Università degli Studi di Padova

Machine Learning
2024/2025

AMCO
Artificial Intelligence, Machine Learning and Control Research Group

Lecture #11 Classification & k-Nearest Neighbour

Gian Antonio Susto

Before starting: Exam – theoretic/numeric exercise part

A 45-60 minutes exam, multiple choices (main reference: slides)

➔ We'll make a 'simulation' during next week lecture (lecture 14 – 27th of March)

➔ The lecture will also be a recap of the first part of the course: if there are topics that you'd like to be discussed, let me know

We are preparing some example exercises that we'll be shared with you.

Before starting: Exam – programming part

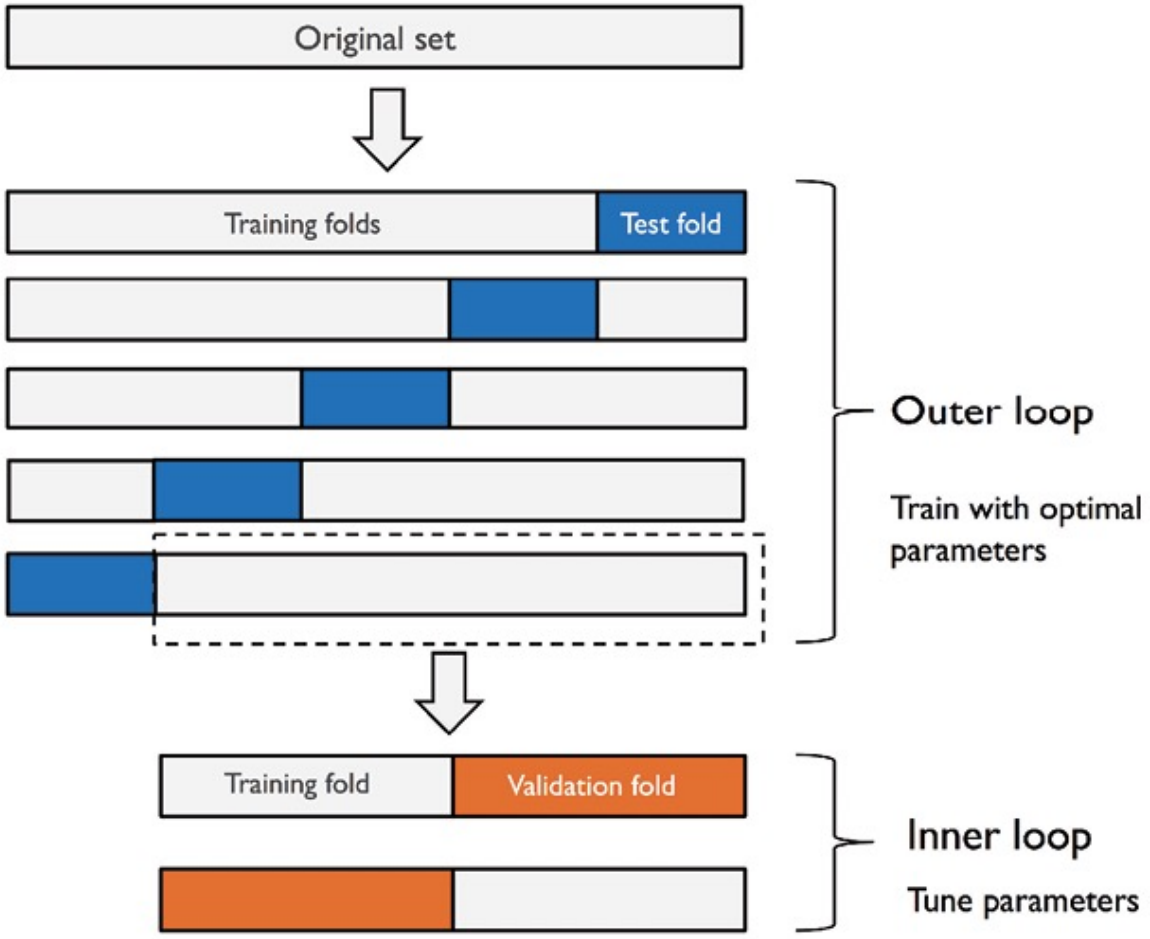
A 3-4 hours exam (done 15 minutes after the theoretic part)

- ➔ What we do in the labs are good examples of what will be asked during the exam
- ➔ Since we'd like you to concentrate on the ML commands, we'll provide you with a 'cheat sheet' with the basic python commands (remember you'll also have the help)
- ➔ By Easter (20th of April), we'll provide you with an example of the exam to be done at home: we will correct in class together (around the 15th of May)
- ➔ There will be a couple of Labs/lectures (on week 13, 19-23 of May) as additional lab simulations

Recap: Nested CV for Hyperparameter Tuning

Nested cycle of CV

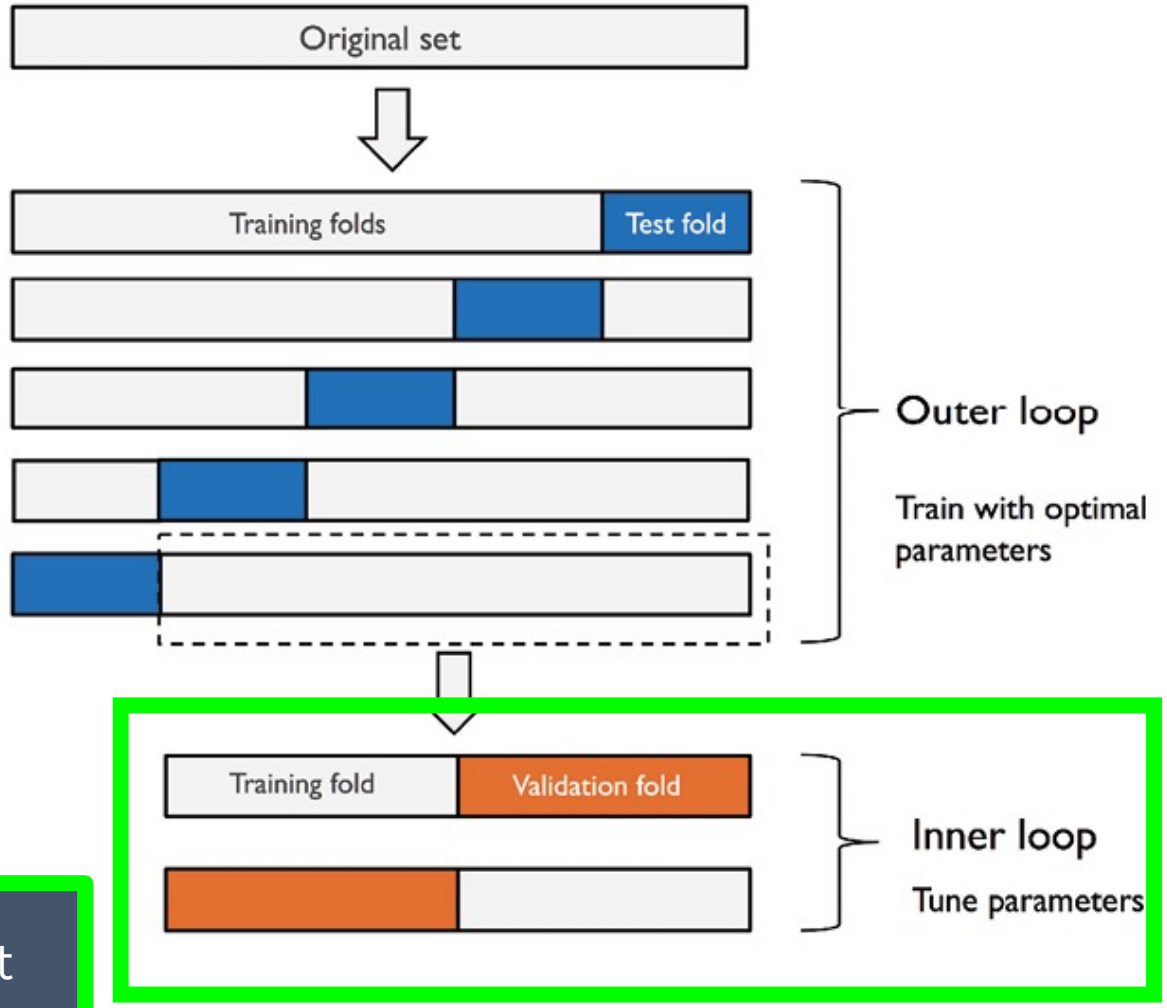
1. Inner
   - Training data for model construction
   - Validation data for choosing the hyperparameter(s)
2. Outer
   - Training data (training+validation) for model building
   - Test data for performance evaluation



Recap: Nested CV for Hyperparameter Tuning

Nested cycle of CV

1. Inner
   - Training data for model construction
   - Validation data for choosing the hyperparameter(s)
2. Outer
   - Training data (training+validation) for model building
   - Test data for performance evaluation



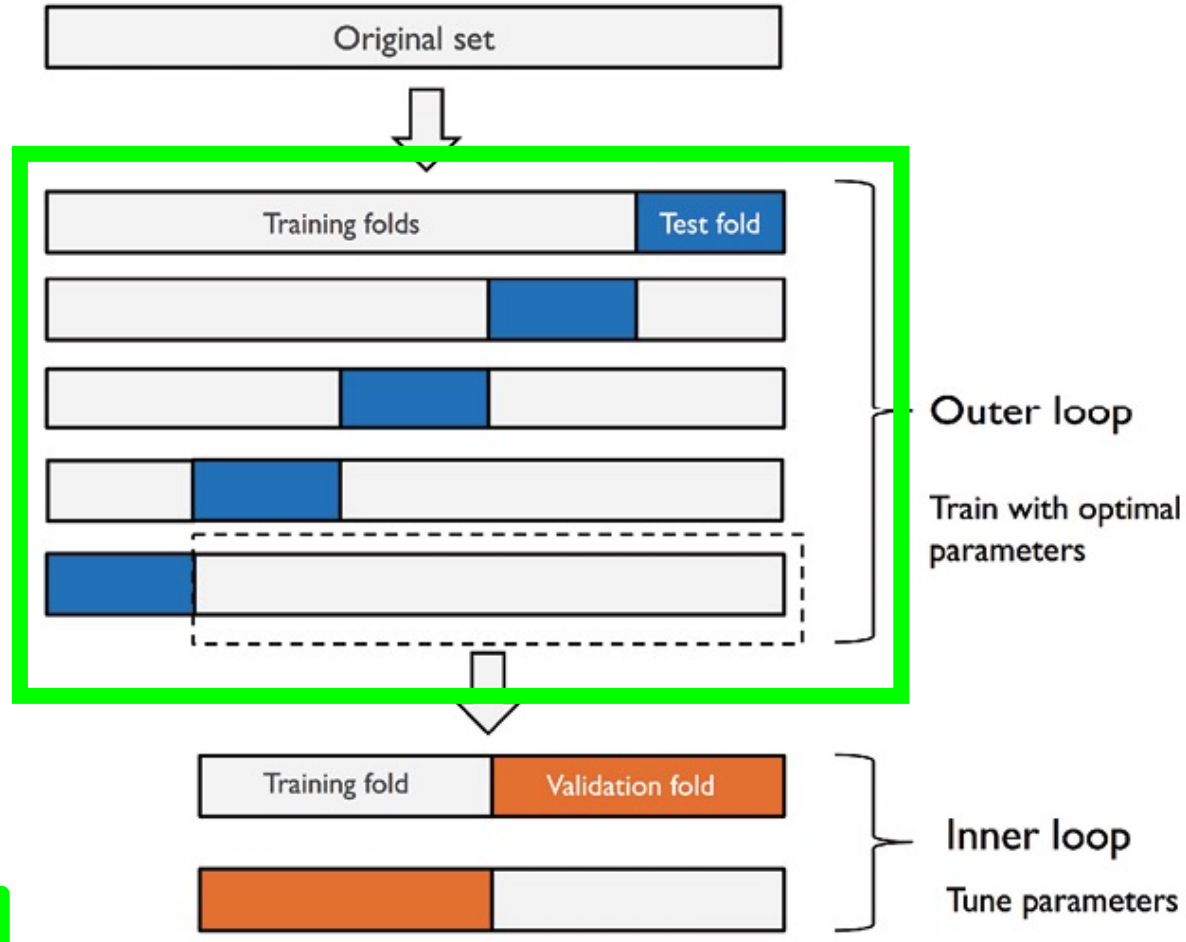
We are answering the question: what is the best hyperparameter for this approach?

Recap: Nested CV for Hyperparameter Tuning

Nested cycle of CV

1. Inner
   - Training data for model construction
   - Validation data for choosing the hyperparameter(s)
2. Outer
   - Training data (training+validation) for model building
   - Test data for performance evaluation

We are answering the question: what the performance will be?

