MACHINE LEARNING BASICS

Clayton Leite – Aalto University

Expected Outcomes

- Learn the basic concepts of machine learning and its use for human activity recognition
- Learn the most common machine learning methods: k-Nearest Neighbors, Support Vector Machines, Decision Trees, and Random Forest

Quick Survey

- How many of you have heard about Machine Learning (ML)?
- How many of you have taken a course in programming?
- How many of you have taken a course in ML?
- Can you describe the terms classification and clustering? What is the difference between them?

01 TRADITIONAL PROGRAMMING

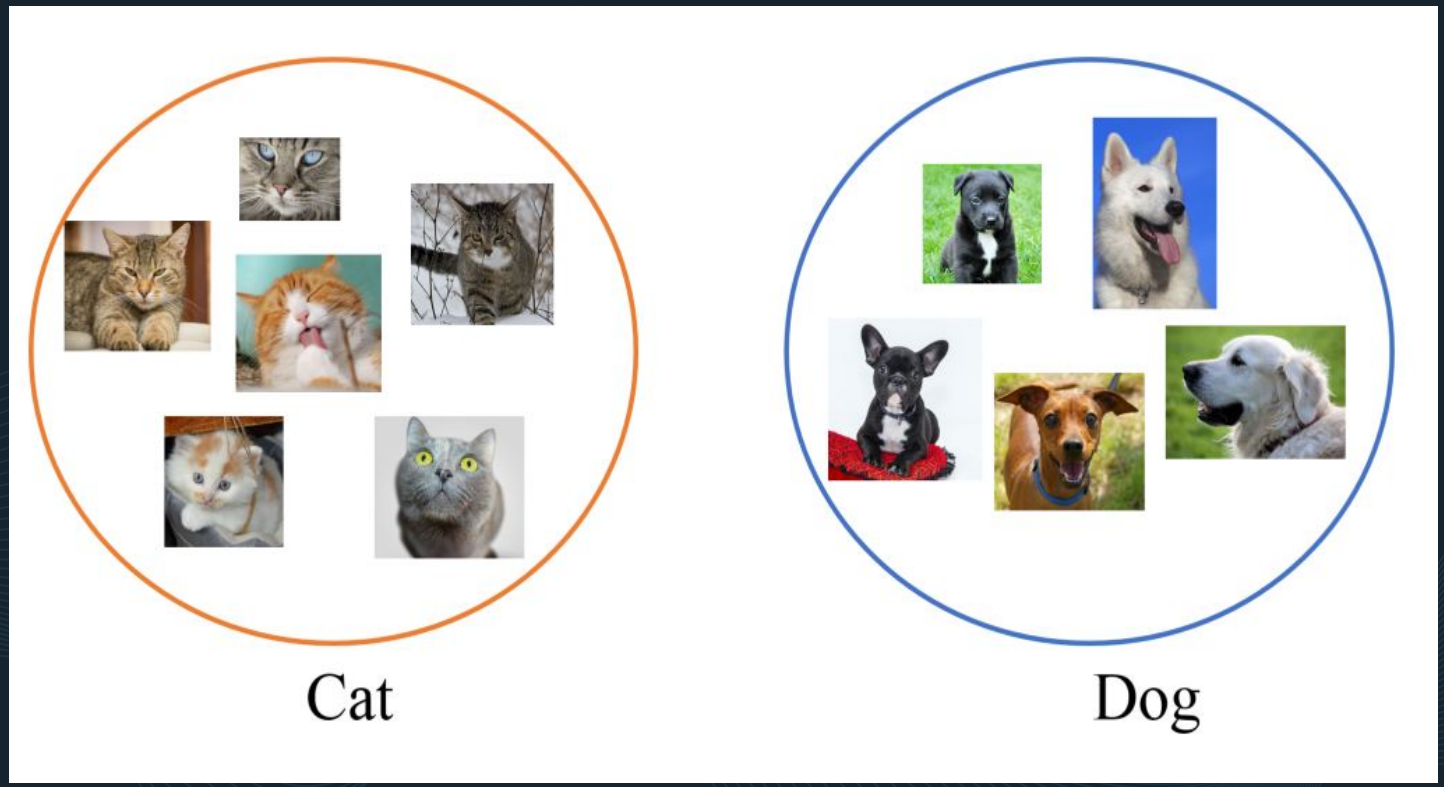
Let’s first talk about **classification, traditional programming, and the concept of machine learning**

Sorting data

- Can you separate the dogs from the cats?
- Easy, right?

Classification

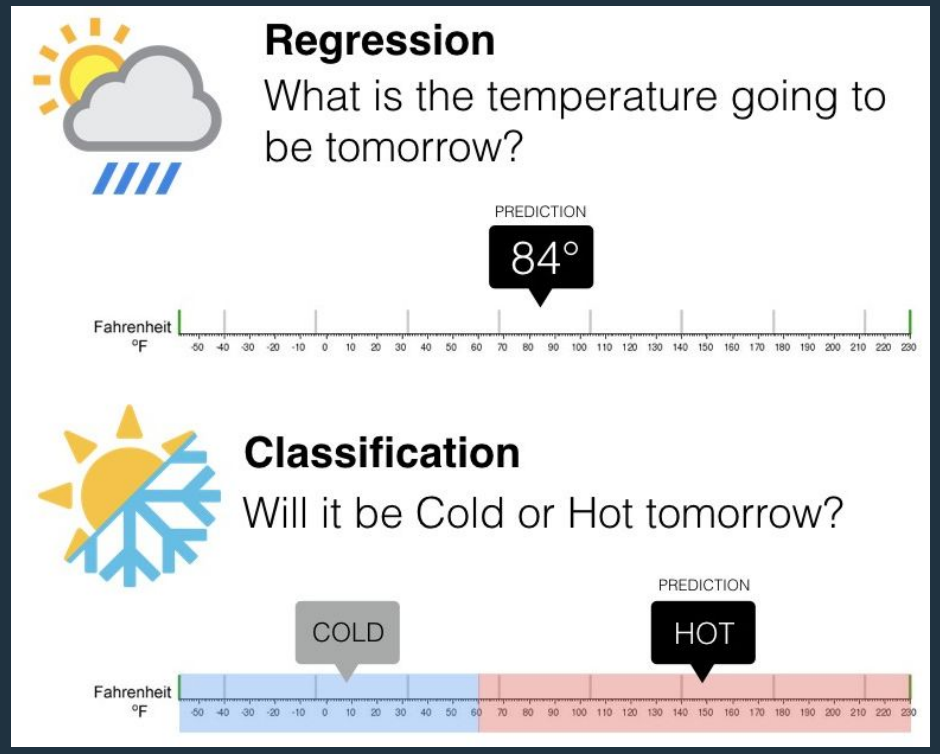
- **Classification** is the task of sorting data into different groups or classes based on their characteristics. **Example**: images of dogs vs. cats



Classification vs. Regression

- As we now know, **classification** means sorting into groups (categories)

- **Regression** means predicting continuous values. For example, predicting house prices based on features such as square footage, number of bedrooms, and location.

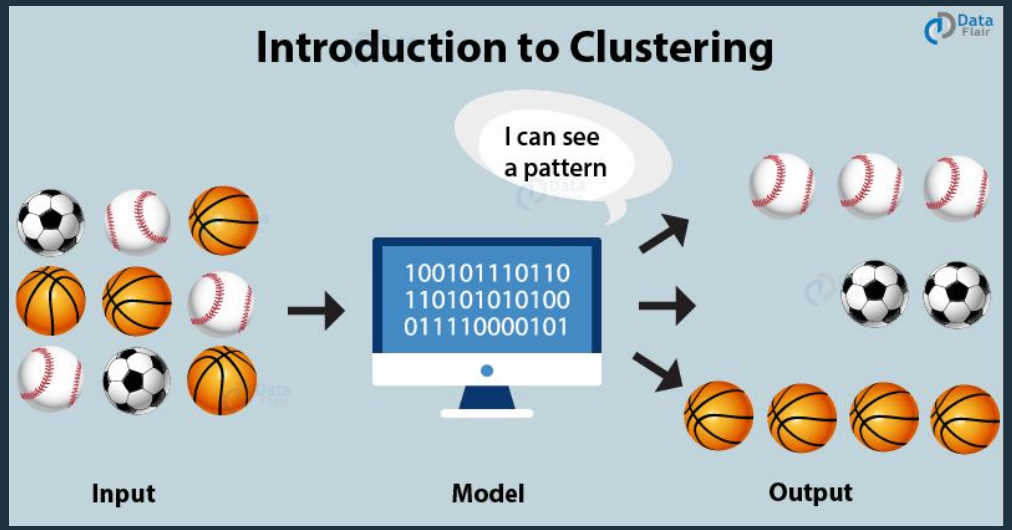


Clustering

- Supervised learning is when the machine learn from examples and their correct class (named **label**)

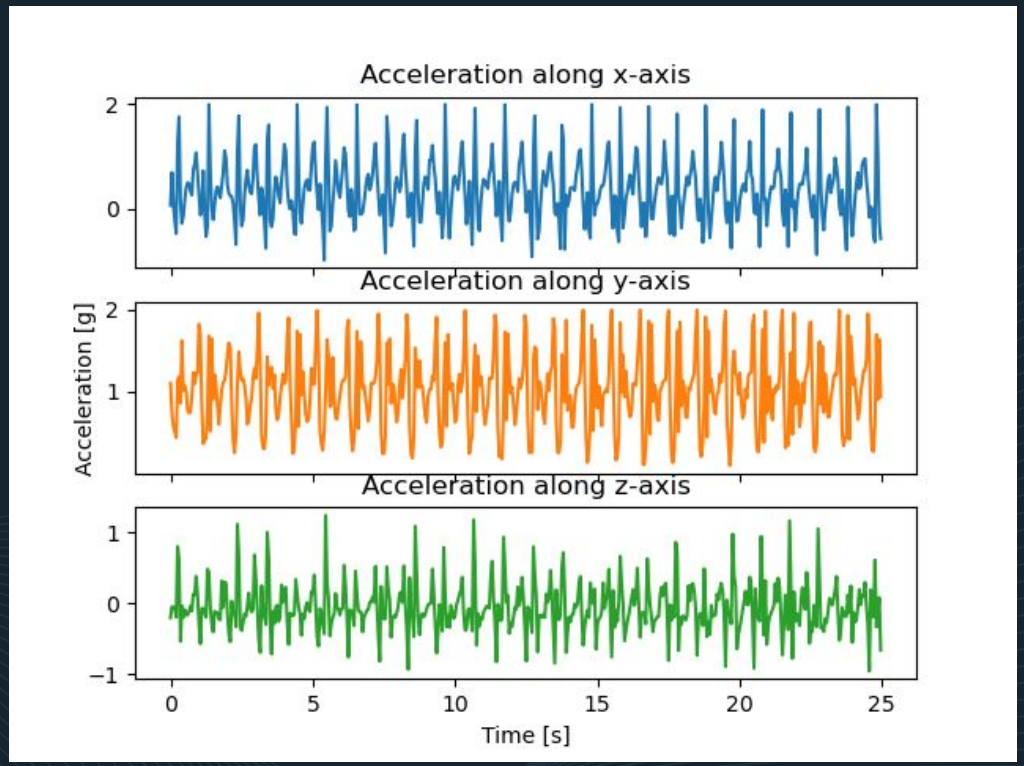
- An example of unsupervised learning is **clustering**.

- **Clustering** is a way of finding patterns and organizing things that seem alike together



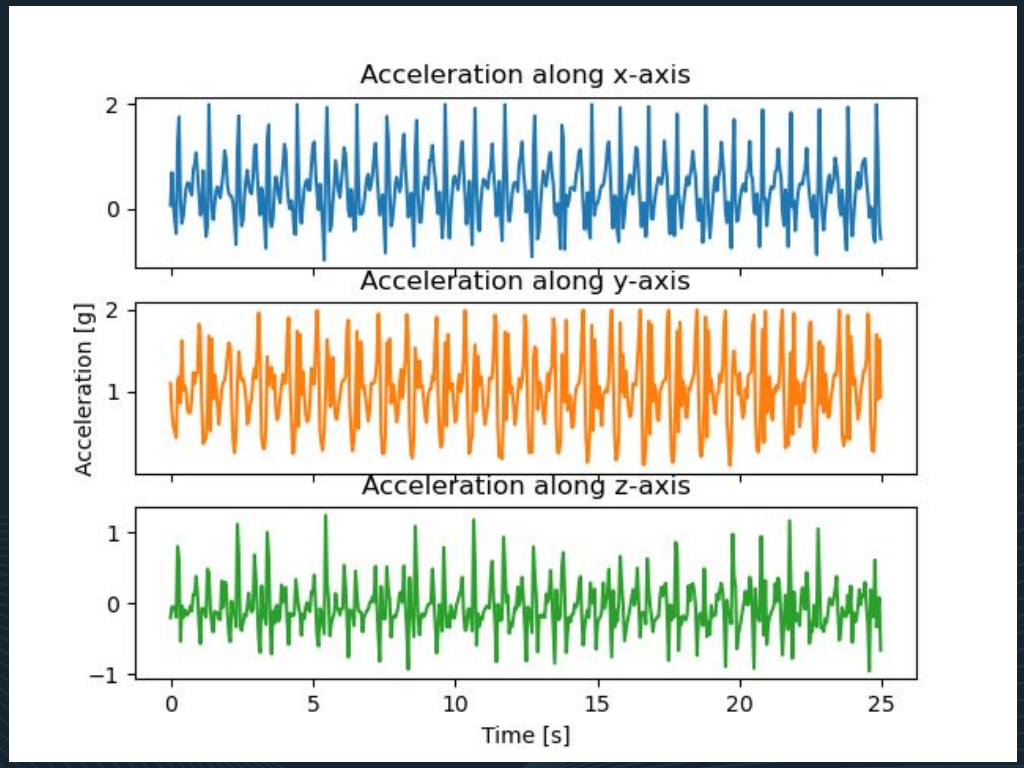
Classification

- **Now let's focus on classification**
- **One example closer to us:** classifying hand gestures based on accelerometer readings.



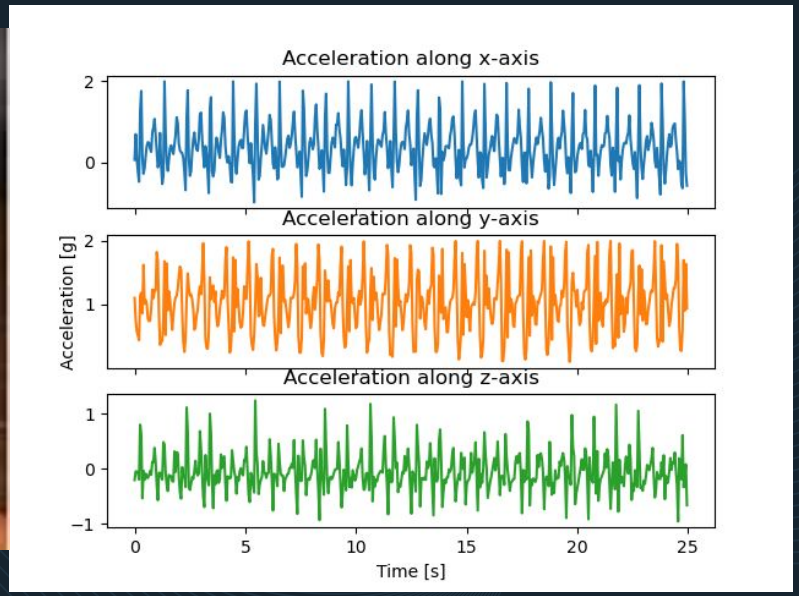
Classification

- Considering this scenario, how to classify accelerometer data between (let's say) two different hand gestures?



Traditional Programming

- **The traditional way:** involves writing explicit rules that transform input data into an output.
- **How exactly?** Look at the data and find patterns that correlate the input with the output. **Not always easy**.



Traditional Programming

- Suppose the two gestures are punching the air and raising the hand.
- Can you identify the corresponding gesture for each of the graphs below?

Traditional Programming

- How do you identify them?
- Punching has significantly higher total accelerometer magnitude (in white).

