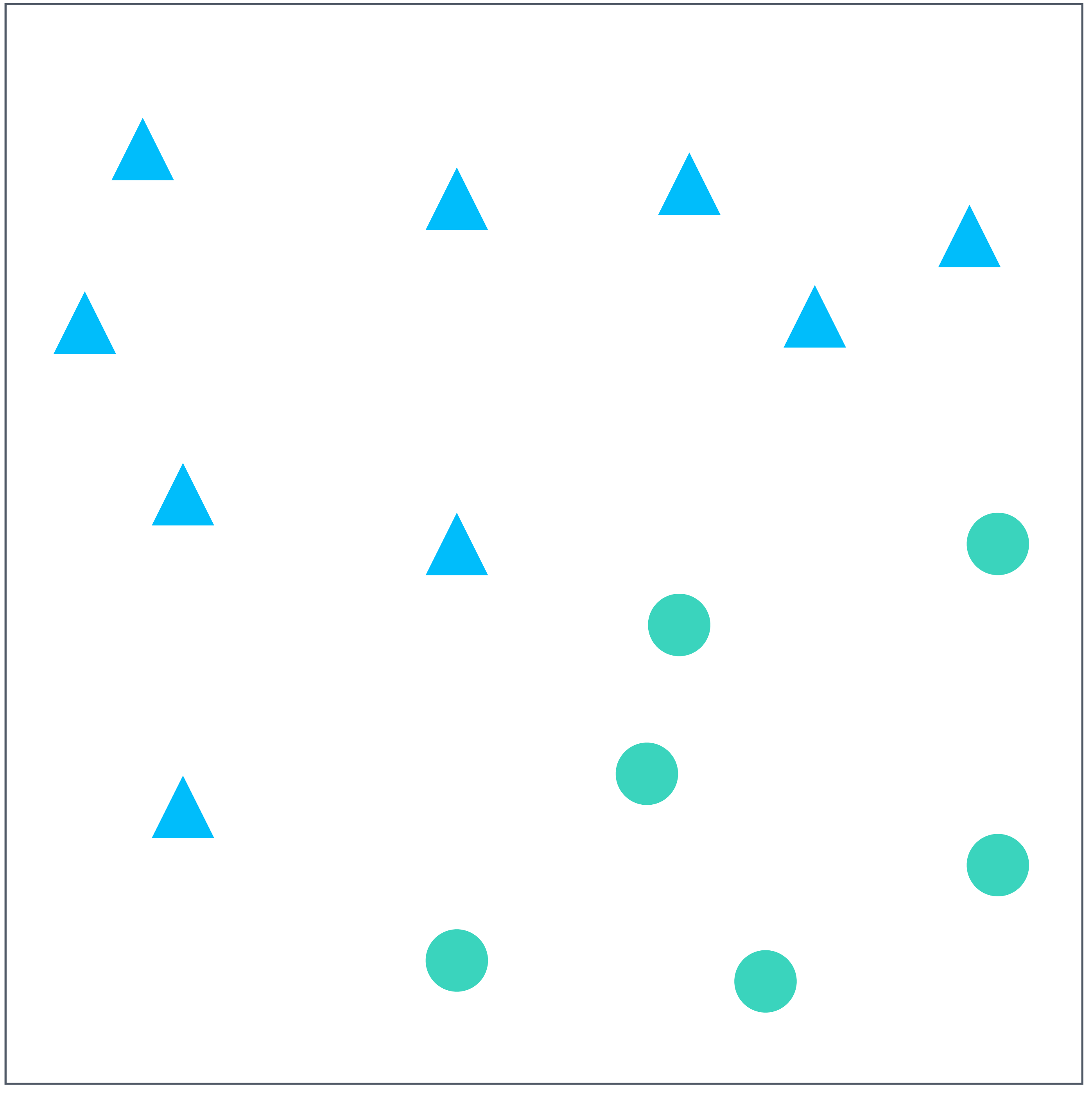
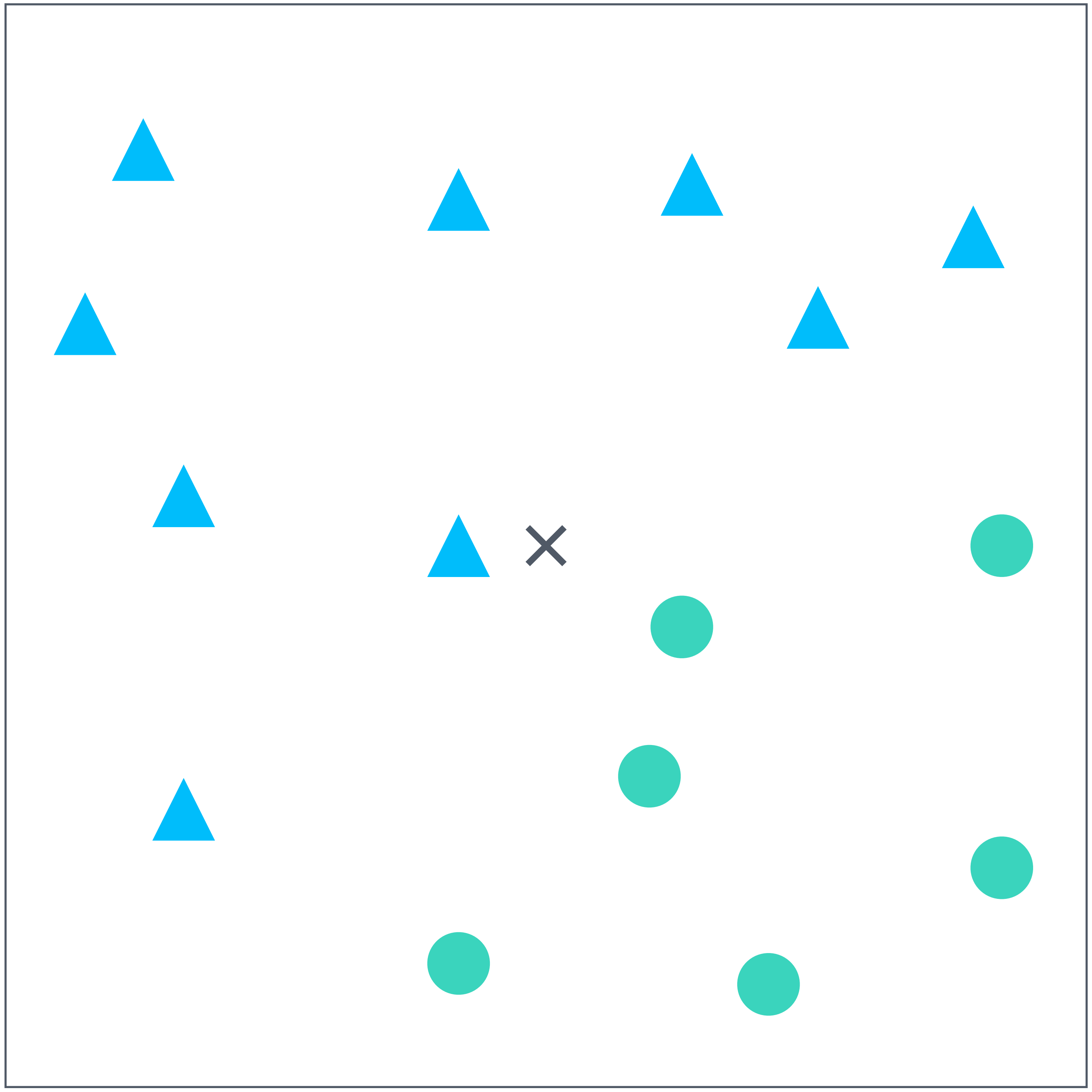
KNN and Decision Trees

Machine Learning Crash Course
UCL
6.10.2017

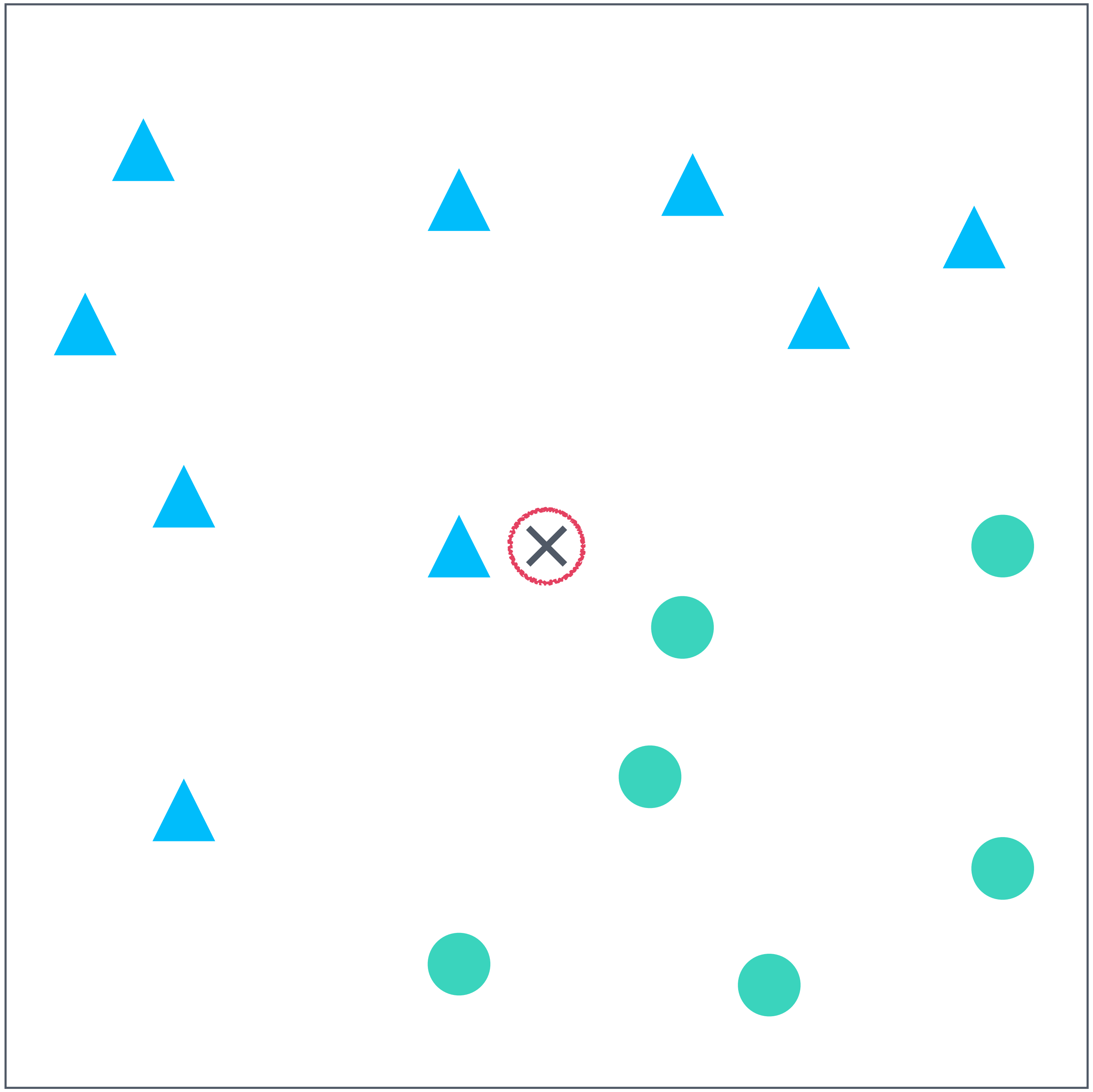
K-Nearest Neighbours (KNN)