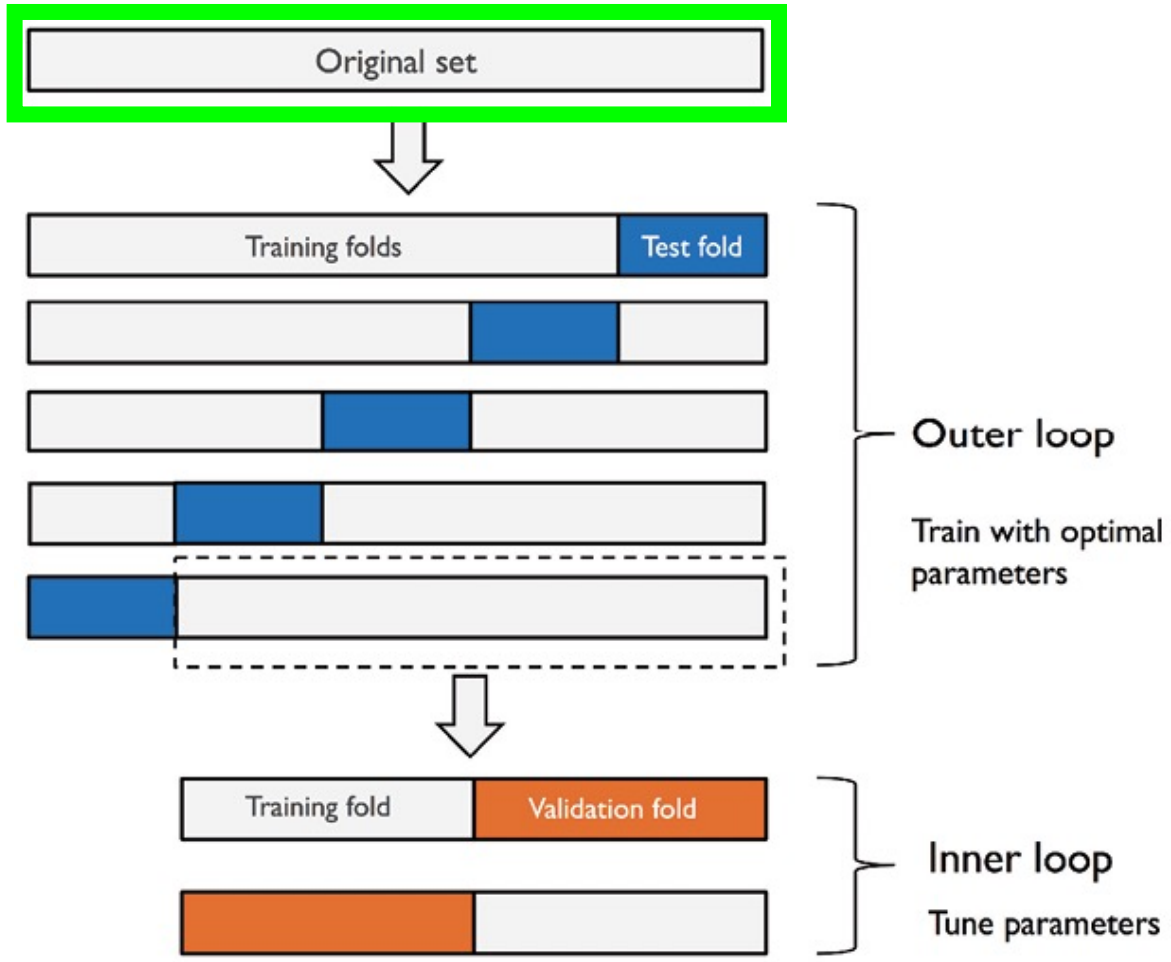


Recap: Nested CV for Hyperparameter Tuning

We are answering the question: what will be the 'final' model?

Nested cycle of C

1. Inner
   - Training data for model construction
   - Validation data for choosing the hyperparameter(s)
2. Outer
   - Training data (training+validation) for model building
   - Test data for performance evaluation



Supervised Tasks

Setup: available historical data

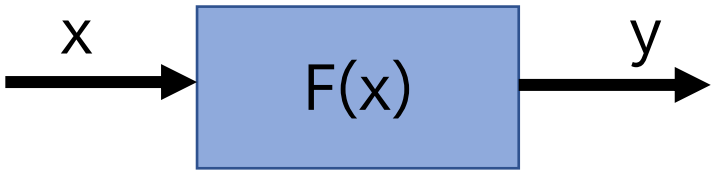
Data: (x – 'input', y – 'output')

Objective: earn a map/function that, when fed with new 'x', provides output estimates of 'y'

Depending on the nature of the output, we distinguish two subclasses of problems:

- If y is a continuous variable -> Regression Problem

- If y is a categorical variable -> Classification Problem



An example of a Classification Task

- Goal: estimating the Iris type
- Thanks to an historical data of *n* data sample with information such as
  - Class (‘setosa’, ‘virginica’, ’versicolur’) (output - *Y*)
  - Sepal length (input - *X*)
  - Sepal width (input - *X*)
  - Petal length (input - *X*)
  - Petal width (input - *X*)

An example of a Classification Task

- Goal: estimating the Iris type
- Thanks to an historical data of *n* data sample with information such as
  - Class ('setosa', 'virginica', 'versicolur') (output - *Y*)
  - Sepal length (input - *X*)
  - Sepal width (input - *X*)
  - Petal length (input - *X*)
  - Petal width (input - *X*)

The number of classes is indicated with *C* (in this case *C = 3*).

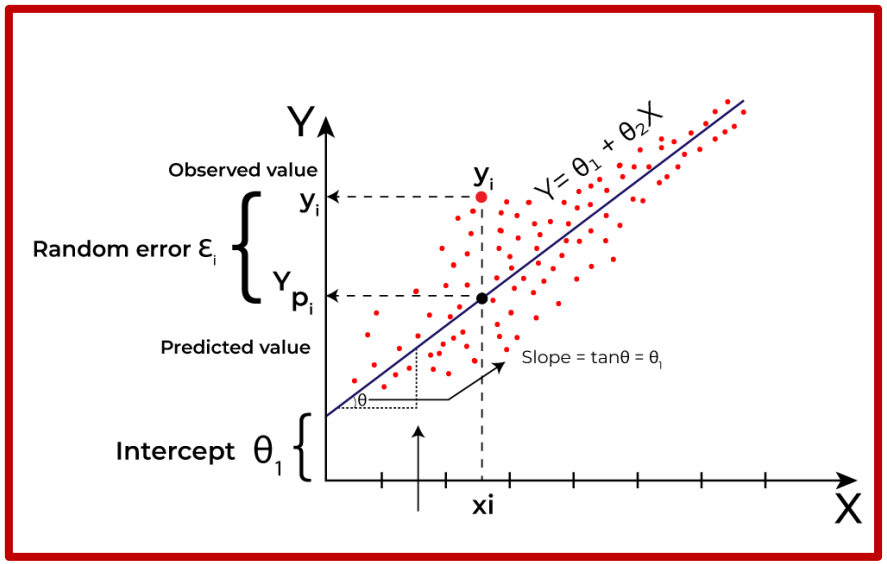
If *C = 2*, we are dealing with a 'binary classification' problem (ie. spam vs. not spam)

If *C > 2*, we are dealing with a 'multi-class classification' problem

An example of a Classification Task

- Goal: recognizing a song from a small (3-4 sec.) data sample
- Currently handling a *C = 10^8* class problem
- Historically, first results on Shazam talked about a ‘digital footprint (X)’: mainly a feature engineering approach made the solution feasible!
- Target, the song name

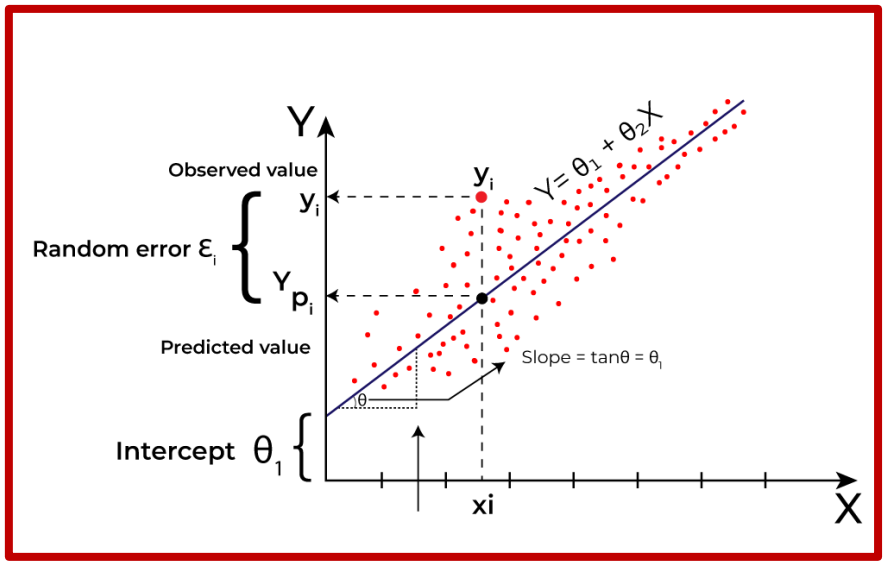
We'll use classification to show the variety of ML approaches form

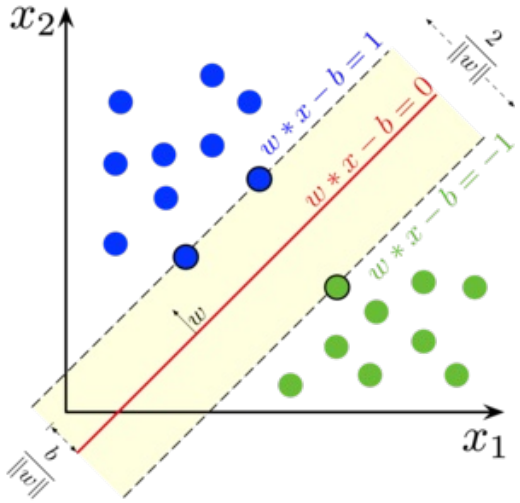


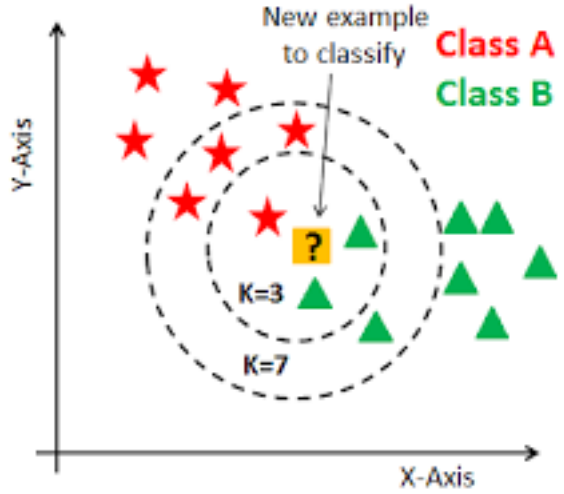
Up until now, all the regression approaches (OLS, Ridge Regression, LASSO, Elastic Net) we have seen they all shared the same form:

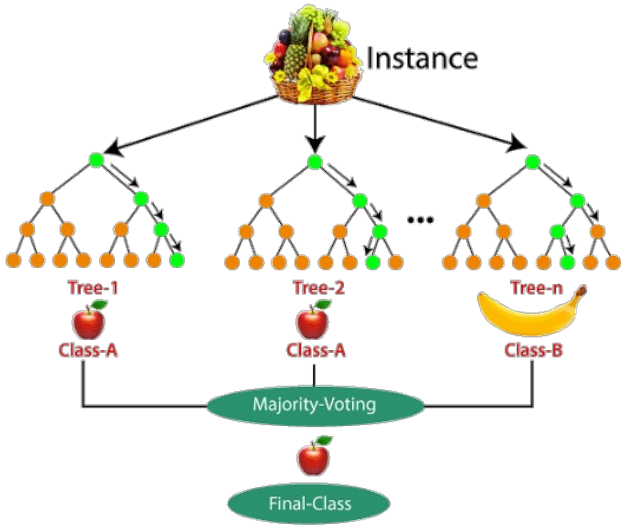
$$F(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

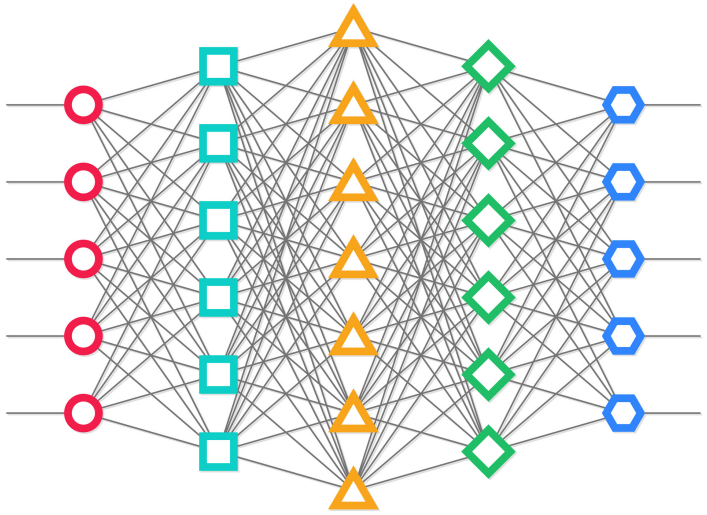
We'll use classification to show the variety of ML approaches form











How to evaluate a classifier?

With regression, we have seen two evaluation metrics

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Does it make sense on classification tasks? Other ideas?

Classification rate

The classification rate (or accuracy) is a performance metric used in classification tasks to measure the proportion of correctly classified instances over the total number of instances in a dataset.

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

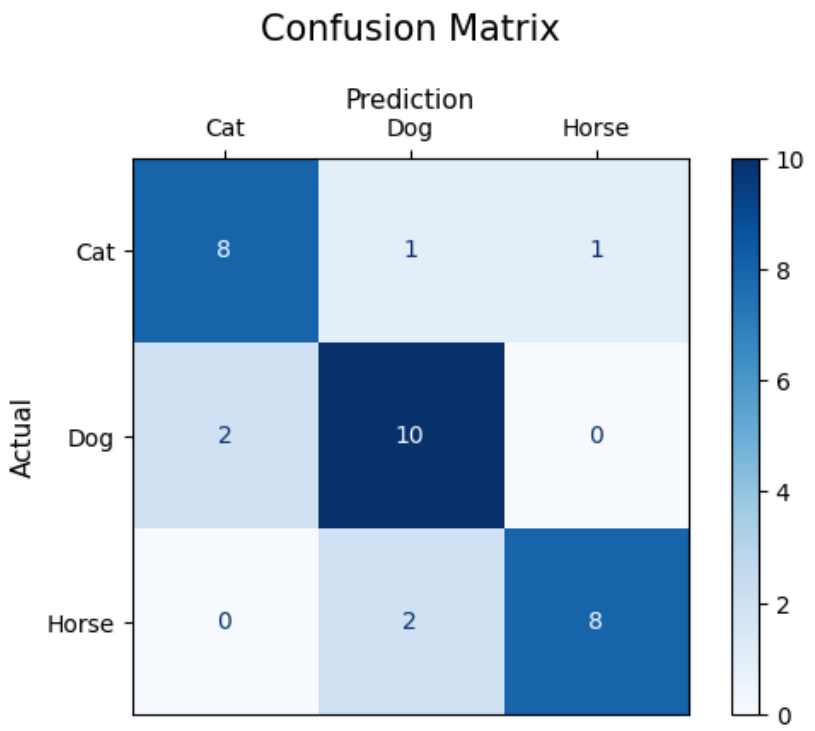
- A classification rate of 1 (100%) means a perfect classifier
- A classification rate of 1/C (100/C%) in classification means the model is performing as well as random guessing (in binary classification 0.5)

Classification rate

The classification rate (or accuracy) is a performance metric used in classification tasks to measure the proportion of correctly classified instances over the total number of instances in a dataset.

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- A classification rate of 1 (100%) means a perfect classifier
- A classification rate of 1/C (100/C%) in classification means the model is performing as well as random guessing (in binary classification 0.5)
- Confusion matrix is a useful representation

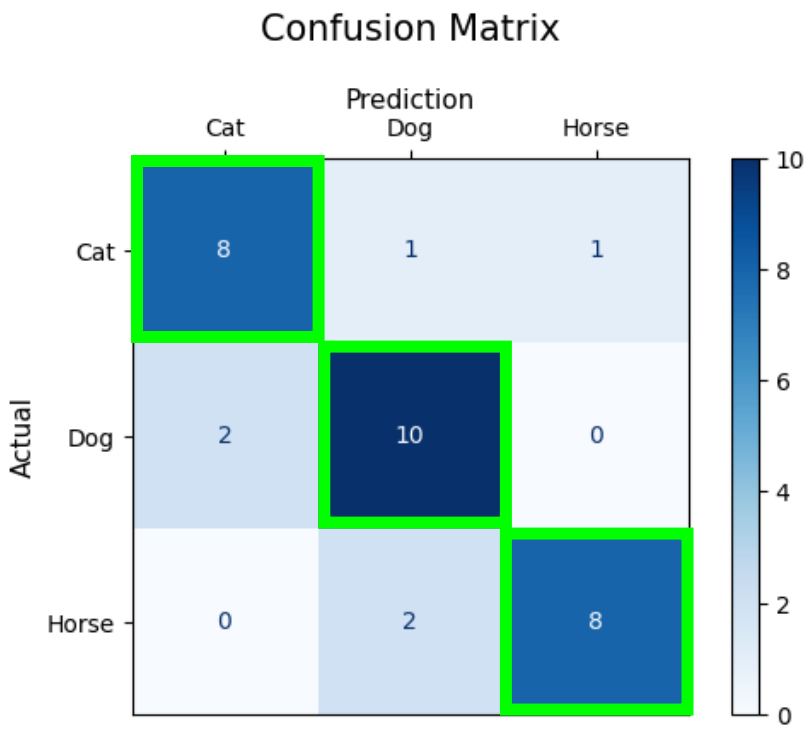


Classification rate

The classification rate (or accuracy) is a performance metric used in classification tasks to measure the proportion of correctly classified instances over the total number of instances in a dataset.

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- A classification rate of 1 (100%) means a perfect classifier
- A classification rate of 1/C (100/C%) in classification means the model is performing as well as random guessing (in binary classification 0.5)
- Confusion matrix is a useful representation



Binary classification: ‘Positive’ and ‘Negative’ classes

In binary classification, the terms "positive" and "negative" refer to how classes are assigned in the problem. These terms are used to define false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN) when evaluating model performance.

- The positive class is the one that represents the condition or outcome of interest.
- The negative class typically represents the absence of the condition.

Problem	Positive Class (1)	Negative Class (0)
Disease diagnosis	Patient **has** the disease	Patient **does not** have the disease
Spam detection	Email is **spam**	Email is **not spam** (ham)
Credit risk	Loan **defaults**	Loan **does not default**
Fraud detection	Transaction is **fraudulent**	Transaction is **legitimate**

Binary classification: ‘Positive’ and ‘Negative’ classes

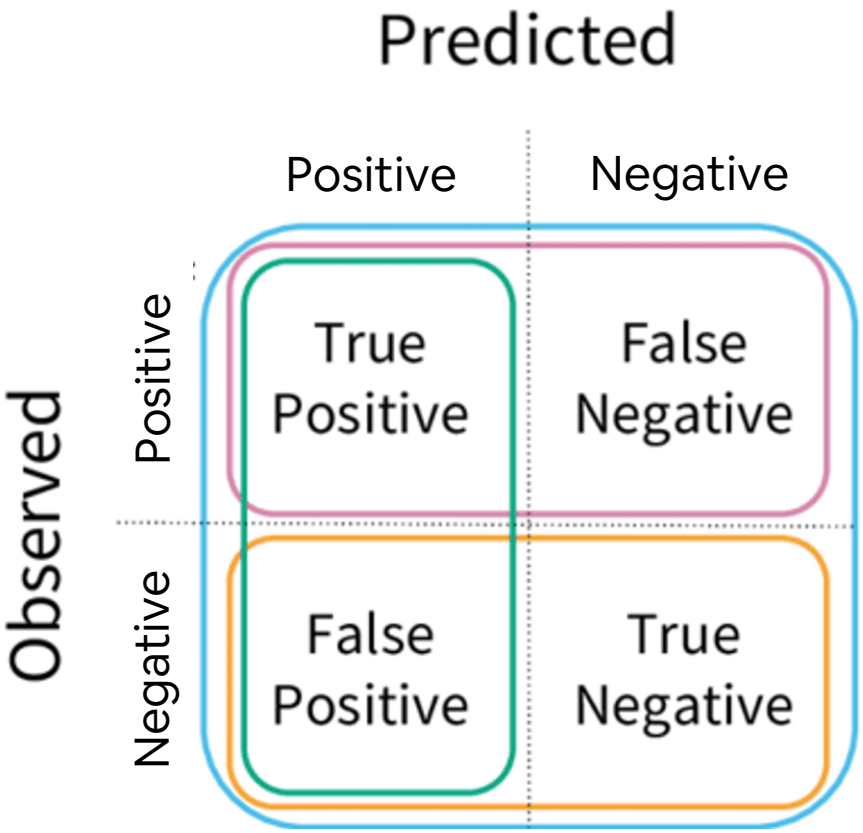
In binary classification, the terms "positive" and "negative" refer to how classes are assigned in the problem. These terms are used to define false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN) when evaluating model performance.

- The positive class is the one that represents the condition or outcome of interest.
- The negative class typically represents the absence of the condition.

Type I error (false positive)

Type II error (false negative)

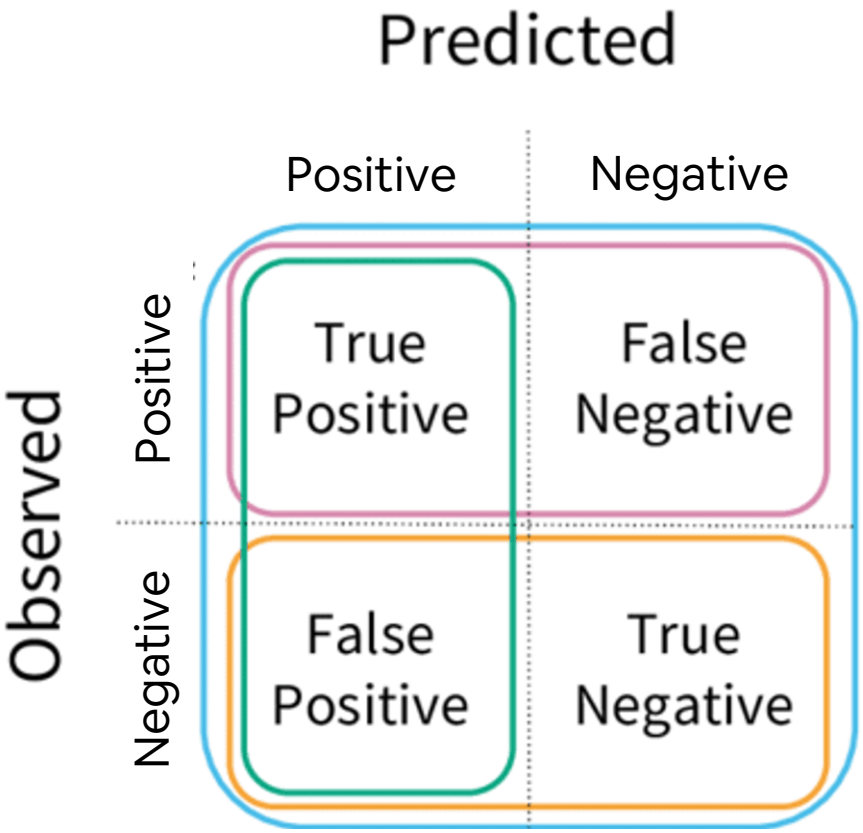
Binary classification: ‘Positive’ and ‘Negative’ classes



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The classification rate! Not always the best metric...

Binary classification: ‘Positive’ and ‘Negative’ classes



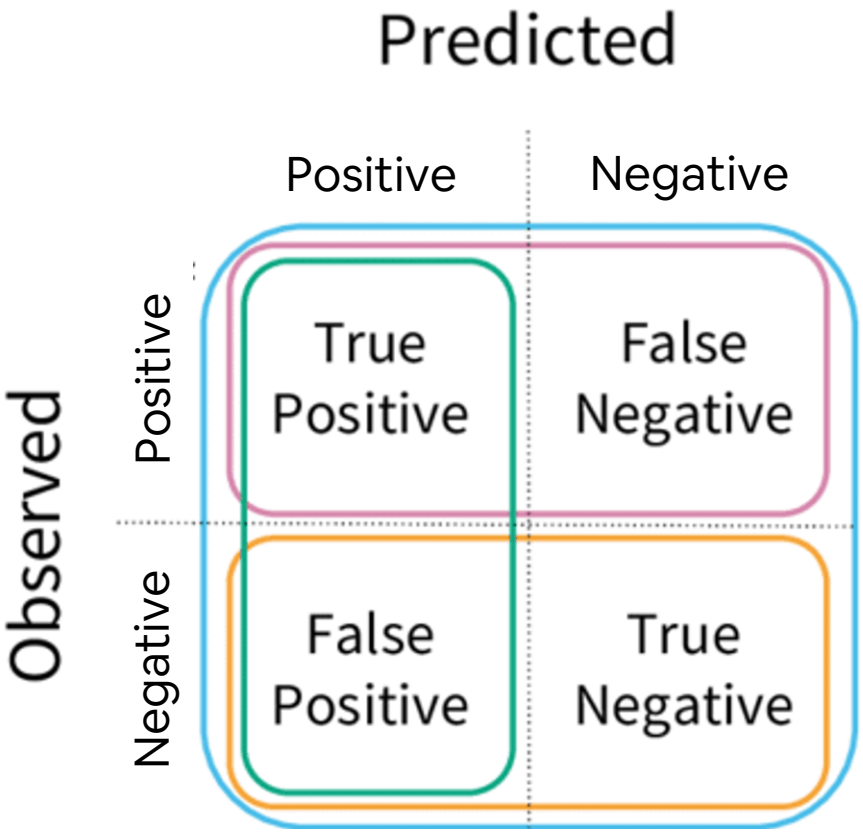
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The classification rate! Not always the best metric...