PUNCHING

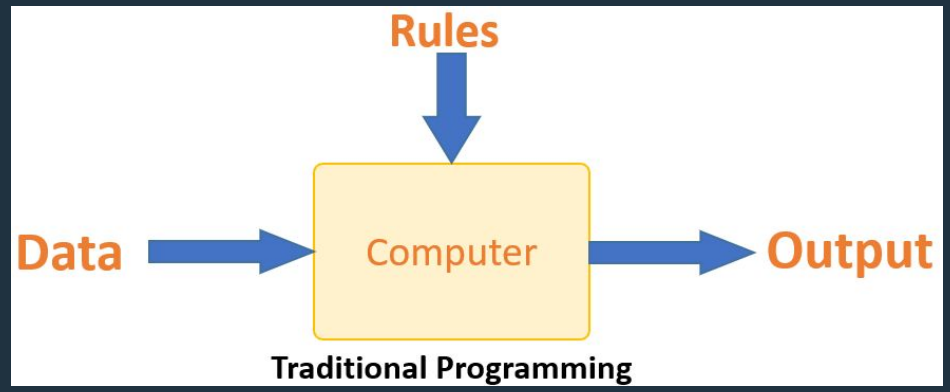
RAISING THE HAND

Traditional Programming

- What if we had more than 10 sensors placed on the individual’s body?
- What if we wanted to recognize more than 10 activities (e.g., walking, running, playing football, going upstairs, and going downstairs)?
- Traditional programming requires **significant human labor and expertise**

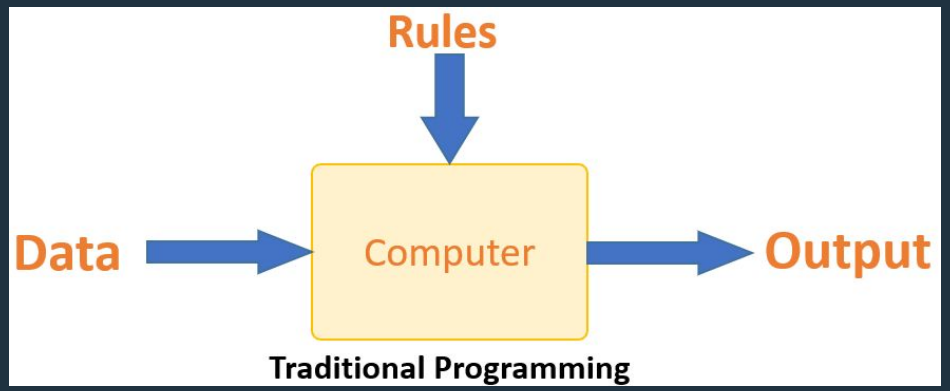
Traditional Programming

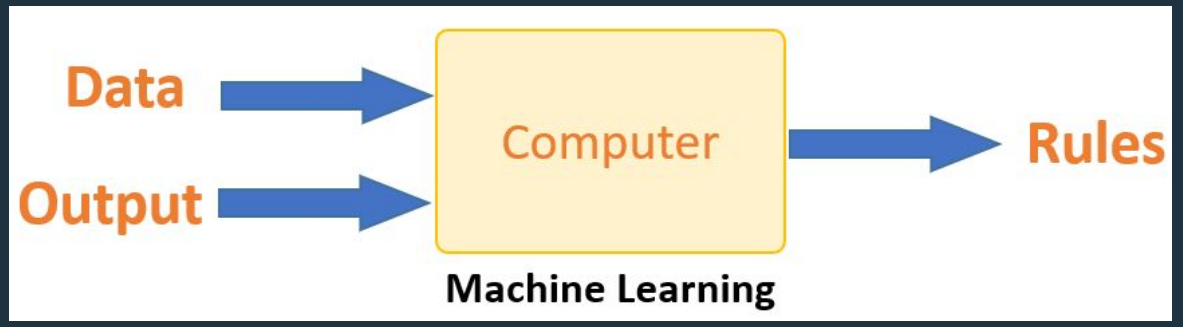
- Traditional programming works well for very simple tasks (e.g., **binary** classification of very simple gestures with very few sensors)



The Concept of Machine Learning

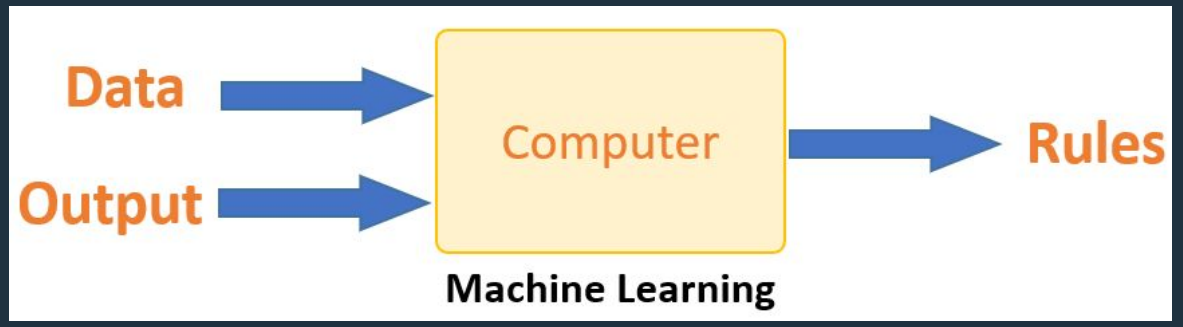
- **Machine learning**: automatically learn the rules with a set of samples (data + desired output)
- Useful for handling large and complex data sets or tasks that are difficult to program manually





The Concept of Machine Learning

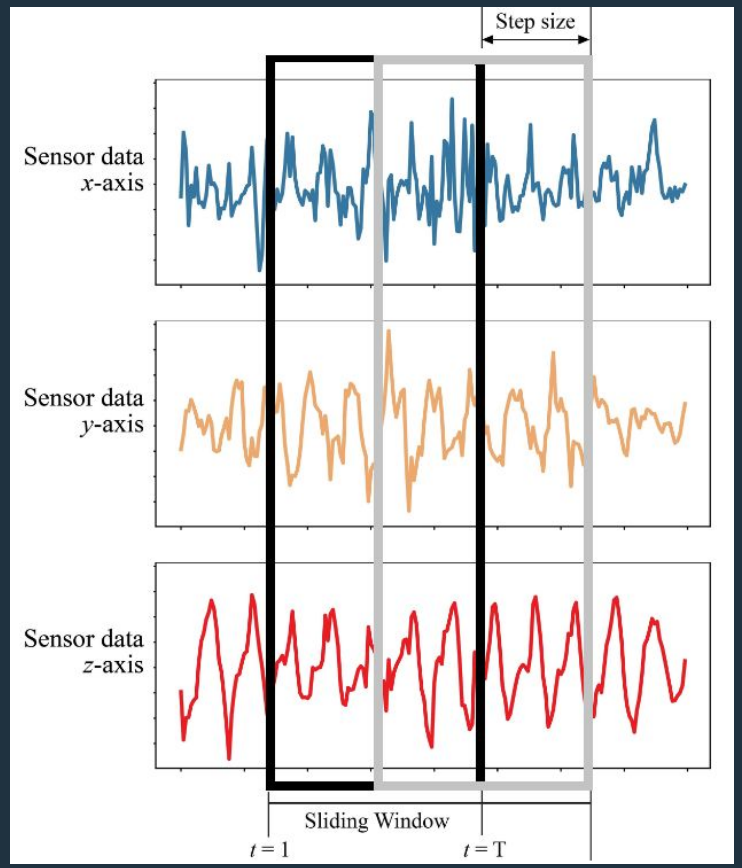
- **In the example of classification:** provide lots of samples of the classes we want to recognize.
- Typically, the more samples, the better.



Sliding Windows

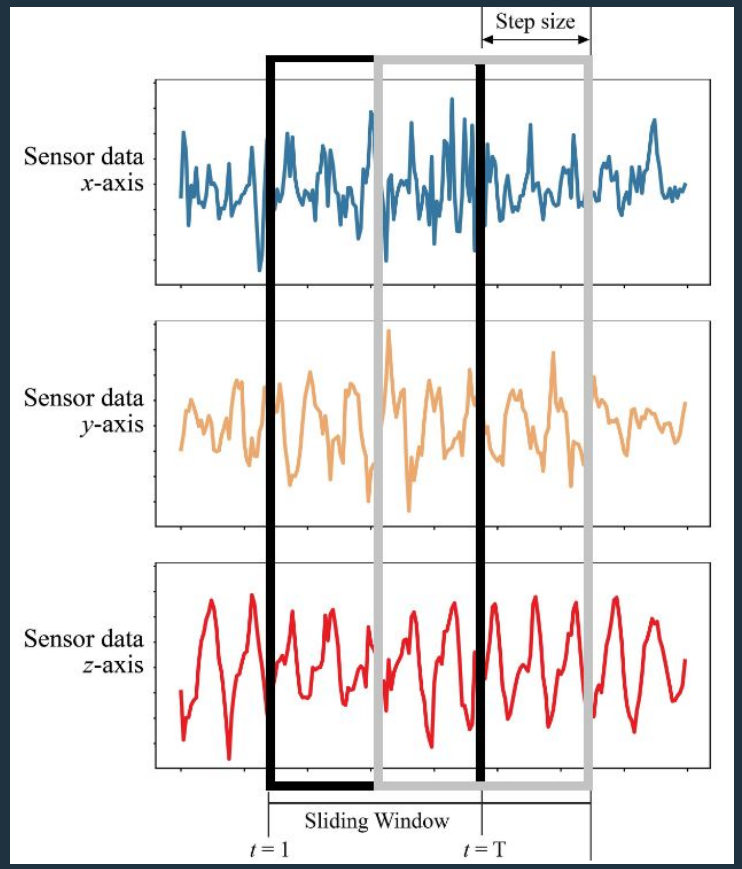
- Did you notice that we had to analyze a sequence of data to perform the classification?
- We call this sequence of data **a window**
- This window slides across the data, thus being called a **sliding window**

Sliding Windows



- Typically, slidings windows overlap with each other.
- The window length and step size are parameters defined by you.
- **What are the benefits and drawbacks of a larger overlap? How about a smaller overlap?**

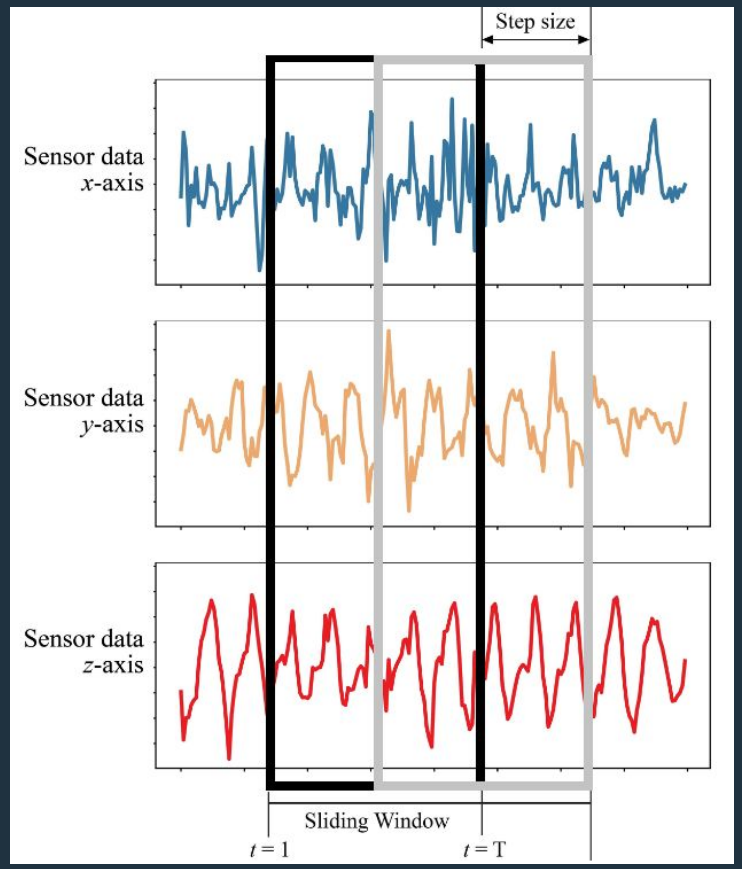
Sliding Windows



- **Larger overlap**: **more** frequent analysis but **larger** computational resources utilization

- **Smaller overlap**: **less** frequent analysis but **smaller** computational resources utilization

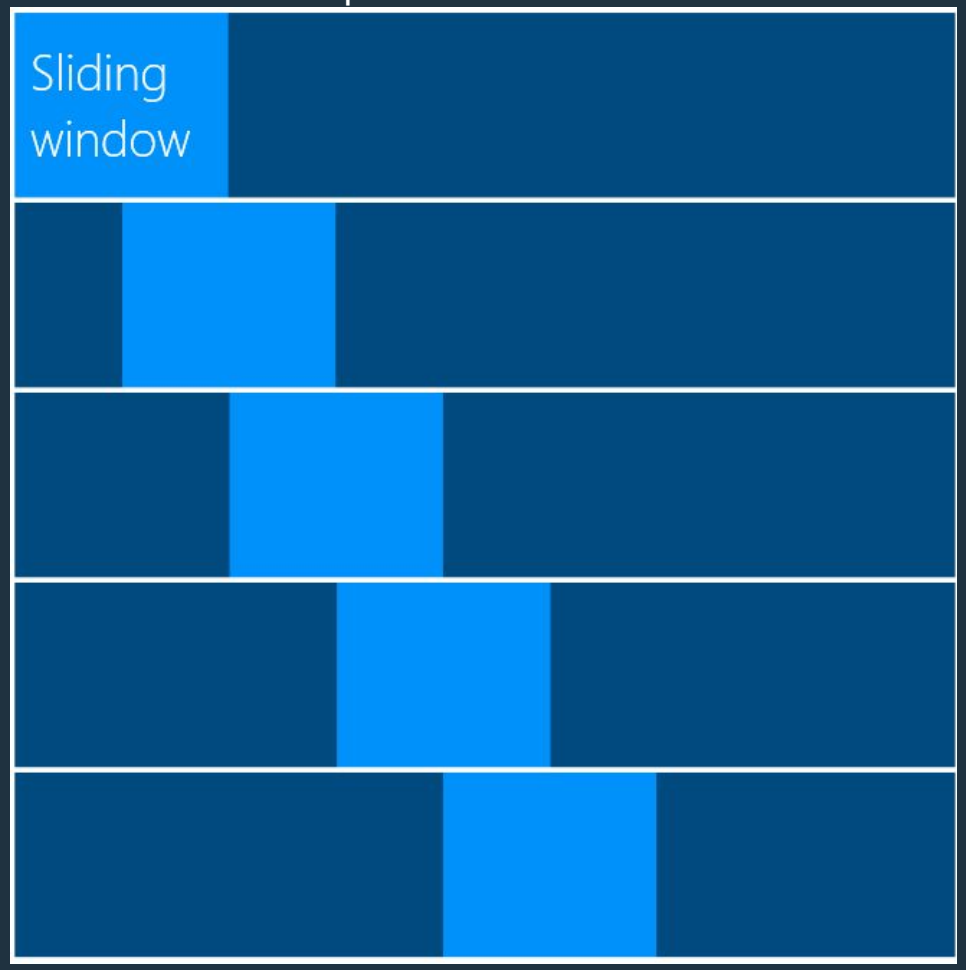
Sliding Windows



- The process of splitting the data stream into sliding windows is named **segmentation**
- Depending on the length, a single sliding window can encompass multiple classes
- And, in such case, the predominant class is the class we assign to that window.
- No guideline for choosing window length and step size. **Typically**, values between 0.5 to 5 seconds are used for window length, with overlap varying from 50% to 80%

Sliding Windows

Sliding from left to right. Notice the overlap



Sliding Windows

- In a very simple case (2 classes, 3 sensor channels), it can be simple to define a good sliding window length.
- How about when we have 3 IMUs (27 sensor channels) and 18 classes?

02 THE BASICS OF MACHINE LEARNING

Let’s talk about features, training, inference, and other terms.

The Phases of Machine Learning (ML)

- **How to create an ML algorithm?**

  1. Various data samples (input + **ground truth**) need to be collected. These samples form the so-called **training set**
  2. The programmer needs to define something called **features**
  3. …

- **Ground truth or label** is the correct class for each sample

PUNCHING

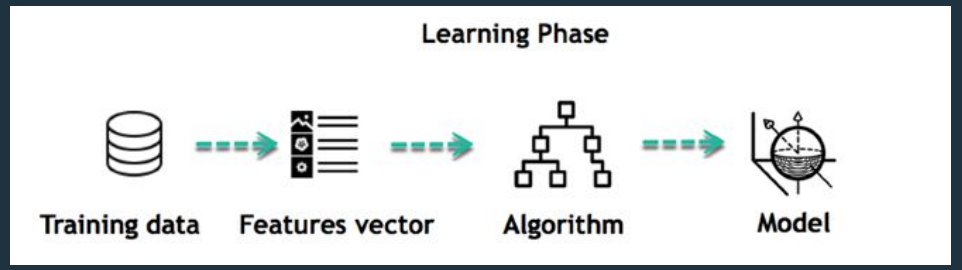
RAISING THE HAND

The Phases of Machine Learning (ML)

- **Features** are representations extracted from the input data.
- **Examples of features**: maximum value, more examples?

