COM2004/3004

Data Driven Computing

Non-parametric classifiers Part 2

Dr. Po Yang

The University of Sheffield
po.yang@sheffield.ac.uk

Outline

Recap

Condensed Nearest Neighbour Classifier (The Hart Algorithm)

Confusion Matrices

Recap

Comparison between parametric and non-parametric classifiers

Parametric:

Non-Parametric:

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.
- **Model Complexity:** Fixed model complexity based on assumptions.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.
- **Model Complexity:** Flexible, complexity can grow with data.

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.
- **Model Complexity:** Fixed model complexity based on assumptions.
- **Training Data:** Parameters estimated from the data.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.
- **Model Complexity:** Flexible, complexity can grow with data.
- **Training Data:** Stores or summarizes training data.

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.
- **Model Complexity:** Fixed model complexity based on assumptions.
- **Training Data:** Parameters estimated from the data.
- **Generalization:** May not generalize well if assumptions are violated.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.
- **Model Complexity:** Flexible, complexity can grow with data.
- **Training Data:** Stores or summarizes training data.
- **Generalization:** Adapts to various data distributions.

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.
- **Model Complexity:** Fixed model complexity based on assumptions.
- **Training Data:** Parameters estimated from the data.
- **Generalization:** May not generalize well if assumptions are violated.
- **Overfitting:** Prone to overfitting when assumptions fail.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.
- **Model Complexity:** Flexible, complexity can grow with data.
- **Training Data:** Stores or summarizes training data.
- **Generalization:** Adapts to various data distributions.
- **Overfitting:** Less prone to overfitting due to flexibility.

Comparison between parametric and non-parametric classifiers

Parametric:

- **Assumptions:** Specific assumptions about data distribution.
- **Model Complexity:** Fixed model complexity based on assumptions.
- **Training Data:** Parameters estimated from the data.
- **Generalization:** May not generalize well if assumptions are violated.
- **Overfitting:** Prone to overfitting when assumptions fail.
- **Efficiency:** Computationally efficient due to fixed model structure.

Non-Parametric:

- **Assumptions:** Few to no assumptions about data distribution.
- **Model Complexity:** Flexible, complexity can grow with data.
- **Training Data:** Stores or summarizes training data.
- **Generalization:** Adapts to various data distributions.
- **Overfitting:** Less prone to overfitting due to flexibility.
- **Efficiency:** Can be computationally intensive, especially with large datasets.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

Weaknesses:

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

Weaknesses:

- **Computational Complexity:** KNN can be computationally expensive.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

Weaknesses:

- **Computational Complexity:** KNN can be computationally expensive.
- **Memory Usage:** High memory usage due to storing the entire dataset.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

Weaknesses:

- **Computational Complexity:** KNN can be computationally expensive.
- **Memory Usage:** High memory usage due to storing the entire dataset.
- **Sensitive to Outliers:** Prone to influence from outliers.

The Strengths and Weaknesses of KNN

Strengths:

- **Simplicity:** KNN is easy to understand and implement.
- **Non-Parametric:** It works well with a variety of data distributions.
- **Adaptability:** Can capture complex decision boundaries.
- **No Training Period:** Doesn't require a separate training phase.

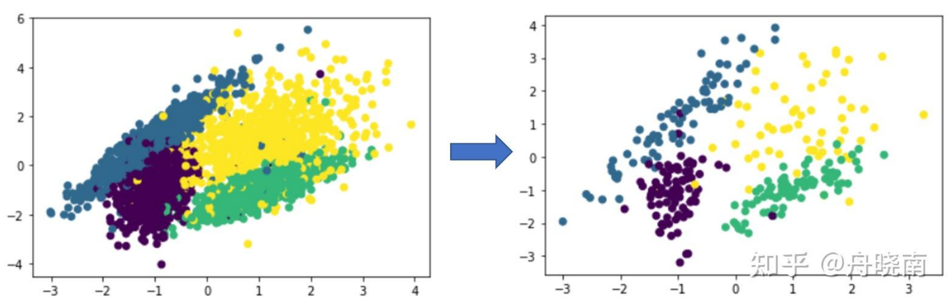
Weaknesses:

- **Computational Complexity:** KNN can be computationally expensive.
- **Memory Usage:** High memory usage due to storing the entire dataset.
- **Sensitive to Outliers:** Prone to influence from outliers.
- **Determining Optimal 'K':** Choosing the right 'K' can be challenging.