	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	27	6	81.81
Non-Spam (Actual)	10	57	85.07
Overall Accuracy			83.44

	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	0	10	0.0
Non-Spam (Actual)	0	990	100.0
Overall Accuracy			99

Binary classification: ‘Positive’ and ‘Negative’ classes



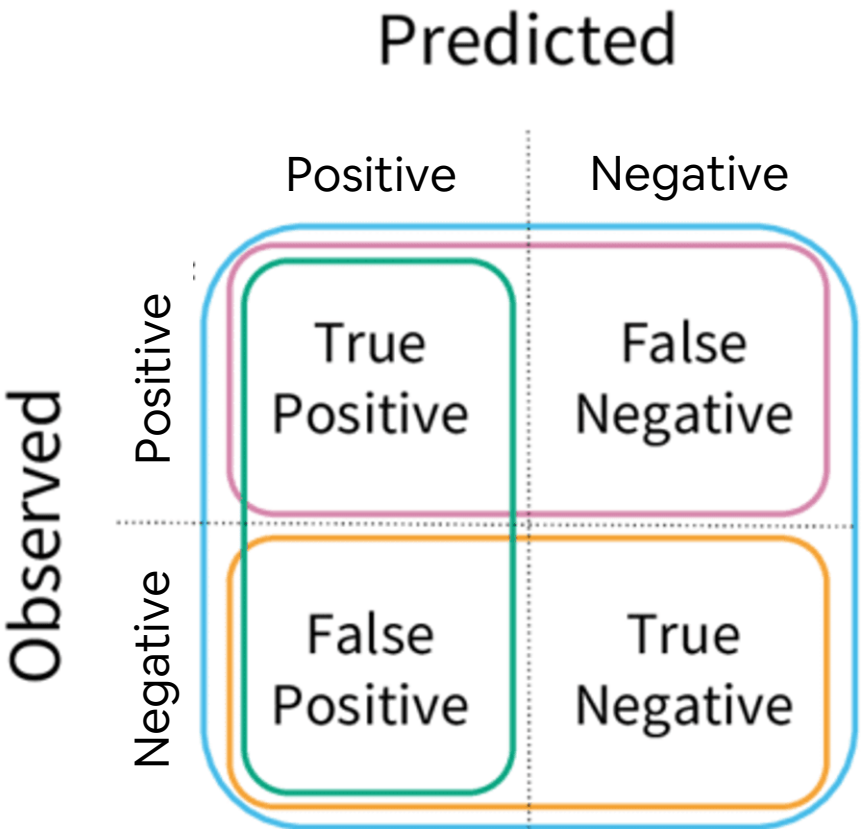
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity tells us how well a model avoids false alarms by correctly identifying the negative cases.

It is important when false positives (FP) are costly or problematic.

Binary classification: ‘Positive’ and ‘Negative’ classes



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

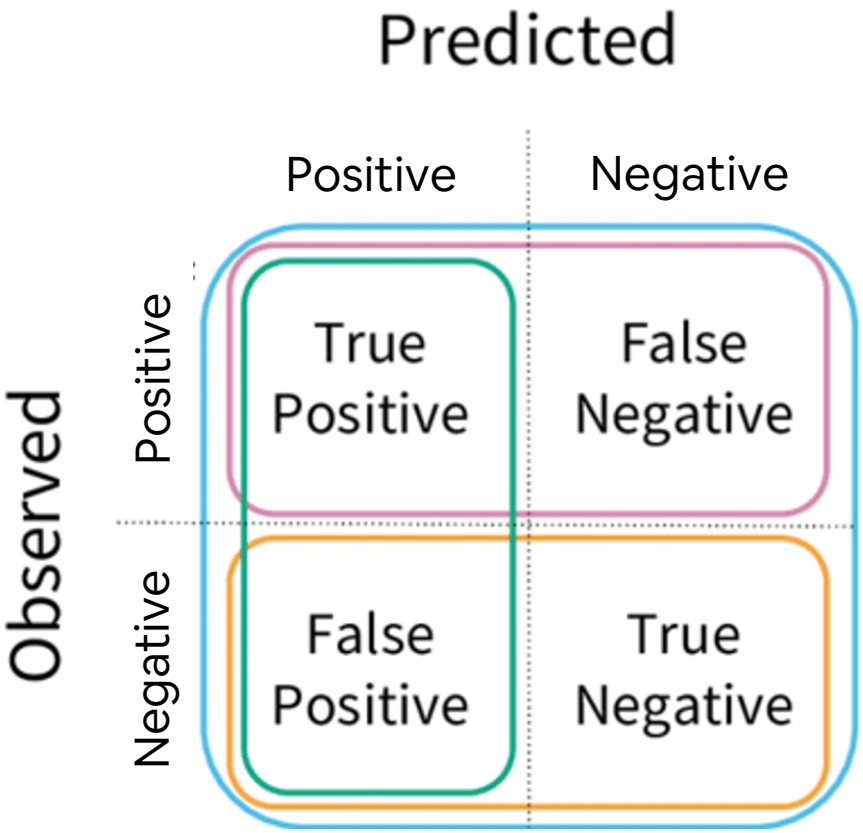
$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision tells us how reliable the positive predictions are. It is important when false positives (FP) are costly or misleading.

We choose this instead of specificity if we want to focus on the quality of positive predictions instead of the negative ones

Binary classification: ‘Positive’ and ‘Negative’ classes



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall (also called Sensitivity or True Positive Rate) measures how well a model identifies actual positive cases.

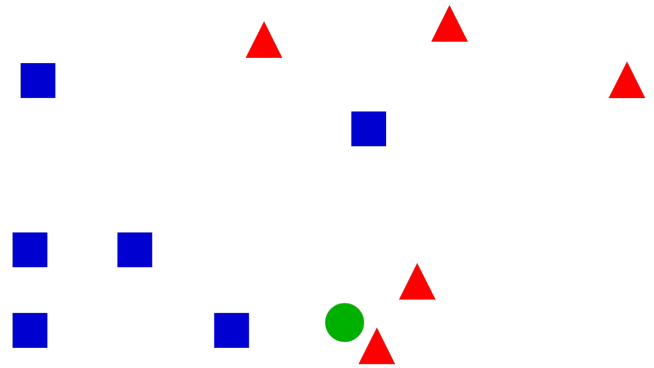
It answers the question: ‘Of all the actual positive cases, how many did the model correctly classify?"

Simplest classification approach: k-Nearest Neighbors (k-NN)

Simply putting it:

‘A new observation is assigned to the class that appears most frequently among its k nearest neighbors in the training data.’

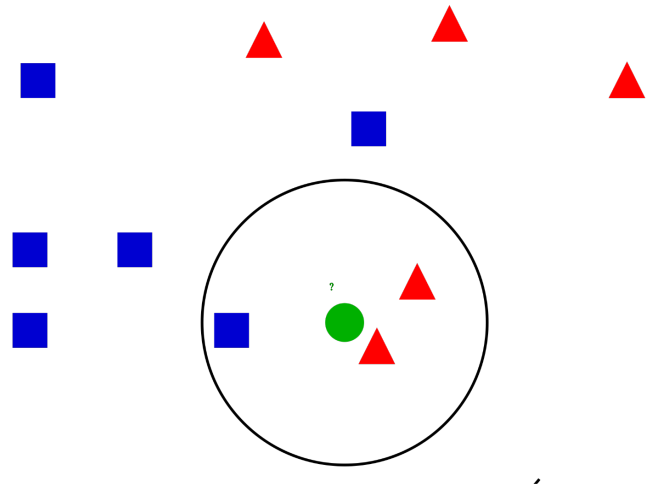
Green: new data point
Blue and Red: historical data of 2 different classes



Simplest classification approach: k-Nearest Neighbors (k-NN)

Simply putting it:

‘A new observation is assigned to the class that appears most frequently among its k nearest neighbors in the training data.’



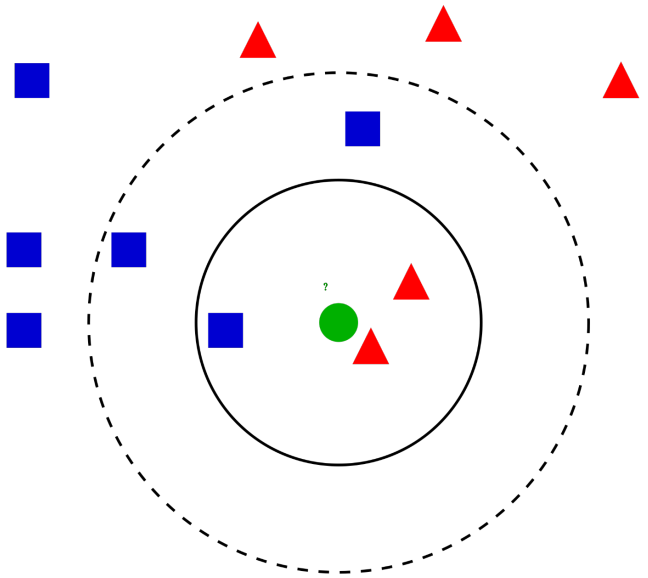
Green: new data point
Blue and Red: historical data of 2 different classes

k = 1, classified as Red
k = 3, classified as Red

Simplest classification approach: k-Nearest Neighbors (k-NN)

Simply putting it:

‘A new observation is assigned to the class that appears most frequently among its k nearest neighbors in the training data.’



Green: new data point
Blue and Red: historical data of 2 different classes

k = 1, classified as Red
k = 3, classified as Red
k = 5, classified as Blue

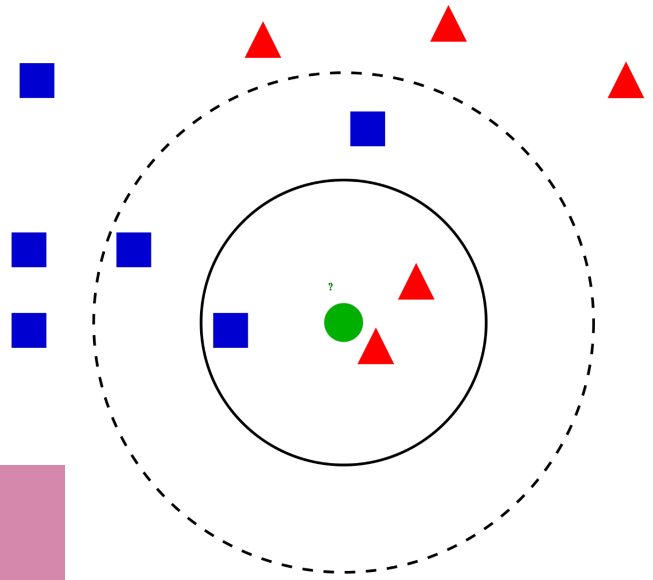
Simplest classification approach: k-Nearest Neighbors (k-NN)

Simply putting it:

‘A new observation is assigned to the class that appears most frequently among its k nearest neighbors in the training data.’