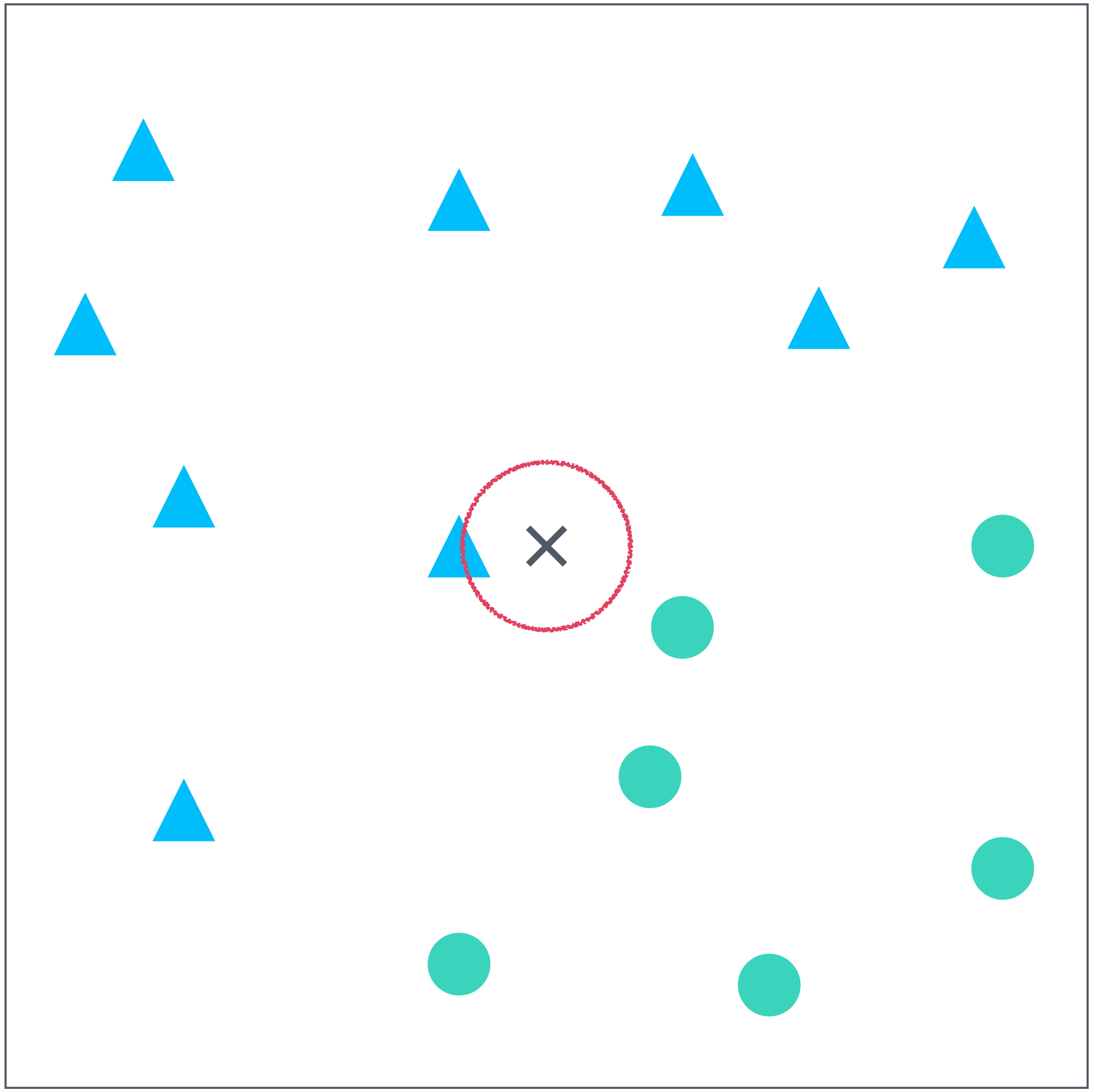
KNN



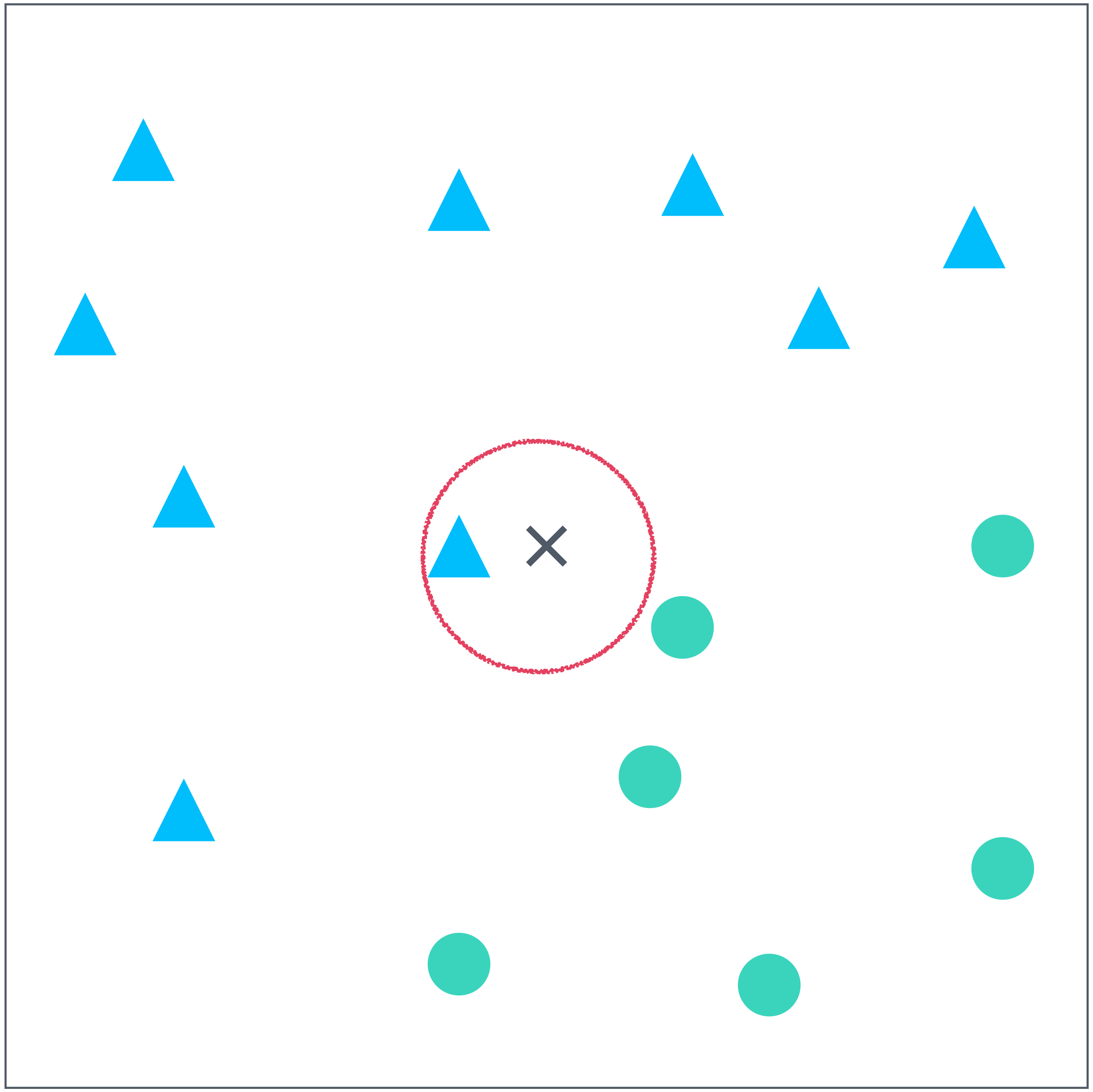
KNN



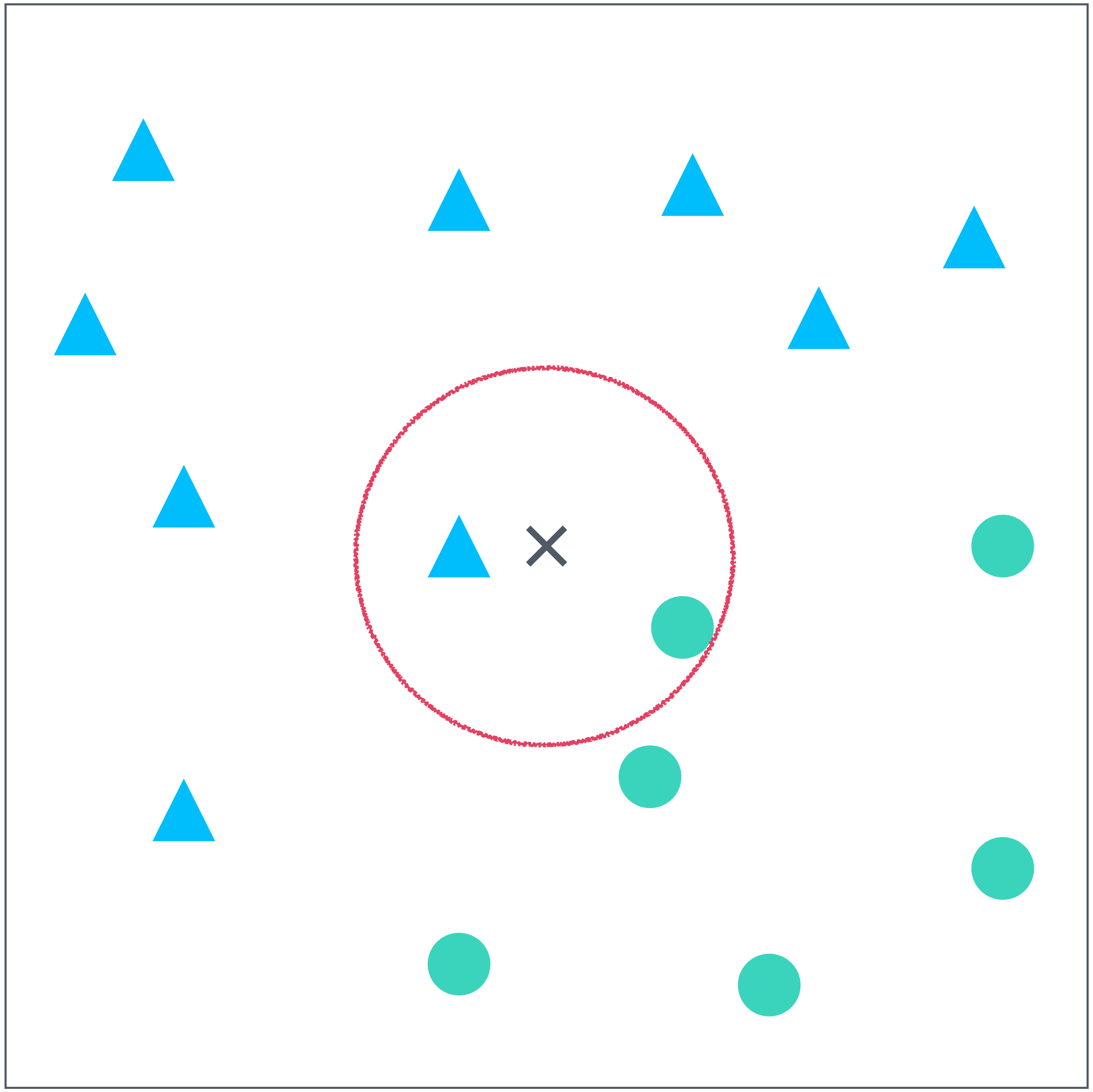
KNN



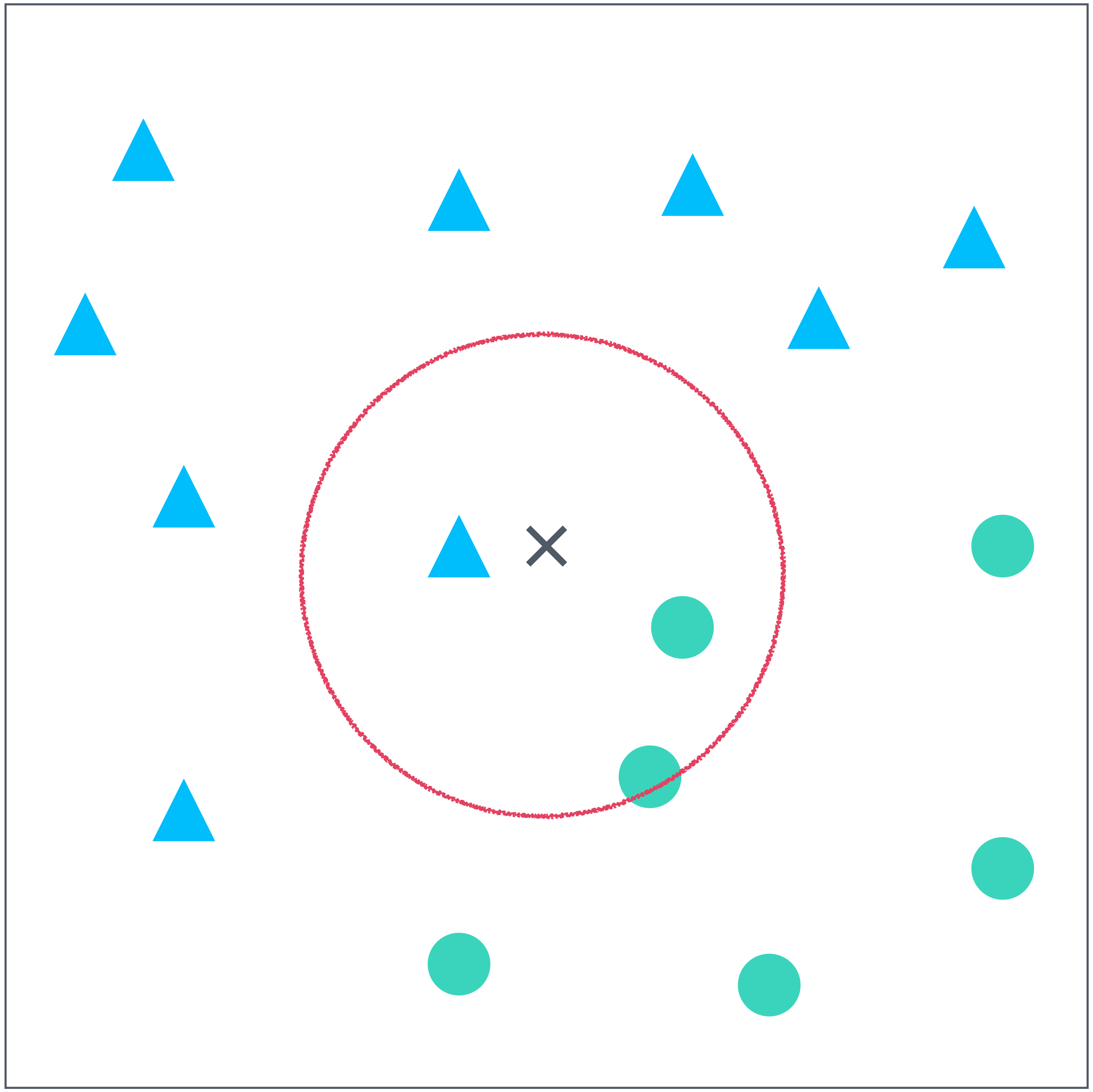
KNN



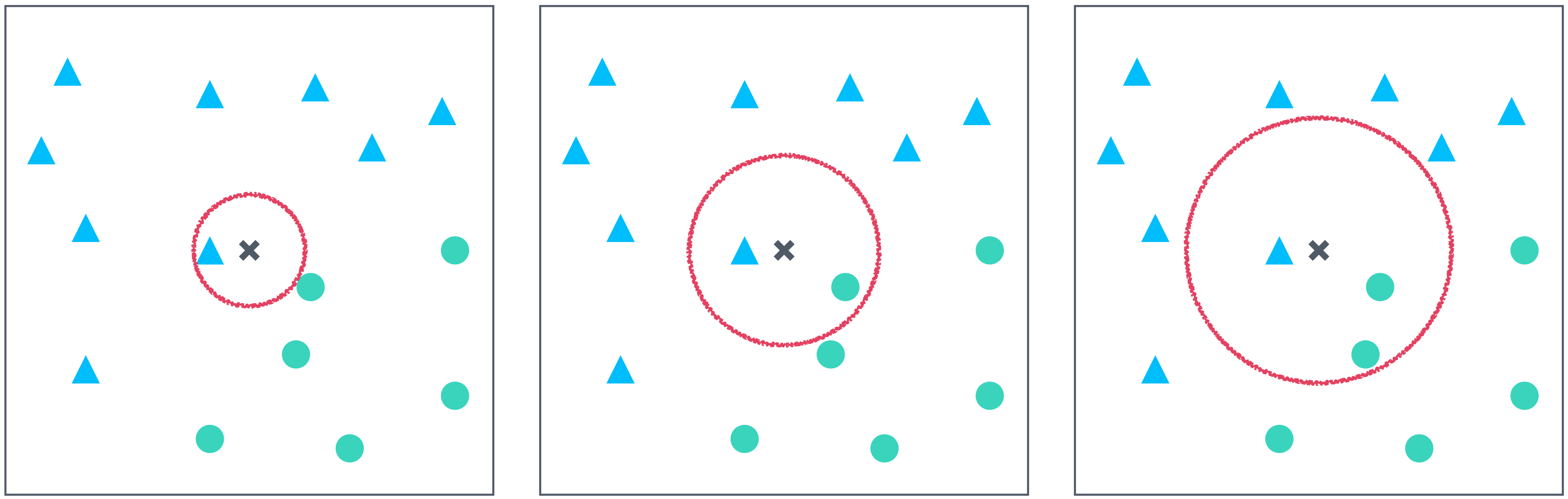
KNN



KNN



KNN



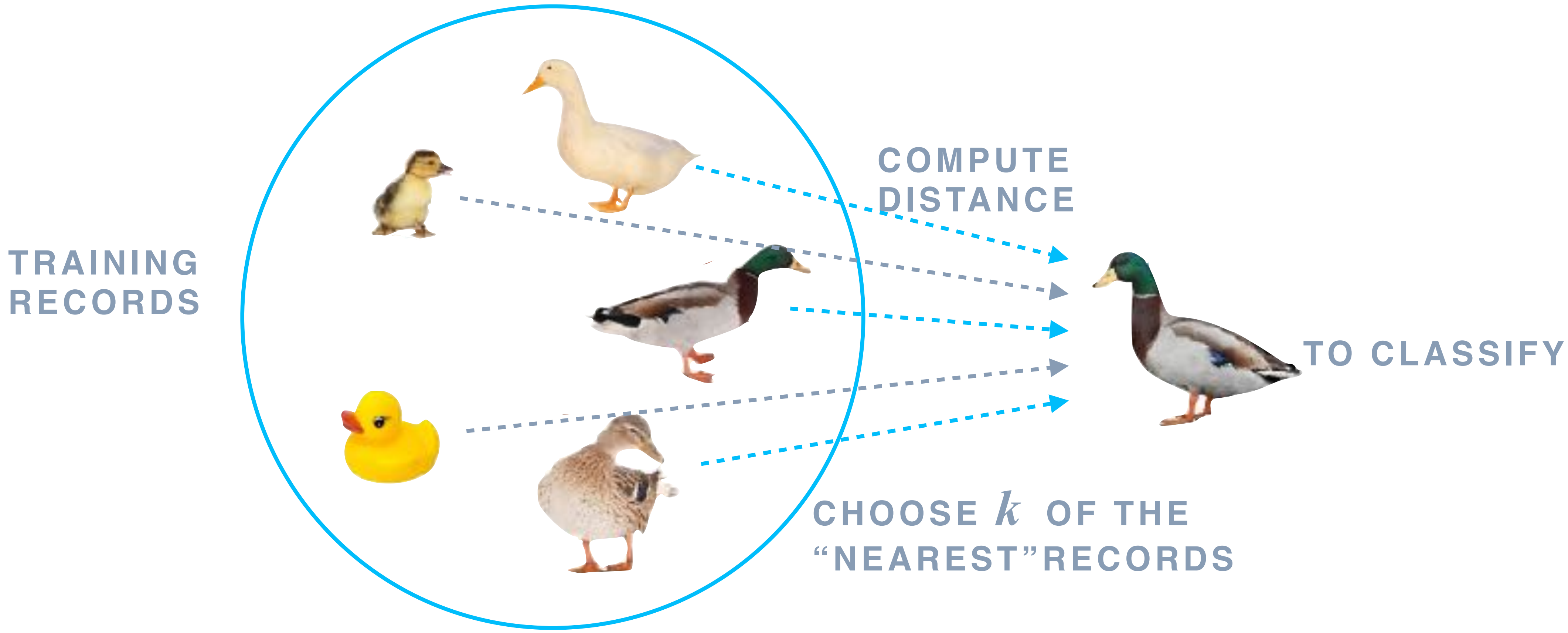
1-nearest neighbour | 2-nearest neighbours | 3-nearest neighbours

k=1	▲
k=2	?
k=3	●

KNN

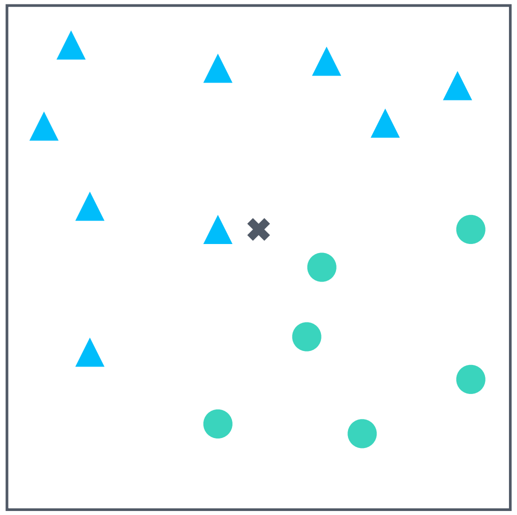
General Idea

"If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck"



KNN

Requirements



What do we need to classify this new point?

