kNN

February 8, 2025

1 k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) is a simple, yet powerful, non-parametric algorithm used for both classification and regression. Its basic principle is based on the idea that similar data points exist in close proximity. In this section, we will explain k-NN in detail, including its mathematical foundations, algorithmic steps, and practical considerations such as distance metrics and the choice of k.

1.1 Basic Principle

The k-NN algorithm works under the assumption that data points that are close together in feature space tend to belong to the same class (for classification) or have similar values (for regression). Given a new query point x_q, the algorithm finds the k data points in the training set that are closest to x_q, and then predicts the label (or value) for x_q based on these neighbors.

For classification, the prediction is often made using a **majority vote** among the k neighbors. For regression, the prediction is typically the **average** (or a weighted average) of the neighbors' values.

1.2 Mathematical Formulation

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be a training dataset, where $x_i \in \mathbb{R}^n$ represents the feature vector and y_i the corresponding label or target value.

For a query point x_q, the k-NN algorithm involves the following steps:

1. **Distance Calculation:** Compute the distance between x_q and each x_i in $\mathcal{D}$. A common choice is the Euclidean distance:

$$d(x_q, x_i) = \sqrt{\sum_{j=1}^{n} (x_{q,j} - x_{i,j})^2},$$

   where $x_{q,j}$ and $x_{i,j}$ are the j-th features of x_q and x_i, respectively.

2. **Identify Neighbors:** Sort the distances and select the k data points with the smallest distances to x_q. These are the k nearest neighbors.

3. **Prediction:**

   - **Classification:** Assign the label to x_q by majority voting among the k neighbors. Formally, the predicted class $\hat{y}$ is given by:

$$\hat{y} = \text{mode}(\{y_i : x_i \in \mathcal{N}_k(x_q)\}),$$

where $\mathcal{N}_k(x_q)$ denotes the set of k nearest neighbors.

- **Regression:** Predict the target value as the average of the neighbors' values:

$$\hat{y} = \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x_q)} y_i.$$

1.3 Distance Metrics

While Euclidean distance is commonly used, other distance metrics may be more appropriate depending on the problem:

- **Manhattan Distance (L_1 norm):**

$$d(x, y) = \sum_{j=1}^{n} |x_j - y_j|.$$

- **Minkowski Distance:** A generalized distance metric defined as:

$$d(x, y) = \left(\sum_{j=1}^{n} |x_j - y_j|^p \right)^{1/p},$$

  where $p = 2$ corresponds to the Euclidean distance and $p = 1$ corresponds to the Manhattan distance.

The choice of metric can significantly impact the performance of the k-NN algorithm, especially in high-dimensional spaces.

1.4 Choice of k

The parameter k is crucial:

- A small k makes the model sensitive to noise (high variance), as the prediction is based on a very limited number of neighbors.
- A large k reduces the effect of noise but may smooth over local structure, potentially leading to bias.

Typically, k is chosen via cross-validation or based on domain knowledge. The optimal k balances bias and variance.

1.5 Advantages and Limitations

1.5.1 Advantages:

- **Simplicity:** k-NN is easy to understand and implement.
- **Non-parametric:** It makes no assumptions about the underlying data distribution.
- **Versatility:** Can be used for both classification and regression tasks.

1.5.2 Limitations:

- **Computational Cost:** The need to compute distances to all training points for every prediction can be computationally expensive, especially for large datasets.
- **Curse of Dimensionality:** In high-dimensional spaces, the difference between the nearest and farthest neighbor becomes less significant, which can deteriorate performance.
- **Choice of k:** The performance of the algorithm is sensitive to the choice of k and the distance metric.

