decision boundaries for different k (1 vs 15).

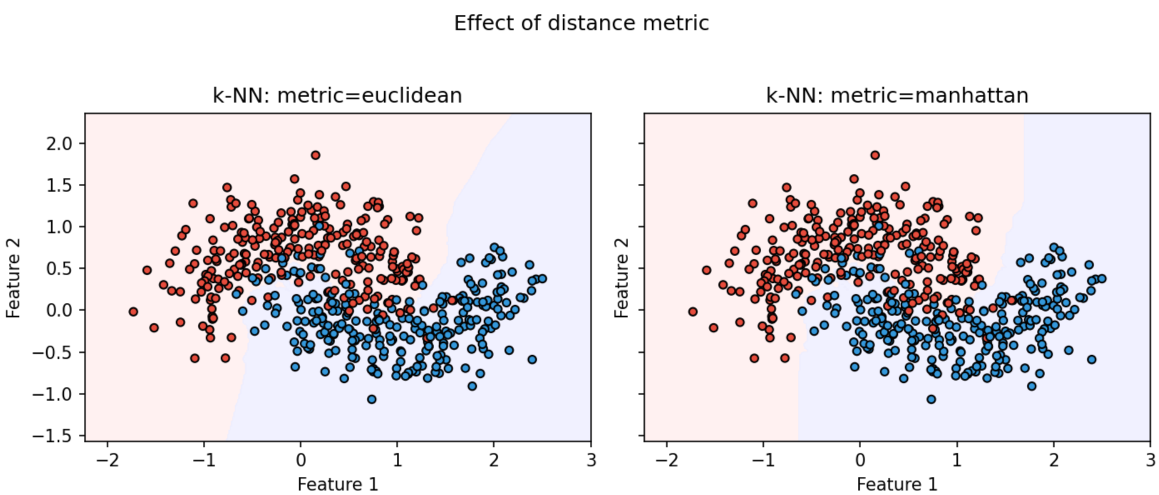

Figure 2: Effect of metric: Euclidean vs Manhattan.

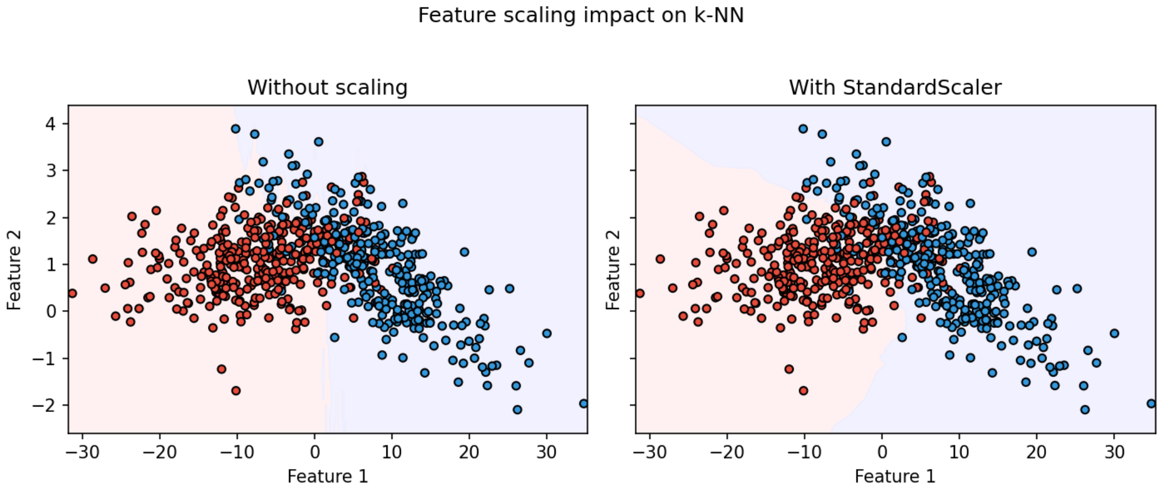

Figure 3: Impact of feature scaling on decision boundary.

