

MACHINE LEARNING

CHAP 7 K-Nearest Neighbor KNN

Jean Tabet
j.tabet@ul.edu.lb

➔ K-NEAREST NEIGHBOR KNN

Introduction

- K-Nearest Neighbors (KNN) is a supervised machine learning algorithm generally used for classification but can also be used for regression tasks.
- It works by finding the "k" closest data points (neighbors) to a given input.
- It makes predictions based on the majority class (for classification) or the average value (for regression).
- Since KNN makes no assumptions about the underlying data distribution it makes it a non-parametric and instance-based learning method.

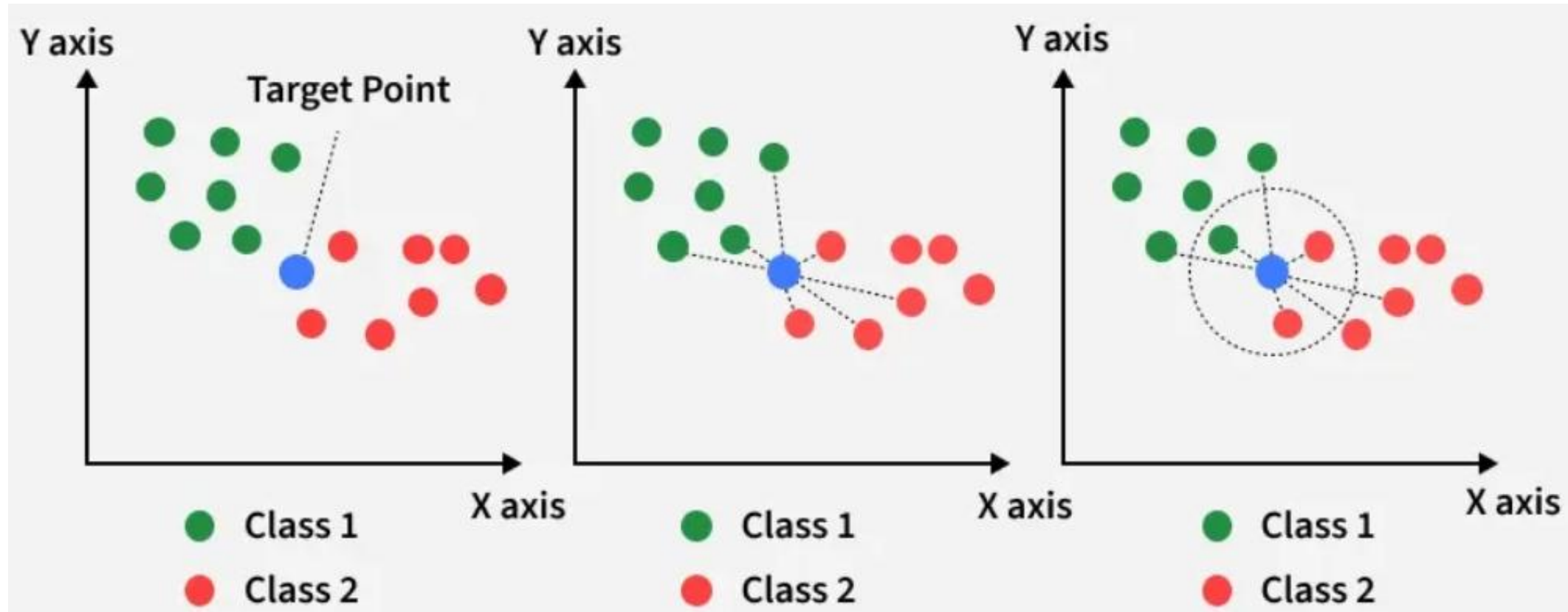
➔ K-NEAREST NEIGHBOR KNN

Applications

- Image Recognition (e.g., handwritten digit classification)
- Recommendation Systems
- Customer Segmentation
- Fraud Detection
- Medical Diagnosis