Condensed Nearest Neighbour Classifier (The Hart Algorithm)

The idea

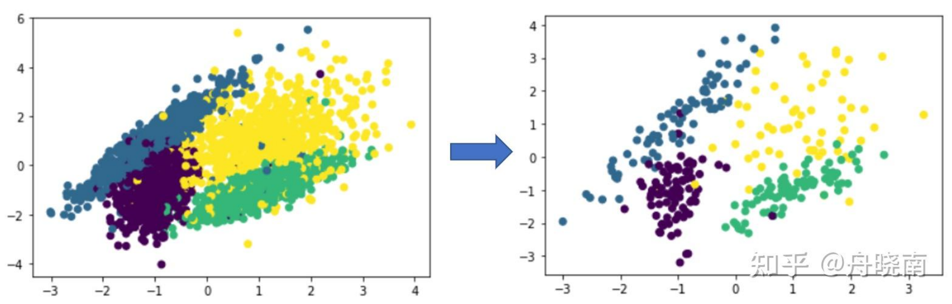
- The k-nearest neighbour m can be very slow if the training database is large.



Condensed nearest neighbour – Idea

The idea

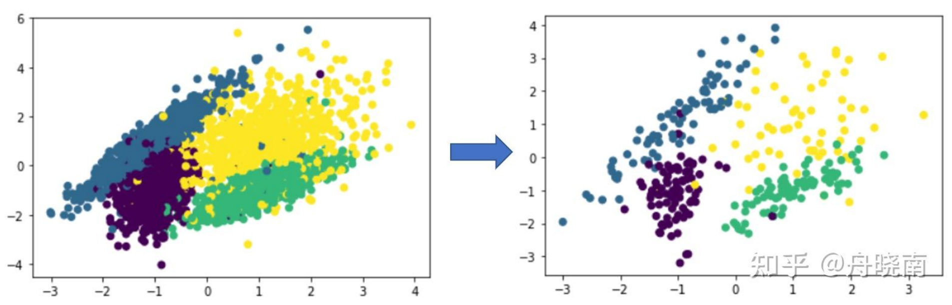
- The k-nearest neighbour m can be very slow if the training database is large.
- Many points will be ‘redundant’ i.e., having no influence on the decision boundary.



Condensed nearest neighbour – Idea

The idea

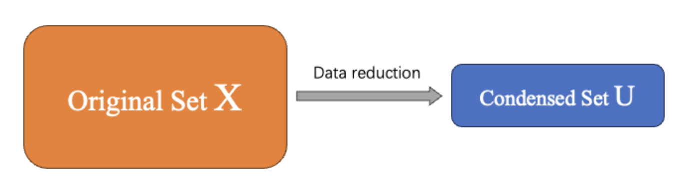
- The k-nearest neighbour m can be very slow if the training database is large.
- Many points will be 'redundant' i.e., having no influence on the decision boundary.
- If we can remove these points we can speed up classification (and reduce the classifier's memory footprint).



The Hart Algorithm for Data Reduction

Given an original training set X, remove a point $\mathbf{x_i}$ from X and add it to a `CondensedSet` U, then proceed iteratively,

1. For each element $\mathbf{x_i}$ in X
   - classify $\mathbf{x_i}$ using U as the training data.
   - If $\mathbf{x_i}$ is misclassified, remove it from X and add it to U
2. If any $\mathbf{x_i}$ has been added to U then go back to step 1
3. Replace X with U.



Example: Hart Algorithm with Euclidean Distance

Let's consider a simple example to illustrate the Hart algorithm using Euclidean distance.

Original Dataset X:

$$\begin{aligned} \mathbf{x_1} &: \quad [1, 2] - \text{Class A} \\ \mathbf{x_2} &: \quad [3, 4] - \text{Class B} \\ \mathbf{x_3} &: \quad [2, 1] - \text{Class A} \end{aligned}$$

Using the Hart algorithm with Euclidean distance and $K = 1$:

1. Start with $X = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}$ and an empty `CondensedSet` $U = \{\}$.
2. Randomly select $\mathbf{x_1}$ and add it to $U = \{\mathbf{x_1}\}$.

Example: Hart Algorithm with Euclidean Distance (Cont.)

Data Points $X = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}$: [1, 2] - Class A; [3, 4] - Class B; [2, 1] - Class A

Now $X = \{\mathbf{x_2}, \mathbf{x_3}\}$, $U = \{\mathbf{x_1}\}$.

3. Iterate through the remaining data points $X = \{\mathbf{x_2}, \mathbf{x_3}\}$ and compute the Euclidean distance from each point in the original dataset to the nearest point in $U = \{\mathbf{x_1}\}$.
   - Euclidean distance between $\mathbf{x_2}$ and $\mathbf{x_1}$:

$$d(\mathbf{x_1}, \mathbf{x_2}) = \sqrt{(3-1)^2 + (4-2)^2} = \sqrt{8}.$$

   Since $\mathbf{x_2}$ (Class B) is not correctly classified by U, it is added to $U = \{\mathbf{x_1}, \mathbf{x_2}\}$ and hence $X = \{\mathbf{x_3}\}$.

Example: Hart Algorithm with Euclidean Distance (Cont.)

Data Points $X = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}$: [1, 2] - Class A; [3, 4] - Class B; [2, 1] - Class A

Now $X = \{\mathbf{x_3}\}, \quad U = \{\mathbf{x_1}, \mathbf{x_2}\}$.

4. Repeat step 3 for the remaining data points X:
   - Euclidean distance between $\mathbf{x_3}$ and $\mathbf{x_1}, \mathbf{x_2}$:

$$d(\mathbf{x_1}, \mathbf{x_3}) = \sqrt{(2-1)^2 + (1-2)^2} = \sqrt{2}$$

$$d(\mathbf{x_2}, \mathbf{x_3}) = \sqrt{(3-2)^2 + (4-1)^2} = \sqrt{10}$$

   Since $\mathbf{x_3}$ (Class A) is correctly classified by U, it is not added to $U = \{\mathbf{x_1}, \mathbf{x_2}\}$ and hence $X = \{\}$.

Example: Hart Algorithm with Euclidean Distance (Cont.)

Data Points: [1, 2] - Class A; [3, 4] - Class B; [2, 1] - Class A

Now $X = \{\}, \quad U = \{\mathbf{x_1}, \mathbf{x_2}\}$.

5. The U now contains the reduced dataset:

CondensedSet U:

$$\begin{aligned} \mathbf{x_1} &: \quad [1,2] - \text{Class A} \\ \mathbf{x_2} &: \quad [3,4] - \text{Class B} \end{aligned}$$

This condensed dataset retains the necessary information for classification while being smaller than the original dataset.

Conclusion: Condensed Nearest Neighbour

Definition

The Condensed Nearest Neighbour (CNN) classifier is a data reduction technique commonly used in combination with k-Nearest Neighbour (k-NN) classification. It aims to create a smaller, more efficient dataset while preserving essential information for classification.

Conclusion: Condensed Nearest Neighbour

Definition

The Condensed Nearest Neighbour (CNN) classifier is a data reduction technique commonly used in combination with k-Nearest Neighbour (k-NN) classification. It aims to create a smaller, more efficient dataset while preserving essential information for classification.

Key Features

- Data Reduction: CNN reduces the size of a dataset by selecting a subset of informative instances.

Conclusion: Condensed Nearest Neighbour

Definition

The Condensed Nearest Neighbour (CNN) classifier is a data reduction technique commonly used in combination with k-Nearest Neighbour (k-NN) classification. It aims to create a smaller, more efficient dataset while preserving essential information for classification.

Key Features

- Data Reduction: CNN reduces the size of a dataset by selecting a subset of informative instances.
- Classification Focus: Although it works with various classifiers, it's often used with k-NN for efficient classification.

Conclusion: Condensed Nearest Neighbour

Definition

The Condensed Nearest Neighbour (CNN) classifier is a data reduction technique commonly used in combination with k-Nearest Neighbour (k-NN) classification. It aims to create a smaller, more efficient dataset while preserving essential information for classification.

Key Features

- Data Reduction: CNN reduces the size of a dataset by selecting a subset of informative instances.
- Classification Focus: Although it works with various classifiers, it's often used with k-NN for efficient classification.
- Retaining Information: CNN retains instances that can be confidently classified using their own features.

Conclusion: Condensed Nearest Neighbour

Definition

The Condensed Nearest Neighbour (CNN) classifier is a data reduction technique commonly used in combination with k-Nearest Neighbour (k-NN) classification. It aims to create a smaller, more efficient dataset while preserving essential information for classification.

Key Features

- Data Reduction: CNN reduces the size of a dataset by selecting a subset of informative instances.
- Classification Focus: Although it works with various classifiers, it's often used with k-NN for efficient classification.
- Retaining Information: CNN retains instances that can be confidently classified using their own features.
- Iterative Process: It iterates through the dataset, selecting instances, and continues until no more instances can be confidently classified.

Condensed Nearest Neighbour Classifier (The Hart Algorithm)

Notes

Condensed Nearest Neighbour

Notes

- The Hart algorithm might be very slow, i.e. requiring a lot of passes through the data. $\mathcal{O}(N^3)$ But it is only run once, i.e., it is like a training stage cost.
- The Hart algorithm is not decision boundary consistent ...
- ... nor is it guaranteed to find a minimal set.
- Different results depending on order points are considered in.
- Condensation removes redundancy so outlier data can become a problem; e.g., if using $k - NN$ may no longer be able to use large k
- Many variants have been developed, e.g., Reduced NN, Edited NN, Modified CNN, Fast CNN, etc, etc, (eg., see: Angiulli, "Fast CNN", ICML 2005)

Condensed Nearest Neighbour Classifier (The Hart Algorithm)

Summary

Summary

- The standard nearest neighbour algorithm can be impractical for very large training datasets
- Large datasets can be condensed using the Hart algorithm
- The smaller dataset will have similar (but not identical) decision boundaries to the original.