Green: new data point
Blue and Red: historical data of 2 different classes

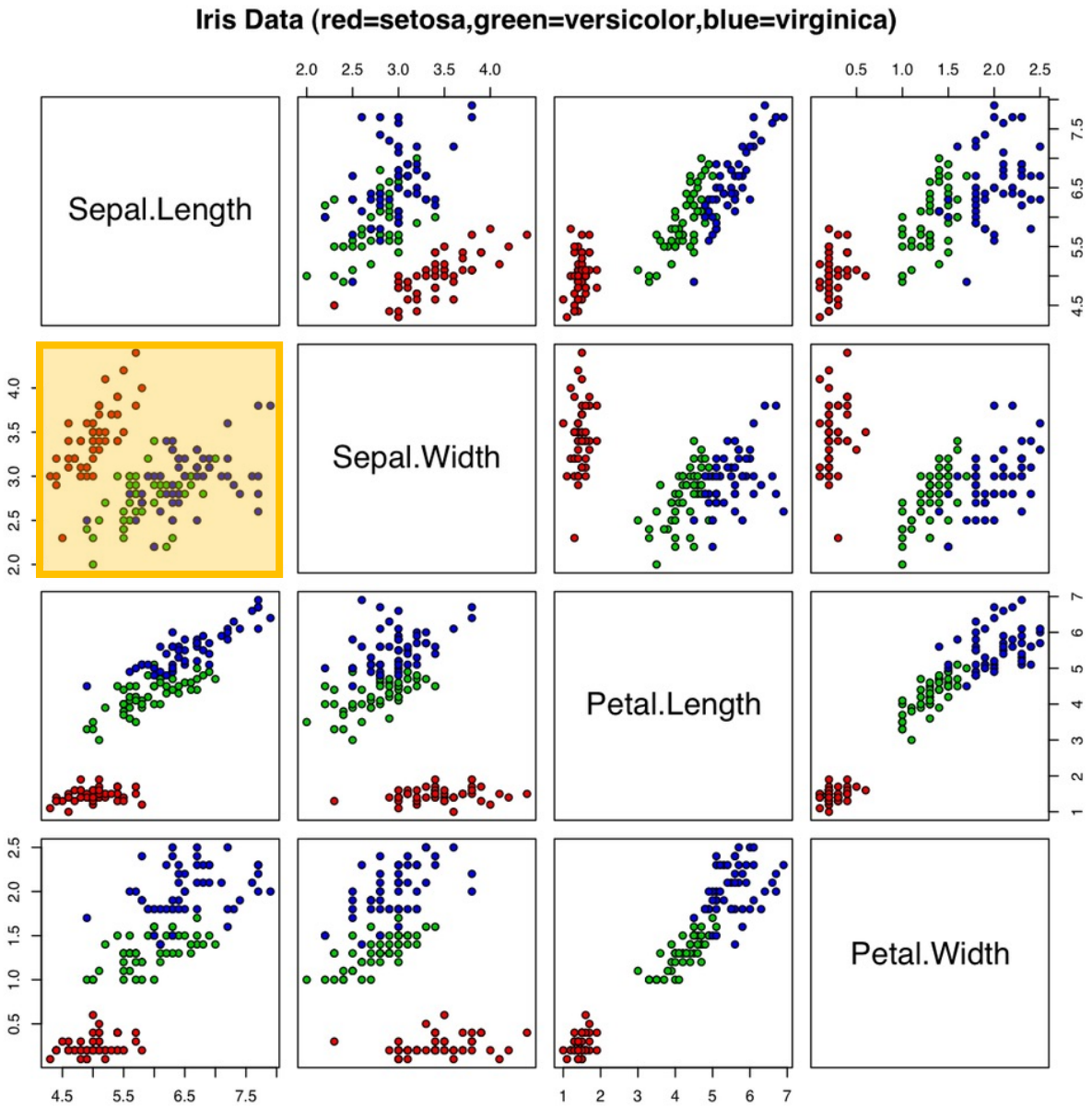
k = 1, classified as Red
k = 3, classified as Red
k = 5, classified as Blue



k is an hyperparameter!

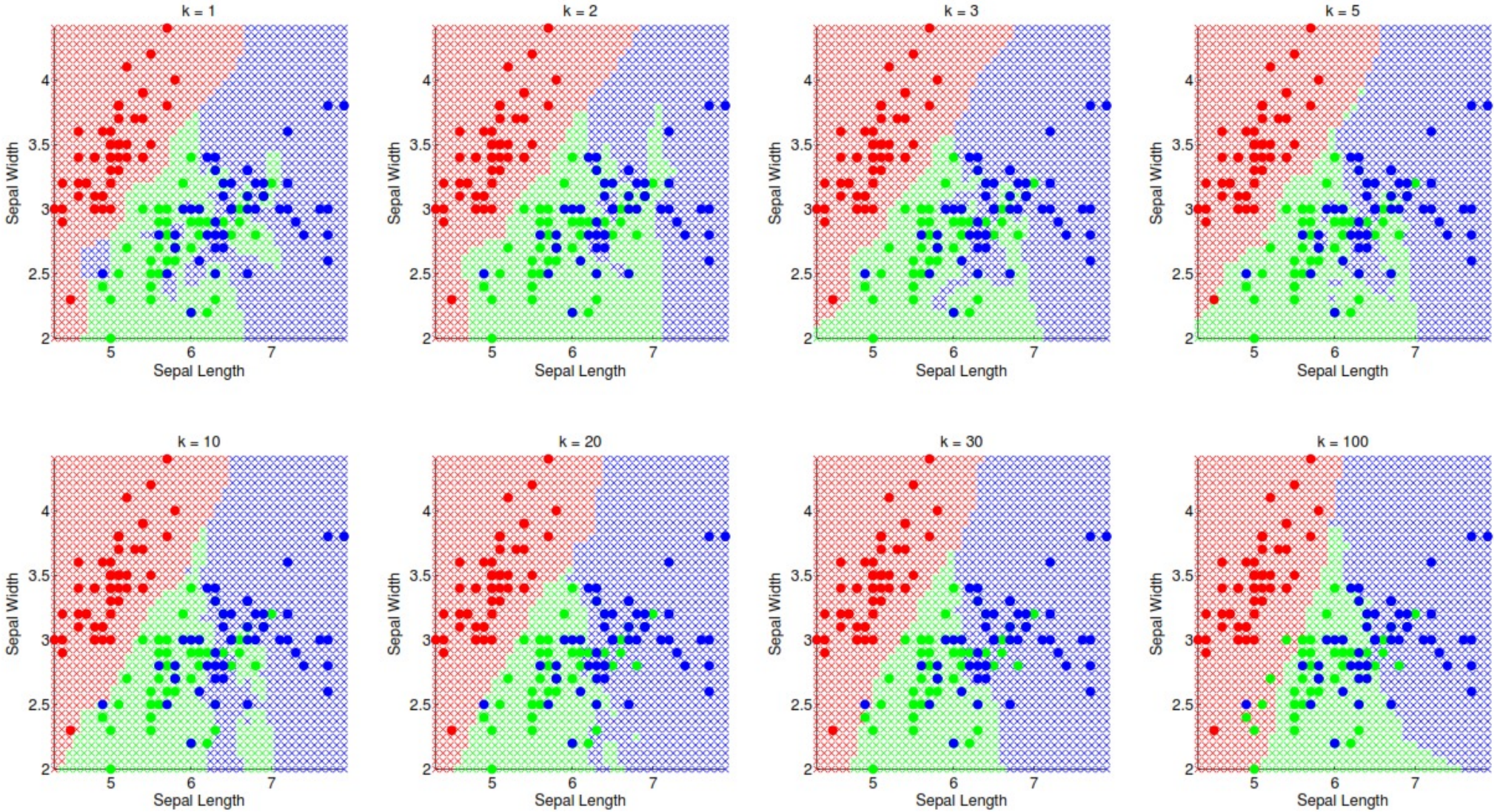
With different choices we get different results and effects!

k-Nearest Neighbors (k-NN) in action: Iris Dataset



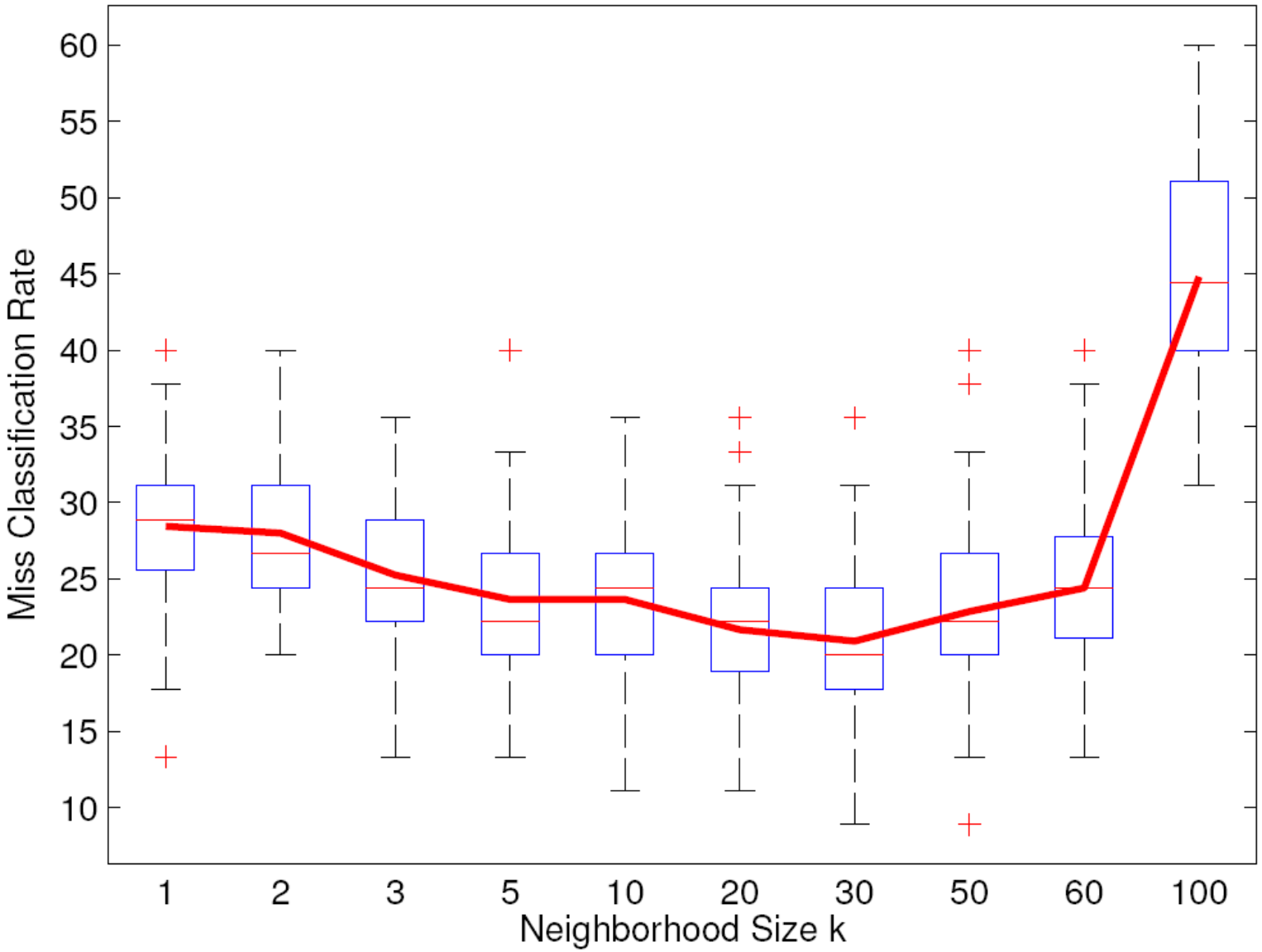
Wikipedia

k-Nearest Neighbors (k-NN) in action: Iris Dataset

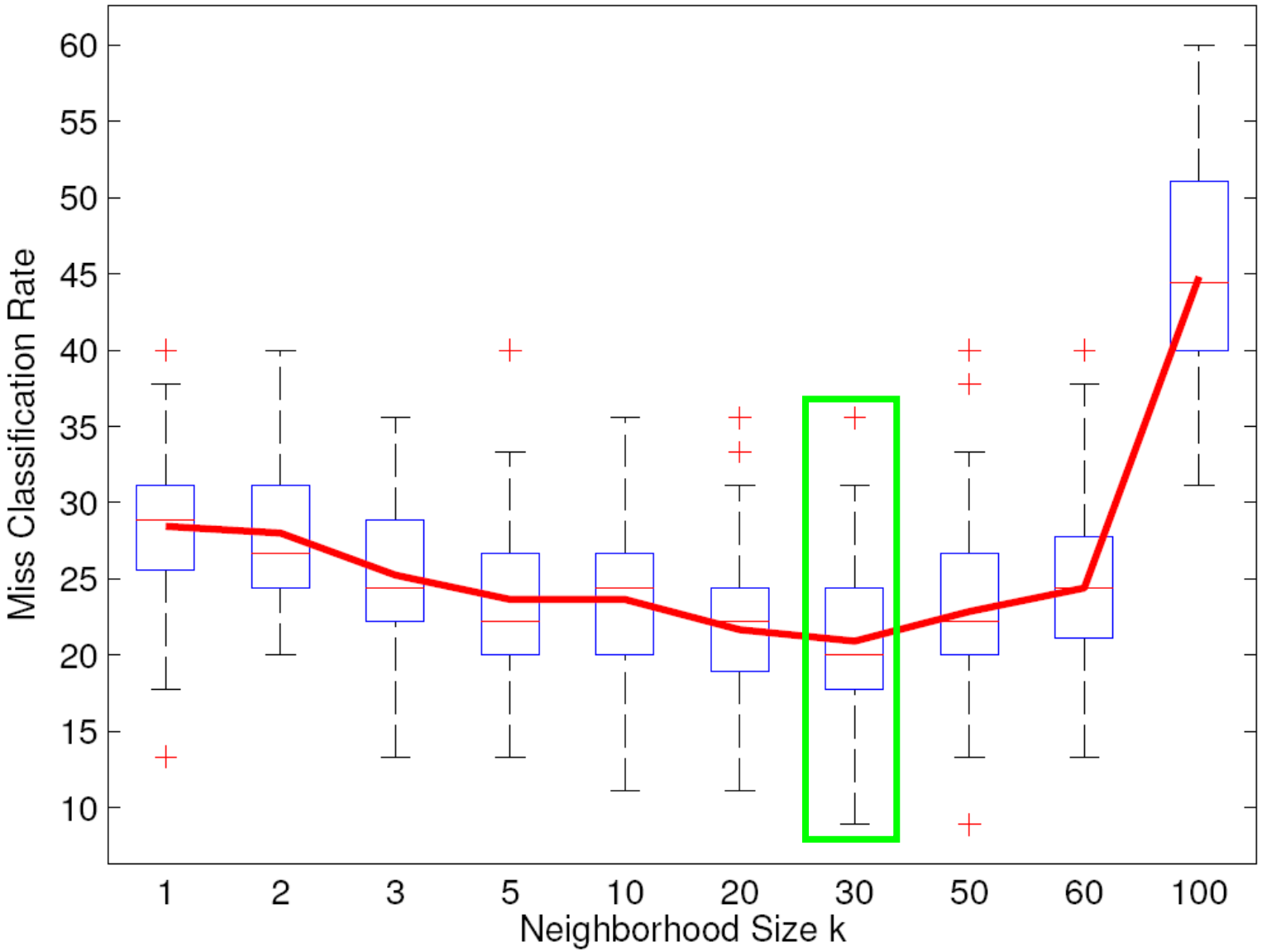


Which one do you think is the best?

