

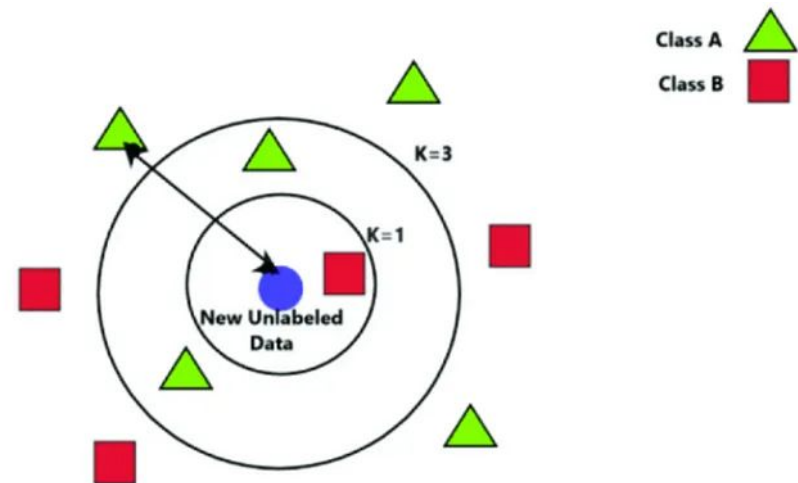
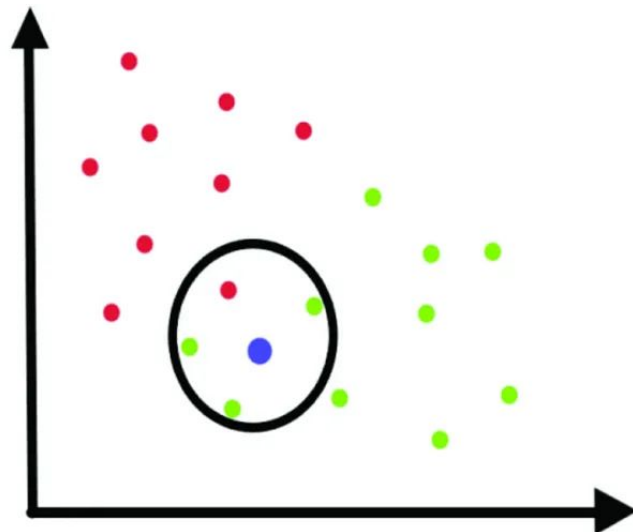


Lesson_9: The KNN algorithm

Ali Aburas, PhD

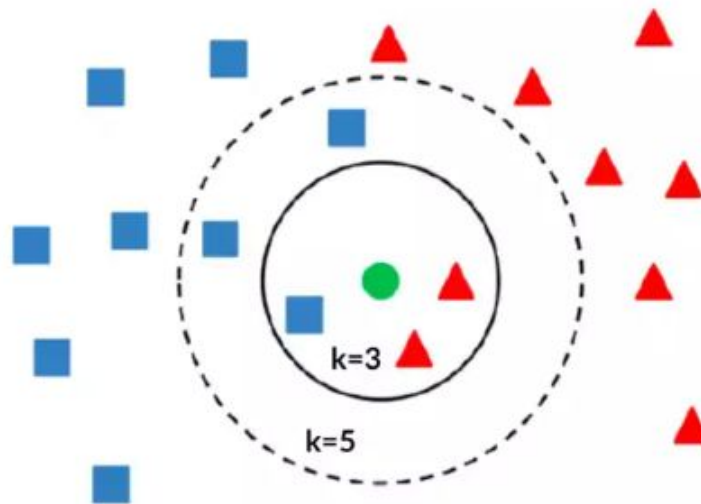
The k-NN algorithm

- **Assumption:** Similar Inputs have similar outputs.
- The K-Nearest Neighbors (KNN) algorithm stands out for its simplicity and effectiveness
- KNN is a **non-parametric** algorithm that can be used for both classification and regression.
- It doesn't require training in the traditional sense but makes decisions based on the training data when it's time to predict.



Choosing the Right Number of Neighbors

- The key to using KNN effectively is selecting the right value for “K,” which is the number of neighbors.
- If we choose $K = 1$, makes our model more prone to outliers and overfitting.
- Value of 1 means that we will consider only the closest (or nearest) neighbor to predict the class for our testing sample, and in majority of the cases it will lead to **overfitting**.
- If we choose a larger K, we can get a more general idea by considering a majority vote.



The k-nearest neighbors algorithm (KNN)

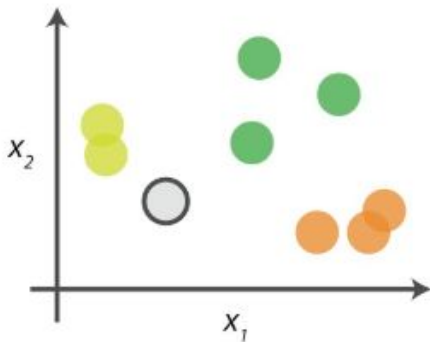


1. Data Collection and Preparation:
2. Choosing a Value for k:
3. Distance Calculation
4. Majority Vote (Classification) or Average (Regression):
 - a. For classification tasks: If the problem is binary, count the occurrences of each class among the k nearest neighbors and assign the class with the highest count to the testing data point.
 - b. For regression tasks: Take the average of the target values of the k nearest neighbors and assign this average as the predicted value for the testing data point.
5. Prediction and Evaluation:
 - a. Repeat steps 3-5 for all data points in the testing set.
 - b. Evaluate the algorithm's performance using appropriate metrics such as accuracy (for classification) or mean squared error (for regression).
6. Model Tuning and Validation:
7. Test the algorithm with different values of k to find the optimal value that provides the best performance on the validation set.

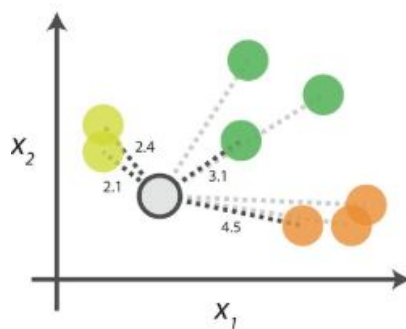
The k-nearest neighbours algorithm (KNN)

- When classifying an unseen dataset using a trained K-nn algorithm, it looks through the training data and finds the k training examples that are closest to the new example.
- It then assigns a class label to the new example based on a majority vote between those k training examples.
- if K is equal to 3, the algorithm will select the three closest data points to each case and classify it based on a majority vote based on the classes that those three adjacent points hold.

0. Look at the data



1. Calculate distances



2. Find neighbours

Point Distance	
...	2.1 → 1st NN
...	2.4 → 2nd NN
...	3.1 → 3rd NN
...	4.5 → 4th NN

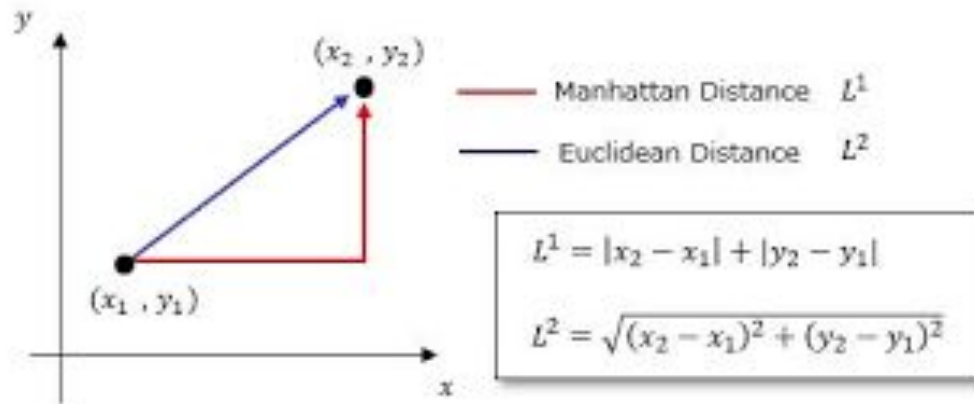
3. Vote on labels

Class	# of votes
	2
	1
	1

→ Class wins the vote!
Point is therefore predicted to be of class .

How Does KNN Work?