- **A feature vector** is a list of more than one feature

- **Feature extraction** is the process of extracting these representation from the input data

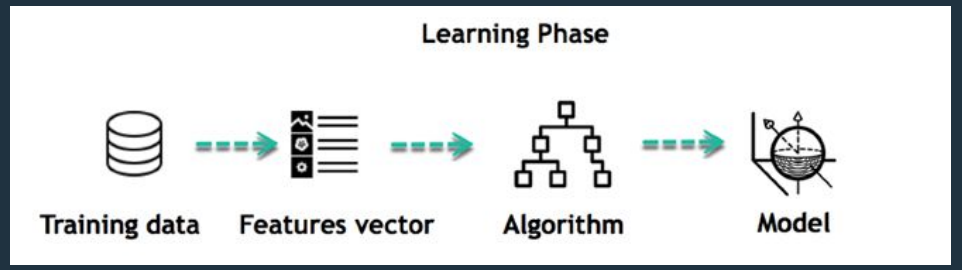


The Phases of Machine Learning (ML)

- **Features** are representations extracted from the input data.
- **Examples of features**: maximum value, minimum value, mean, standard deviation, zero-crossing rate, peak-to-peak amplitude.

- **A feature vector** is a list of more than one feature

- **Feature extraction** is the process of extracting these representation from the input data



The Phases of Machine Learning (ML)

- After extracting the features, the training set should look like this
- A list where the columns (except for the last one) represent features
- The last column is the ground truth
- Each row represents a sample (also called data point).
- Each sample is obtained by extracting features off a sliding window

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

The Phases of Machine Learning (ML)

- **How to create an ML algorithm?**

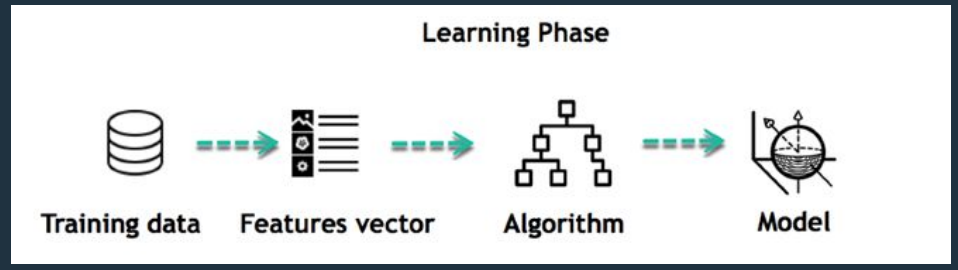
  1. Various data samples (input + output) need to be collected. These samples form the so-called **training set**

  2. The programmer needs to define something called **features**

  3. Feature extraction

  4. The programmer chooses the algorithm (we will study these later) and start the learning (training)

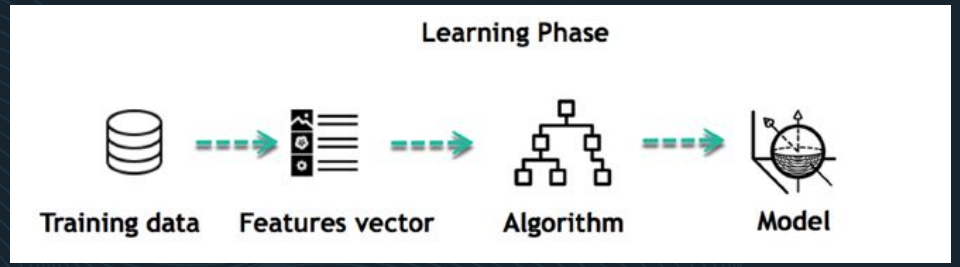
- The training (or learning) is when the machine learning algorithm learns to **predict** classes



The Phases of Machine Learning (ML)

- When the training is over, you have a **model**. The model is a set of instructions that is able to predict classes from the input data.

- **When is the training over?** You define it. We will learn more about this later on.



The Phases of Machine Learning (ML)

- What is next? Validation!

- **Validation** is when you evaluate the model. For this, you use validation data (must not be the same as training data).

- During the validation, you pass validation data as input to the model and compare its output with **the ground truth**. In other terms, you perform **inference** on the validation data.

The Phases of Machine Learning (ML)

- The results of the validation (e.g., accuracy) can be used to make decisions about the algorithm or features.

- For instance, if the validation accuracy is low (let’s say, 40%), then we might want to modify the algorithm, train for a little longer, or better select features.

- After performing modifications, we restart the training.

The Phases of Machine Learning (ML)

- The outcome of the new training is a new model, which is hopefully better.

- **If not**, we restart all over again.

- What if the validation accuracy is never good enough? In this case, you might have bigger problems. Maybe it is **impossible** to predict the classes from the sample data

The Phases of Machine Learning (ML)

- What bigger problems?

1. Maybe it is **impossible** to predict the classes from the data because the sensors are faulty or do not provide enough information for this task.
2. Maybe the training data were not sufficient
3. There could be problems in the ground truth (erroneous ground truth)
4. Errors in the code

The Phases of Machine Learning (ML)

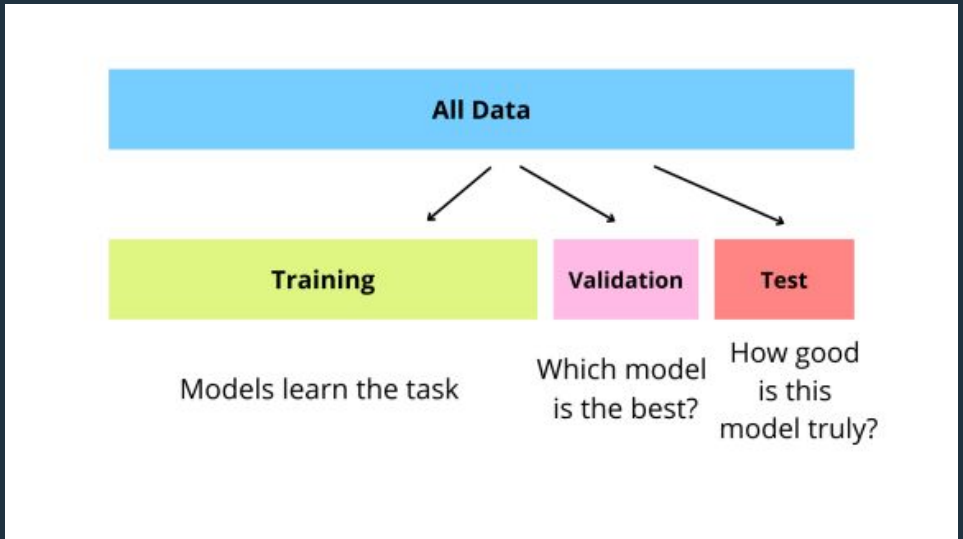
- Let’s suppose the validation accuracy is good enough. What is next?

- **The final test**. We evaluate the final **performance (accuracy)** on a new set of data – the so-called test set.

- Why not get the final performance of the model on the validation set?
- In other words, why is the validation performance not the final one?

The Phases of Machine Learning (ML)

- The validation performance is an overly optimistic estimate of the model's actual **generalization capability**; because the model has been tailored during the training to achieve the highest validation performance possible.



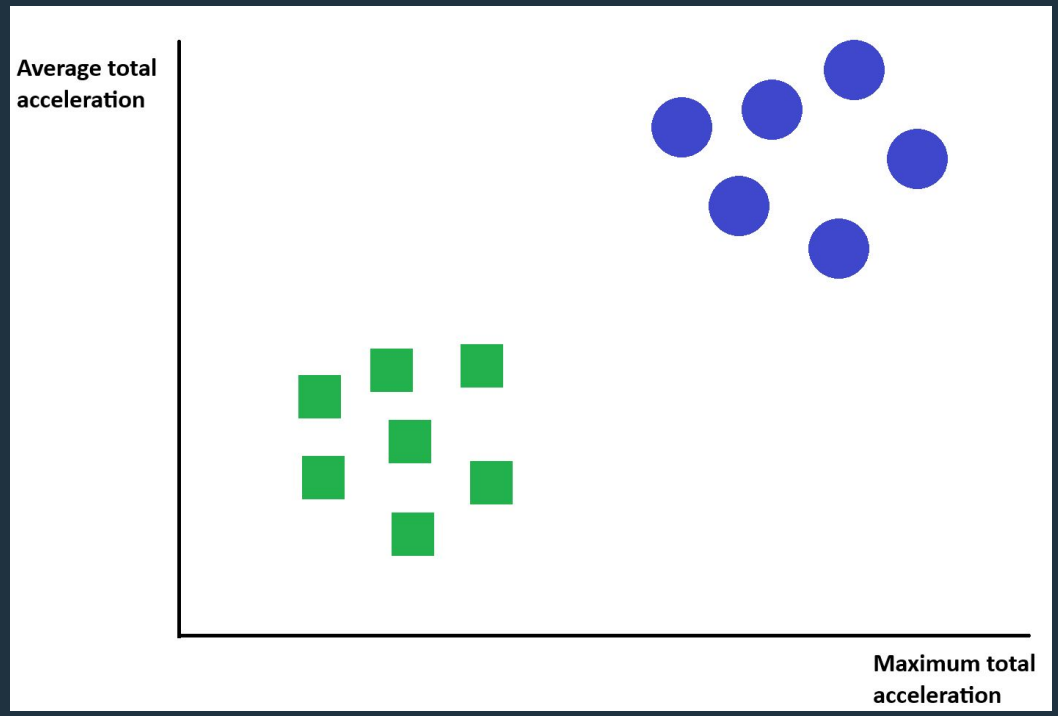
The Phases of Machine Learning (ML)

- Can we cite all the steps again?

03 k NEAREST NEIGHBORS

Support Vector Machines

- Let’s learn the first classification algorithm
- Let’s consider again the punching vs. raising hand problem.
- Suppose we utilize the features of **maximum total acceleration** and **average total acceleration**
- We then plot the samples in a 2D graph

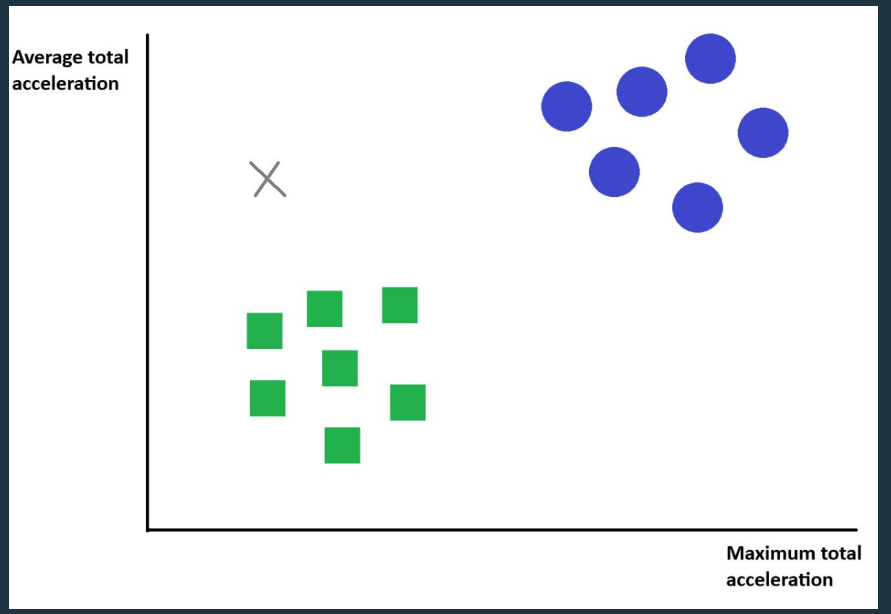


k Nearest Neighbors (kNN)

- In the graph, each green square represents a sample of the “raising hand” class, whereas each blue circle represents a sample of the “punching” class.

- Now suppose we want to find out the class of a certain sample (represented as X in the image)

- Can you tell an easy way to classify X?



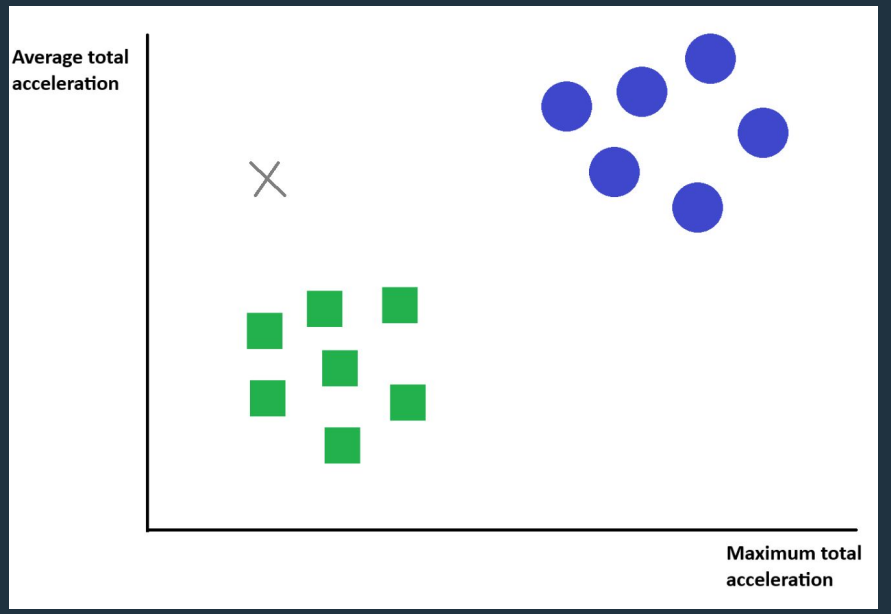
k Nearest Neighbors (kNN)

- In the graph, each green square represents a sample of the “raising hand” class, whereas each blue circle represents a sample of the “punching” class.

- Now suppose we want to find out the class of a certain sample (represented as X in the image)

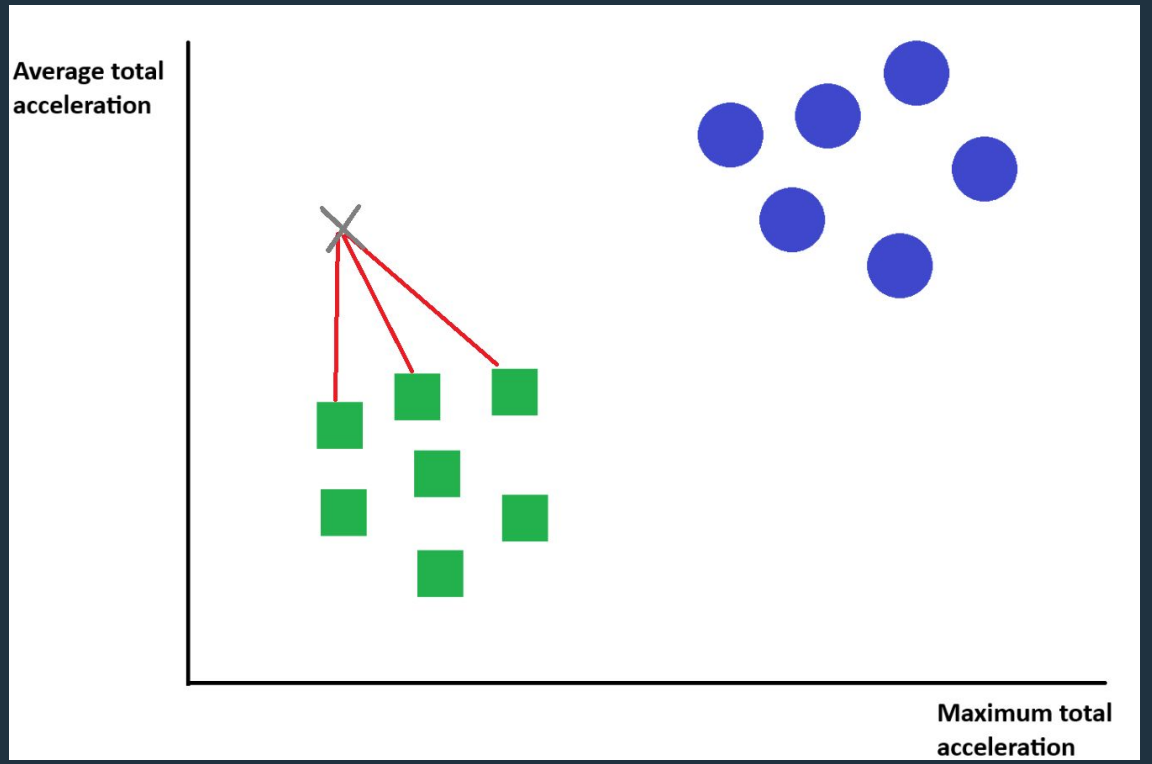
- KNN works by finding the K nearest neighbors

  to the sample we want to predict the class



k Nearest Neighbors (kNN)

- Let’s make k = 3. We then find the 3 nearest neighbors to X.
- Check the classes of the nearest neighbors. The majority class is the class of X.

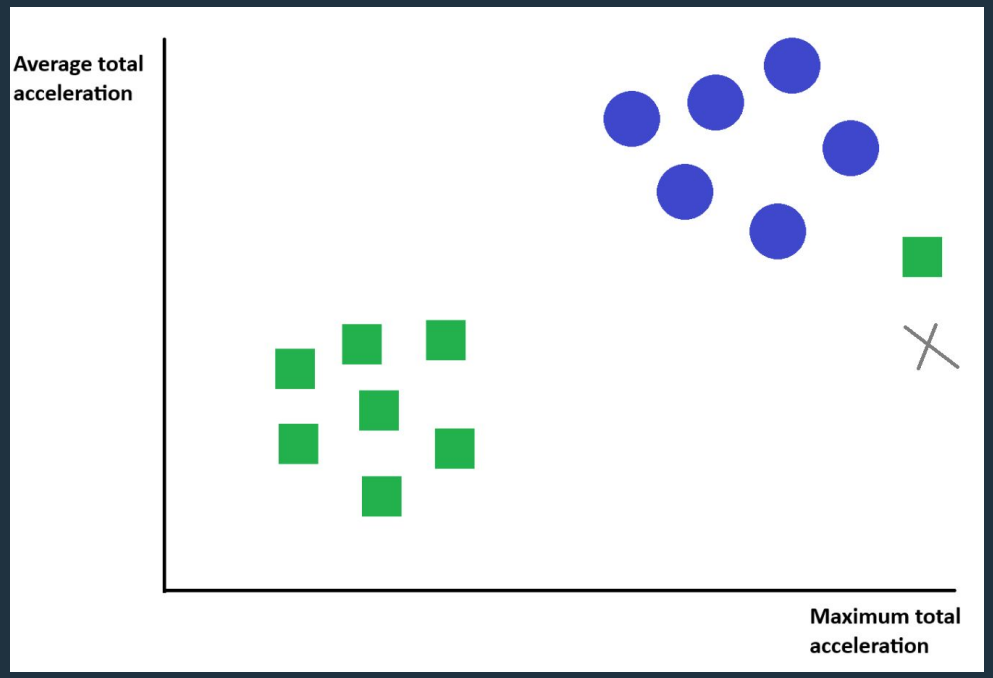


k Nearest Neighbors (kNN)

- The performance in kNN can be sensitive to the choice of the number of neighbors k.
- Picking the right k value is important for accurate predictions.
- Choosing an odd number for k prevents ties

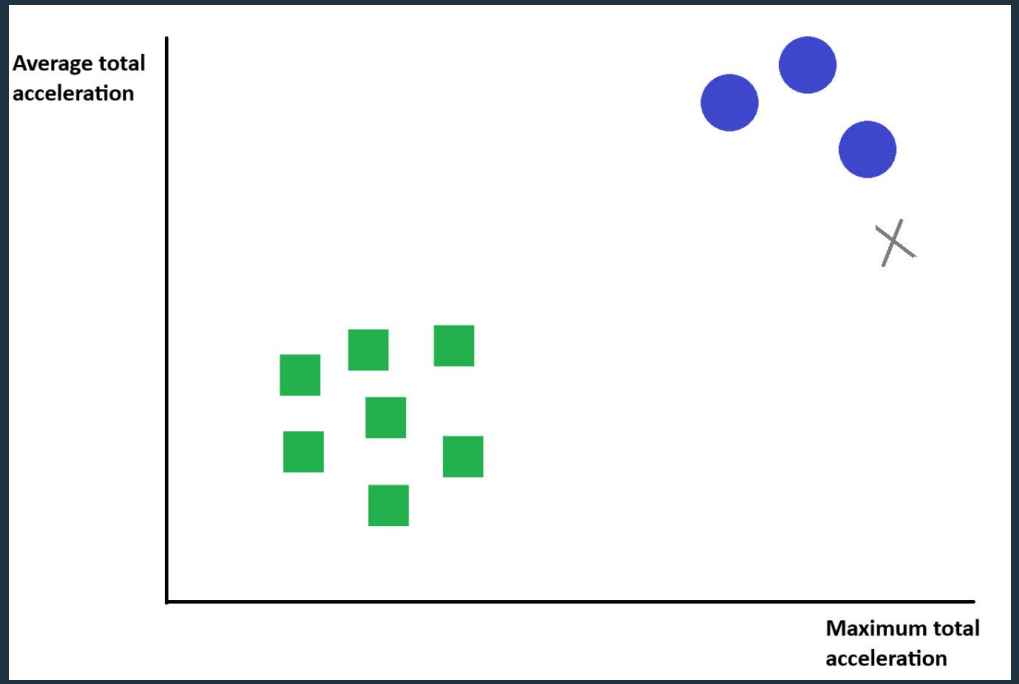
k Nearest Neighbors (kNN)

- Low value for k → sensitive to outliers
- In the following case, the predicted class would be the green square for k = 1 due to the outlier close to X



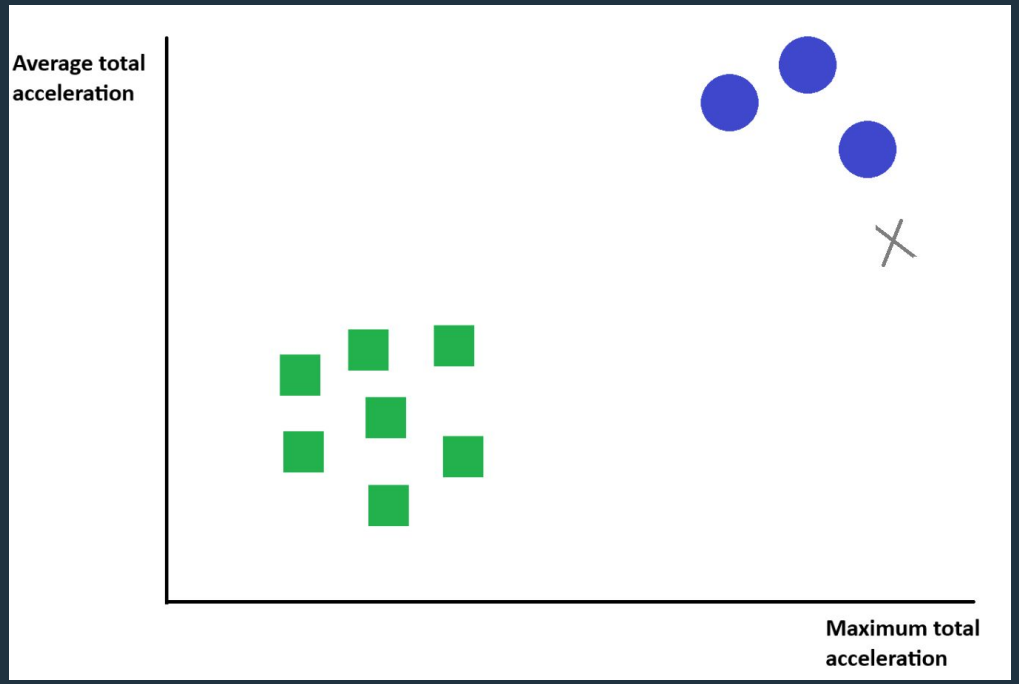
k Nearest Neighbors (kNN)

- High value for k → bias
- What if we had **much more** samples for the green square class?



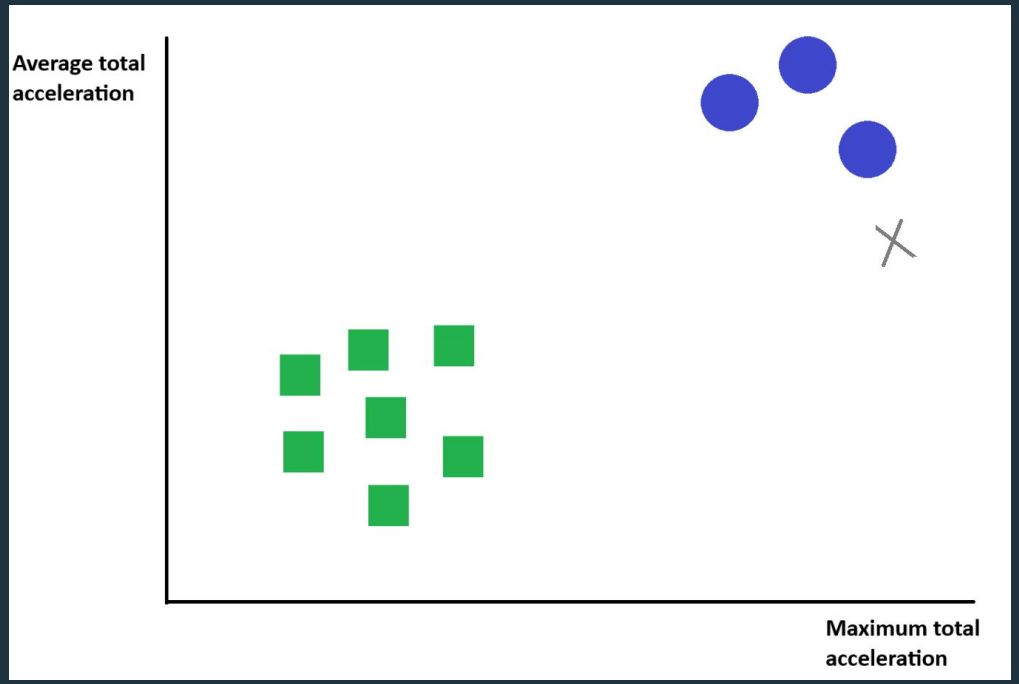
k Nearest Neighbors (kNN)

- High value for k → bias
- What if we had **much more** samples for the green square class?
- If we choose k = 7, inevitably more samples of the green square class would be present in the list of 7 nearest neighbors.



k Nearest Neighbors (kNN)

- How to choose the best k? A good starting value for k is sqrt(N), where N is the number of samples.
- In this case, N = 10. So, sqrt(10) = 3.16, k = 3. Hence, we would avoid bias.



k Nearest Neighbors (kNN)

- k = sqrt(N) is just a starting point.
- The optimal k is selected using the validation set.
- The way it is done is by trying different values of k and finding the accuracy of the KNN classifier with the validation set.
- The larger the validation set, the more reliable is the accuracy.

k Nearest Neighbors (kNN)

- kNN can be **computationally expensive** when the dataset grows larger, since it needs to search through all stored data points to find nearest neighbors.

- kNN is useful when you have small to medium-sized datasets

- So, if we have a small dataset, how do we select the best k?

k Nearest Neighbors (kNN)

- Like mentioned before, a large validation set provides better accuracy estimate.

- But, if we have a small dataset and want to use KNN, how do we do?

- **K-Fold Cross Validation**

K-Fold Cross Validation

- **Note the capital K, differently than kNN**

- **K-Fold Cross Validation** is a technique for model performance evaluation. It replaces the traditional validation approach.

- Randomly split the training set into K equal-sized subsets

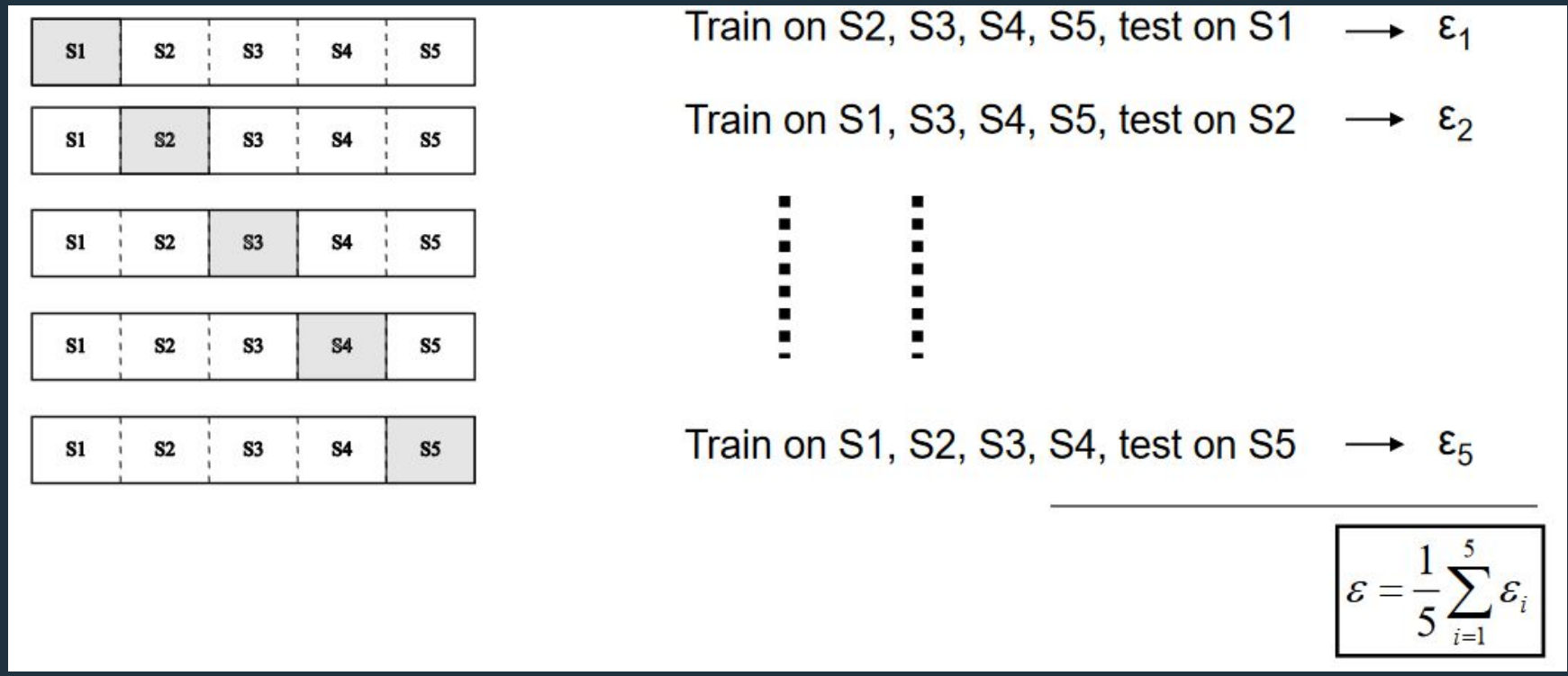
  The subsets should have similar class distribution

- Perform learning and testing K times

K-Fold Cross Validation

- Calculate the accuracy for each subset, average the accuracies.

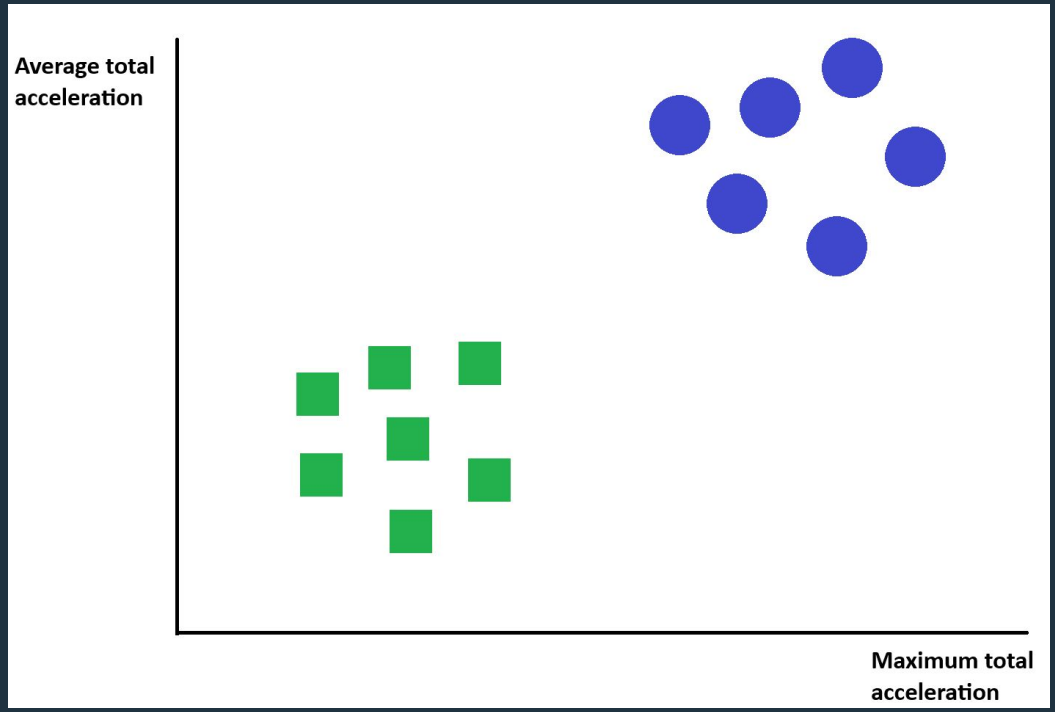
- Repeat this K-Fold Cross Validation process for different values of k (not capital) and select the k that provides the best average accuracy.



