k-Nearest Neighbors for Continuous Variables

Er Zhao, Shenyu Li, Shan Meng, Zifeng Li

1. Overview of the Algorithm

K-Nearest Neighbors (kNN) is a simple, intuitive, and widely used machine learning algorithm for both classification and regression tasks. It belongs to the family of instance-based learning or lazy learning algorithms. This means that the model doesn't explicitly learn a function during training but rather memorizes the training data and makes predictions based on the similarity of new data to the stored data points.

The core idea behind kNN is that similar data points tend to have similar outcomes or labels. In the case of classification, the class of a test point is determined by the majority class of its k nearest neighbors in the training set, and in the case of regression, the output is typically the average or weighted average of the target values of those neighbors.

1.1 The History of kNN

The k-nearest neighbors (kNN) algorithm originated in the 1950s when Evelyn Fix and Joseph L. Hodges Jr. introduced the "nearest neighbor rule" in a 1951 U.S. Air Force report. This rule classified points based on the distance to their closest neighbors. In 1967, Thomas M. Cover and Peter E. Hart formalized the approach in their paper *"Nearest Neighbor Pattern Classification,"* demonstrating that, with a large enough dataset, the nearest neighbor rule could achieve an error rate close to the optimal Bayes error rate. This established kNN as a credible method in pattern recognition and statistical classification.

For decades, kNN saw limited use due to computational constraints—it required storing the entire dataset and performing expensive searches during prediction. However, advancements in computing power in the 1980s and 1990s addressed these limitations, making kNN a staple in machine learning courses for its simplicity, intuitive approach, and strong theoretical foundations.

Today, kNN remains a foundational algorithm, both as a teaching tool for beginners and as a benchmark for developing more advanced machine learning techniques.

1.2 Steps in kNN

To apply the k-nearest neighbors (kNN) algorithm, we begin by choosing the value of k, which determines how many neighbors we consider. Next, we calculate the distance between our test point and each data point in the training set, then select the k closest points. If we are dealing with classification, the test point is assigned to the class that appears most frequently among these neighbors. For regression, the predicted value is the average of the values of the k nearest neighbors.

One of the main advantages of kNN is that it’s simple to understand and implement, making it a good starting point for beginners. It is a non-parametric method, meaning it doesn’t make any assumptions about how the data is distributed. Additionally, kNN is effective for both classification and regression tasks, making it a versatile choice for many different problems.

However, kNN also comes with several drawbacks. It can be computationally expensive for large datasets since it must compute distances to all training points for each prediction. The algorithm’s performance depends heavily on the choice of k and how features are scaled. It also struggles in high-dimensional spaces, where data becomes sparse, a phenomenon known as the “curse of dimensionality.”

1.3 Examples of kNN in Real Life

1.3.1 Recommendation Systems

In an online store, a recommendation system might use kNN to suggest products to customers. It looks at the customer's past purchases and identifies other customers who bought similar products. The system can then recommend items purchased by those similar customers.
How it works: The algorithm calculates the "distance" between a user's purchase history and other users' purchase histories. The top k closest users (neighbors) determine the items to recommend.

1.3.2 Medical Diagnosis

A kNN algorithm might be used to classify a patient's medical condition based on symptoms. For instance, if a patient reports certain symptoms (such as fever, cough, and fatigue), the system can predict whether they might have a condition like the flu or COVID-19 by comparing the patient's symptoms to historical patient data with known diagnoses.
How it works: The algorithm calculates the "distance" between the test patient's symptom profile and the training set (which contains data on patients with known diagnoses). The majority class of the k nearest neighbors (e.g., "flu" or "COVID-19") will be assigned as the prediction.

2. Illustrative Example

Imagine you are running an online store and you have data about your customers. Each data point represents a single customer’s shopping habits measured by two simple features:

- Number of items purchased per year
- Average price per item purchased

You want to classify customers into two categories:

- Bargain Hunters: who buy lower-cost items more frequently
- Premium Shoppers: who buy fewer but more expensive items.

Here’s an example of how you might assign each data point to known categories.