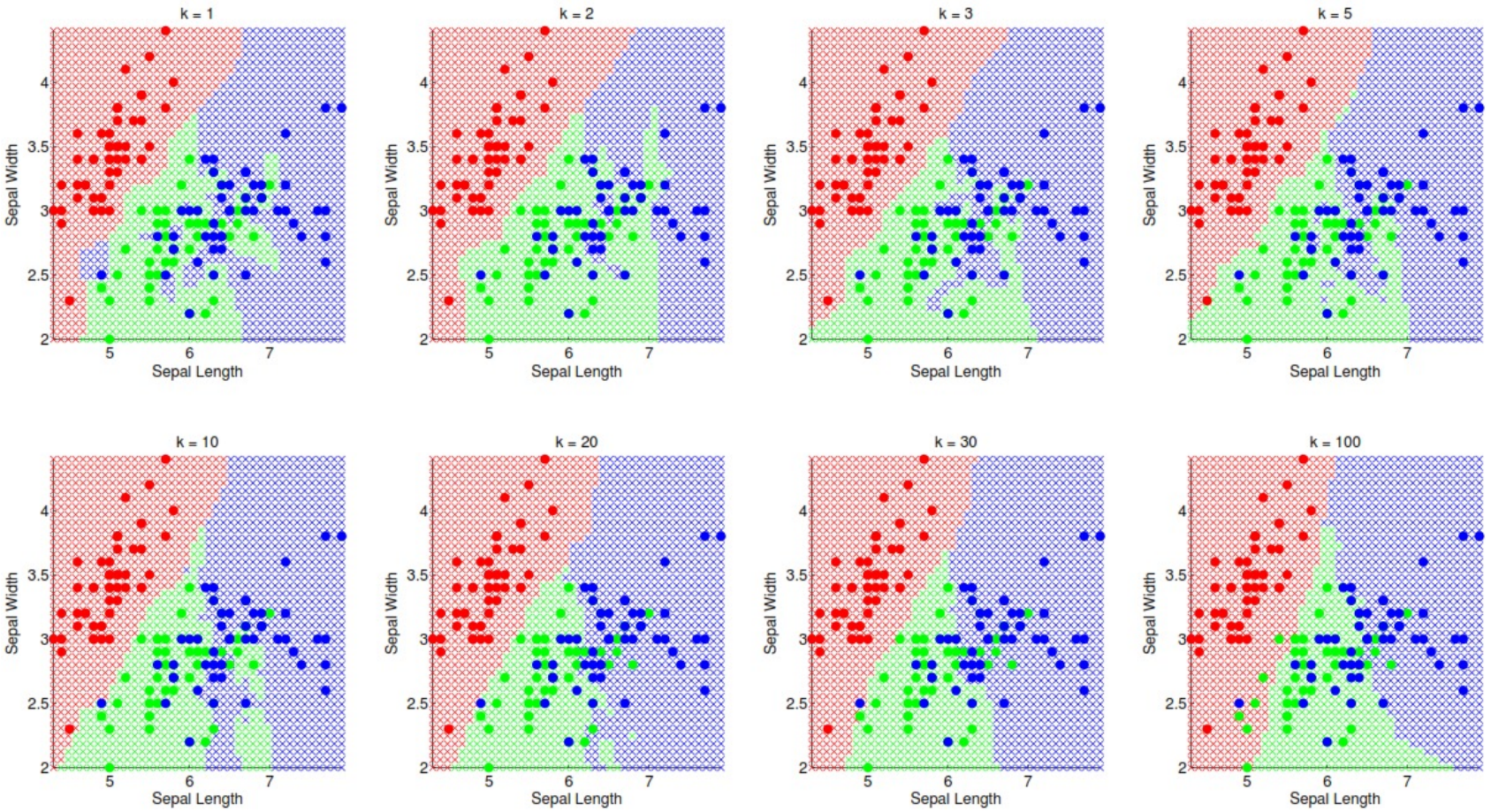
k-Nearest Neighbors (k-NN) in action: Iris Dataset



k-Nearest Neighbors (k-NN) in action: Iris Dataset

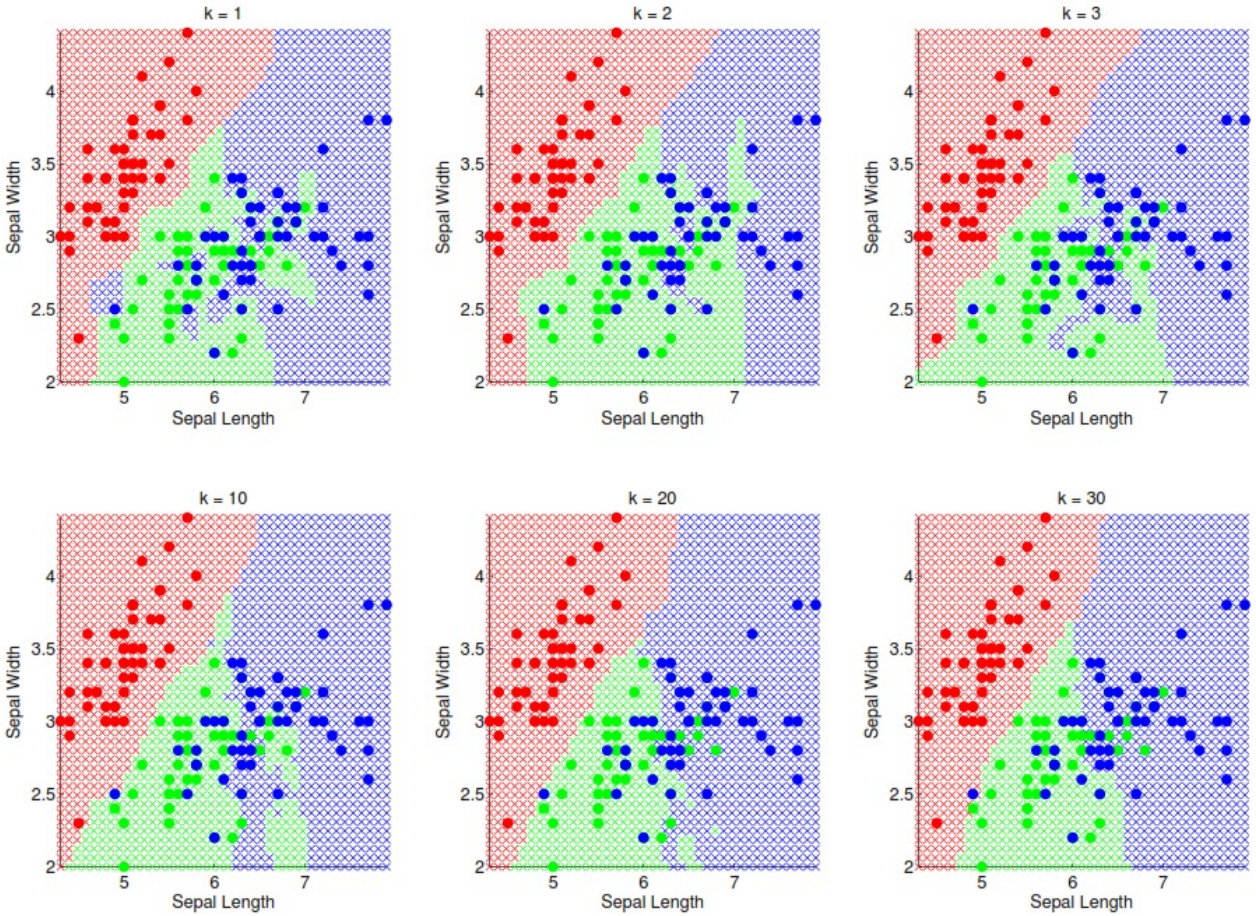


k-Nearest Neighbors (k-NN) in action: Iris Dataset

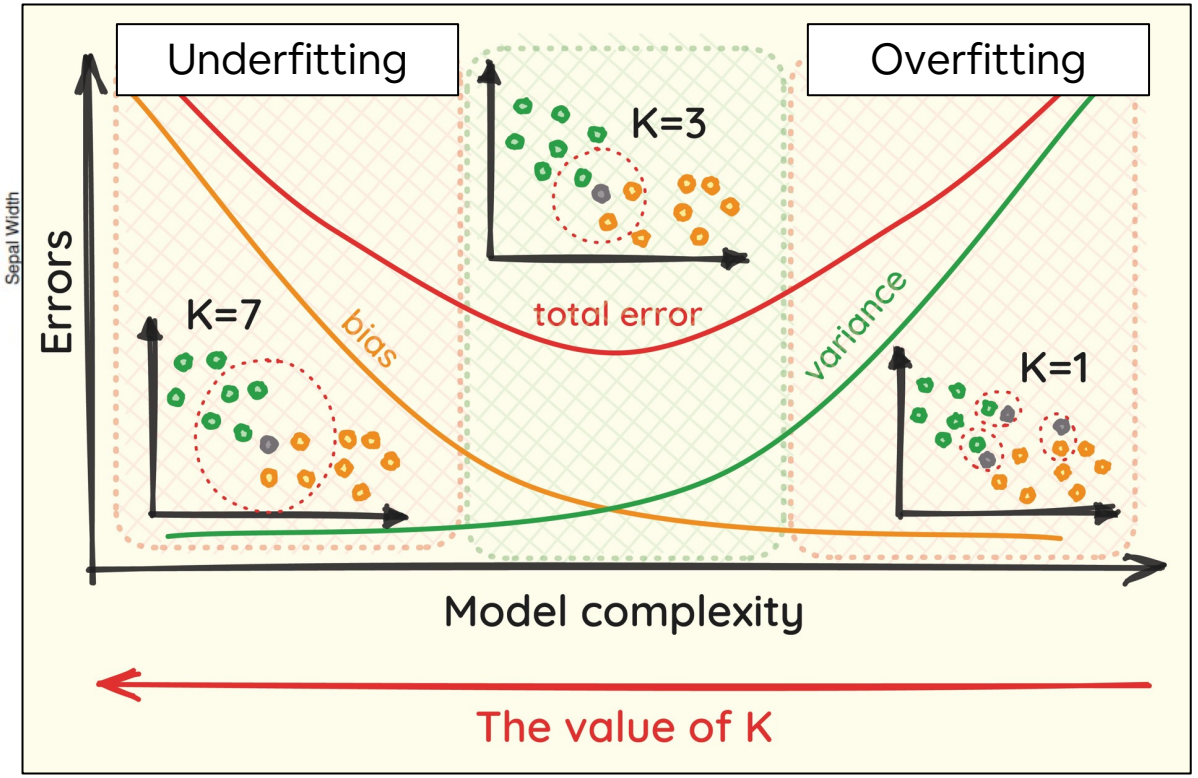


With the choice of *k* we can also regulate some behaviour we have previously encountered: ideas?

k-Nearest Neighbors (k-NN) in action: Iris Dataset



With the choice of *k* we can also regulate some behaviour we have previously encountered: ideas?



k-Nearest Neighbors (k-NN): comments

- Instance-based learning (memory-based learning): instead of an 'explicit' model, new observations are compared with previous instances.

    -> May not be really good for edge solutions!

k-Nearest Neighbors (k-NN): comments

- Instance-based learning (memory-based learning): instead of an 'explicit' model, new observations are compared with previous instances.

    -> May not be really good for edge solutions!

- Lazy learning: there is no explicit training, the computational cost is in the prediction step.