KNN

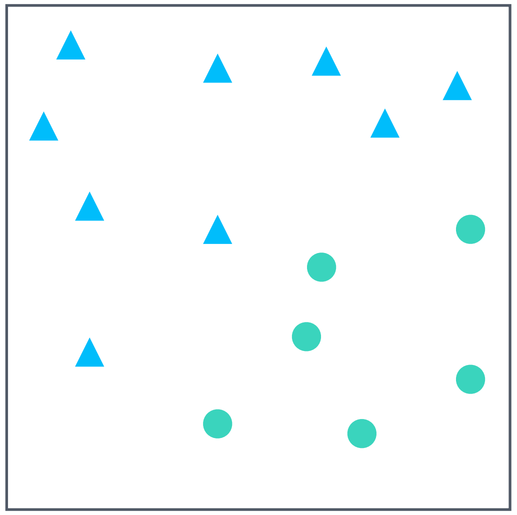
Requirements

1. Training Data

KNN

Requirements

1. Training Data



KNN

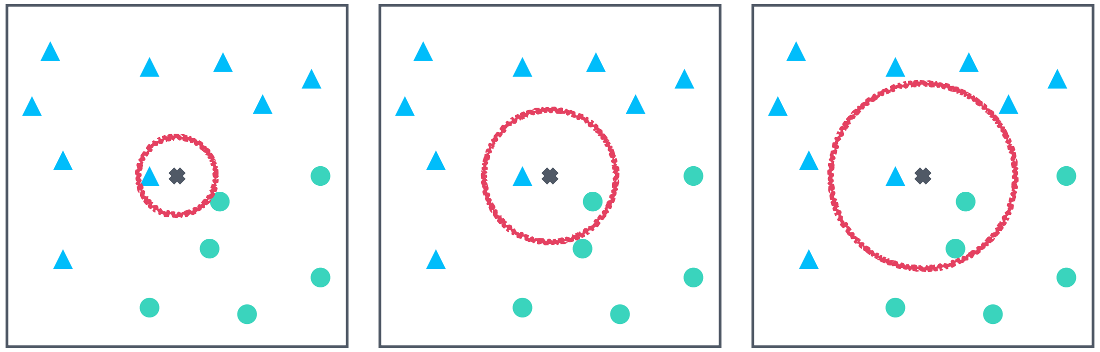
Requirements

1. Training Data
2. Number of nearest-neighbours "K" to consider

KNN

Requirements

1. Training Data
2. Number of nearest-neighbours "K" to consider



KNN

Requirements

1. Training Data
2. Number of nearest-neighbours “K” to consider
3. Distance Metric (wont discuss much)

KNN

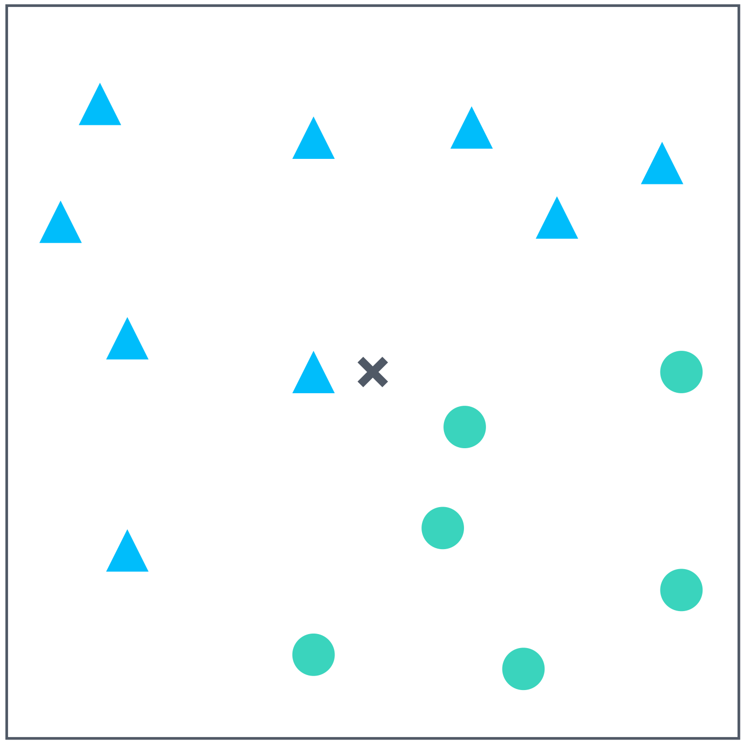
Algorithm Outline

In order to classify an unlabelled point:

KNN

Algorithm Outline

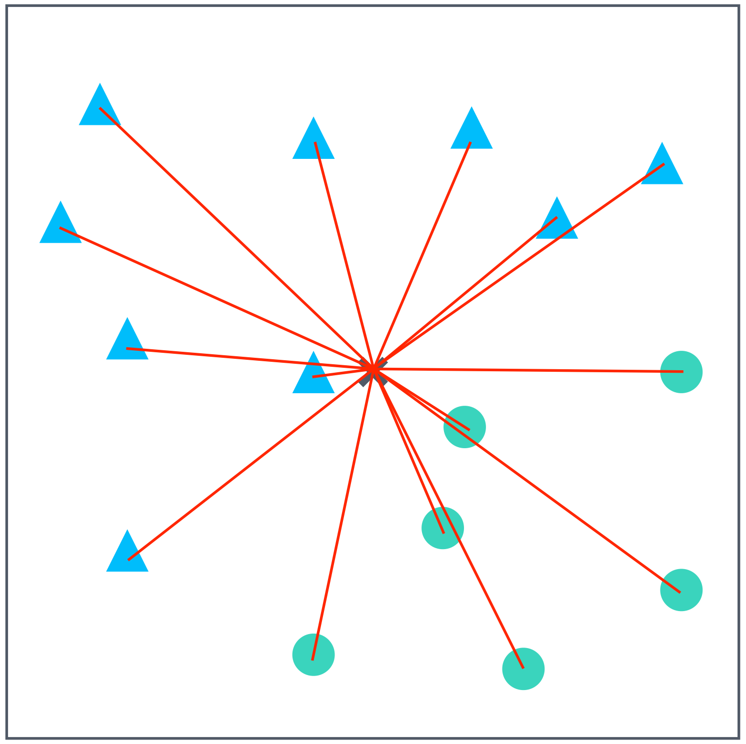
In order to classify an unlabelled point:



KNN

Algorithm Outline

1. Compute Distance to all training examples



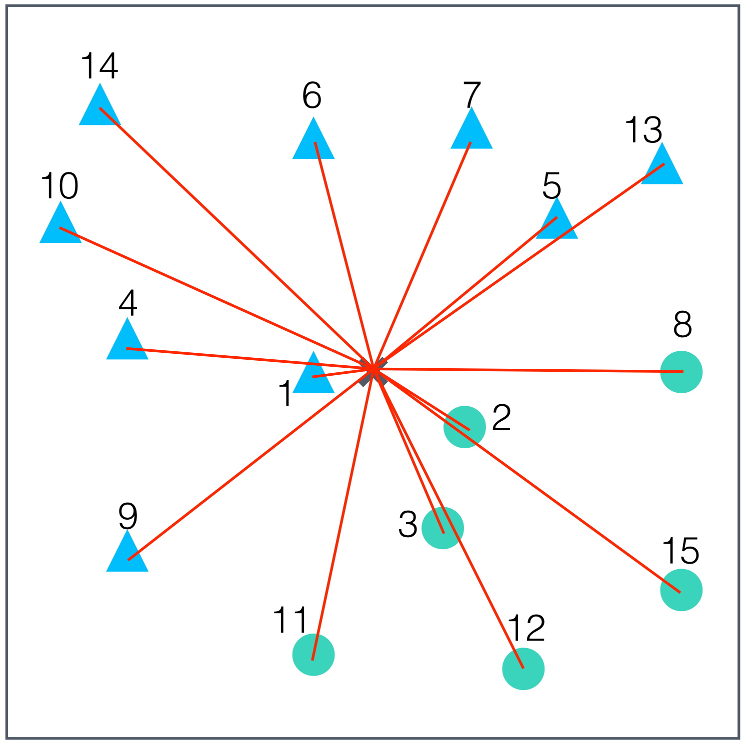
$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

Here we have used a so-called Euclidean Metric

KNN

Algorithm Outline

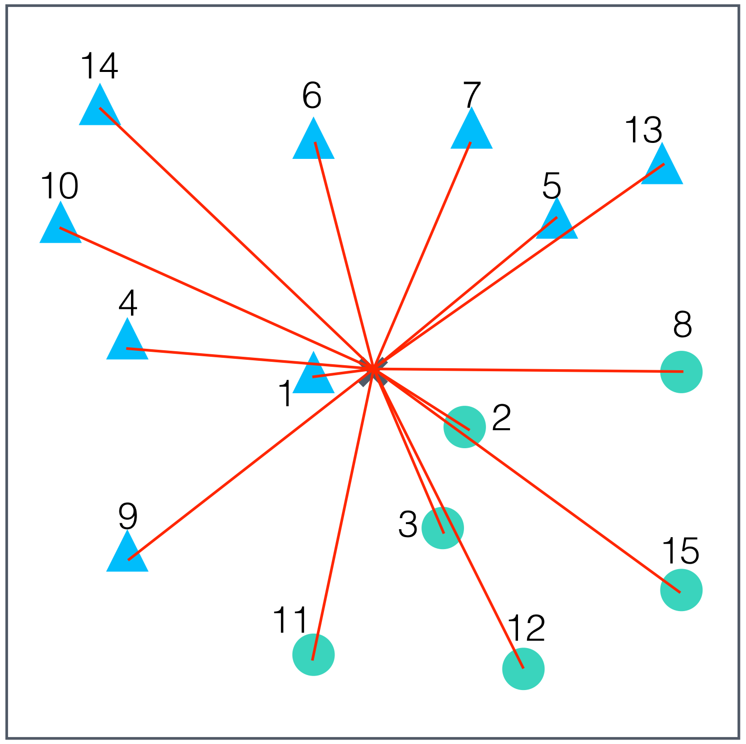
2. Rank training instances in order of distance