Confusion Matrices

What is a Confusion Matrix?

A confusion matrix is a fundamental tool in classification tasks.

- It helps assess the performance of a classification model.

What is a Confusion Matrix?

A confusion matrix is a fundamental tool in classification tasks.

- It helps assess the performance of a classification model.
- It provides a clear picture of correct and incorrect predictions.

Why is it Called a "Confusion" Matrix?

The name "confusion matrix" arises because it helps us understand the confusion between actual and predicted classes.

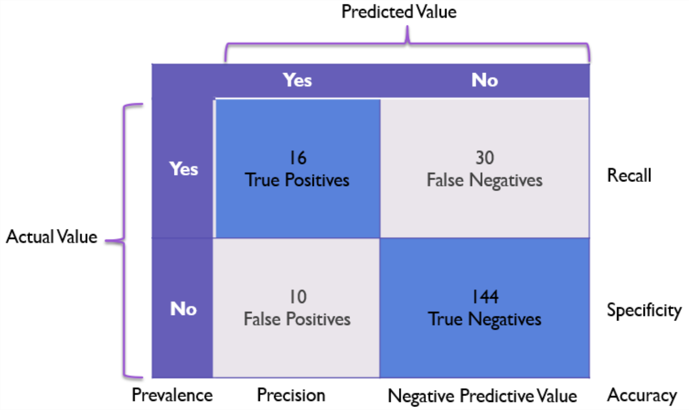
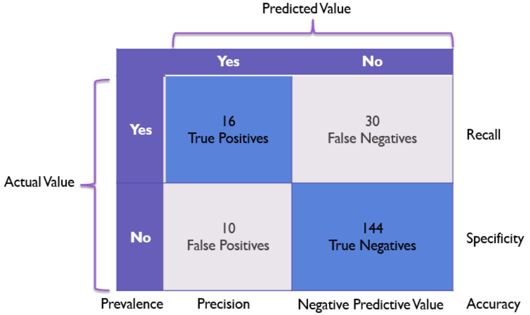


Figure : Example Confusion Matrix

Basic Elements of a Confusion Matrix

A confusion matrix is typically a 2x2 table with four elements:

- True Positives (TP): Correctly predicted positive instances.
- True Negatives (TN): Correctly predicted negative instances.
- False Positives (FP): Incorrectly predicted positive instances (Type I error).
- False Negatives (FN): Incorrectly predicted negative instances (Type II error).



Toy Example: Disease Diagnosis

Let's consider a toy example.

	Actual Disease Status	
Predicted Disease Status	Disease (+)	Healthy (-)
Disease (+)	**42 (TP)**	8 (FP)
Healthy (-)	3 (FN)	**47 (TN)**

To interpret the confusion matrix in disease diagnosis:

- **True Positives (TP)**: We correctly diagnosed 42 individuals with the disease.
- **False Positives (FP)**: We incorrectly diagnosed 8 healthy individuals as having the disease.
- **False Negatives (FN)**: We missed 3 cases of the disease and diagnosed them as healthy.
- **True Negatives (TN)**: We correctly identified 47 healthy individuals.

Comparison of Confusion matrix

What is a good confusion matrix?

- The elements are mainly concentrated on the diagonal.

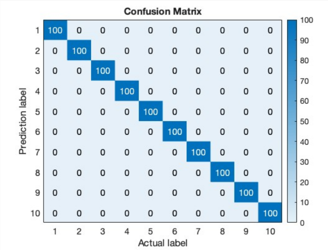


Figure : Perfect result

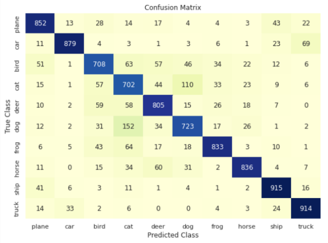


Figure : Good result

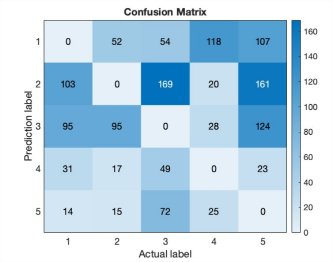


Figure : Terrible result

Confusion Matrices

Summary

Summary

- The confusion matrix is a convenient way to look at the classifiers performance.
- Correct responses appear along the diagonal
- Incorrect responses are off the diagonal.
- Note, it is often approximately symmetric because if a pair of classes are similar confusions happen in both directions.