→ K-NEAREST NEIGHBOR KNN

Introduction



➔ K-NEAREST NEIGHBOR KNN

Mathematical formulation

- Classification:

$$\hat{y} = \arg \max_c \sum_{i \in N_k(x)} \mathbf{1}(y_i = c)$$

- Regression:

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

- Distance:

$$d(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

➔ K-NEAREST NEIGHBOR KNN

Choosing the value of k

- The value of k in KNN decides how many neighbors the algorithm looks at when making a prediction.
- Choosing the right k is important for good results.
- If the data has lots of noise or outliers, using a larger k can make the predictions more stable.
- But if k is too large the model may become too simple and miss important patterns and this is called underfitting.

- Small k → sensitive to noise (overfitting)
- Large k → smooth model (underfitting)

➔ K-NEAREST NEIGHBOR KNN

Statistical Methods for Selecting k

Cross-Validation: Cross-Validation is a good way to find the best value of k is by using k-fold cross-validation.

- This means dividing the dataset into k parts. The model is trained on some of these parts and tested on the remaining ones. This process is repeated for each part.
- The k value that gives the highest average accuracy during these tests is usually the best one to use.

Elbow Method: In Elbow Method we draw a graph showing the error rate or accuracy for different k values.

- As k increases the error usually drops at first. But after a certain point error stops decreasing quickly.
- The point where the curve changes direction and looks like an "elbow" is usually the best choice for k.

➔ K-NEAREST NEIGHBOR KNN

Distance Metrics Used in KNN Algorithm

1. Euclidean Distance

Euclidean distance is defined as the straight-line distance between two points in a plane or space.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^d (x_j - X_{i_j})^2}$$

2. Manhattan Distance

This is the total distance you would travel if you could only move along horizontal and vertical lines like a grid or city streets.

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

3. Minkowski distance (default)

Minkowski distance is a general distance measure controlled by a parameter p. When p=2, it becomes Euclidean distance, and when p=1, it becomes Manhattan distance.

➔ K-NEAREST NEIGHBOR KNN

Choosing the value of k

Step 1: Selecting the optimal value of K

Step 2: Calculating the distance between data points in the dataset and target point.

Step 3: Finding the K Nearest Neighbors

Step 4: Voting for Classification or Taking Average for Regression.

- The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called majority voting.
- In regression, the algorithm takes the average of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.

➔ K-NEAREST NEIGHBOR KNN

Limitations of KNN

- High computational cost for large datasets
- Sensitive to feature scaling
- Poor performance in high-dimensional space
- Requires storing the entire training dataset

➔ K-NEAREST NEIGHBOR KNN

Implementing KNN in Python

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model = KNeighborsClassifier(n_neighbors=3)
```

```
model.fit(X_train, y_train)
```

```
predictions = model.predict(X_test)
```

```
print(predictions)
```

➔ K-NEAREST NEIGHBOR KNN

Tuning hyperparameters

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
```

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
grid = GridSearchCV(KNeighborsClassifier(),
    param_grid, cv=5)
grid.fit(X_train, y_train)
```

```
print("Best Parameters:", grid.best_params_)
```

1. `n_neighbors` (`k`)

- **What it does:** Controls how many nearest neighbors vote for the output.
- **Effect:**
 - Smaller `k`: More flexible (low bias, high variance)
 - Larger `k`: Smoother, more stable (high bias, low variance)
- **Typical range:** 1 to 50 (use odd numbers to avoid ties in binary classification)

2. `weights`

- **Options:** `'uniform'` or `'distance'`
- **Effect:**
 - `'uniform'`: All neighbors contribute equally.
 - `'distance'`: Closer neighbors have more influence.
- **Tip:** Use `'distance'` when closer points are more likely to be correct.

3. `metric`

- **What it does:** Defines how distance is measured.
- **Common choices:**
 - `'minkowski'` (default, general form for Euclidean and Manhattan)
 - `'euclidean'`: Straight-line distance
 - `'manhattan'`: Sum of absolute differences
 - `'cosine'`: Angle between vectors (for text or high-dimensional data)