KNN

<u>Algorithm Outline</u>

3. Extract “K” closest points



K =1 : ▲

K =2 : ▲ ●

K =3 : ▲ ● ●

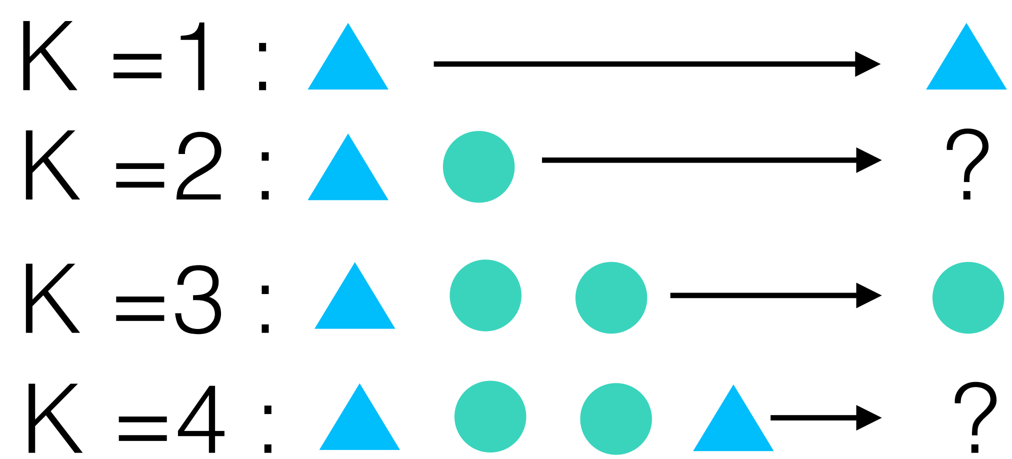
K =4 : ▲ ● ● ▲

etc

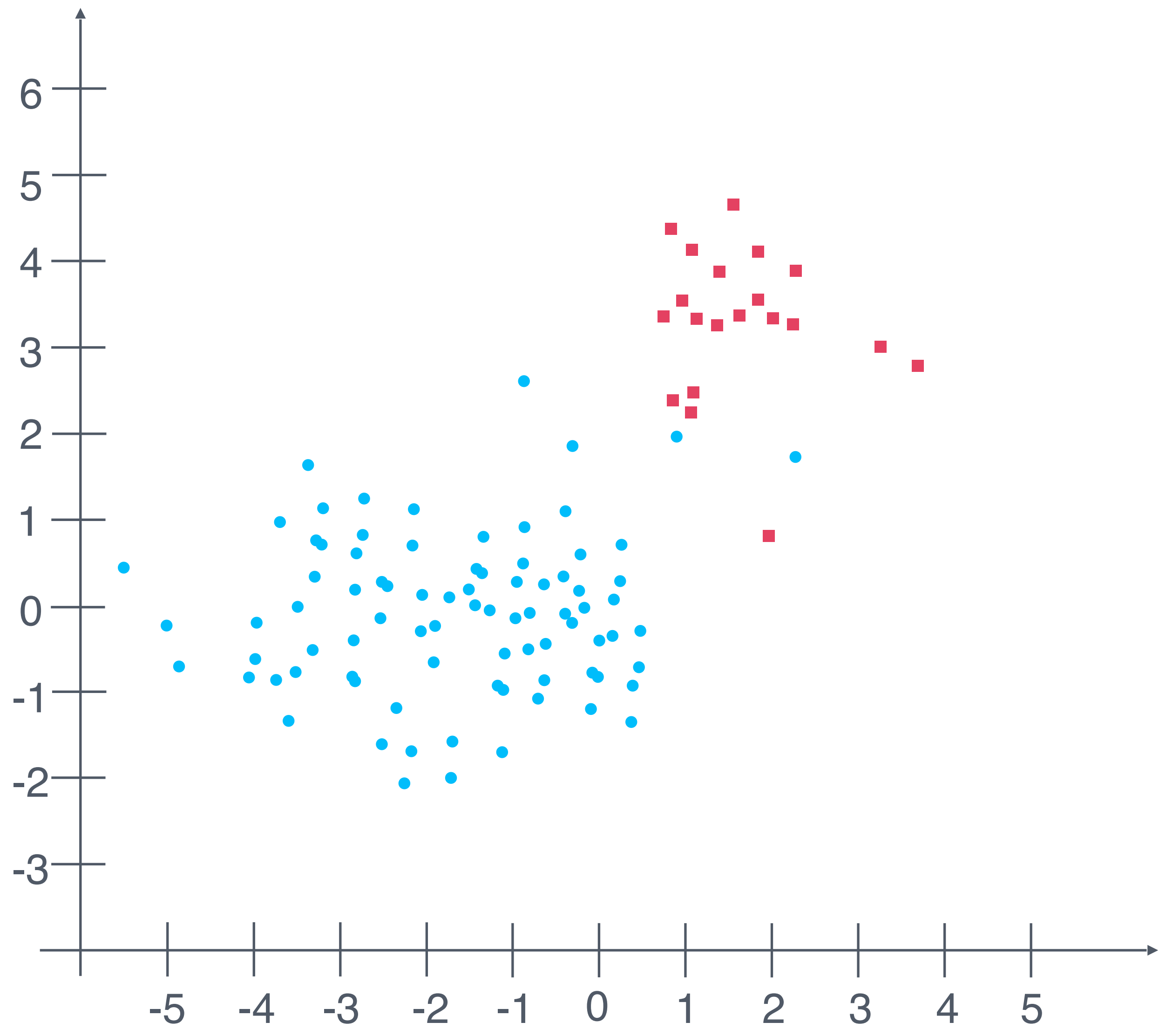
KNN

Algorithm Outline

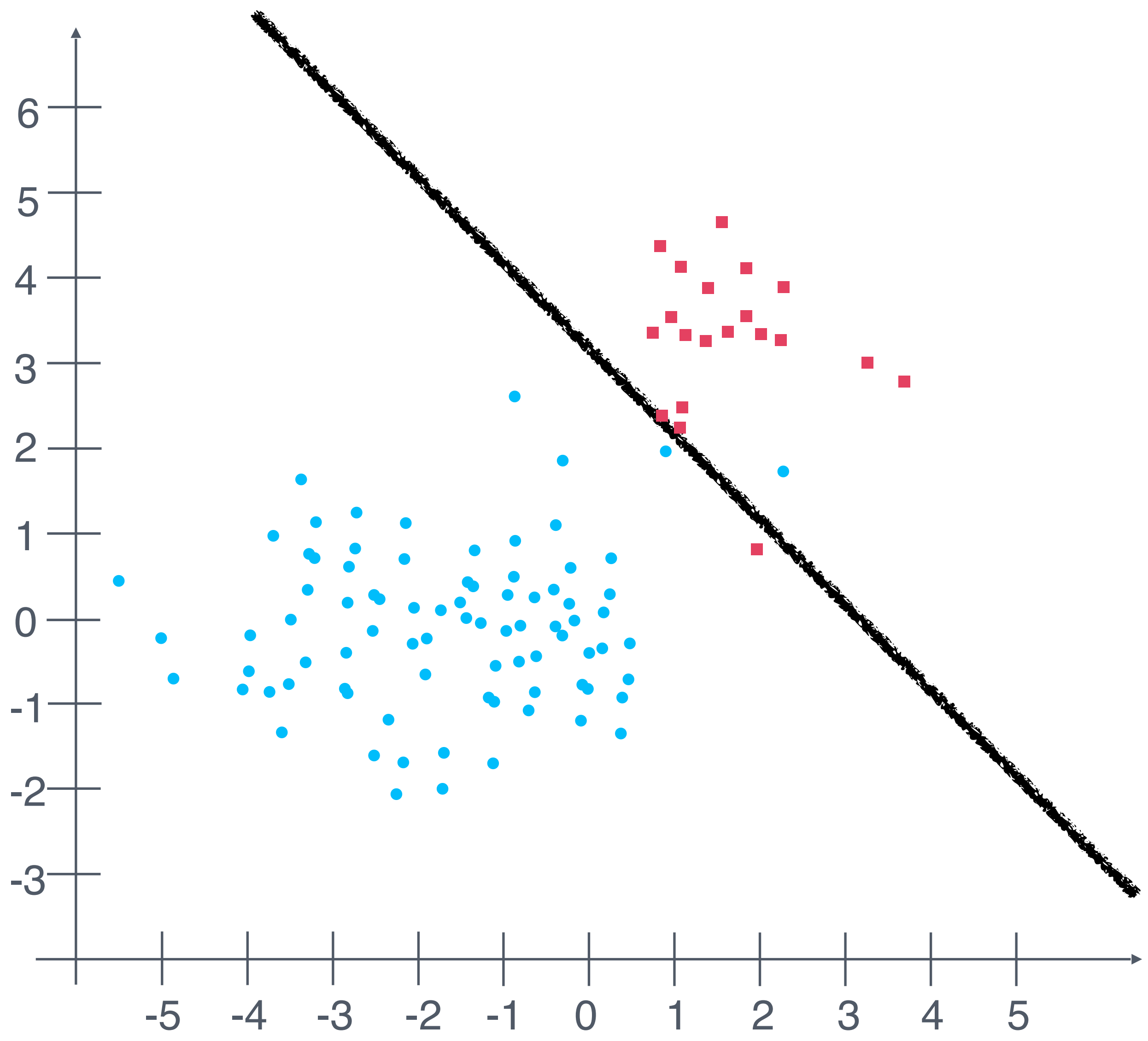
4. Take a majority vote



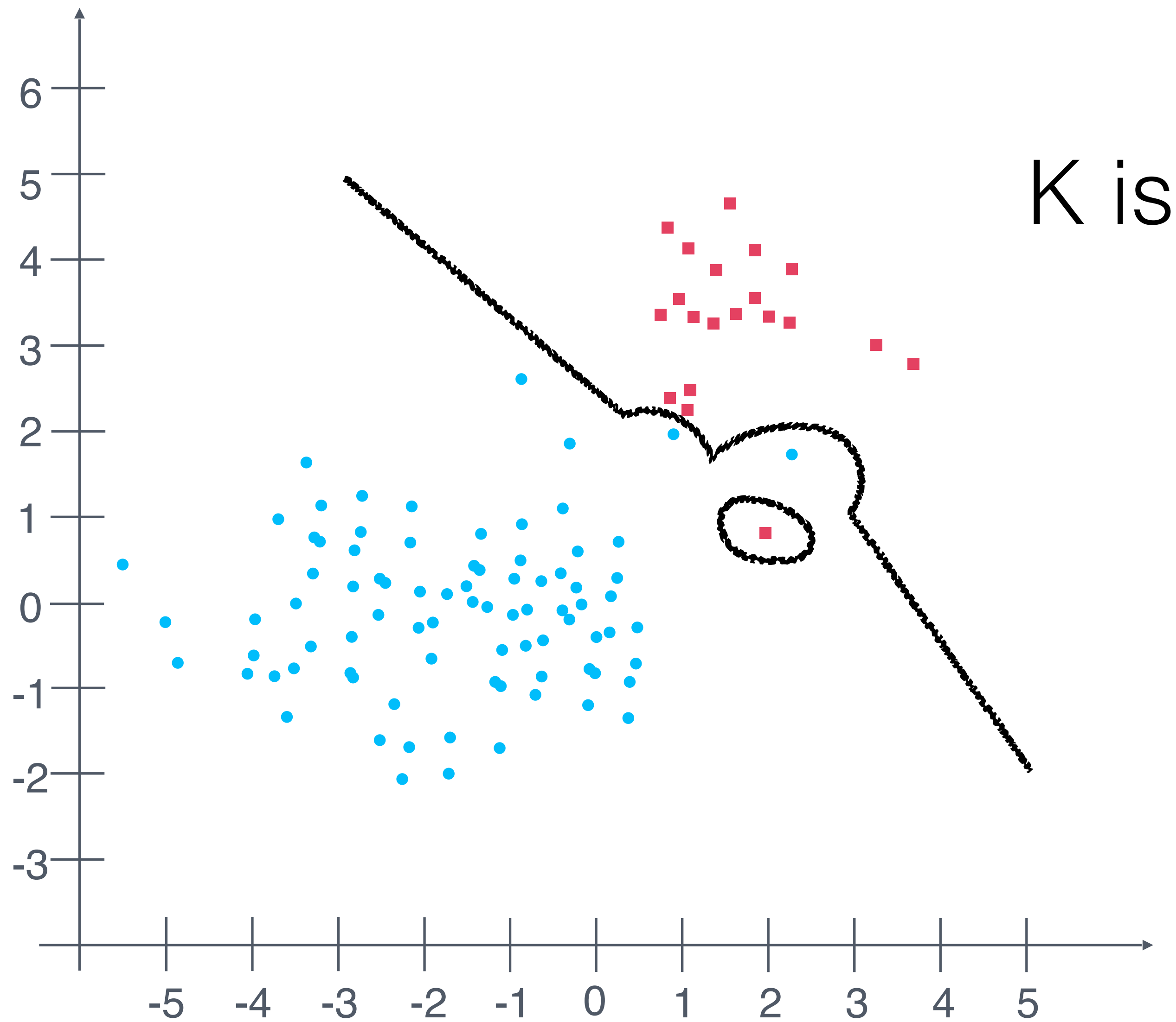
Setting K



Setting K



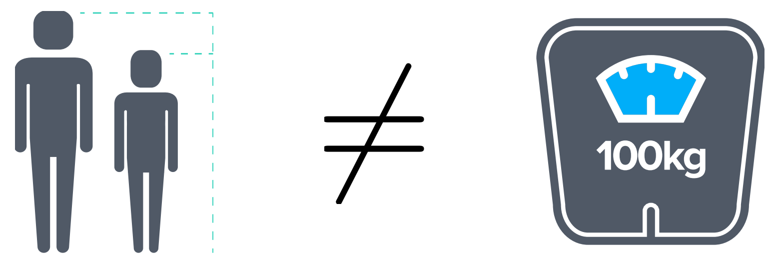
Setting K



K is too low, overfit

Feature Scaling

- Need to <u>scale</u> features (various ways of doing this)



- Potential need to consider correlated features

Cross Validation

Three way data split

KNN

Pros and Cons

PROS

- Quick for prototyping a few points as model doesn't need training
- Simple to understand and explain (use geometric arguments, no equations)
- Sometimes competitive with state of the art classifiers

CONS

- Performance highly dependent on K and distance metrics. Cross-validation time can offset performance benefits
- Accuracy degraded by noisy/irrelevant attributes
- Inefficient if needed to classify large data sets

Exercise

1. Open a browser (preferably **Chrome**) and navigate to https://sherlockml.com/.

2. Access the project on SherlockML from Session 1 & 2.

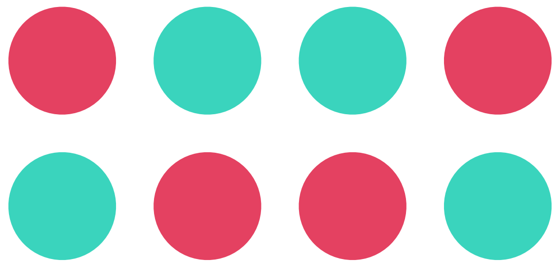
3. Once it's finished spinning up, open a terminal [“New” -> “Terminal”] and type the command “git clone https://github.com/ilyafeige/RandomForests_UCL.git”. This will create a new directory called “RandomForests” in your workspace. If you cannot see it, click the refresh wheel-like button.

If you don't have a server: spin up a Server by clicking on the link (“You have no server instances in this project. Click here to add your first server”) at the bottom of the page.

4. Navigate to the “RandomForests” directory, and click to launch the Jupyter Notebook “taiwanese_credit_cards.ipynb”.

To get around SherlockML, visit: http://docs.sherlockml.com/.

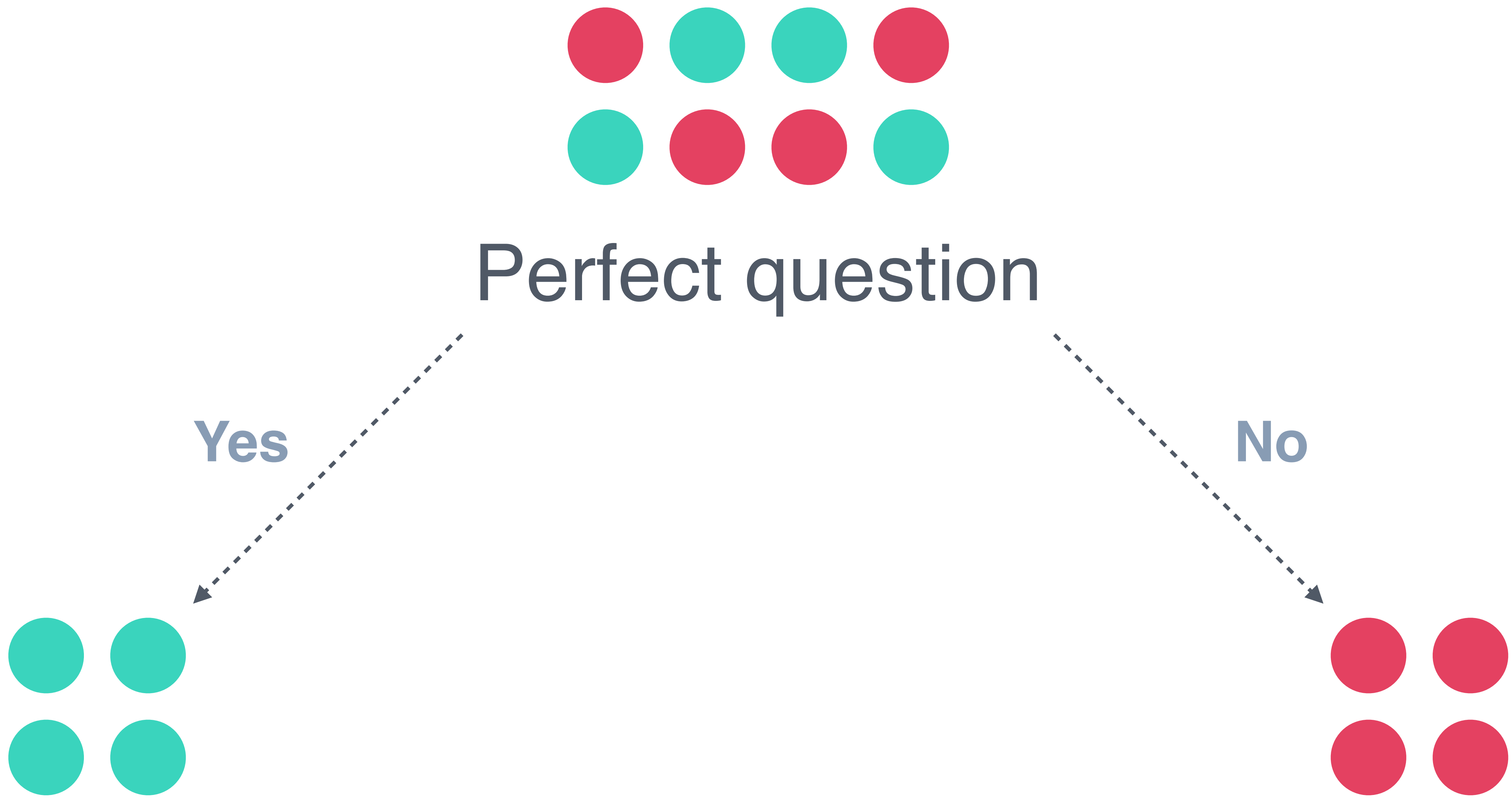
Decision Trees



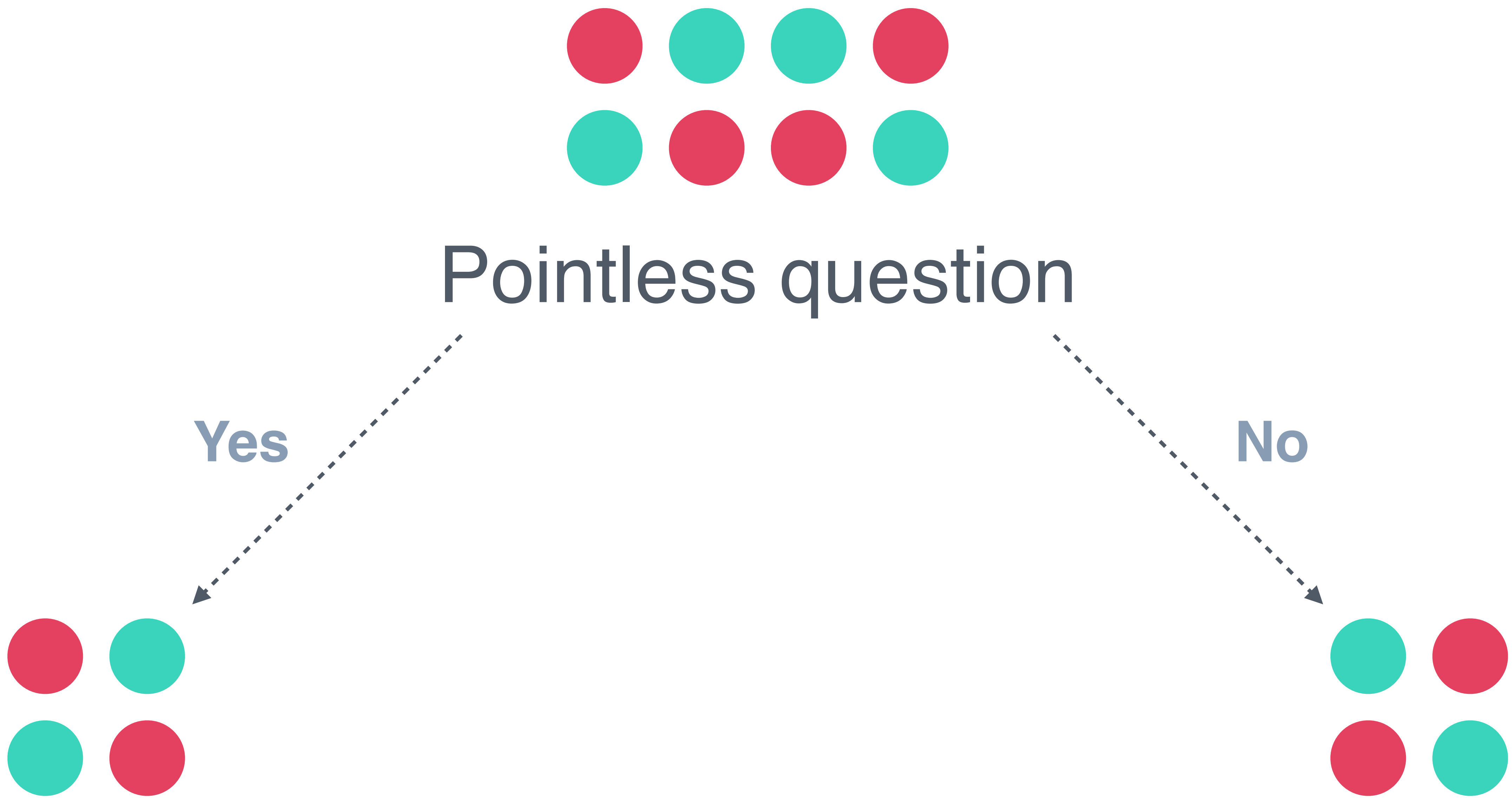
Here we have some labelled training data

Each instance has some features associated with it

Decision Trees

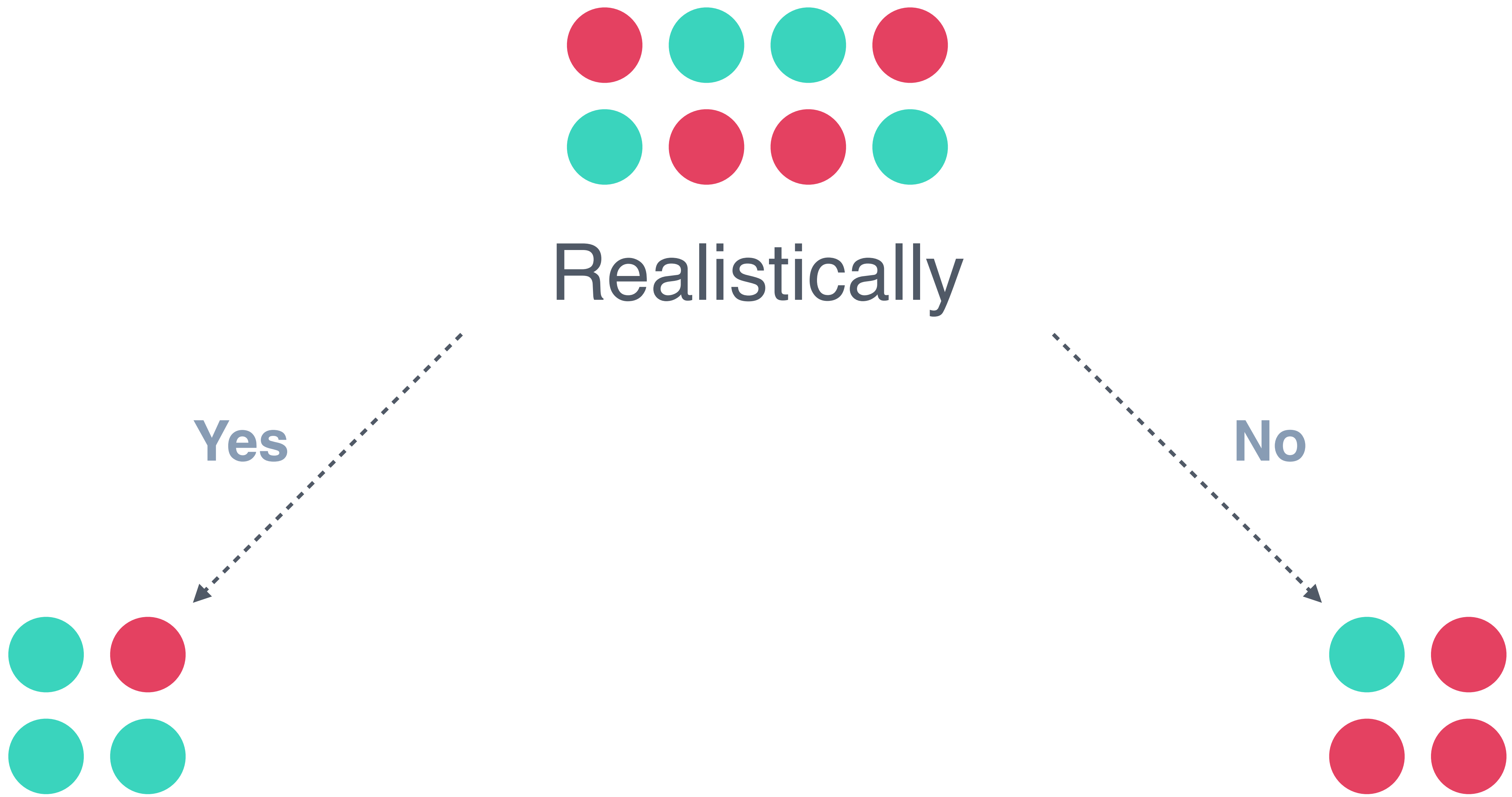


Decision Trees

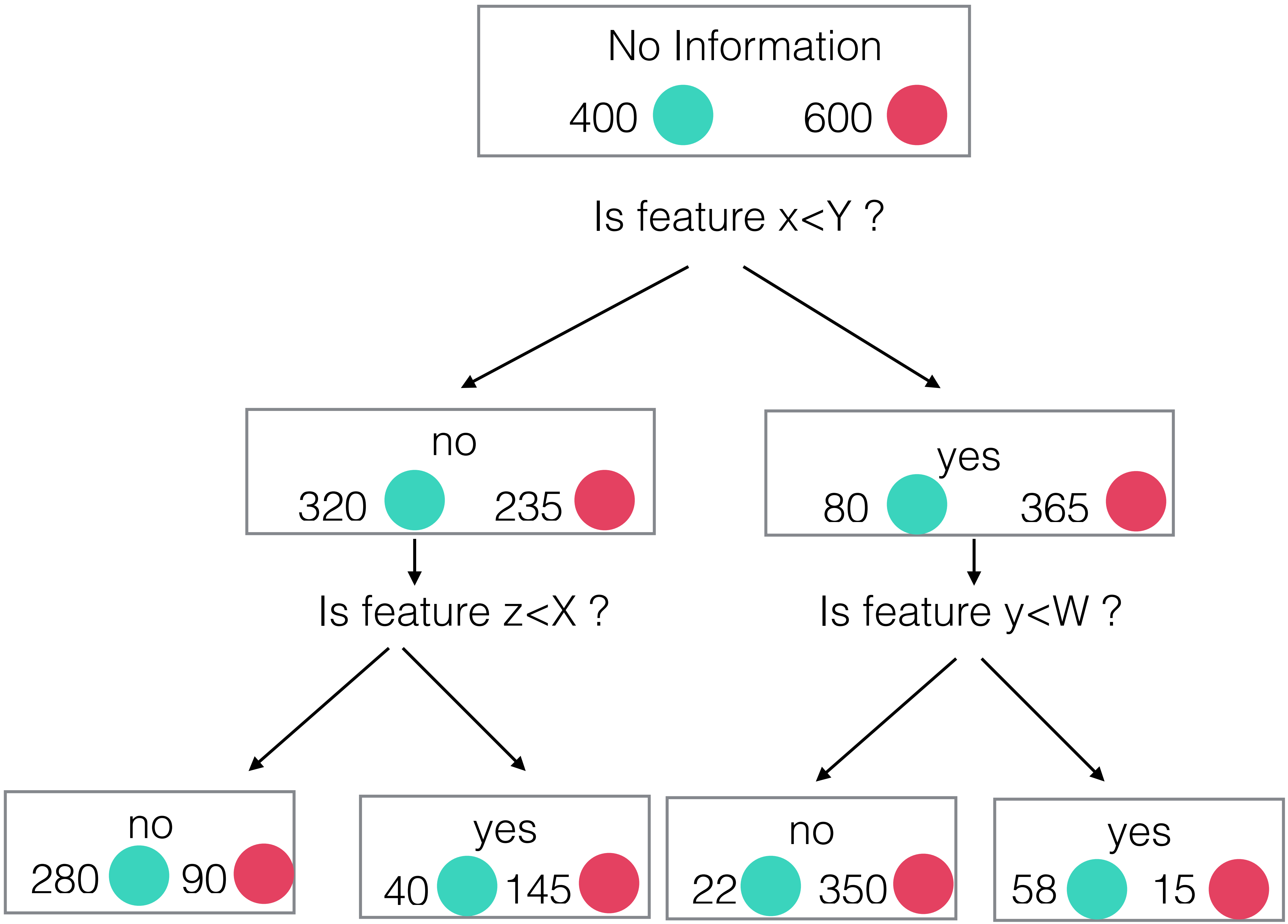


Decision Trees

Decision Trees



Decision Trees

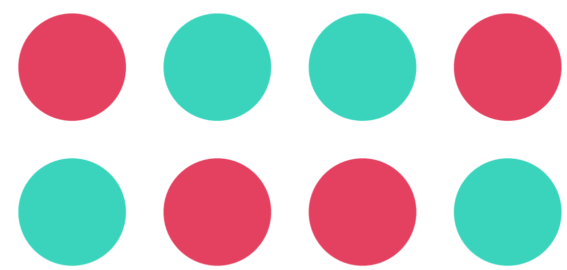


Entropy

Claim: $$H = -\sum_i p_i \log_2 p_i$$ is the formula which tells you how good a split is

Entropy

Claim: $$H = -\sum_i p_i \log_2 p_i$$ is the formula which tells you how good a split is

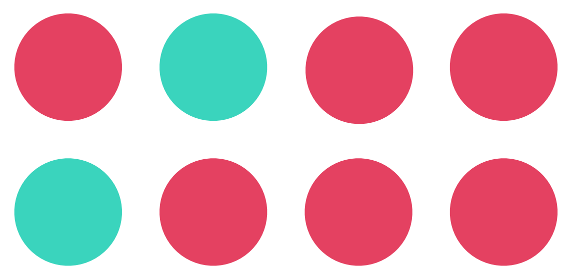


$$p_1 = \frac{1}{2} \qquad p_2 = \frac{1}{2}$$

$$H = -2 \cdot \frac{1}{2} \cdot \log_2 \left(\frac{1}{2} \right) = 1$$

Entropy

Claim: $H = -\sum_i p_i \log_2 p_i$ is the formula which tells you how good a split is

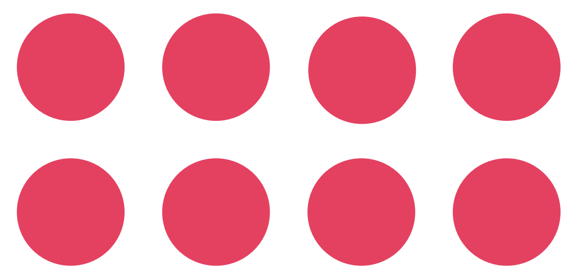


$$p_1 = \frac{3}{4} \quad p_2 = \frac{1}{4}$$

$$H = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) \simeq 0.811$$

Entropy

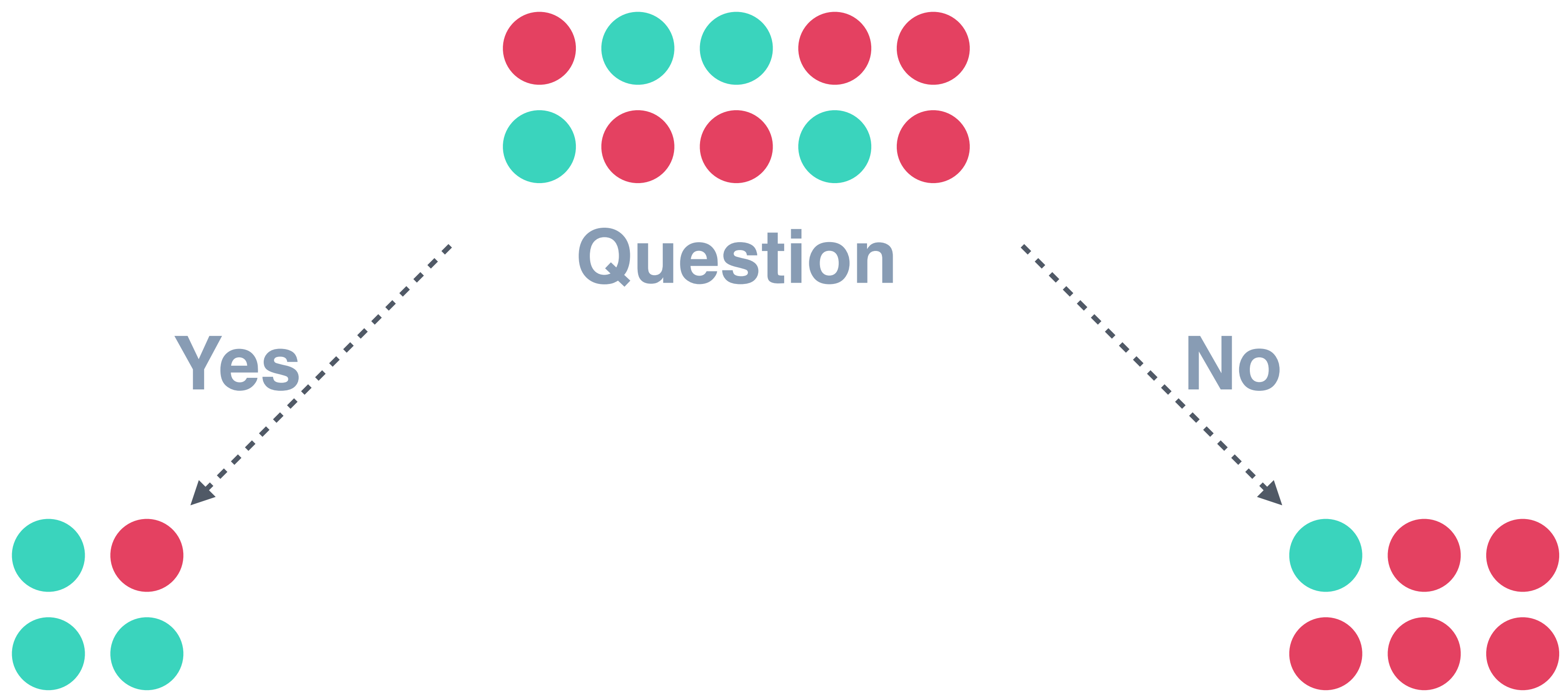
Claim: $$H = -\sum_i p_i \log_2 p_i$$ is the formula which tells you how good a split is



