k-Nearest Neighbors (kNN) Classifier: Theory and Practice

Machine Learning Course

Lecture Overview

Topic: k-Nearest Neighbors Classification
Duration: 2 hours
Learning Objectives:

- Understand the intuition and mathematics behind kNN
- Implement kNN from scratch and using scikit-learn
- Master hyperparameter tuning for kNN
- Apply kNN to real-world classification problems
- Evaluate kNN performance and understand its limitations

1 Introduction to k-Nearest Neighbors

1.1 Basic Intuition

The k-Nearest Neighbors algorithm is based on a simple principle: **"Similar things exist near each other"** or **"Birds of a feather flock together."**

1.2 Formal Definition

kNN is a **non-parametric**, **lazy learning** algorithm used for both classification and regression.

- **Non-parametric**: Makes no assumptions about the underlying data distribution
- **Lazy learning**: Doesn't learn a model during training, simply stores the dataset
- **Instance-based**: Uses the entire dataset for prediction

2 Mathematical Foundation

2.1 Distance Metrics

The core of kNN is calculating distances between data points. Common distance metrics include:

2.1.1 Euclidean Distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

2.1.2 Manhattan Distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

2.1.3 Minkowski Distance (Generalized)

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{3}$$

2.2 kNN Algorithm Formulation

Algorithm 1 k-Nearest Neighbors Classification

procedure $\text{KNN}(X_{train}, y_{train}, x_{new}, k)$
    for each x_i in X_{train} **do**
        $distance_i \leftarrow \text{calculateDistance}(x_i, x_{new})$
    end for
    $indices \leftarrow \text{argsort}(distances)[: k]$ ▷ Get indices of k smallest distances
    $nearest\_labels \leftarrow y_{train}[indices]$
    $prediction \leftarrow \text{mode}(nearest\_labels)$ ▷ Most frequent class
    return $prediction$
end procedure

3 Detailed Example: Medical Diagnosis

3.1 Problem Setup

Let's consider a medical diagnosis problem where we want to predict whether a patient has a specific disease based on two features:

- **Feature 1**: Blood Pressure (normalized 0-1)
- **Feature 2**: Cholesterol Level (normalized 0-1)
- **Target**: Disease (0 = No, 1 = Yes)

3.2 Step-by-Step kNN Calculation

Let's classify a new patient with features: Blood Pressure = 0.55, Cholesterol = 0.45 using k=3.

Table 1: Patient Dataset for Disease Prediction

Patient ID	**Blood Pressure**	**Cholesterol**	**Disease**	**Label**
P1	0.2	0.3	No	0
P2	0.3	0.7	Yes	1
P3	0.4	0.2	No	0
P4	0.5	0.6	Yes	1
P5	0.6	0.4	No	0
P6	0.7	0.8	Yes	1
P7	0.8	0.3	No	0

3.2.1 Step 1: Calculate Distances

Using Euclidean distance: $d = \sqrt{(BP_1 - BP_2)^2 + (Chol_1 - Chol_2)^2}$

Table 2: Distance Calculations for New Patient

Patient	**Distance Calculation**	**Distance**	**Rank**
P1	$\sqrt{(0.55 - 0.2)^2 + (0.45 - 0.3)^2}$	0.38	4
P2	$\sqrt{(0.55 - 0.3)^2 + (0.45 - 0.7)^2}$	0.35	3
P3	$\sqrt{(0.55 - 0.4)^2 + (0.45 - 0.2)^2}$	0.29	2
P4	$\sqrt{(0.55 - 0.5)^2 + (0.45 - 0.6)^2}$	0.16	**1**
P5	$\sqrt{(0.55 - 0.6)^2 + (0.45 - 0.4)^2}$	0.07	**2**
P6	$\sqrt{(0.55 - 0.7)^2 + (0.45 - 0.8)^2}$	0.43	5
P7	$\sqrt{(0.55 - 0.8)^2 + (0.45 - 0.3)^2}$	0.34	**3**

3.2.2 Step 2: Identify k-Nearest Neighbors

For k=3, the nearest neighbors are:

- P5 (distance: 0.07) - Label: 0 (No Disease)
- P4 (distance: 0.16) - Label: 1 (Disease)
- P7 (distance: 0.34) - Label: 0 (No Disease)

3.2.3 Step 3: Majority Voting

Neighbor labels: [0, 1, 0]
Majority class: 0 (No Disease)

3.2.4 Prediction

The new patient is classified as **No Disease** (Class 0).

4 Real-World Application: Iris Flower Classification

4.1 Dataset Overview

The Iris dataset is a classic benchmark containing measurements of three iris flower species:

- **Features**: Sepal length, Sepal width, Petal length, Petal width
- **Classes**: Setosa, Versicolor, Virginica
- **Samples**: 150 (50 per class)

4.2 Complete Implementation with Scikit-learn

`https://www.kaggle.com/code/xvivancos/tutorial-knn-in-the-iris-data-set#k-nn-executio`
`https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassif`
`html`

5 Advanced kNN Concepts

5.1 Distance-Weighted kNN

In standard kNN, all k neighbors have equal vote. In distance-weighted kNN, closer neighbors have more influence:

$$\text{Weight}(x_i) = \frac{1}{d(x_i, x_{new}) + \epsilon} \tag{4}$$

$$\text{Class Probability} = \frac{\sum_{i \in \text{neighbors}} \text{Weight}(x_i) \cdot 1(y_i = c)}{\sum_{i \in \text{neighbors}} \text{Weight}(x_i)} \tag{5}$$

5.2 kNN for Regression

kNN can also be used for regression by taking the average of the k-nearest neighbors:

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i \tag{6}$$

6 Hyperparameter Tuning and Model Selection

6.1 Choosing the Optimal k

The value of k significantly impacts model performance:

Table 3: Impact of k Value on kNN Performance

k Value	Advantages	Disadvantages
k=1	Highly flexible, captures local patterns	Very sensitive to noise, overfitting
Small k	Captures fine-grained patterns	High variance, sensitive to outliers
Large k	Smooth decision boundary, robust to noise	May miss important local patterns
k=n	Always predicts majority class	Complete underfitting

6.2 Cross-Validation for k Selection

7 Strengths and Limitations

7.1 Advantages of kNN

- **Simple and intuitive**: Easy to understand and implement
- **No training phase**: Fast "training" (just storing data)
- **Adapts easily**: Can learn complex patterns without strong assumptions
- **Natural handling of multi-class problems**
- **Provides confidence scores** through class probabilities

7.2 Limitations and Challenges

- **Computationally expensive**: Prediction time grows with dataset size
- **Memory intensive**: Requires storing entire training dataset
- **Sensitive to feature scaling**: Features must be properly normalized
- **Curse of dimensionality**: Performance degrades in high dimensions
- **Sensitive to irrelevant features**: All features contribute equally to distance

7.3 When to Use kNN

8 Performance Optimization Techniques

8.1 Efficient kNN with KD-Trees

For large datasets, KD-Trees can significantly speed up nearest neighbor searches:

Table 4: kNN Application Scenarios

Good for kNN	Moderate for kNN	Poor for kNN
Small to medium datasets	Large datasets with indexing	Very large datasets
Low-dimensional data	Medium-dimensional data	High-dimensional data
Non-linear relationships	Complex decision boundaries	Simple linear relationships
Multi-class problems	Binary classification	Real-time applications
Prototype development	Research and exploration	Production systems with speed requirements

9 Practical Tips and Best Practices

9.1 Data Preprocessing for kNN

1. **Feature Scaling**: Always scale features (StandardScaler, MinMaxScaler)
2. **Handle Missing Values**: Impute or remove missing values
3. **Feature Selection**: Remove irrelevant features to improve performance
4. **Dimensionality Reduction**: Consider PCA for high-dimensional data

9.2 Model Selection Guidelines

- Start with k=5 as a reasonable default
- Use odd k values to avoid ties in binary classification
- Perform cross-validation to find optimal k
- Consider distance-weighted voting for better performance
- Use different distance metrics for different data types

10 Conclusion and Summary

10.1 Key Takeaways

- kNN is a simple yet powerful **instance-based** learning algorithm
- The choice of **k** and **distance metric** significantly impacts performance
- **Feature scaling** is crucial for kNN to work properly
- kNN suffers from the **curse of dimensionality** in high-dimensional spaces
- For large datasets, use **efficient data structures** like KD-Trees
- kNN is excellent for **prototyping** and **multi-class problems**

10.2 Further Reading

- Explore **radius-based neighbors** for density-based classification
- Study **locally weighted regression** for kNN-based regression
- Investigate **approximate nearest neighbor** algorithms for very large datasets
- Learn about **metric learning** to adapt distance metrics to specific problems

Exercise

Implement kNN to classify handwritten digits from the MNIST dataset and compare its performance with other classification algorithms. Experiment with different values of k, distance metrics, and preprocessing techniques to achieve the best possible accuracy.