    -> Great for adaptive solutions or when we start with few data

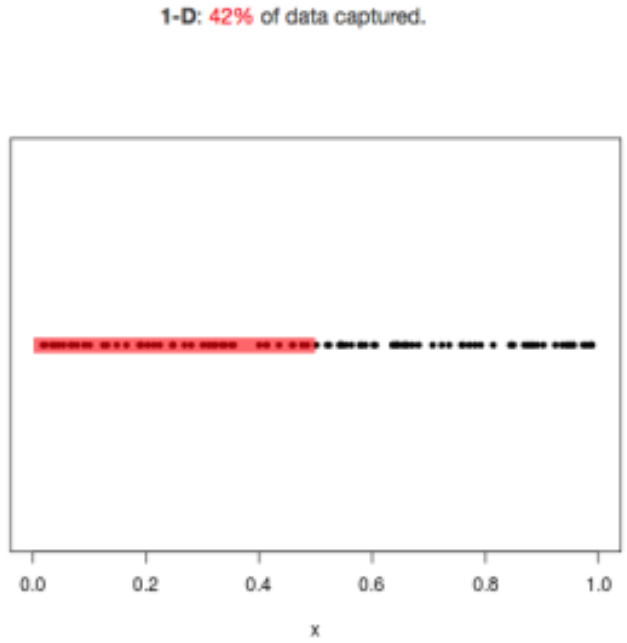
    -> Not ideal if the prediction should be fast!

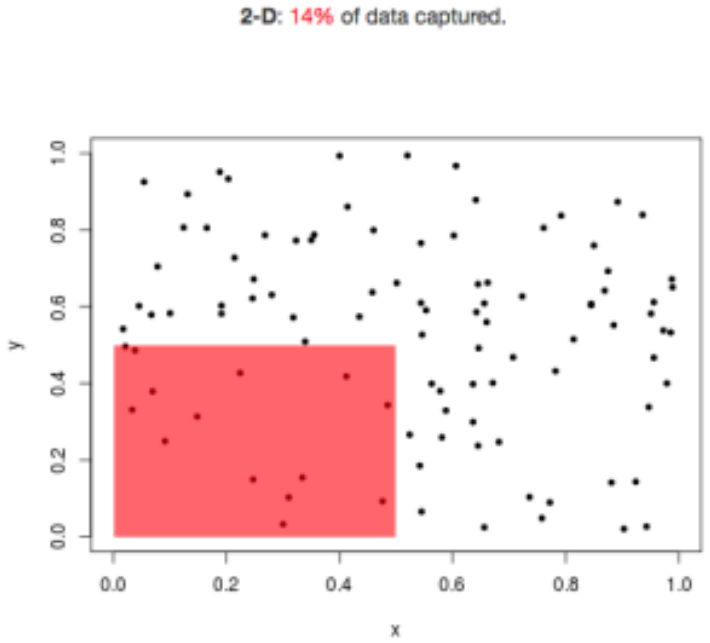
k-Nearest Neighbors (k-NN): comments

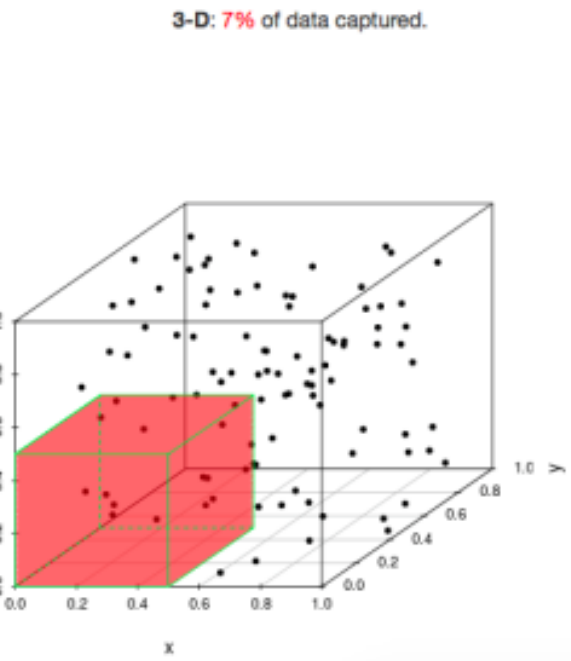
❌ Slow for large datasets (it computes distances for every new prediction).
❌ Sensitive to irrelevant features (feature selection or scaling is crucial).
❌ Not great for high-dimensional data (curse of dimensionality).

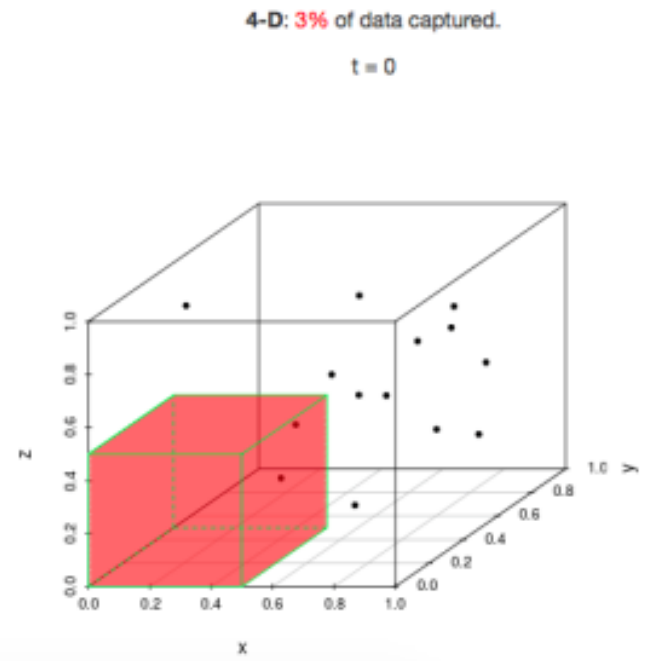
k-Nearest Neighbors (k-NN): comments

❌ Slow for large datasets (it computes distances for every new prediction).
❌ Sensitive to irrelevant features (feature selection or scaling is crucial).
❌ Not great for high-dimensional data (curse of dimensionality).







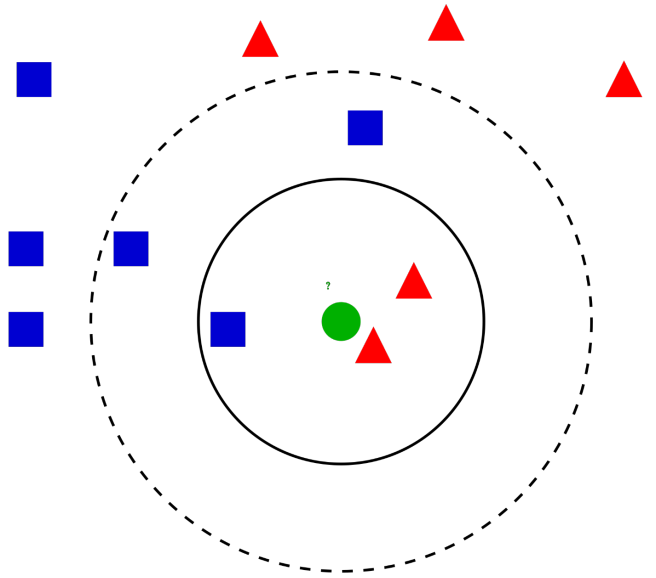


Curse of dimensionality

The curse of dimensionality refers to the problems that arise when dealing with data in high-dimensional spaces. As the number of dimensions (features) increases, data behaves in unexpected ways, making many machine learning tasks harder and less efficient.

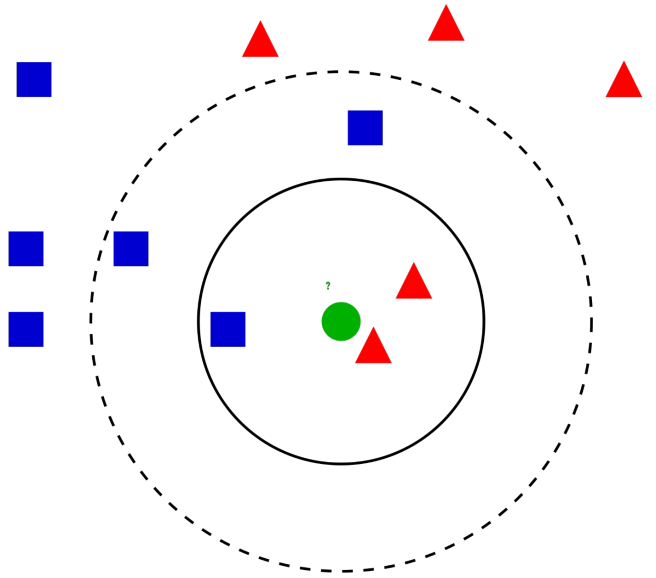
k-Nearest Neighbors (k-NN): the algorithm

1. **Choose/Given a value for k** (number of neighbors to consider).
2. **Compute the distance** between the new data point and all training points (commonly using **Euclidean distance**).
3. **Select the k nearest neighbors** (data points closest to the new point).
4. **Make a prediction**:
    1. **For classification**: Assign the class that appears most frequently among the k neighbors (**majority voting**).



k-Nearest Neighbors (k-NN): the algorithm

1. **Choose/Given a value for k** (number of neighbors to consider).
2. **Compute the distance** between the new data point and all training points (commonly using **Euclidean distance**).
3. **Select the k nearest neighbors** (data points closest to the new point).
4. **Make a prediction**:
    1. **For classification**: Assign the class that appears most frequently among the k neighbors (**majority voting**).



Euclidean Distance (most used):

$$d(A, B) = \sqrt{\sum (A_i - B_i)^2}$$

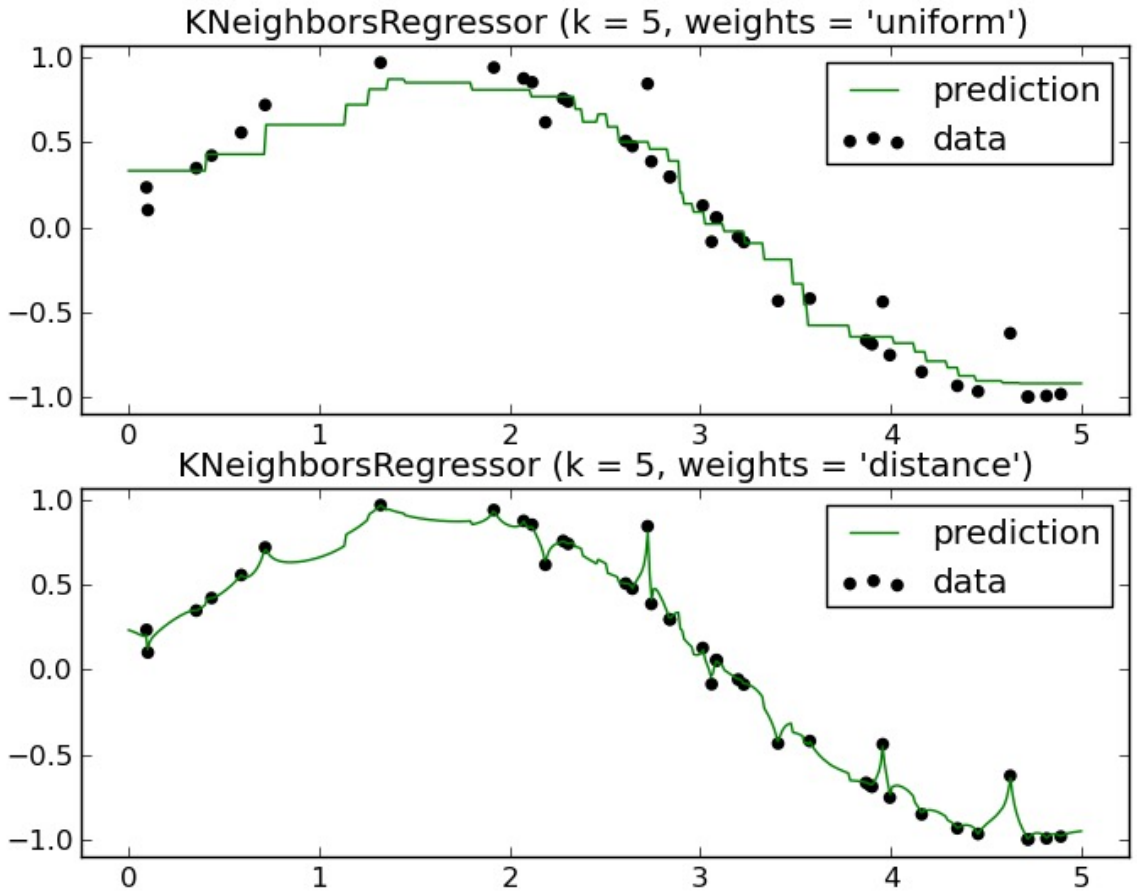
Manhattan Distance:

$$d(A, B) = \sum |A_i - B_i|$$

Cosine Similarity (for text data).

