Comprehensive Notes on K-Nearest Neighbors (KNN)

Prepared by Aashu Kumar

November 7, 2025

Contents

Introduction

K-Nearest Neighbors (KNN) is one of the simplest yet most powerful algorithms in machine learning, particularly used for classification and regression tasks. It belongs to the family of **instance-based learning algorithms**, meaning it does not build an explicit model but makes decisions based on the entire training dataset.

Key Idea:

Given a data point whose label is unknown, KNN finds the *K nearest data points* (neighbors) in the training set and assigns the most common label among them.

Assumptions

- KNN assumes data lies in a metric space, where distance between points can be meaningfully computed.
- Each training data point has an associated label (e.g., + or −). KNN can also handle multi-class problems.
- The algorithm requires a hyperparameter (K), typically an odd number, representing the number of neighbors considered for classification.

Geometric Intuition

Consider the following example dataset:

S. No	Age	Salary (in Thousands)	Purchased
1	25	20	N
2	63	120	Y
3	33	75	N
4	42	100	Y

When plotted on a 2D scatter plot (Age vs. Purchase), each data point represents a person.

- Green points (Y) indicate customers who purchased the product.
- Red points (N) indicate customers who did not purchase.

Classification Cases:

Case 1: K = 1 The new data point is classified based on its single nearest neighbor.

Case 2: K = 3 The majority label among the 3 nearest points decides the class.

Case 3: K = 5 The majority label among the 5 nearest points decides the class.

Hence, K controls the decision boundary smoothness. A small K can lead to overfitting, while a large K may lead to underfitting.

Mathematical Formulation

For a new point x, we calculate its distance to all training points using a distance metric such as:

- **Euclidean Distance:** $d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- **Manhattan Distance:** $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$
- **Minkowski Distance:** $d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$

Then, select the (K) nearest points and determine the label by majority voting.

Building KNN from Scratch

1. Load and preprocess the dataset (handle missing values, normalize features).
2. Choose a suitable distance metric.
3. Select a value for (K).
4. For each test sample:
    - Compute distances to all training samples.
    - Sort distances and pick the top (K) nearest neighbors.
    - Perform majority voting to assign the label.

Building and Deploying a KNN Model

1. Import `scikit-learn`: `from sklearn.neighbors import KNeighborsClassifier`
2. Split data into training and test sets.
3. Fit model: `model.fit(X`$_train, y_train$)
3. `Predict:` `y`$_pred = model.predict(X_test)$
3. `Evaluate using accuracy, confusion matrix, or cross-validation.`
4. `Deploy using pickle or joblib for serialization.`

Advanced Topics in KNN

- **Weighted KNN:** Closer neighbors have higher influence.
- **Distance Metrics:** Choice affects decision boundaries significantly.
- **Dimensionality Reduction:** Techniques like PCA can improve efficiency.
- **Curse of Dimensionality:** In high dimensions, distances become less meaningful.
- **Feature Scaling:** Standardization or normalization is critical for fair distance measurement.
- **KNN for Regression:** Predicts continuous values by averaging neighbor outputs.

Important Points to Remember

- Suitable for **low noise data**.
- A **lazy learner** — no explicit training phase.
- Can be used for both **classification and regression**.
- **Non-parametric:** No assumption about data distribution.
- **Instance-based:** Entire training data must be stored and referenced during prediction.

Limitations of KNN

- **Computationally expensive:** Requires calculating distances to all training samples for every prediction.
- **Memory-intensive:** Needs to store the entire training dataset.
- **Sensitive to irrelevant or redundant features.**
- **Does not work well with high-dimensional data** due to the curse of dimensionality.
- **Choice of K and distance metric** can significantly impact performance.
- **Slow for large datasets** — prediction time scales poorly with dataset size.
- **Sensitive to feature scaling:** Features with larger magnitude dominate the distance metric.
- **Imbalanced data problem:** Class with more samples can dominate the prediction.

Applications

- Recommendation Systems
- Document Retrieval
- Gene Expression Research
- Handwriting and Image Recognition
- Credit Risk Analysis
- Medical Diagnosis

How to Choose K?

Method 1:

- $K = \sqrt{N}$, where N is the number of training samples.
- Prefer odd values of (K) to avoid ties.
- Alternatively, use cross-validation or trial-and-error for optimal K.
- Use Grid Search to tune K systematically.

Performance Optimization Techniques

- Use **KD-Trees** or **Ball Trees** to reduce distance computation time.
- Apply **Dimensionality Reduction** (PCA, LDA) to reduce computational burden.
- Use **Approximate Nearest Neighbors (ANN)** for faster queries on large datasets.
- Combine with **feature selection methods** to remove irrelevant features.

References

- Christopher M. Bishop, *Pattern Recognition and Machine Learning*.
- Tom Mitchell, *Machine Learning*.
- Scikit-learn](https://scikit-learn.org/stable/modules/neighbors.htmlScikit-learn) documentation on KNN
- Towards](https://towardsdatascience.com/understanding-k-nearest-neighbors-knn-8821aa8a60c8Towards) Data Science - Understanding KNN

"What I cannot create, I do not understand." – Richard Feynman