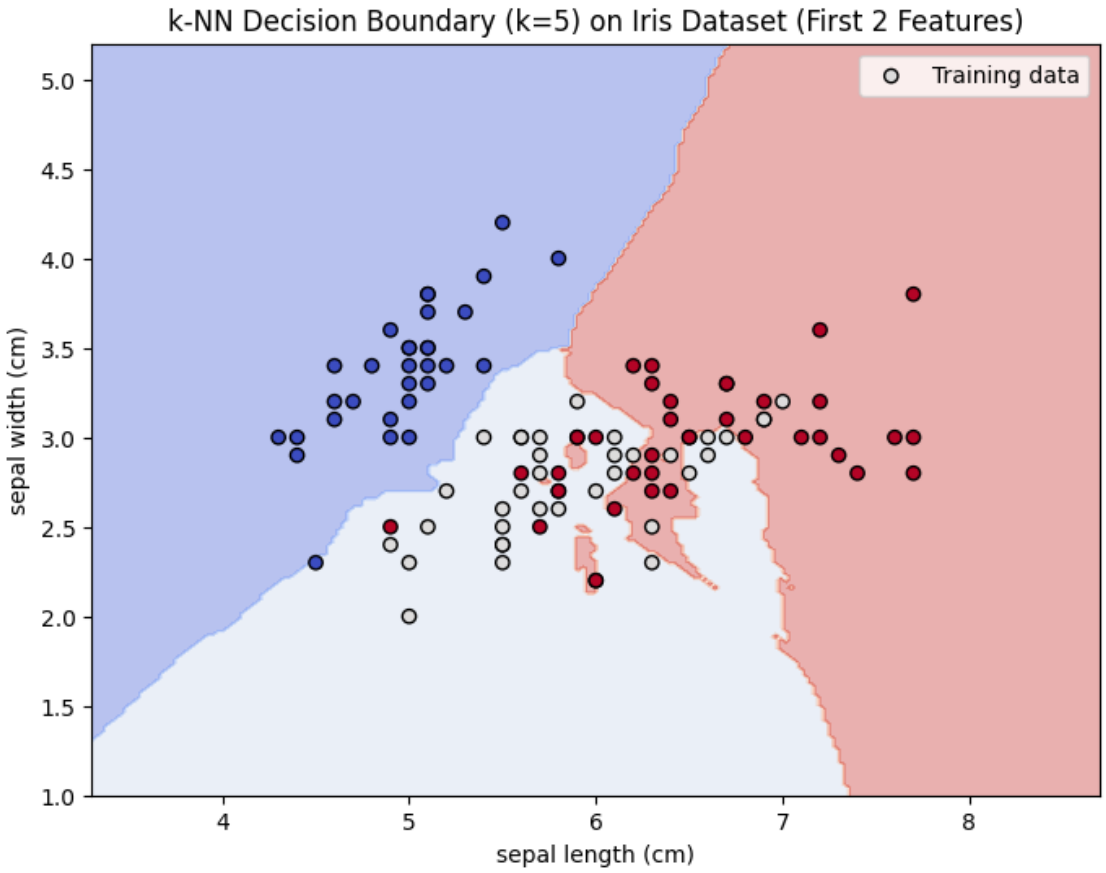
1.6 Python Example: k-NN Classification on the Iris Dataset

In this example, we use the Iris dataset to demonstrate k-NN classification. We will use scikit-learn's implementation of k-NN to classify iris species based on four features.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data  # features
y = iris.target  # labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Initialize k-NN classifier with k=5
k = 5
knn = KNeighborsClassifier(n_neighbors=k)

# Fit the model
knn.fit(X_train, y_train)

# Predict on the test set
y_pred = knn.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with k={k}: {accuracy:.2f}")

# For visualization purposes, plot the decision boundaries for the first two␣
 ↪features
import matplotlib.colors as colors
```

```
# Reduce the data to 2 dimensions
X_train_2d = X_train[:, :2]
X_test_2d = X_test[:, :2]

knn_2d = KNeighborsClassifier(n_neighbors=k)
knn_2d.fit(X_train_2d, y_train)

# Create a mesh grid
x_min, x_max = X_train_2d[:, 0].min() - 1, X_train_2d[:, 0].max() + 1
y_min, y_max = X_train_2d[:, 1].min() - 1, X_train_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max,␣
 ↪200))

# Predict over the mesh grid
Z = knn_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.coolwarm)
plt.scatter(X_train_2d[:, 0], X_train_2d[:, 1], c=y_train, cmap=plt.cm.
 ↪coolwarm, edgecolor='k', label='Training data')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('k-NN Decision Boundary (k=5) on Iris Dataset (First 2 Features)')
plt.legend()
plt.show()
```

```
Accuracy with k=5: 1.00
```



1.7 Summary

The k-Nearest Neighbors algorithm is a straightforward yet effective method for both classification and regression tasks. Its effectiveness relies on the assumption that similar points exist close to each other in feature space. By carefully choosing the distance metric and the number of neighbors k, one can achieve good performance, although care must be taken in high-dimensional settings due to the curse of dimensionality.

[ ]:

[ ]: