MACHINE LEARNING LECTURE

How K-Nearest Neighbors Sees the World: Understanding Decision Boundaries and the Role of *k*

Introduction:

K-Nearest Neighbors (KNN) is one of the simplest and most intuitive classification algorithms in machine learning. Despite its simplicity, KNN provides powerful insights into how models partition feature space and how the choice of hyperparameters—especially the number of neighbors, *k*—affects predictive performance.
This tutorial aims to teach the conceptual and practical aspects of KNN through visualization. In particular, we focus on **decision boundaries**, which reveal how the model "sees" different regions of the feature space. By examining decision boundaries for different values of *k*, analysing the bias–variance trade-off, and exploring the roles of feature scaling, weighting schemes, and cross-validation, this tutorial provides a practical foundation for understanding and applying KNN in your own work.

The target audience includes learners with basic familiarity in Python, NumPy, Matplotlib, and scikit-learn, but the conceptual explanations are presented from first principles.

How KNN Works and Why Decision Boundaries Matter:

KNN is a **non-parametric** classification method. Instead of estimating parameters or training a model, KNN simply stores the training data. To classify a new point, the algorithm:

1. Computes the distance from the new point to all training points (typically using Euclidean distance).
2. Selects the *k* nearest neighbors.
3. Predicts the majority class among these neighbors.

Because KNN relies directly on distances, the structure of the data determines its behavior. Unlike linear models that impose a fixed type of boundary, KNN can approximate complex shapes. This flexibility makes it ideal for visual demonstrations.

The **decision boundary** is the curve that separates regions of different predicted classes in the feature space. Visualizing this boundary reveals the model's structure and highlights how sensitive KNN can be to noise, outliers, and the choice of *k*.

Experimental Setup:

To study KNN visually, we use a two-dimensional dataset created with the *make_moons* function from scikit-learn. This dataset contains two interleaving half-circles that cannot be separated linearly, making it ideal for examining non-linear decision boundaries.

The steps include:

- Generating 800 data points with moderate noise
- Splitting data into training and test sets
- Scaling the features using standardization

Scaling is essential because KNN depends entirely on the notion of distance. If one feature has a larger numeric range than another, it may dominate the distance computation and distort the model's behaviour.

A scatter plot of the dataset shows two intertwined clusters with some overlap. This setting enables us to observe how different values of *k* control the smoothness or complexity of the resulting decision boundary.

Visualizing Decision Boundaries:

To understand KNN's behaviour, we generate decision boundary plots by:

1. Creating a grid that spans the range of the two features.
2. Using the trained KNN model to predict the class for each point on the grid.
3. Colouring grid regions based on predicted class.
4. Overlaying the original training data on top.

The resulting plots show how KNN assigns regions of space to each class. Where two colours meet, the algorithm is uncertain, and the boundary is formed.

To support accessibility, colour-blind-friendly palettes should be used (for example, soft red versus soft blue), and figures should include clear titles and axis labels.

This boundary-plotting method is reused throughout the tutorial to compare model behaviour under different hyperparameter choices.

How Changing *k* Transforms the Boundary:

The value of *k* dramatically shapes the behaviour of KNN:

- **Small *k* (e.g., 1 or 3):**
  Boundaries are highly irregular and follow the training data closely.
  The classifier is extremely flexible and may overfit noise.
- **Moderate *k* (e.g., 5 to 15):**
  The boundary becomes smoother.
  Small pockets of mislabeled or noisy data lose influence.
  Test accuracy usually improves.
- **Large *k* (e.g., 50):**
  The model becomes overly smooth.
  Fine structure is lost, and the decision boundary may ignore true patterns.
  Predictions tend to reflect overall class proportions.

Comparing boundary plots for $k = 1$, 5, 15, and 50 clearly illustrates these effects. When $k = 1$, the classification regions form intricate shapes that mirror local noise. By contrast, for $k = 50$, nearly the entire space may be predicted as one class except near dense regions of the other.

These visualisations provide a direct and intuitive demonstration of how KNN transitions from highly flexible to overly rigid as *k* increases.