04 SUPPORT VECTOR MACHINES

k Nearest Neighbors (kNN)

- Consider again the same scenario and let’s plot each sample from the training set

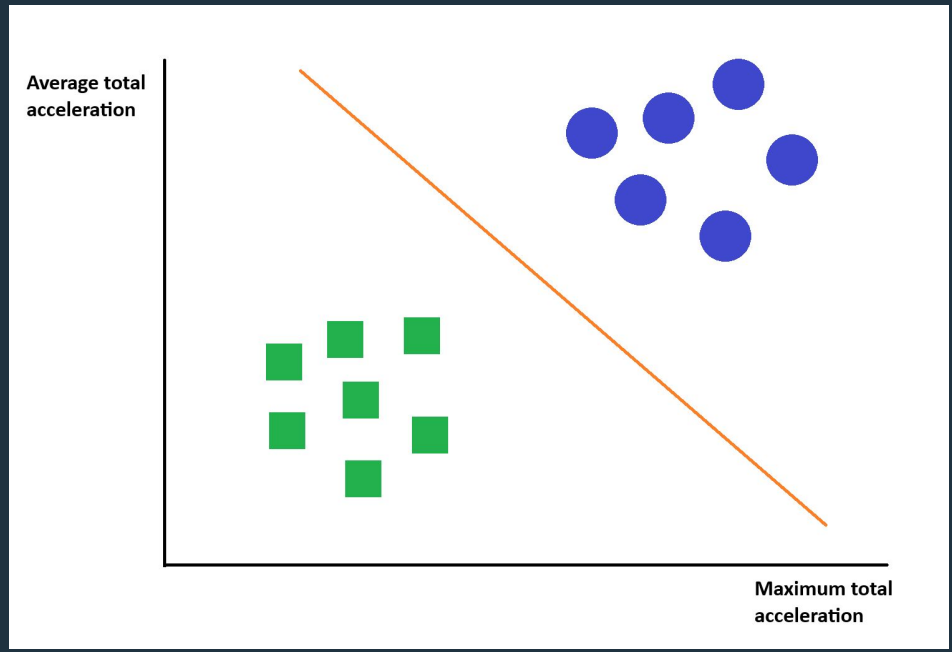


Support Vector Machines

- In the graph, each green square represents a sample of the “raising hand” class, whereas each blue circle represents a sample of the “punching” class.

- Support Vector Machines draw a **line** to divide these two classes.

- But where is it best to draw the line?

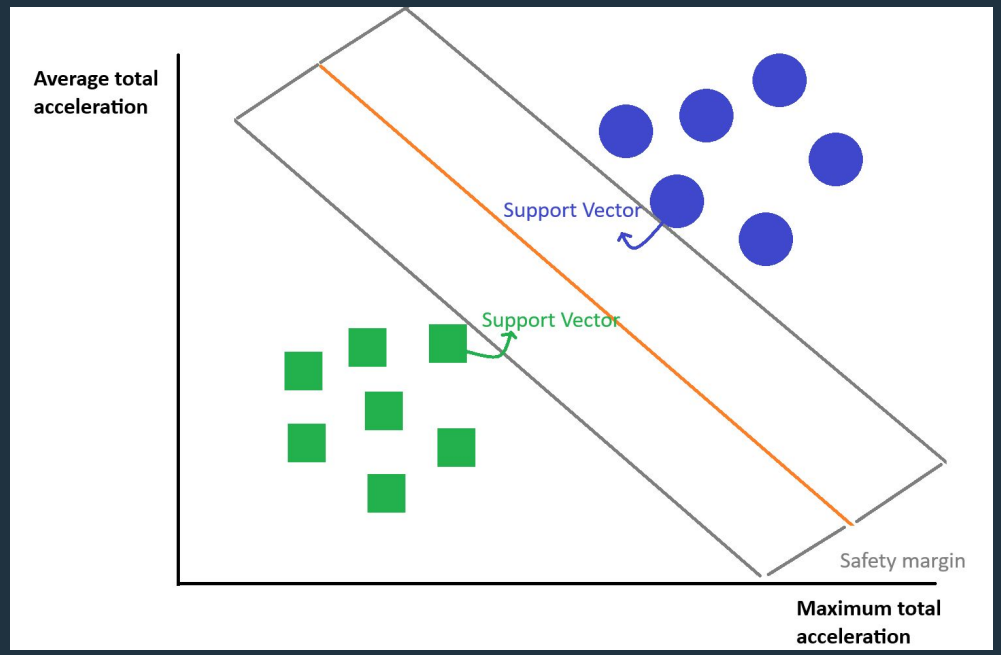


Support Vector Machines

- Smart way of drawing the line: you choose the line that has the most space between it and the nearest sample of each class.

- This space is like a safety margin.

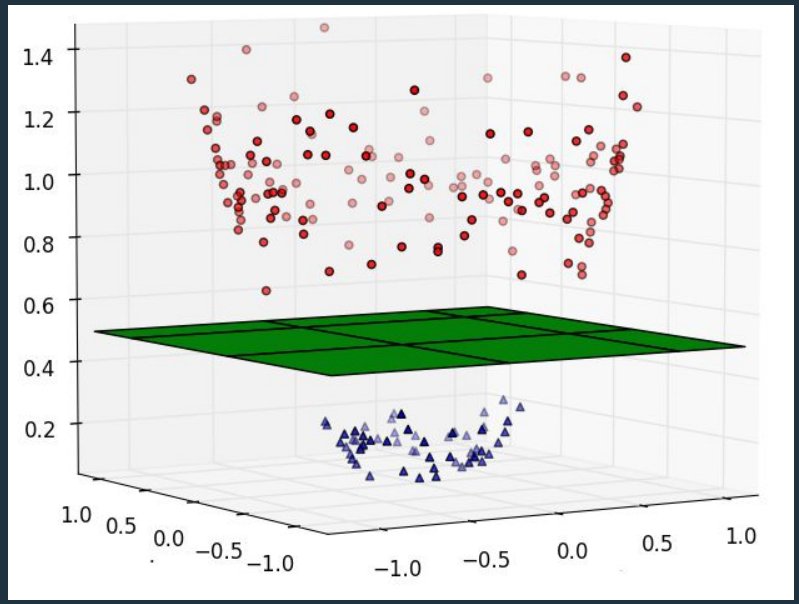
- The closest “punching” and “raising hand” samples are the **support vector**s.



Support Vector Machines

- What if we have more than 2 features? In the case of 3 features, the space is three-dimensional and instead of a line, we have a plane. Above 3 features, we have a hyperplane.

- The math behind finding the line, plane, or hyperplane is not explored in this course. We can use existing coding libraries for this task.



Support Vector Machines

- What if we have more than 2 classes? We have two strategies for that: **One-vs-One (OvO)** and **One-vs-All (OvA)**.

- In the strategy OvO, we create an individual classifier for each pair of classes.

- Hence, if we have 5 classes, in **OvO** we will have 10 classifiers.

- During prediction, each binary classifier votes for one of the classes.

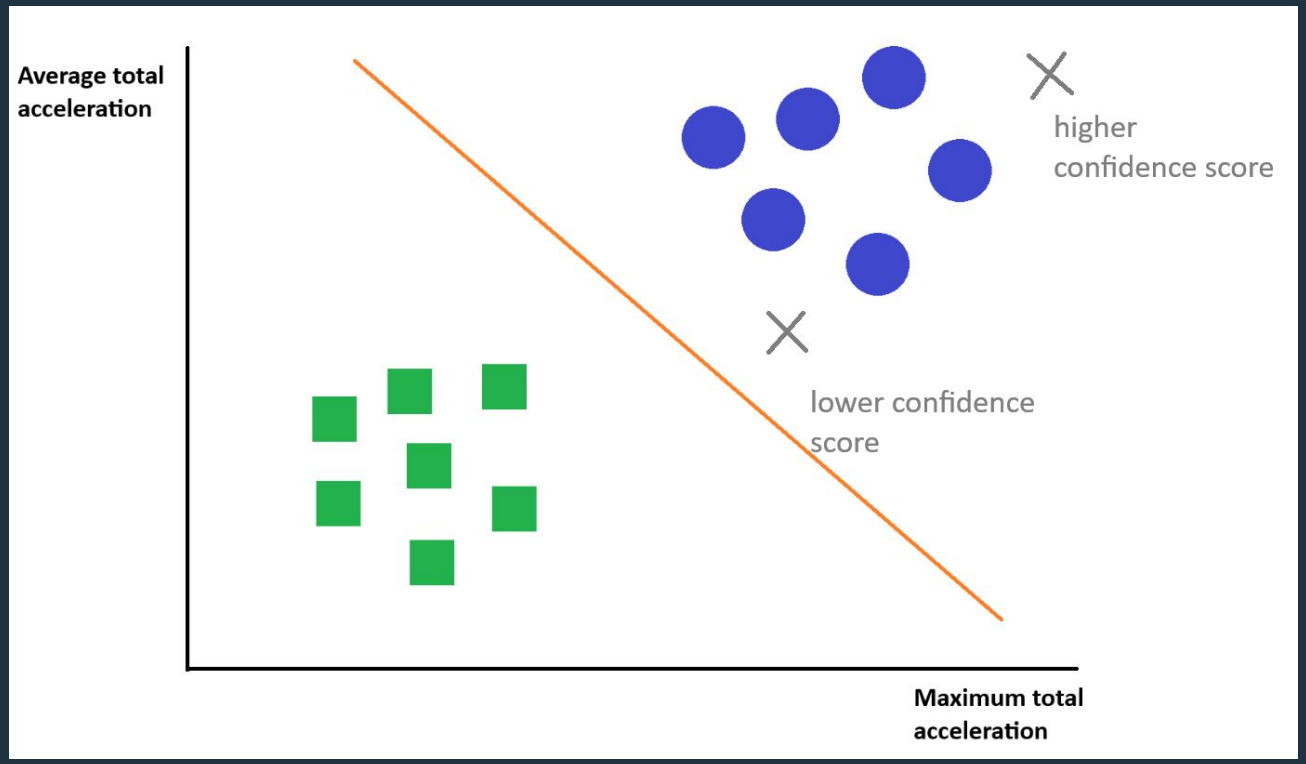
- The class with the **most** votes is the predicted class.

Support Vector Machines

- **OvA** is simpler. In this strategy, the number of classifiers is the number of classes.
- You train a binary SVM for each class versus the rest of the classes.
- During prediction, each binary classifier gives a **confidence score** for its class.
- The class with the highest confidence score is the predicted class.

Support Vector Machines

- **The confidence score** is a measure of how close to the line (plane, or hyperplane) are the predicted data.



Support Vector Machines

- For those familiar with mathematics, the SVM is defined as a hyperplane of equation **ω**·**x**+b = 0

- The vector **ω** and the constant **b** consists of the coefficients or weights

- Denoting **x'** **as** the feature vector of a certain sample, **ω**·**x'**+ b is the decision. That is, positive values of **ω**·**x'**+ b represent one class, whereas negative values represent another class

- The confidence score is calculate as the distance between **x'** and the hyperplane defined by **ω**·**x**+b = 0

Support Vector Machines

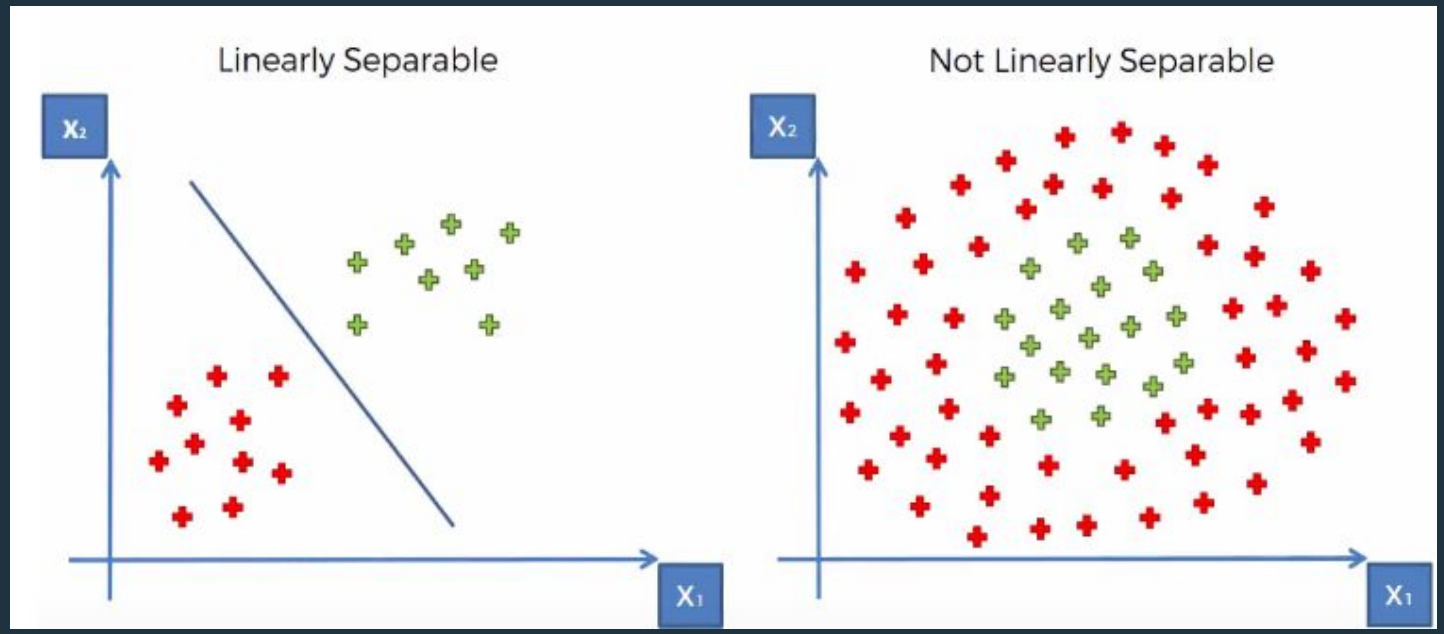
- The confidence score is calculate as the distance between $\mathbf{x}'$ and the hyperplane defined by $\mathbf{w} \cdot \mathbf{x} + b = 0$

- This is equal to: $|\mathbf{w} \cdot \mathbf{x}' + b| / ||\mathbf{w}||$

- It is possible to obtain a probability score (instead of distance) of a feature vector pertaining to a certain class. We will learn a bit more about it in the exercise sessions.