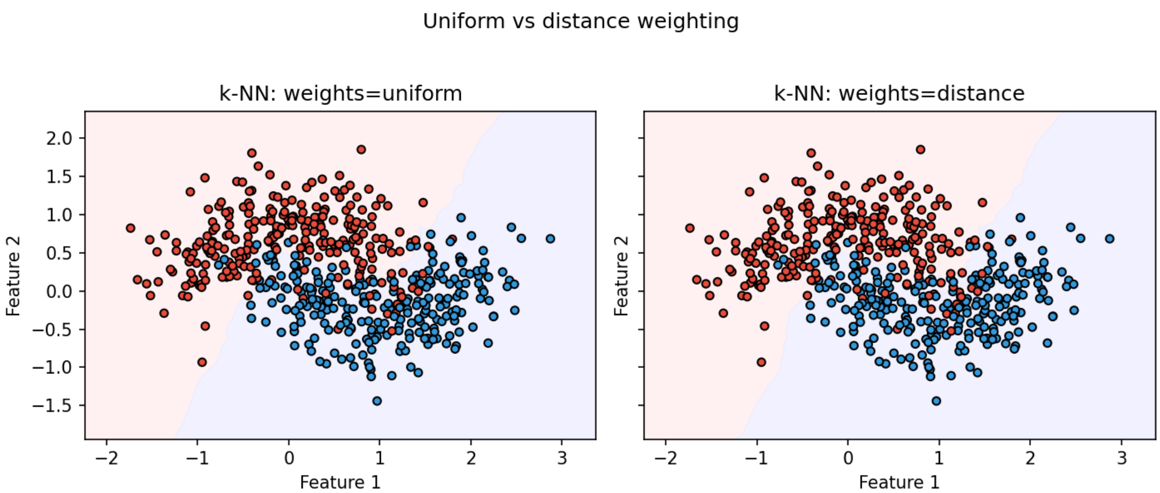

Figure 4: Uniform vs distance weighting in k-NN.

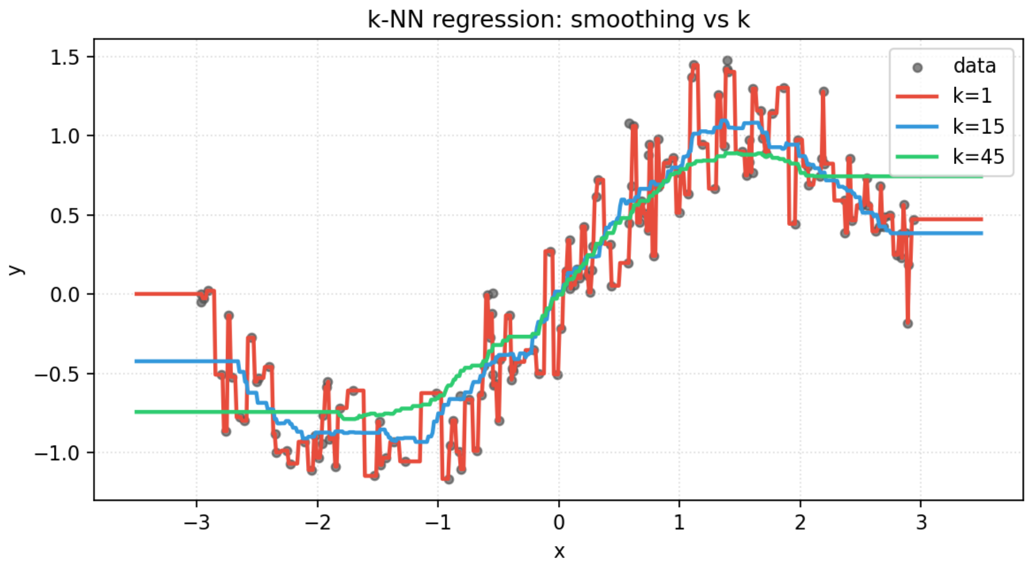

Figure 5: k-NN regression: smoothing effect as k increases.

6 Summary

k-NN is a simple yet powerful baseline for both classification and regression when features are well-scaled and dimensionality is moderate. Select k, metric, and weighting via validation, and use scaling to ensure distances are meaningful.