k-Nearest Neighbours Method

Shreeyesh Menon

1 Introduction

Many predictive modelling tasks in economics rely on flexible, nonparametric methods that make minimal assumptions about functional forms. The **k-Nearest Neighbours (kNN)** algorithm is one of the simplest and most intuitive examples of such a method. Given a new observation, kNN looks for the k most similar examples in the training data and constructs a prediction based on their outcomes.

Despite its simplicity, kNN is a powerful baseline tool and an important conceptual stepping stone toward more advanced machine learning algorithms.

2 The Basic Idea

Suppose we observe training pairs

$$\{(x_i, y_i)\}_{i=1}^{n},$$

where $x_i \in \mathbb{R}^d$ represents a vector of features (e.g., income, age, education) and y_i is the associated outcome (class label or continuous value).

Given a new point x_{new}, kNN works as follows:

1. Compute the distance from x_{new} to every training point.
2. Select the k nearest neighbours.
3. For regression: predict the average of their y_i values.
4. For classification: predict the most frequent class among the neighbours.

The model is "nonparametric" because no explicit functional form (such as linear or logit) is assumed.

3 Distance Metrics

The choice of distance metric affects which points are considered "nearest." The most common is Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_{il} - x_{jl})^2}.$$

Other metrics include:

- **Manhattan distance:** $d(x_i, x_j) = \sum_{l=1}^{d} |x_{il} - x_{jl}|$
- **Minkowski distance:** a generalization of the above
- **Mahalanobis distance:** accounts for variable scaling and correlation

Standardizing features is essential when variables are on different scales.

4 kNN for Regression

For regression, the kNN estimate of the function $f(x)$ at a point x_{new} is:

$$\hat{f}(x_{\text{new}}) = \frac{1}{k} \sum_{i \in N_k(x_{\text{new}})} y_i,$$

where $N_k(x_{\text{new}})$ is the set of indices corresponding to the k nearest neighbours.

This estimator is essentially a local average. Small k yields a very flexible estimator; large k gives a smoother one.

5 kNN for Classification

For classification, the predicted class is the majority vote:

$$\hat{y}(x_{\text{new}}) = \arg\max_{c} \sum_{i \in N_k(x_{\text{new}})} \mathbf{1}(y_i = c).$$

One can also compute estimated class probabilities:

$$\hat{p}(c \mid x_{\text{new}}) = \frac{1}{k} \sum_{i \in N_k(x_{\text{new}})} \mathbf{1}(y_i = c).$$

This makes kNN useful when uncertainty quantification is desirable.

6 The Role of k and the Bias–Variance Tradeoff

The choice of k controls model complexity:

- **Small k (e.g., $k = 1$):**
  - Highly flexible, low bias
  - High variance, sensitive to noise
- **Large k:**

  - Smoother predictions, higher bias
  - Lower variance

Thus, selecting k is an example of the classic **bias–variance tradeoff**. In practice, cross-validation is used to pick k.

In any modeling exercise, we often face the challenge of selecting model *hyperparameters*, or parameters that govern how the model learns from data.

For the k-Nearest Neighbours (kNN) algorithm, the critical hyperparameter is the number of neighbours, k.

7 Train–Test Split

Before introducing cross-validation, let's address the terms training and validation metrics.

The dataset is divided into a training set, used to fit the model, and a test set, used to evaluate prediction accuracy. Training error refers to mis-classification/prediction mistakes made by the error for training data samples. This represents how well the model is able to "fit the data" shown to it.

However, our goal is to have a model that can perform well over data that it hasn't seen during training. Hence, the meaningful metric for measuring model performance is the validation error, or the magnitude of the errors that the model makes when predicting out-of-sample. For this reason, we keep a subsample of the data separate and call this the *validation set*. This process is called a train-test split.

However, for choosing hyperparameters such as k, a single train–test split is often unstable because the evaluation depends heavily on that particular split.

8 k-Fold Cross-Validation

k-fold cross-validation is a more robust method to estimate out-of-sample performance. The procedure is:

1. Split the training data into K roughly equal-sized folds.
2. For each fold $j = 1, \ldots, K$:
   - Treat fold j as the validation set.
   - Train the model on the remaining $K - 1$ folds.
   - Compute the prediction error on the validation fold.
3. Average the validation errors across all K folds.

The result is an estimate of the model's out-of-sample error for a given hyperparameter value.

Common choices are $K = 5$ or $K = 10$, balancing computational cost and performance stability.

9 Selecting the Optimal k for kNN

To choose the best number of neighbours k in kNN, we perform cross-validation across a grid of candidate values:

1. Choose a set of candidate values, e.g. $k \in \{1, 2, 3, \ldots, 25\}$.
2. For each value of k, run K-fold cross-validation and compute the average error:

$$\mathrm{CV}(k) = \frac{1}{K} \sum_{j=1}^{K} \mathrm{Err}_j(k),$$

   where $\mathrm{Err}_j(k)$ is the validation error on fold j.
3. Select the value of k that achieves the lowest cross-validation error:

$$k^* = \arg\min_k \mathrm{CV}(k).$$

The selected k^* reflects the best tradeoff between bias and variance for the given dataset.

10 Weighted kNN

A common extension is distance-weighted kNN, where closer neighbours receive larger weights. One popular weighting is the inverse-distance scheme:

$$\hat{f}(x_{\text{new}}) = \frac{\sum_{i \in N_k(x_{\text{new}})} w_i y_i}{\sum_{i \in N_k(x_{\text{new}})} w_i}, \qquad w_i = \frac{1}{d(x_{\text{new}}, x_i)}.$$

This can improve performance when neighbours differ substantially in proximity.

11 The Curse of Dimensionality

A major limitation of kNN is the **curse of dimensionality**. As the number of features d grows:

- all points become far from one another,
- distances become less informative,
- the nearest neighbour may be very far away,
- prediction quality deteriorates.

For high-dimensional economic data, dimensionality reduction (PCA, feature engineering, etc.) is often crucial.

12 Computational Considerations

Because kNN requires computing distances to all training points at prediction time, it is computationally expensive for large datasets. Popular optimizations include:

- **KD-trees** and **Ball Trees** for faster neighbour searches
- Approximate nearest neighbour methods
- Dimensionality reduction prior to applying kNN

In most practical scenarios with moderately sized data, standard implementations (e.g., in `scikit-learn`) are sufficient.