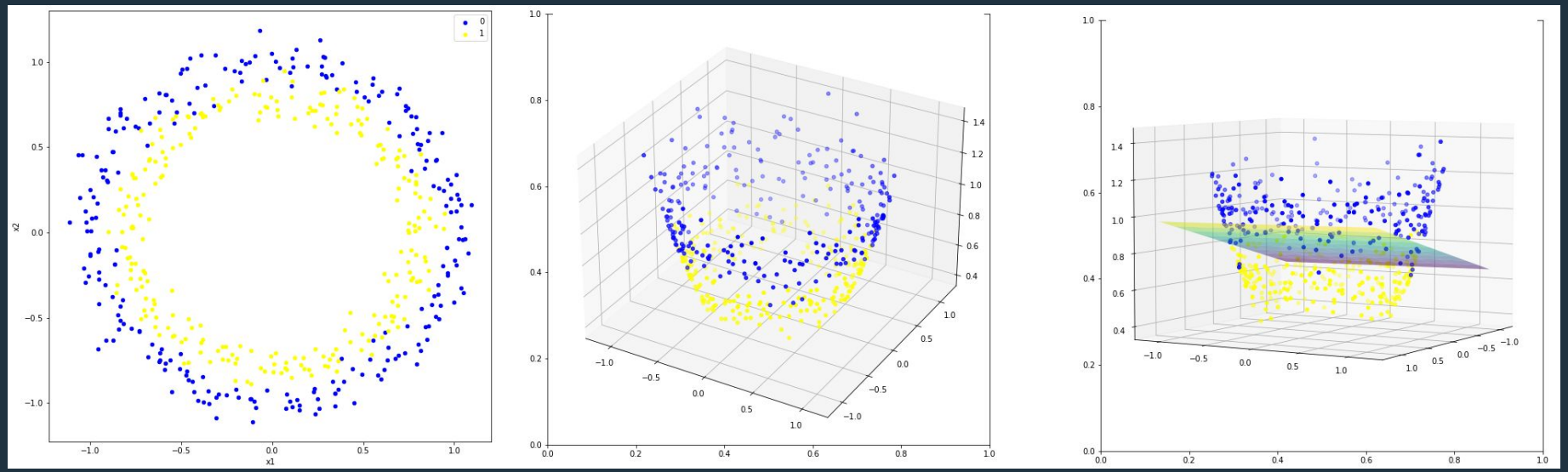
Support Vector Machines

- What if the data are not linearly separable?
- **Kernel trick:** map the feature vectors into a higher dimensional space



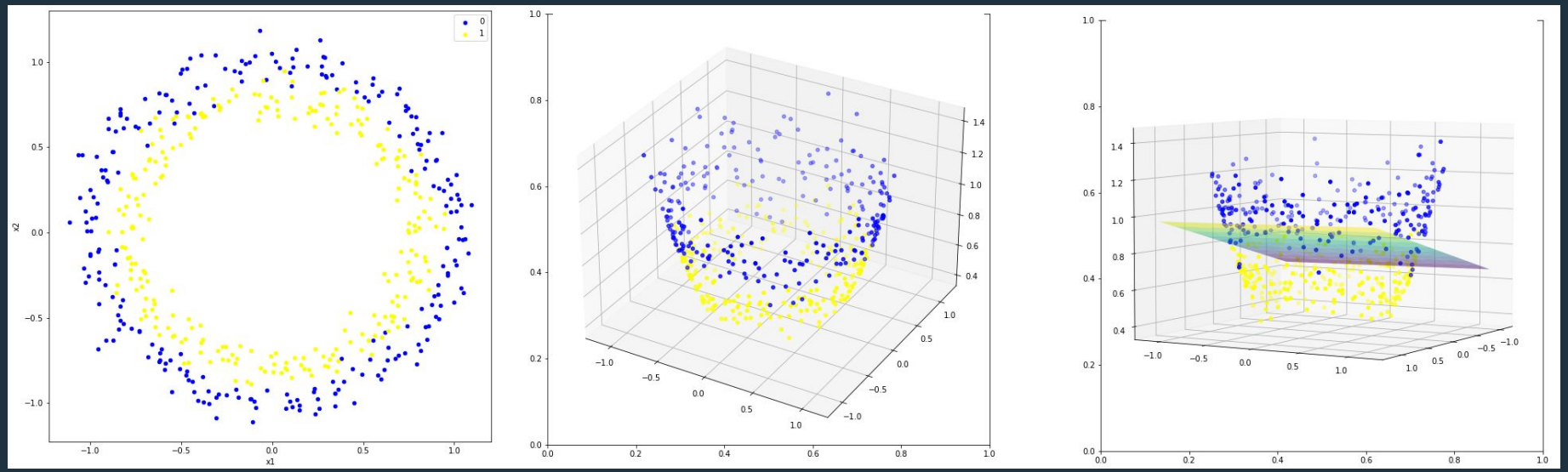
Support Vector Machines

- Consider the 2D data on the left figure. Blue and yellow represent different classes.
- A simple kernel trick is the create an additional dimension (a third dimension) for all the feature vectors.



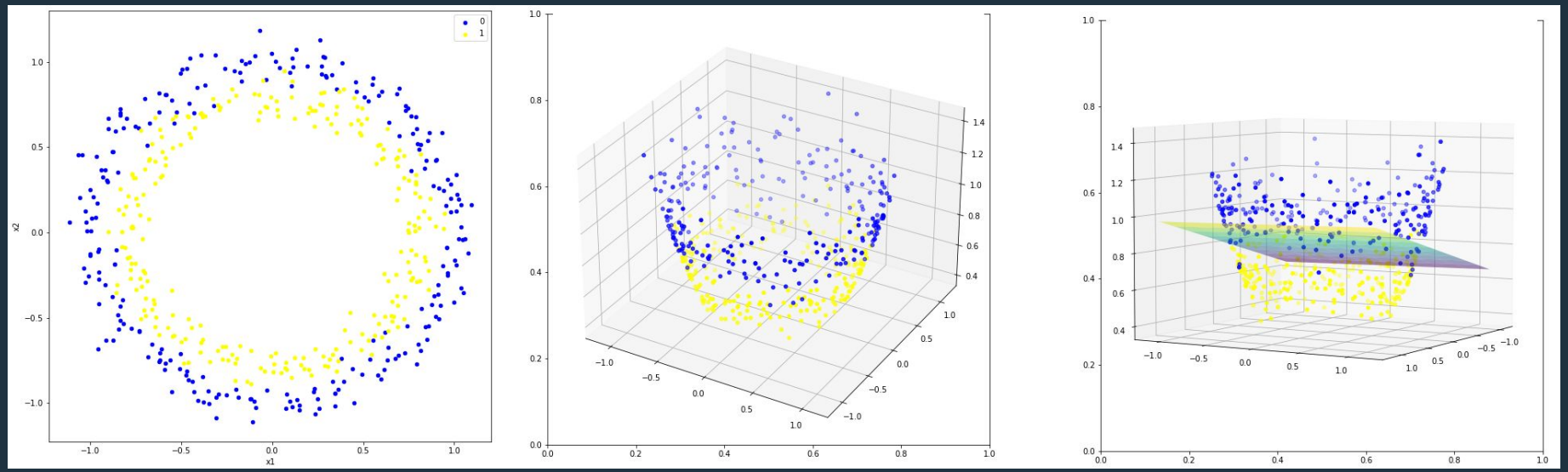
Support Vector Machines

- What could be the third dimension for a feature vector?



Support Vector Machines

- In this special case, the third dimension could simply be the distance of the sample to the center (0, 0)
- Hence, the data becomes linearly separable in 3D
- We go from an accuracy of ~50% to ~90%



Support Vector Machines

- The most commonly used kernel functions (responsible for performing the transforming into higher dimensions) are **polynomial** and **radial basis function**

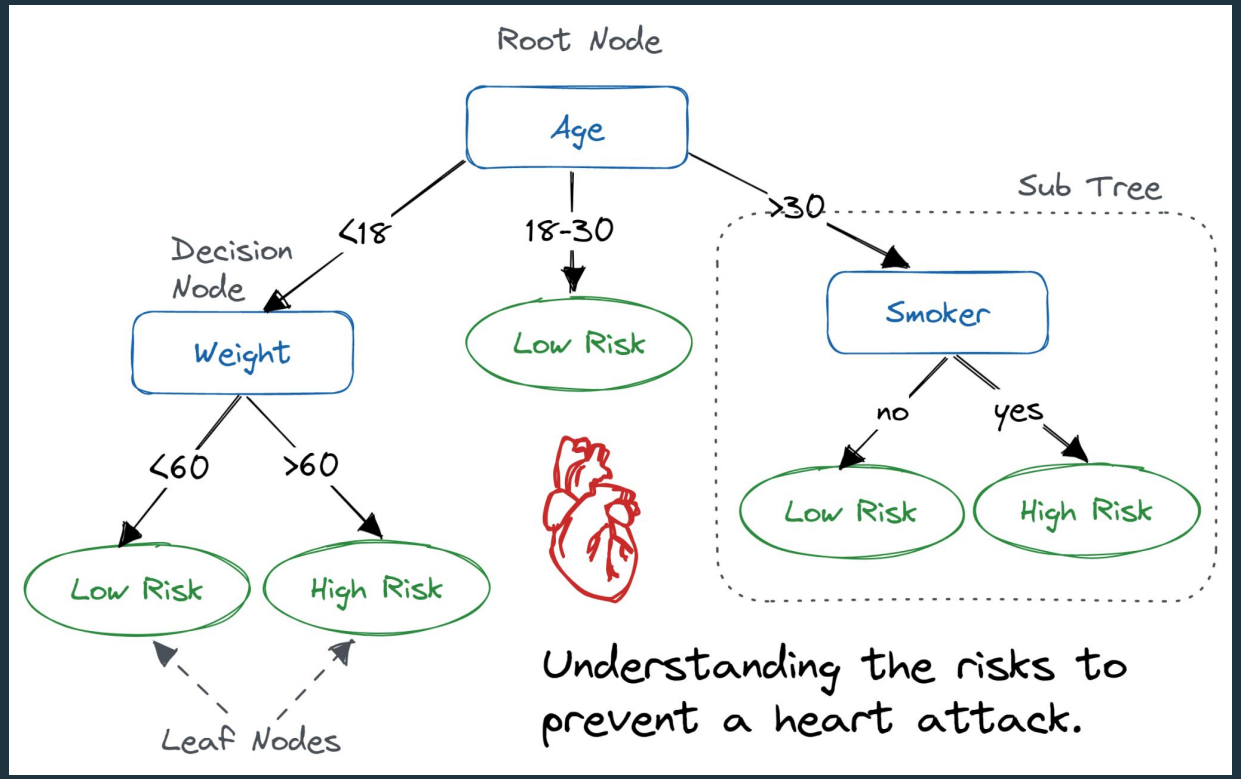
- We will skip their mathematical details. Just keep in mind the possibility of more complex kernels to handle nonlinear data.

- In the exercise section, we will explore these kernel functions in an easy way (no math required).

05 DECISION TREES

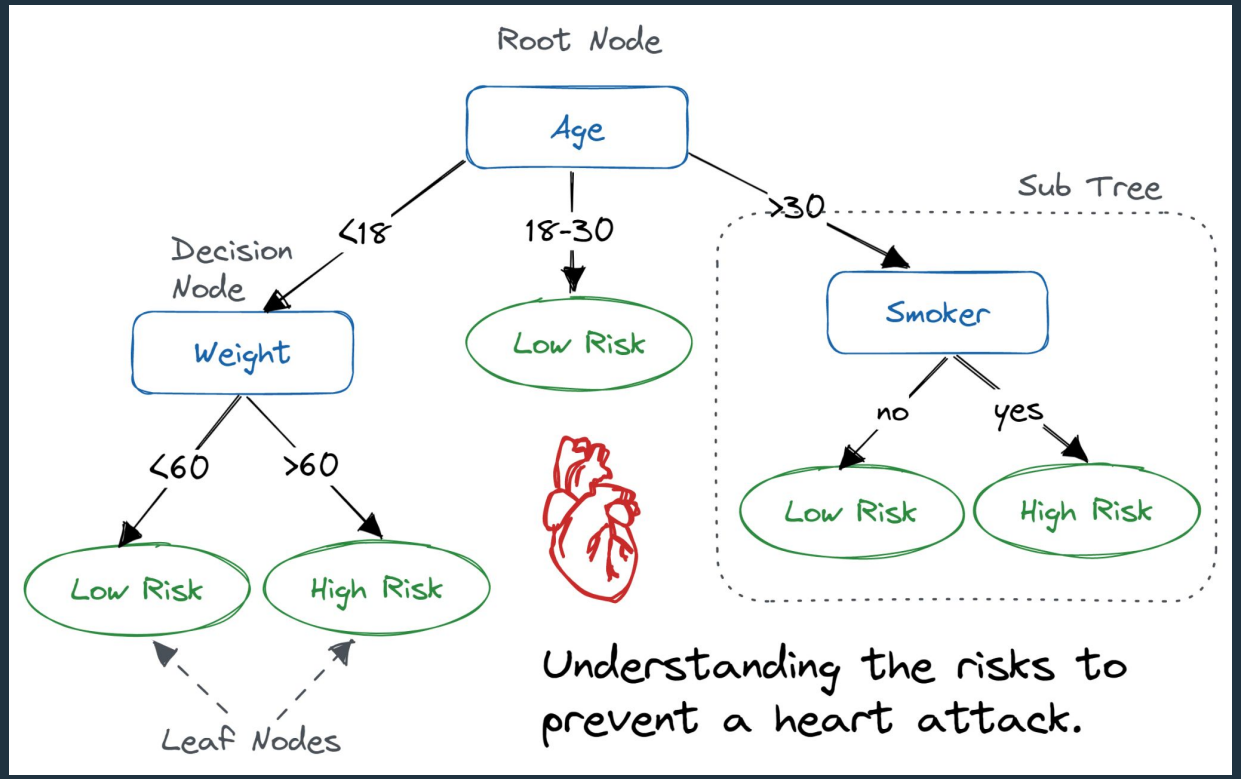
Decision Trees

- Decision Trees are hierarchical structures resembling trees
- The process starts at the **root node** (which is also a decision node)
- The decision node poses a question regarding a certain feature



Decision Trees

- Splitting happens at the decision node, leading to another decision node or, eventually, a leaf node (where the tree ends)
- Note that the tree can ask multiple times about a certain feature. Not only once.



Decision Trees

- How to create a decision tree?
- First, let’s place the data in a structure like a matrix or table
- Consider the problem of classifying between walking and running based on the features of *“mean total acceleration”* and *“mean total angular speed”*.

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

Decision Trees

- The next step is to split the data considering the features individually.
- In our case, we have two splits since we have two features.

Mean Total Acceleration	Class
3.2	walking
2.7	running
4.1	walking
0.9	walking
3.8	running
1.1	walking
4.3	running
2.4	walking
0.5	running
4.7	running

Mean Total Angular Speed	Class
1.8	walking
3.6	running
2.0	walking
4.4	walking
1.5	running
2.7	walking
3.9	running
0.7	walking
4.2	running
3.1	running

Decision Trees

- We then sort both splits according to the values of the features

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

Mean Total Angular Speed	Class
0.7	walking
1.5	running
1.8	walking
2.7	walking
2.0	walking
3.1	running
3.6	running
4.2	running
4.4	walking
3.9	running

Decision Trees

- Now, let's consider the feature "mean total acceleration"
- How do we create a decision rule that best splits the following data?
- For instance, let's consider the following decision rule:

  Is the mean total acceleration ≤ 0.7?

  Yes: 0 walking, 1 running

  No: 5 walking, 4 running

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

- By doing this, we are separating instances of the data into two groups: **yes** and **no**

- The best split leaves us with two groups (yes and no) that are the most **pure**

Decision Trees

- How we we quantify **impurity**? With a measure called “gini impurity”
- The gini impunity measures how mixed or impure a group of things is.

  Is the mean total acceleration ≤ 0.7?

  Yes: 0 walking, 1 running

  No: 5 walking, 4 running

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

- Gini impurity:

  $1 - (\text{prob of class walking})^2 - (\text{prob of class running})^2$

- Gini impurity for branch **Yes**: $1 - 0^2 - 1^2 = 0$
- Gini impurity for branch **No**: $1 - (5/9)^2 - (4/9)^2 = 0.4938$
- Weighted average of gini impurity for the decision: $(1/10)*0 + (9/10)*0.4938 = 0.444$

Decision Trees

- Now, let's consider the following decision rule:

  Is the mean total acceleration ≤ 1.0?

  Yes: 1 walking, 1 running

  No: 4 walking, 4 running

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

- Gini impurity:

  1 - (prob of class walking)² - (prob of class running)²

- Gini impurity for branch **Yes**: 1 - (1/2)² - (1/2)² = 0.5
- Gini impurity for branch **No**: 1 - (4/8)² - (4/8)² = 0.5
- Weighted average of gini impurity: (2/10)*0.5 + (8/10)*0.5 = 0.5

Decision Trees

- We proceed to test all the possible splitting points (mean of consecutive values):

  0.7, 1.0, 1.75, 2.55, 2.95, 3.5, 3.95, 4.2, 4.5

- We choose the splitting point that has the lowest gini impurity
- In our case, this was when mean total acceleration was ≤ 3.5

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

Decision Trees

- Now, let's consider the following decision rule:

  Is the mean total acceleration ≤ 4.2?