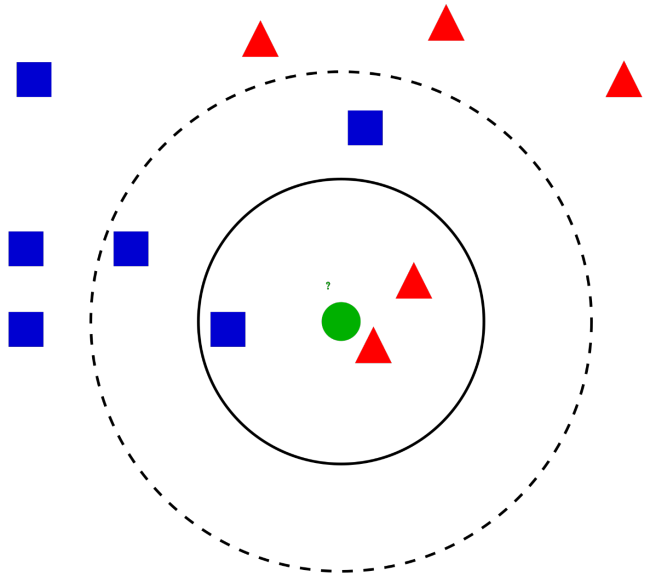
k-Nearest Neighbors (k-NN): the algorithm

1. **Choose/Given a value for k** (number of neighbors to consider).
2. **Compute the distance** between the new data point and all training points (commonly using **Euclidean distance**).
3. **Select the k nearest neighbors** (data points closest to the new point).
4. **Make a prediction**:
    1. **For classification**: Assign the class that appears most frequently among the k neighbors (**majority voting**).
    2. **For regression**: Take the **average** (or weighted average) of the neighbors' values.



k-Nearest Neighbors (k-NN): a non Euclidian norm

1. **Choose/Given a value for k** (number of neighbors to consider).
2. **Compute the distance** between the new data point and all training points (commonly using **Euclidean distance**).
3. **Select the k nearest neighbors** (data points closest to the new point).
4. **Make a prediction**:
    1. **For classification**: Assign the class that appears most frequently among the k neighbors (**majority voting**).
    2. **For regression**: Take the **average** (or weighted average) of the neighbors' values.



Euclidean Distance (most used):

$$d(A, B) = \sqrt{\sum (A_i - B_i)^2}$$

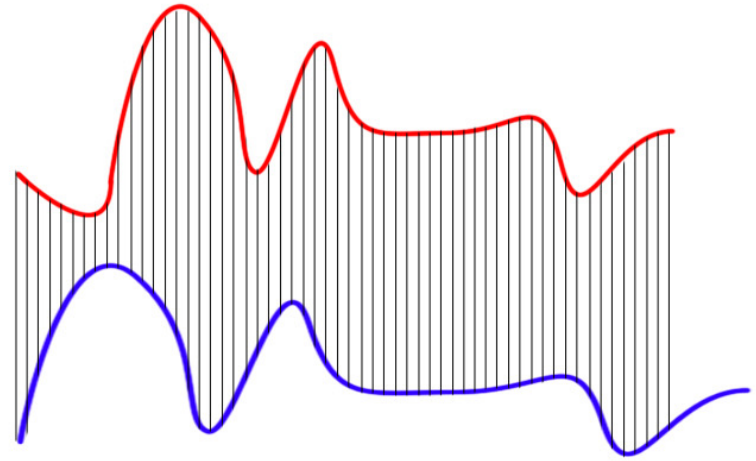
Manhattan Distance:

$$d(A, B) = \sum |A_i - B_i|$$

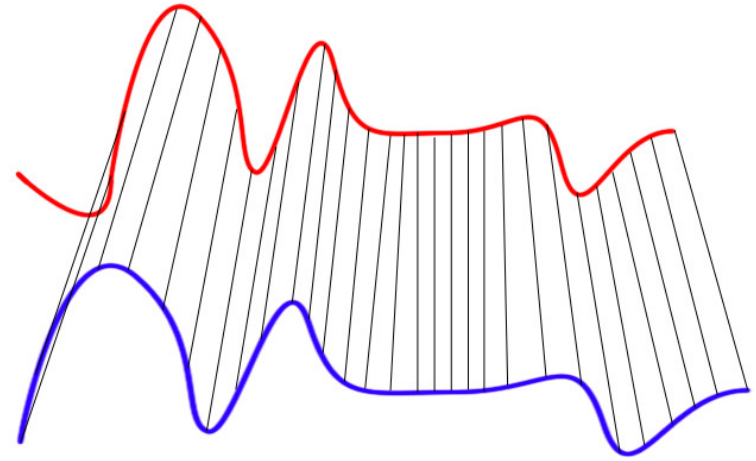
Cosine Similarity (for text data).

k-Nearest Neighbors (k-NN): a non-Euclidian norm (only for theory)

- One of the most commonly used approaches for classifying time series is 1-NN with Dynamic Time Warping (DTW).
- DTW is an algorithm for matching time series with different sampling rates or lengths.
- DTW also provides a similarity index for time sequences, making it useful as a distance metric for k-NN with time series.



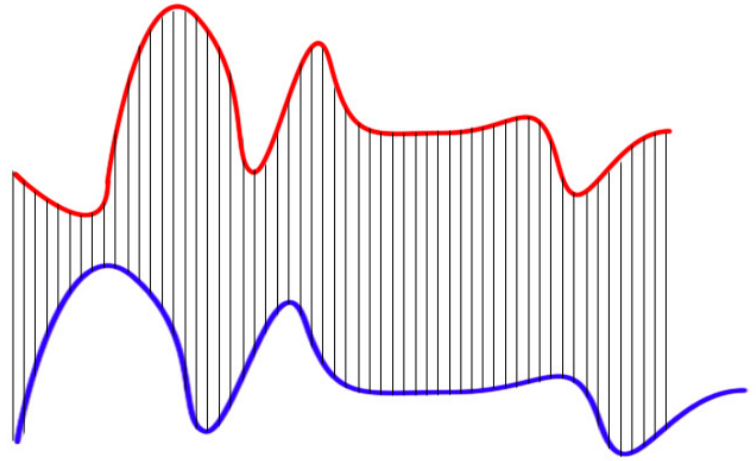
Euclidean Matching



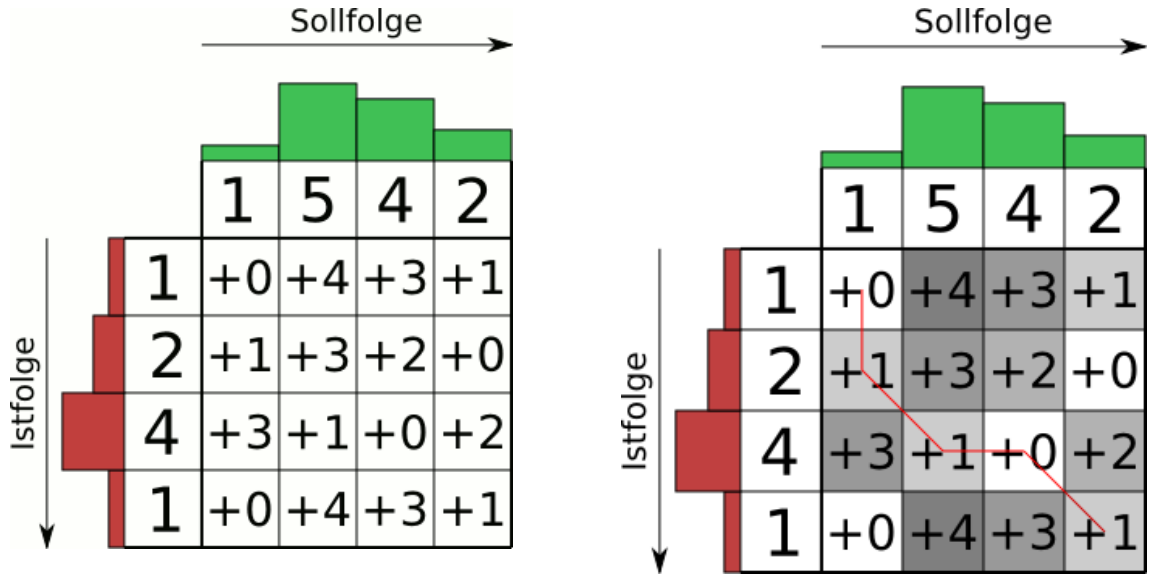
Dynamic Time Warping Matching

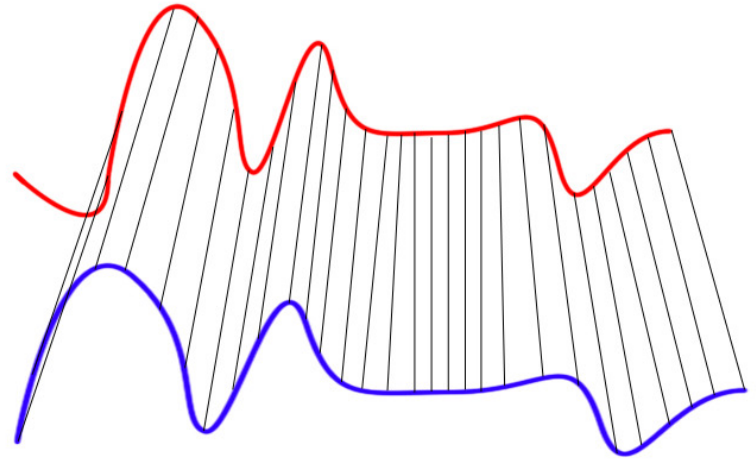
k-Nearest Neighbors (k-NN): a non-Euclidian norm (only for theory)

- One of the most commonly used approaches for classifying time series is 1-NN with Dynamic Time Warping (DTW).
- DTW is an algorithm for matching time series with different sampling rates or lengths.
- DTW also provides a similarity index for time sequences, making it useful as a distance metric for k-NN with time series.



Euclidean Matching





Dynamic Time Warping Matching

Categorical Inputs: One-hot encoding

- We may encounter categorical inputs, not only outputs

Label Encoding

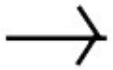
Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

Categorical Inputs: One-hot encoding

- We may encounter categorical inputs, not only outputs
- One-hot encoding is a technique used to represent categorical variables as binary vectors. Each category is converted into a vector where only one element is "1" (indicating the presence of that category), while all other elements are "0".

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

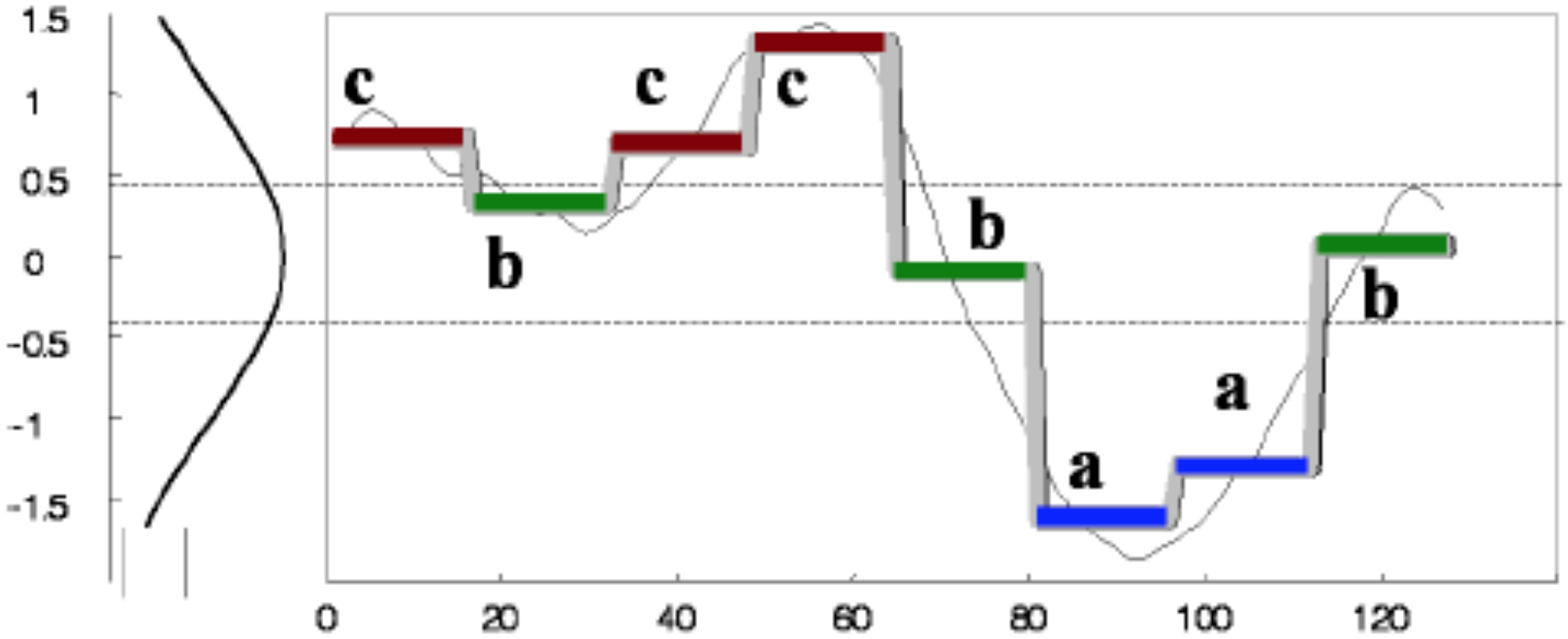


One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Classification of a continuous target

In some applications, we may want to move from a regression task to a classification one!



Thank you!

Gian Antonio Susto