Customer Habits	Number of items	Average price	Categories
Customer1	40	36	Bargain Hunters
Customer2	30	30	Bargain Hunters

Customer3	32	45	Bargain Hunters
Customer4	37	55	Bargain Hunters
Customer5	40	47	Bargain Hunters
Customer6	52	40	Bargain Hunters
Customer7	48	59	Premium Shoppers
Customer8	60	67	Premium Shoppers
Customer9	68	50	Premium Shoppers
Customer10	40	65	Premium Shoppers
Customer11	55	55	Premium Shoppers
Customer12	58	68	Premium Shoppers

Customer1: (40, 36) - This customer buys a moderate number of items (40 a year) at a modest average price (36).
Customer2: (30, 30) - Fewer items and a relatively low price point might suggest a Bargain Hunter.
…
Customer7: (48, 59) - Quite a few items at a high price point - more Premium.
…

Now imagine you have a new unidentified customer U:
Unidentified customer U: (42, 60)
This new customer buys 42 items per year at an average price of 60. You want to decide if U is more like a Bargain Hunter or a Premium Shopper.

We can use kNN to classify this customer step by step:

- Step 1: Determine the distance (e.g. Euclidean distance) between the observation and all the data points in the training set.
- Step 2: Sort the distances.
- Step 3: Identify the k closest neighbors.
- Step 4: Determine the class of the new observation based on the group majority of the k nearest neighbors.

Let's follow these steps on our example data where we like to predict the class of a new observation – Unidentified customer U: (42, 60), with a "Number of items" concentration of 42 and an "Average price" concentration of 60.

In this example, we set the value of k to 5, which means we check the class of the 5 closest neighbors.

Step 1. We begin by calculating the Euclidean distance to all the data points. Let's calculate the distance between the new observation and customer 1. In two dimensions, the Euclidean distance can be calculated by the following formula:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

We plug in the x and y coordinates of the new data point and for data point customer1.

$$\sqrt{(60 - 36)^2 + (42 - 40)^2} = 24.1$$

We see that the Euclidean distance between these two data points is 24.1. We fill in this distance in the table.

Customer Habits	Number of items	Average price	Categories	Distance
Customer1	40	36	Bargain Hunters	24.1
Customer2	30	30	Bargain Hunters	
Customer3	32	45	Bargain Hunters	
Customer4	37	55	Bargain Hunters	
Customer5	40	47	Bargain Hunters	
Customer6	52	40	Bargain Hunters	
Customer7	48	59	Premium Shoppers	
Customer8	60	67	Premium Shoppers	
Customer9	68	50	Premium Shoppers	
Customer10	40	65	Premium Shoppers	
Customer11	55	55	Premium Shoppers	
Customer12	58	68	Premium Shoppers	

Following the same formula, we can calculate the Euclidean distance between the new observation and data point customer2, which is 32.3.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = \sqrt{(60 - 30)^2 + (42 - 30)^2} = 32.3$$

So on and so on, we can calculate the Euclidean distances between the new observation and each data point customer, and then fill out the table as below.

Customer Habits	Number of items	Average price	Categories	Distance
Customer1	40	36	Bargain Hunters	24.1
Customer2	30	30	Bargain Hunters	32.3
Customer3	32	45	Bargain Hunters	18.0
Customer4	37	55	Bargain Hunters	7.1
Customer5	40	47	Bargain Hunters	13.2

Customer6	52	40	Bargain Hunters	22.4
Customer7	48	59	Premium Shoppers	6.1
Customer8	60	67	Premium Shoppers	19.3
Customer9	68	50	Premium Shoppers	27.9
Customer10	40	65	Premium Shoppers	5.4
Customer11	55	55	Premium Shoppers	13.9
Customer12	58	68	Premium Shoppers	17.9

Step 2. Once we have calculated the Euclidean distances between the new observation and all the data points, we will sort this table based on the distances as below.