  Yes: 5 walking, 3 running

  No: 0 walking, 2 running

Mean Total Acceleration	Class
0.5	running
0.9	walking
1.1	walking
2.4	walking
2.7	running
3.2	walking
3.8	running
4.1	walking
4.3	running
4.7	running

- Gini impurity:

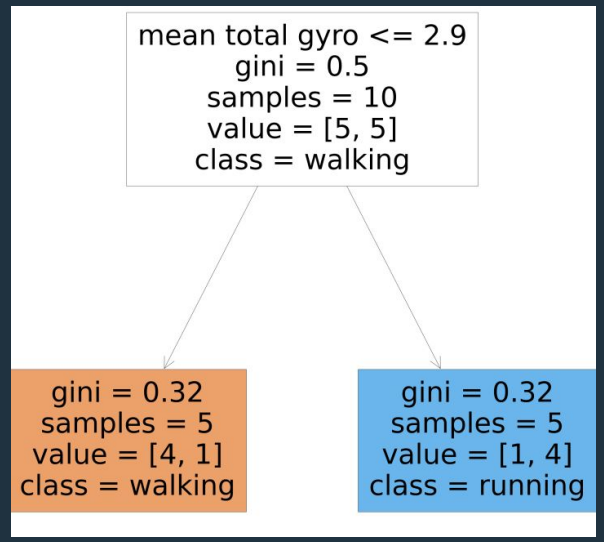
  1 - (prob of class walking)² - (prob of class running)²

- Gini impurity for branch **Yes**: 1 - (5/8)² - (3/8)² = 0.468
- Gini impurity for branch **No**: 1 - (0)² - (1)² = 0
- Average gini impurity for the decision: (8/10)*0.468 + (2/10)*0 = **0.374**

Decision Trees

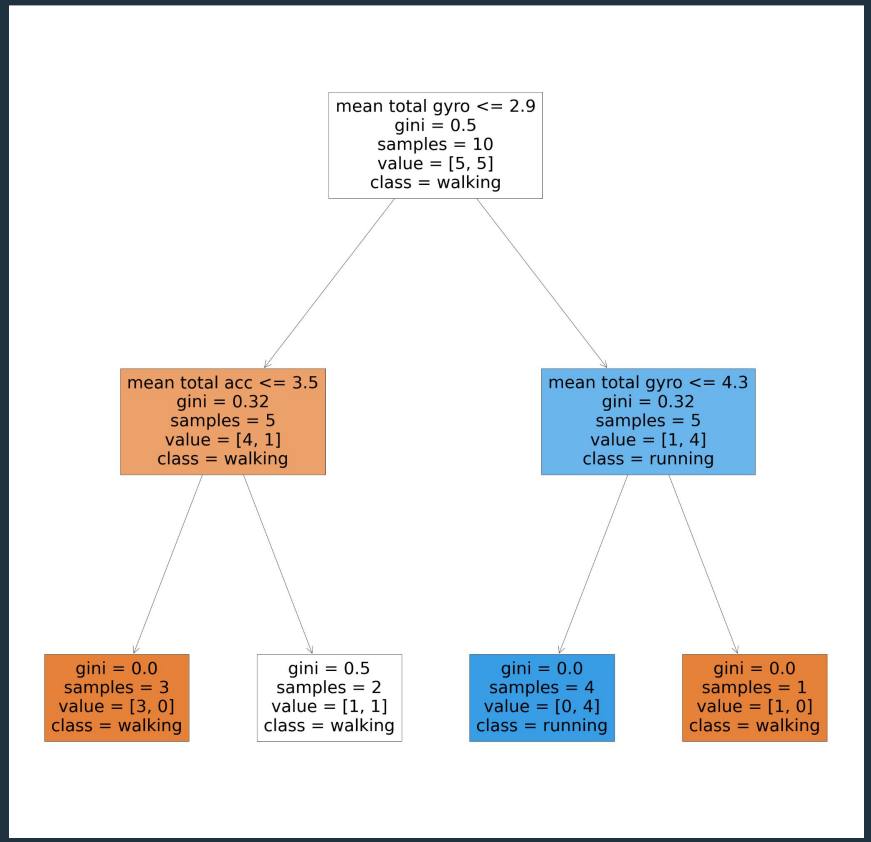
- We proceed to test all possible splitting points for the feature “mean total angular speed”.
- We find the best impurity when the mean total angular speed is ≤ 2.9
- This results in a gini impurity of 0.32

- Remember that, for the first feature, our best gini impurity was 0.374
- For this reason, we will choose “*mean total angular speed ≤ 2.9*” to be the root node



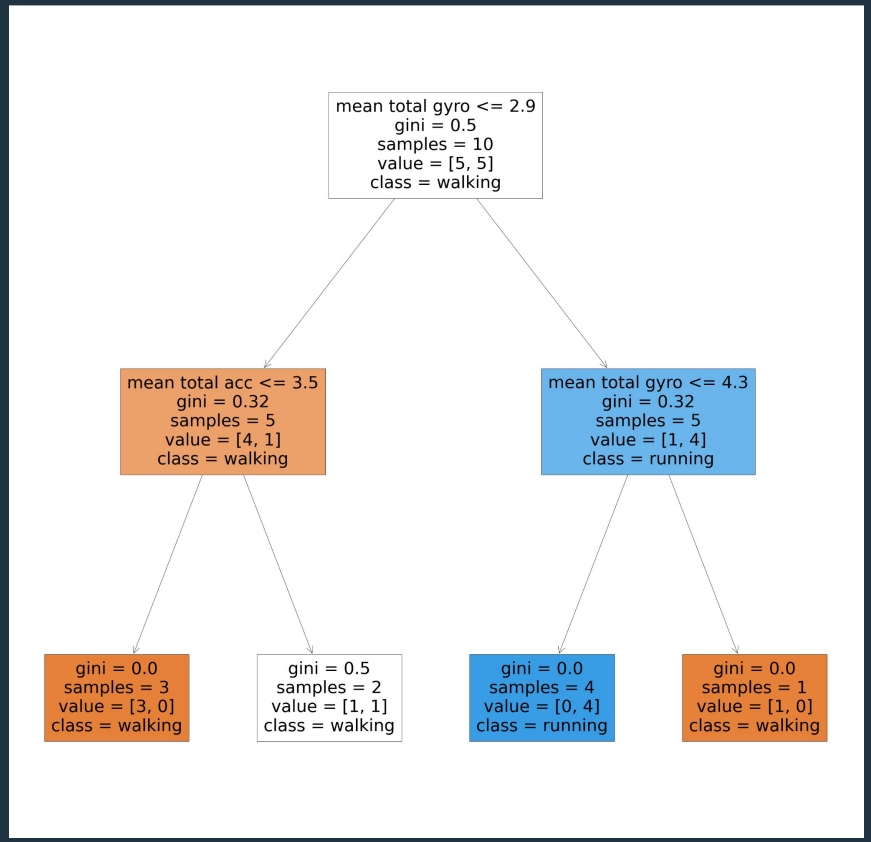
Decision Trees

- We now have two branches, giving us two portions of the dataset. The portion where the mean total angular speed ≤ 2.9 and the portion where > 2.9
- We do the same process for both portions of the dataset.



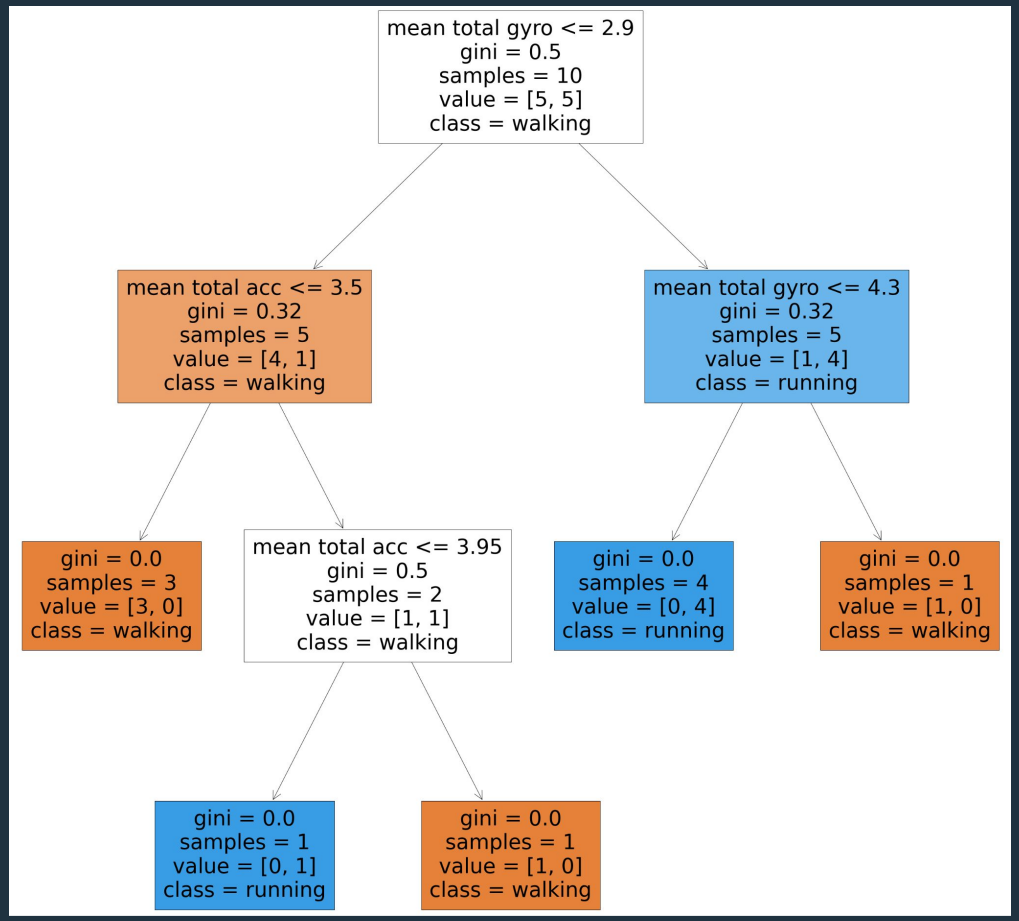
Decision Trees

- Our tree has now depth 2.
- We can keep going or stop here. We decide to keep going to have purer leaf nodes.
- There is only one node that can be further “purified”.



Decision Trees

- All leaf nodes have the lowest possible gini impurity. So we can stop here.



Decision Trees

- When is it best to stop the recursive process of creating the decision tree?
- There is no general rule
- We can choose to stop at a certain depth (for instance, max. depth 5) or when the maximum gini impurity of the leaf nodes is lower than a predefined threshold.

- Low depth: possibility of underfitting
- Large depth: possibility of overfitting

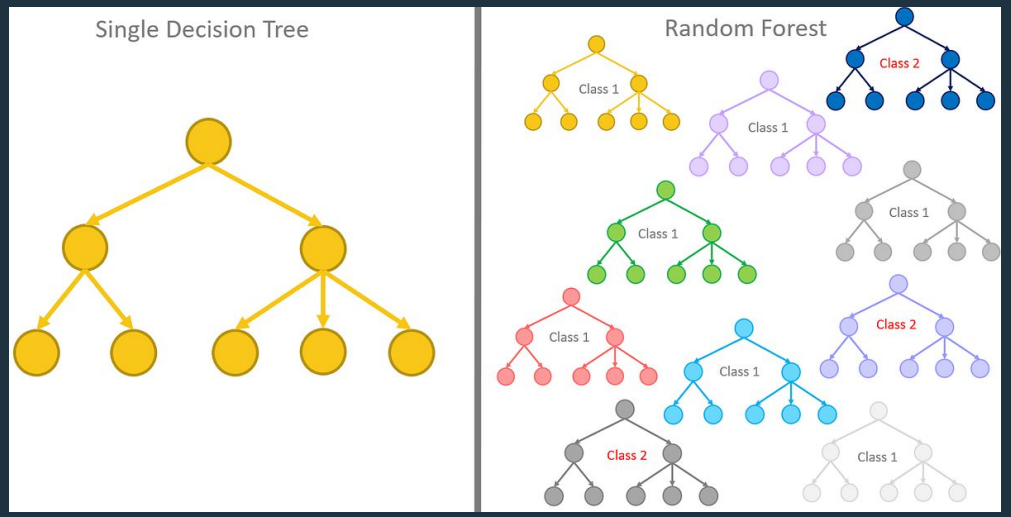
- What do we do? Validation to find the best model!

06 RANDOM FOREST

Random Forest

- Random Forest is a group of decision trees, each of them employed to solve the same problem.

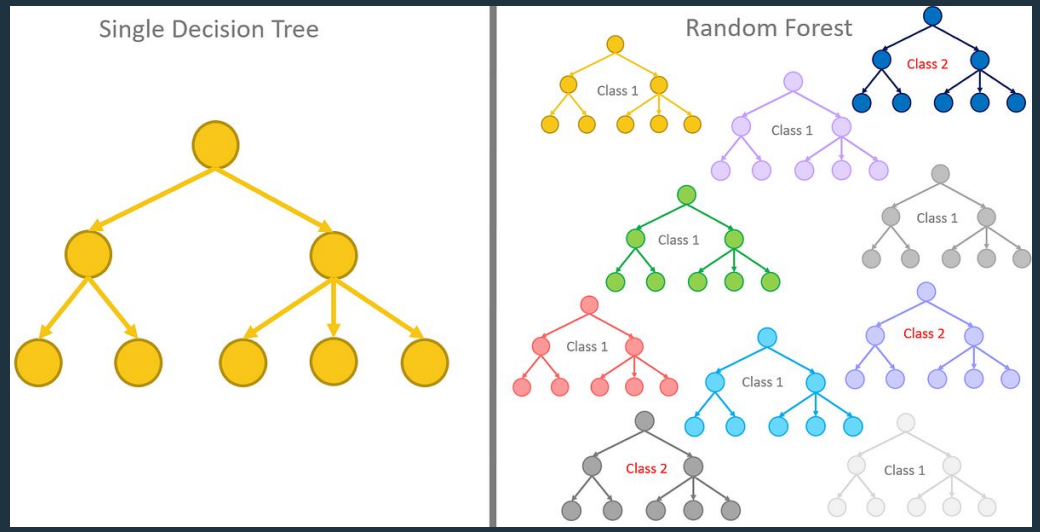
- Hence, instead of building only one decision tree for our classification problem, we build several decision trees.



Random Forest

- When we need to classify data, we use all the decision trees individually. Each of them will provide us with a class.

- Our final class will be the majority class.



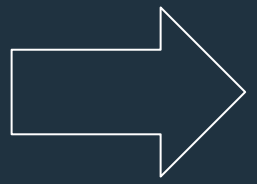
Random Forest

- How do we use randomness to create multiple different decision trees?
- Initially, we specify the number of decision trees we desire within the random forest.
- Consider the “running” and “walking” classification task and suppose that we choose the random forest to have five decision trees.
- Next, we employ randomness to generate five distinct subsets from the original training dataset. Each of these subsets serves as the training data for an individual decision tree.

Random Forest

- To further increase variability we allow data points to repeat within the subsets

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



Subset 1

Mean Total Acceleration	Mean Total Angular Speed	Class
3.8	1.5	running
0.5	4.2	running
4.7	3.1	running
4.1	2.0	walking
2.7	3.6	running

Subset 2

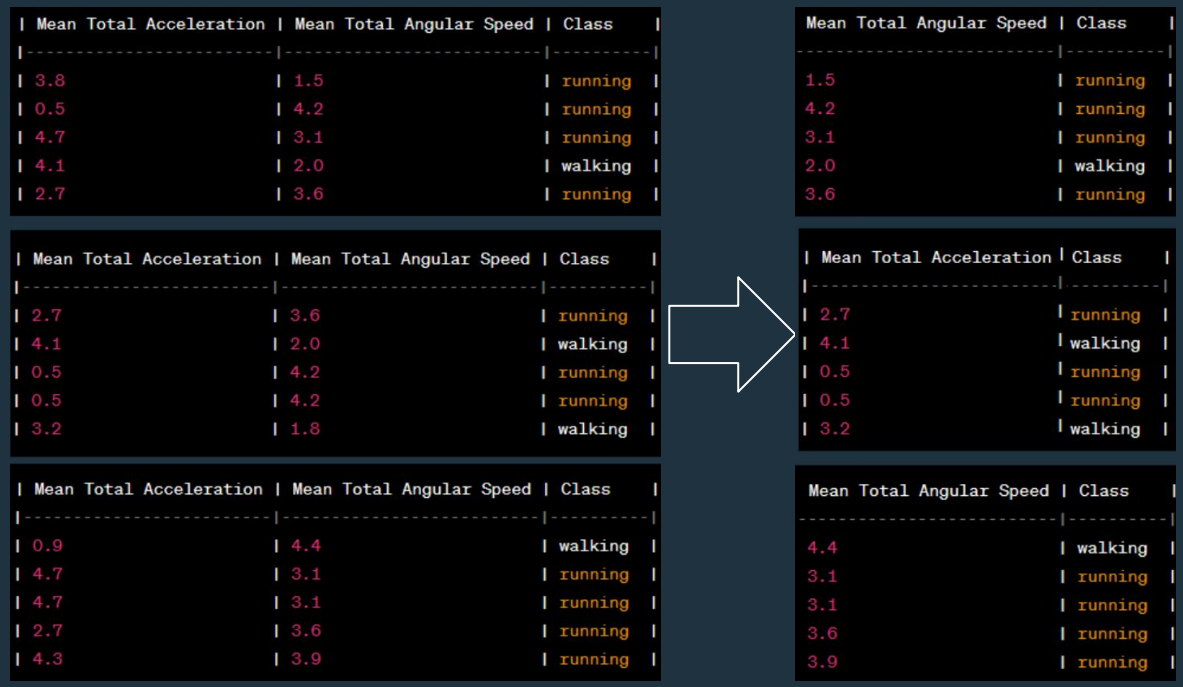
Mean Total Acceleration	Mean Total Angular Speed	Class
2.7	3.6	running
4.1	2.0	walking
0.5	4.2	running
0.5	4.2	running
3.2	1.8	walking

Subset 3

Mean Total Acceleration	Mean Total Angular Speed	Class
0.9	4.4	walking
4.7	3.1	running
4.7	3.1	running
2.7	3.6	running
4.3	3.9	running

Random Forest

- Additionally, we randomly exclude certain features from each subset, which helps diversify our RF.
- The number of features in each subset is another decision we need to make.
- In our case, where we only have two features, we can remove a maximum of one feature from each subset.