The Bias–Variance Trade-Off in KNN:

The behaviour described above can be interpreted through the bias–variance trade-off:

- **Small *k* → low bias, high variance**
  The model can fit complex patterns but may change drastically with small perturbations in training data.
- **Large *k* → high bias, low variance**
  The model becomes smoother and more stable but may fail to capture meaningful structure.

To measure this relationship, we compute training and test accuracy across a range of *k* values. Typical behaviour includes:

- Very small *k* yields extremely high training accuracy (near 1.0) but lower test accuracy due to overfitting.
- Test accuracy peaks at intermediate *k* values (often between 5 and 15).
- Very large *k* reduces both training and test accuracy due to underfitting.

Plotting accuracy as a function of *k* produces a curve that clearly shows the optimal region. This empirical demonstration reinforces the conceptual discussion and strengthens understanding of generalisation.

Distance Weighting: Uniform vs. Distance-Based:

KNN allows different weighting schemes for how neighbours vote:

- **Uniform weighting:**
  Each neighbour contributes equally.
- **Distance weighting:**
  Closer neighbours have stronger influence, reducing the impact of distant or noisy points.

By applying both weighting schemes at various *k* values, we can observe how the decision boundary shifts. Distance weighting often produces smoother, more stable boundaries, especially in regions with overlapping classes. It also tends to improve test accuracy for moderate values of *k* because the model becomes more sensitive to local structure without overreacting to noise.

Selecting *k* Using Cross-Validation:

In practical applications, the best value of *k* should not be selected by visual inspection. Instead, cross-validation provides a principled way to determine which value performs best on unseen data.

Grid search with cross-validation evaluates multiple combinations of:

- Values of *k* (usually odd numbers to avoid ties)
- Weighting schemes (“uniform” or “distance”)

For each combination, the training data is split into several folds, and performance is averaged across them. The combination with the highest cross-validated accuracy is selected.

After identifying the optimal parameters, the model is refitted on the full training dataset and evaluated on the test set. This method ensures honest assessment and prevents overfitting to the test set.

The Importance of Feature Scaling:

Because KNN depends entirely on distance, feature scaling is crucial. Features measured on different scales can distort distance computations. For example, a feature in the range 0–1000 will dominate a feature in the range 0–1, even if the latter is more important for classification.

Experiments comparing models with and without standardization demonstrate large differences:

- Without scaling, accuracy typically drops, and decision boundaries appear stretched or skewed.
- With scaling, the model behaves more sensibly, accuracy improves, and boundaries become more regular.

This practical demonstration reinforces a key lesson: always standardize features when using distance-based algorithms.

Multiple Datasets: Where KNN Succeeds and Struggles:

To highlight the strengths and weaknesses of KNN, we repeat the experiments on multiple datasets:

- **Moons:**
  Complex, curved structure; KNN performs well with moderate *k*.
- **Circles:**
  Concentric circular patterns; again, KNN can capture the circular boundaries effectively.
- **Blobs:**
  Linearly separable clusters; even simple classifiers perform well.

This comparison helps illustrate when KNN is appropriate:

- Low-dimensional datasets
- Smooth but non-linear structure
- Moderate sample sizes
- Situations where interpretability and simplicity are desirable

KNN struggles when dimensionality is high, when distance is not a meaningful metric, or when dataset size becomes large (because prediction time scales poorly).

Summary and Key Takeaways:

This tutorial examined the K-Nearest Neighbors algorithm through the lens of decision boundary visualisation. By observing how KNN partitions the feature space for different values of *k*, we developed intuition for its strengths, limitations, and tuning strategies.

Key lessons include:

- The value of *k* directly controls the model's flexibility and smoothness.
- KNN exemplifies the bias–variance trade-off.
- Distance weighting can improve performance in overlapping regions.
- Feature scaling is essential for meaningful distance computation.
- Cross-validation provides a principled method for selecting *k*.
- KNN is well-suited for low-dimensional datasets where interpretability and flexible boundaries are beneficial.

Although simple, KNN provides foundational intuition about model behaviour and serves as a useful baseline in practical machine learning tasks.

References

- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21–27.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.
- Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- van der Plas, J. (2016). *Python Data Science Handbook*. O'Reilly.

GITHUB LINK: https://github.com/Sadweep03/ML_KNN/blob/main/ML_KNN.ipynb