Customer Habits	Number of items	Average price	Categories	Distance
Customer10	40	65	Premium Shoppers	5.4
Customer7	48	59	Premium Shoppers	6.1
Customer4	37	55	Bargain Hunters	7.1
Customer5	40	47	Bargain Hunters	13.2
Customer11	55	55	Premium Shoppers	13.9
Customer12	58	68	Premium Shoppers	17.9
Customer3	32	45	Bargain Hunters	18.0
Customer8	60	67	Premium Shoppers	19.3
Customer6	52	40	Bargain Hunters	22.4
Customer1	40	36	Bargain Hunters	24.1
Customer9	68	50	Premium Shoppers	27.9
Customer2	30	30	Bargain Hunters	32.3

Step 3. After we have sorted the customers based on the distances to the new observation, we see that three out of five closest neighbors are of class “Premium Shoppers” whereas two are of class “Bargain Hunters”.

Customer Habits	Number of items	Average price	Categories	Distance
Customer10	40	65	Premium Shoppers	5.4
Customer7	48	59	Premium Shoppers	6.1
Customer4	37	55	Bargain Hunters	7.1

Customer5	40	47	Bargain Hunters	13.2
Customer11	55	55	Premium Shoppers	13.9

Step 4. Since the majority of the k nearest neighbors are of class “Premium Shoppers”, we classify the new observation as “Premium Shoppers”.

We successfully applied the kNN algorithm to classify the new, previously undetermined customer U as a “Premium Shopper”. Based on this classification, we can now tailor our advertising strategies and promotional activities accordingly.

3. Information about Technical Parts

Our group chose to use Python as our project programming language, not only because it is easier to code but also because they have sufficient third library packages for machine learning. For our code, we implemented numpy, pandas, scikit-learn, matplotlib, and seaborn.

- Numpy package provides efficient multi-dimensional array operations. Since the dataset may contain multiple parameters, the numpy package can speed up the calculation.
- Pandas package provides powerful data manipulation, data cleaning and preprocessing, seamless integration with other packages like numpy, matplotlib, and efficient indexing and filtering. Datasets are not always clean enough, and pandas package can handle missing data, grouping, merging and dropping NA or meaningless data.
- Scikit-learn package has a comprehensive machine learning toolkit including classification, regression, model selection and preprocessing dataset. In our project, to validate and test our scratch model, we splitted our data to training data and testing data, which we used train_test_split in sklearn.model_selection.
- Matplotlib and Seaborn are both powerful libraries for data visualization. They allow us to visualize data distributions and gain more intuitive insights from the dataset.

We wrote our KNN models from scratch. There are eight functions in class KNN.

- The first function is __init__(self, k=3, metric=”euclidean”), which is an initializing function, and we determine the default k value is 3 and metric distance is euclidean (L2 norm).
- The second function is euclidean(self, v1, v2), which computes the Euclidean distance between two vectors v1 and v2. It calculates the square root of the sum of squared differences between corresponding elements in the vectors. It is used when the distance metric is set to “euclidean”.
- The third function is manhattan(self, v1, v2), which computes the Manhattan distance (L1 norm) between two vectors v1 and v2. It calculates the sum of the absolute differences between corresponding elements in the vectors. It is used when the distance metric is set to “manhattan”.
- The forth function is fit(self, X, y), which trains the KNN model by storing the training data X and their corresponding labels y as self.X_train and self.y_train. It does not involve learning any parameters but prepares the data for future predictions.

- The fifth function is get_neighbours(self, test_row), which finds the k nearest neighbors for a given test data point test_row. It calculates the distance between the test point and all training points using the specific distance metric (Euclidean or Manhattan). The function returns the k closest neighbors sorted by distance.
- The sixth function is predict(self, X_test), which predicts the class labels for a given set of test data points X_test. For each test point, it finds the k nearest neighbors using the get_neighbours function, determines the majority class among them using the Counter class, and appends the predicted class to the output array.
- The seventh function is score(self, preds, y_test), which calculates the accuracy score of the model's predictions. It compares the predicted labels (preds) to the true labels (y_test) and computes the percentage of correct predictions.
- The last function is plot_predictions(self, X_test, y_test, preds, feature_names = ["Height", "Weight", "Length"]), which visualizes the predictions of the KNN model in a 3D scatter plot. It plots the training data points, the testing data point, and its k nearest neighbors in a 3D place. The function also draws connecting lines between the test point and its neighbors.