- KNN works based on feature similarity.
- To classify a new point, the algorithm calculates the “distance” between the new point and all other points in the dataset.
- The most common way to measure this distance is by using
 - **Euclidean distance:** Calculates the straight-line distance between two points in the feature space
 - **Manhattan Distance:** Calculates the distance between two points by summing the absolute differences between their coordinates along each dimension



- The choice of distance metric should align with the characteristics of your data and the problem you're trying to solve. Experimenting with different distance metrics and evaluating their impact on the KNN algorithm's performance can help you select the most appropriate one for your task

Feature Scaling and Cross-Validation



- One important step in using KNN is feature scaling. If one feature is much larger in value than another, it can dominate the distance calculation.
- For example, if we're using both age and income as features to classify a person's group, the income in thousands could skew the results.
- To avoid this, we normalize the data so that each feature has equal weight.

```
# Example Implementation
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

K-Fold Cross-Validation

- we can use cross-validation to select the best value of K.
- This involves dividing the dataset into multiple parts, training on some, and testing on others to find the most accurate K.
- Finally, the results of these five iterations are averaged to give an overall performance score.

```
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import cross_val_score

# Load the breast cancer dataset
cancer = datasets.load_breast_cancer()

# Create KFold object with 5 folds
kf = KFold(n_splits=5)

# Create a KNN classifier
clf = KNeighborsClassifier(n_neighbors= 5)

# Perform cross-validation and calculate accuracy
scores = cross_val_score(clf, cancer.data, cancer.target, cv=kf)
print(f"K-Fold Accuracy: {scores.mean()} ")
```


Stratified K-Fold

- Suppose your dataset has 80% samples of Class 1 and 20% samples of Class 2. If you use Stratified K-Fold, each fold will have approximately 80% samples of Class 1 and 20% samples of Class 2, ensuring that the model gets a fair representation of both classes during each iteration.
- Without stratification, one-fold might end up with only Class 1 samples and none from Class 2, which would not provide an accurate test of the model's ability to generalize across both classes.

```
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import cross_val_score

# Load the breast cancer dataset
cancer = datasets.load_breast_cancer()

# Create Stratified KFold object with 5 folds
skf = StratifiedKFold(n_splits= 5)

# Create a KNN classifier
clf = KNeighborsClassifier(n_neighbors= 5)

# Perform cross-validation and calculate accuracy
scores = cross_val_score(clf, cancer.data, cancer.target, cv=skf)
print(f"Stratified K-Fold Accuracy: {scores.mean()} ")
```

Advantages of KNN




- **Simple and Intuitive:** KNN is easy to understand and implement. It serves as a good starting point for beginners in machine learning.
- **Non-Parametric:** KNN is a non-parametric algorithm, meaning it doesn't make any assumptions about the underlying data distribution. This makes it versatile and applicable to a wide range of problems.
- **Adaptable to Different Data Types:** KNN can handle numerical and categorical data. It's also suitable for mixed data types, making it flexible in various scenarios.
- **No Training Phase:** KNN doesn't involve an explicit training phase. The algorithm stores the training data so that new data can be classified or predicted instantly.

Disadvantages of KNN



- **Computationally Expensive:** KNN's prediction time grows linearly with the size of the training dataset. For large datasets, predicting with KNN can be slow.
- **Sensitive to Noisy Data:** Outliers and noisy data points can significantly influence the prediction in KNN, especially with small values of “k.”
- **Choosing the Right “k”:** Selecting the appropriate value for “k” is crucial. A small value can lead to overfitting, while a large value can lead to over smoothing and loss of local patterns.
- **Imbalanced Data:** KNN doesn't handle imbalanced datasets well, as the class with more instances might dominate the prediction for new data points.
- **Distance Metric Sensitivity:** KNN's performance heavily depends on the choice of distance metric. Using an inappropriate distance metric can lead to suboptimal results.
- **Curse of Dimensionality:** When we have many features (high-dimensional data), KNN can perform poorly, as irrelevant features add noise to the distance calculation.

What is k-nearest neighbors used for?

- 
- **Classification:** KNN is frequently used for classification tasks where the goal is to assign a class label to a data point based on its neighbors
 - **Regression:** KNN can be used for regression tasks, where the goal is to predict a continuous numerical value based on the values of neighboring data points.
 - **Anomaly Detection:** KNN can help identify outliers or anomalies in a dataset by identifying data points significantly different from their neighbors.
 - **Collaborative Filtering:** In recommendation systems, KNN can identify similar users or items based on their behavior, helping to make personalized recommendations.
 - **Clustering:** KNN can be applied for clustering or grouping similar data points. It's used in data segmentation, customer segmentation, and market analysis.
 - **Image Segmentation:** KNN can segment images into regions with similar characteristics, which is helpful in image processing and computer vision tasks.