Mean Total Acceleration	Mean Total Angular Speed	Class
3.8	1.5	running
0.5	4.2	running
4.7	3.1	running
4.1	2.0	walking
2.7	3.6	running

Mean Total Angular Speed	Class
1.5	running
4.2	running
3.1	running
2.0	walking
3.6	running

Mean Total Acceleration	Mean Total Angular Speed	Class
2.7	3.6	running
4.1	2.0	walking
0.5	4.2	running
0.5	4.2	running
3.2	1.8	walking

Mean Total Acceleration	Class
2.7	running
4.1	walking
0.5	running
0.5	running
3.2	walking

Mean Total Acceleration	Mean Total Angular Speed	Class
0.9	4.4	walking
4.7	3.1	running
4.7	3.1	running
2.7	3.6	running
4.3	3.9	running

Mean Total Angular Speed	Class
4.4	walking
3.1	running
3.1	running
3.6	running
3.9	running

Random Forest

- Next, we proceed to train all the decision trees in our Random Forest.

- Suppose we choose to have decision trees with maximum depth 2.

Subset 1

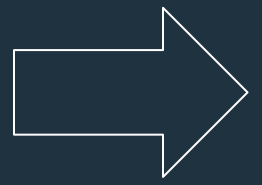
Mean Total Angular Speed	Class
1.5	running
4.2	running
3.1	running
2.0	walking
3.6	running

Subset 2

Mean Total Acceleration	Class
2.7	running
4.1	walking
0.5	running
0.5	running
3.2	walking

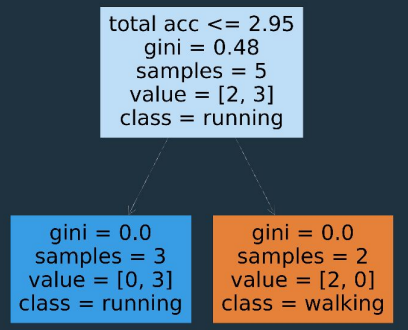
Subset 3

Mean Total Angular Speed	Class
4.4	walking
3.1	running
3.1	running
3.6	running
3.9	running

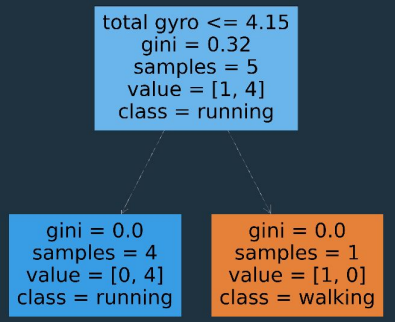


Tree 1

Tree 2



Tree 3



Random Forest

- Apart from the maximum depth of the decision trees, recall that we made specific choices for three other important values.

- That is, the number of trees, the number of data points in each subset, and the number of features considered in each subset.

- Now, the question arises: How can we assess if these choices are suitable and effective for our specific problem?

Random Forest

- One interesting aspect of RF is that we do not need a validation set to evaluate it.
- That means that data points that would normally populate a validation set can be all assigned to the training set.
- But then, how do we evaluate the performance of the RF?

Random Forest

- One interesting aspect of RF is that we do not need a validation set to evaluate it.
- That means that data points that would normally populate a validation set can be all assigned to the training set.
- But then, how do we evaluate the performance of the RF?

Random Forest

- First, let’s go through all data points in the original training set, checking in which subsets they are not present.

Mean Total Angular Speed	Class
1.5	running
4.2	running
3.1	running
2.0	walking
3.6	running

Subset 1

Mean Total Acceleration	Class
2.7	running
4.1	walking
0.5	running
0.5	running
3.2	walking

Subset 2

Mean Total Angular Speed	Class
4.4	walking
3.1	running
3.1	running
3.6	running
3.9	running

Subset 3

Mean Total Acceleration	Mean Total Angular Speed	Class	
3.2	1.8	walking	not in subsets 1 and 3
2.7	3.6	running	present in all subsets
4.1	2.0	walking	not in subset 3
0.9	4.4	walking	not in subsets 1 and 2
3.8	1.5	running	not in subsets 2 and 3
1.1	2.7	walking	not in subsets 1, 2, and 3
4.3	3.9	running	not in subsets 1 and 2
2.4	0.7	walking	not in subsets 1, 2 and 3
0.5	4.2	running	not in subset 3
4.7	3.1	running	not in subset 2

Random Forest

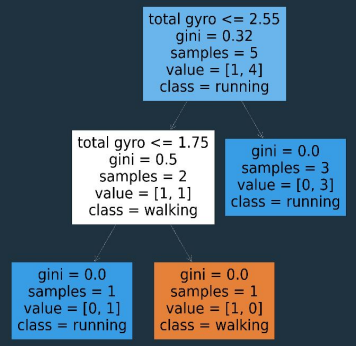
- We proceed to classify each data point using the trees where it was not used during the training

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

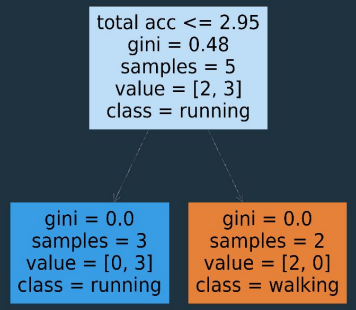
not in subsets 1 and 3	→ run it through trees 1 and 3
present in all subsets	→ not used
not in subset 3	→ run it through tree 3
not in subsets 1 and 2	→ run it through trees 1 and 2
not in subsets 2 and 3	→ run it through trees 2 and 3
not in subsets 1, 2, and 3	→ run it through all trees
not in subsets 1 and 2	→ run it through trees 1 and 2
not in subsets 1, 2 and 3	→ run it through all trees
not in subset 3	→ run it through tree 3
not in subset 2	→ run it through tree 2

Random Forest

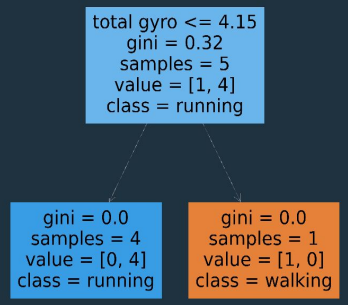
Tree 1



Tree 2



Tree 3

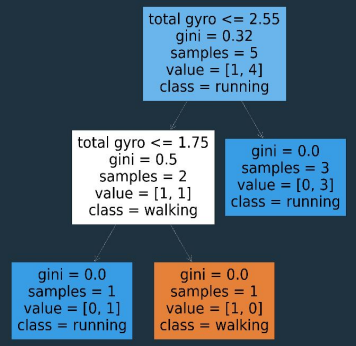


Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

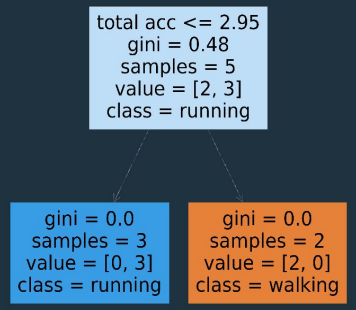
run it through trees 1 and 3 → walking, running
not used
run it through tree 3 → running
run it through trees 1 and 2 → running, running
run it through trees 2 and 3 → running, running
run it through all trees → running, running, running
run it through trees 1 and 2 → running, walking
run it through all trees → running, running, running
run it through tree 3 → walking
run it through tree 2 → walking

Random Forest

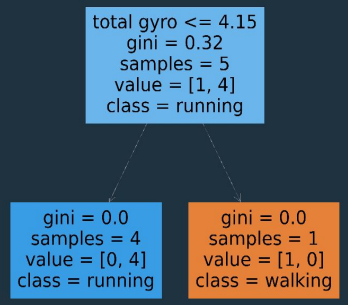
Tree 1



Tree 2



Tree 3



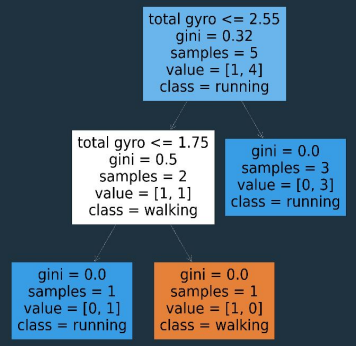
- Next, we find the majority vote for each data point.
- To break ties between trees, we can check the leaf nodes responsible for giving us the prediction.

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

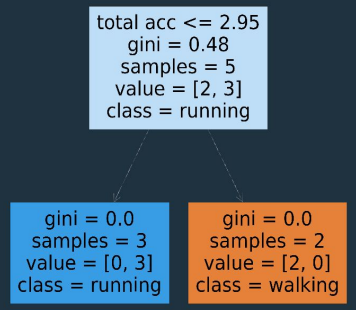
```
trees 1 and 3  → walking, running          →
not used
tree 3         → running                   → running
trees 1 and 2  → running, running          → running
trees 2 and 3  → running, running          → running
all trees      → running, running, running → running
trees 1 and 2  → running, walking          →
all trees      → running, running, running → running
tree 3         → walking                   → walking
tree 2         → walking                   → walking
```

Random Forest

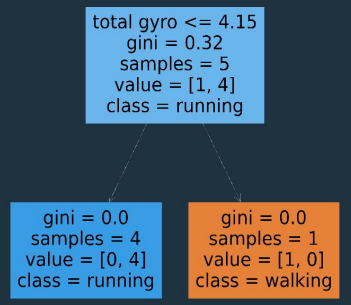
Tree 1



Tree 2



Tree 3



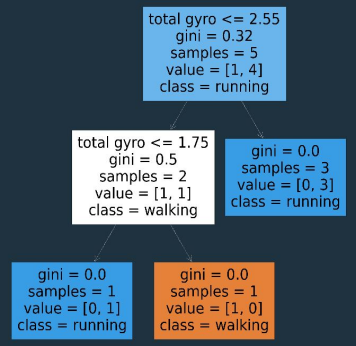
- Tree 1 predicted *(3.2, 1.8)* to be walking and the leaf node responsible for this classification has 100% data points of class walking.
- Tree 3 predicted *(3.2, 1.8)* to be running and the leaf node responsible for this classification contains also contains 100% samples of running.
- Another tie! Which one makes more sense to be the conclusion? Walking or running?

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

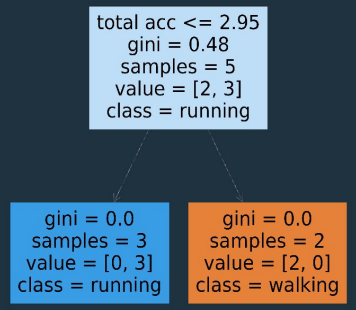
```
trees 1 and 3  → walking, running          →
not used
tree 3         → running                   → running
trees 1 and 2  → running, running          → running
trees 2 and 3  → running, running          → running
all trees      → running, running, running → running
trees 1 and 2  → running, walking          →
all trees      → running, running, running → running
tree 3         → walking                   → walking
tree 2         → walking                   → walking
```

Random Forest

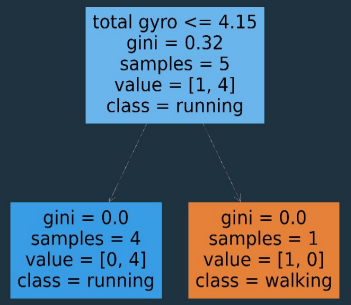
Tree 1



Tree 2



Tree 3



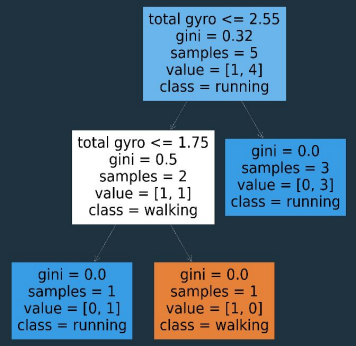
- In this exceptional case of tie, we choose the prediction of Tree 3
- Do you know why?

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

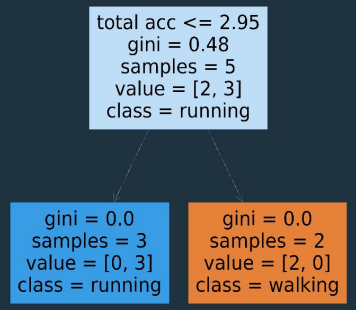
```
trees 1 and 3  → walking, running          → running
not used
tree 3         → running                   → running
trees 1 and 2  → running, running          → running
trees 2 and 3  → running, running          → running
all trees      → running, running, running → running
trees 1 and 2  → running, walking          →
all trees      → running, running, running → running
tree 3         → walking                   → walking
tree 2         → walking                   → walking
```

Random Forest

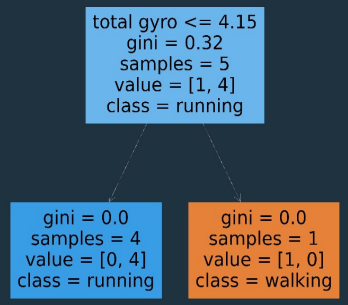
Tree 1



Tree 2



Tree 3

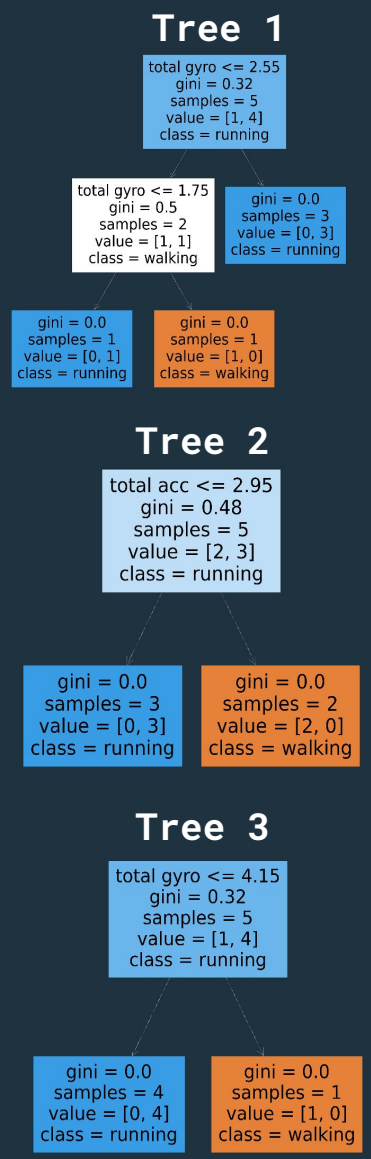


- Because there are more samples in the leaf of Tree 3 than of that of Tree 1
- Similarly, we break the tie in the sample *(4.3, 3.9)*
- We get 8 errors out of 9. An error rate of 88.8% (out-of-bag error)

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

```
trees 1 and 3  → walking, running          → running
not used
tree 3         → running                   → running
trees 1 and 2  → running, running          → running
trees 2 and 3  → running, running          → running
all trees      → running, running, running → running
trees 1 and 2  → running, walking          → running
all trees      → running, running, running → running
tree 3         → walking                   → walking
tree 2         → walking                   → walking
```

Random Forest



- **Terminology**: The subsets are usually called “**bags**”, the data points that are not included in a certain bag are called “**out-of-bag**” data points.
- The process of building **bags** is called **bootstrapping**
- Combining predictions of different decision trees is called **aggregating**
- These two processes are called **bootstrapping aggregating** or shortly **bagging**

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

```
trees 1 and 3  → walking, running          → running
not used
tree 3         → running                   → running
trees 1 and 2  → running, running          → running
trees 2 and 3  → running, running          → running
all trees      → running, running, running → running
trees 1 and 2  → running, walking          → running
all trees      → running, running, running → running
tree 3         → walking                   → walking
tree 2         → walking                   → walking
```

Random Forest

- This OOB error is pretty bad. How can we improve our RF?
- Let’s try increasing the number of samples in the subsets and the number of trees.
- We keep the maximum depth of the trees set to 2 since our dataset is very small.

Random Forest

- Different options and their OOB error:

  5 trees, 6 samples → 30%

  8 trees, 7 samples → 40%

  10 trees, 8 samples → 40%

  12 trees, 9 samples → 30%

- The first and last two options seem to be the best.

- The first option is the simplest and, thus, can be the chosen one.

- What is the next step? Find the performance on the test set.