4. Instructions on How to Run the Program

In this program, we use a K-Nearest Neighbors (KNN) classifier to predict whether an animal is a cat or a dog based on its physical attributes (Height, Weight, and Length). We also visualize the data and predictions using 3D scatter plots. Here's an Instruction of how to run the whole program properly.

- Prerequisites
  Before you run the program, you need to install the necessary Python libraries. To do this, you can run the following command:
  pip install numpy pandas matplotlib seaborn scikit-learn
  You also need the dataset ***CatsAndDogs.csv***, which contains the data for training the model.
- Running the Program
  Once the necessary libraries and dataset are in place, we navigate to the directory containing the Python script and dataset. To run the program, you need to execute the following command:
  python name_project.py

5. Instructions on Required Input Format

The input dataset must consist of continuous variables, and all data should be cleaned and preprocessed to remove missing values, duplicates, and outliers. By following this instruction, you ensure that the input data is in a standardized format, reducing the risk of errors and improving the reliability of the result.

- Data Type: The input dataset should only contain continuous numerical variables. If your data includes non-numerical features, please transform them into continuous values before use.
- Data Cleaning:

    - Missing Values: Your dataset must not contain any missing values. If a value is missing, please remove the record or impute the missing value using an appropriate method (e.g., mean, median, interpolation).
    - Duplicates: Remove any duplicate rows. Each record in the dataset should be unique and represent a distinct observation.
    - Outliers: Identify and remove any outliers that could significantly skew the analysis. Detect and eliminate outlier values.
- File Format & Structure: Prepare the dataset in a common, machine-readable format (e.g., CSV or TSV). Include a single header row that names each variable.
- No Extra Information: Do not include text descriptions, metadata, or additional worksheets within the data file. The dataset should be clean and ready for direct input into the model.

6. Reflection

In this project, implementing the k-Nearest Neighbors (kNN) algorithm from scratch provided insights into its workings, advantages, and limitations. We explored the fundamental steps of the algorithm, including distance calculation, neighbor selection, and classification/prediction, which deepened our understanding of how kNN operates behind the scenes.

First, the decision to use Python proved to be advantageous due to the availability of libraries like NumPy, pandas, scikit-learn, matplotlib, and seaborn, which streamlined tasks such as data manipulation, visualization, and model validation. Writing the algorithm from scratch allowed us to comprehend how the distance metrics (Euclidean and Manhattan) influence the model's predictions and how the choice of hyperparameters, like k, can impact accuracy.

Then, one of the key challenges we encountered was optimizing the code for large datasets, as kNN is computationally expensive, particularly during the prediction phase when distances must be calculated for all training points. Another challenge was selecting the appropriate k, as smaller values of k led to overfitting, while larger values diluted the influence of close neighbors. Experimentation with different k values helped us balance bias and variance effectively.

Finally, visualization tools like 3D scatter plots helped us interpret the results intuitively and analyze how different parameters affected classification boundaries. These visualizations also highlighted potential cons, such as sensitivity to outliers or feature scaling, underscoring the importance of preprocessing steps.

Overall, this project has significantly enhanced our understanding of kNN, its applications, and its limitations. It reinforced the importance of hyperparameter tuning, dataset preprocessing, and computational considerations, equipping us with skills to apply kNN effectively in future machine learning projects.

7. Citations

1. CatsAndDogs dataset:

   https://www.kaggle.com/datasets/scarb7/catsanddogs-dummy

2. k-Nearest Neighbours(KNN) from scratch:

   https://www.kaggle.com/code/samuelcortinhas/k-nearest-neighbours-knn-from-scratc
   h

3. K-Nearest Neighbor(KNN) Algorithm:

   https://www.geeksforgeeks.org/k-nearest-neighbours/