$$p_1 = 1 \quad p_2 = 0$$

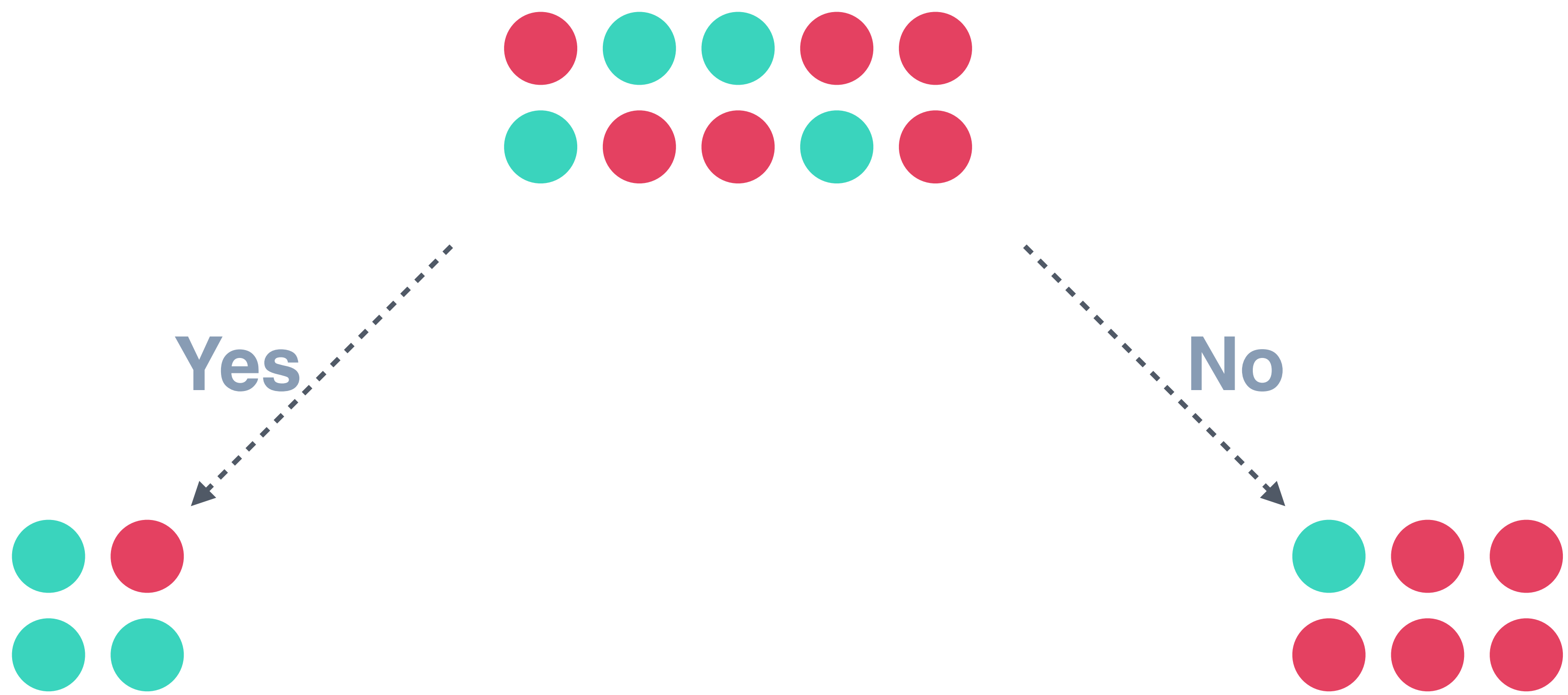
$$H = -\log_2(1) = 0$$

Information Gain



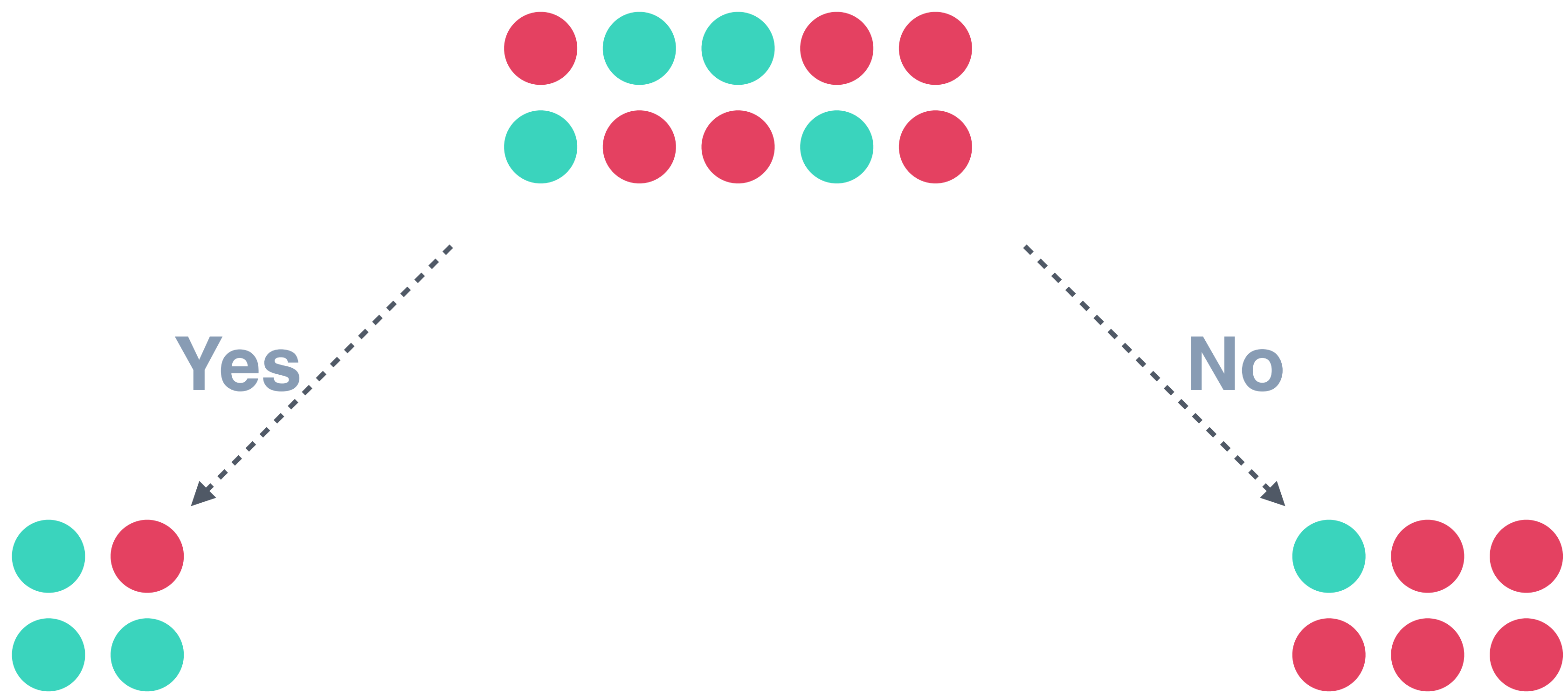
Information Gain

$$-\frac{2}{5}\log_2(\frac{2}{5}) - \frac{3}{5}\log_2(\frac{3}{5}) \simeq 0.97$$



Information Gain

$$-\frac{2}{5}\log_2(\frac{2}{5}) - \frac{3}{5}\log_2(\frac{3}{5}) \simeq 0.97$$



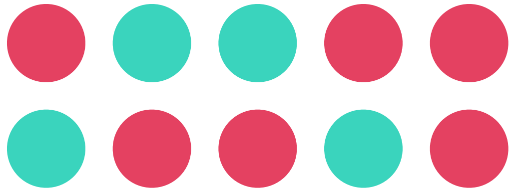
$$-\frac{3}{4}\log_2(\frac{3}{4}) - \frac{1}{4}\log_2(\frac{1}{4}) \simeq 0.81$$

$$-\frac{5}{6}\log_2(\frac{5}{6}) - \frac{1}{6}\log_2(\frac{1}{6}) \simeq 0.65$$

$$H \simeq \frac{2}{5} \cdot 0.81 + \frac{3}{5} \cdot 0.65 = 0.714$$

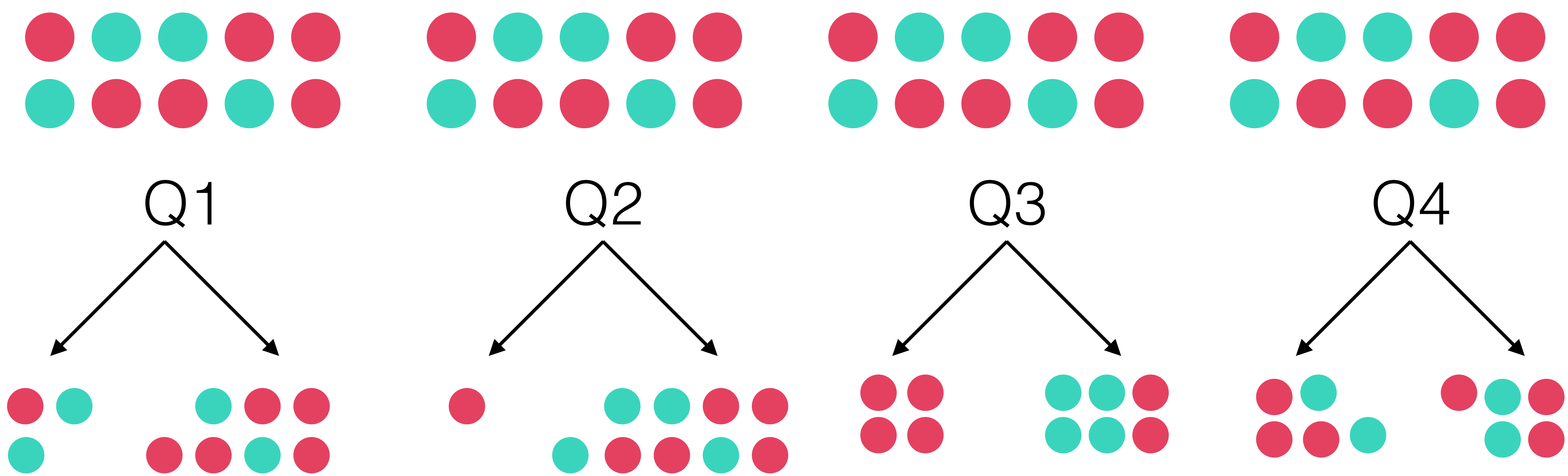
ID3 Training Algorithm

1. Start with all training data at root node



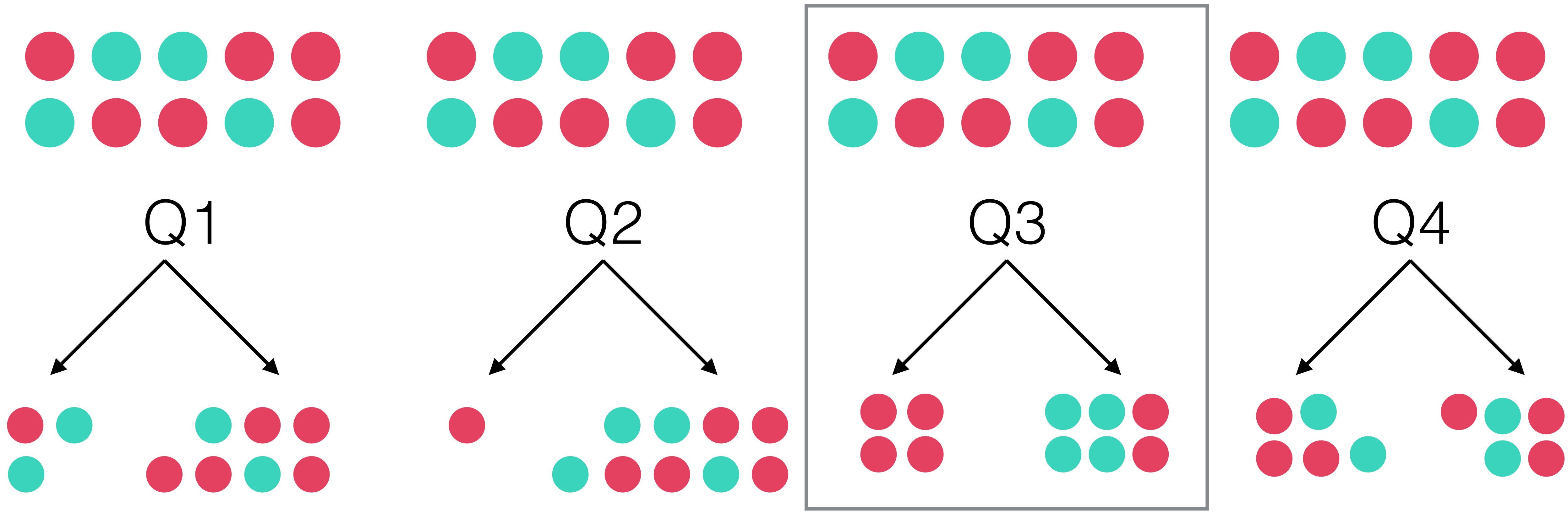
ID3 Training Algorithm

2. Compute impurity metric for all possible splits, split data on “best” generating two nodes



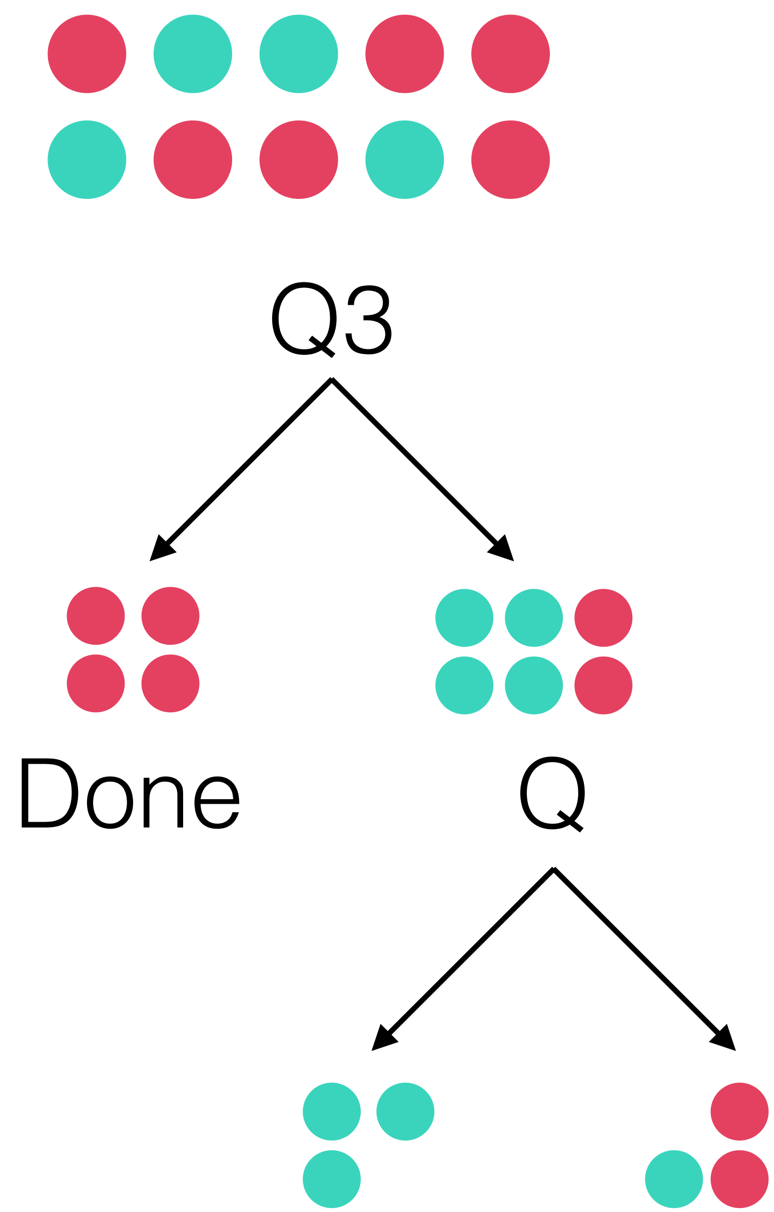
ID3 Training Algorithm

2. Compute impurity metric for all possible splits, split data on “best” generating two nodes



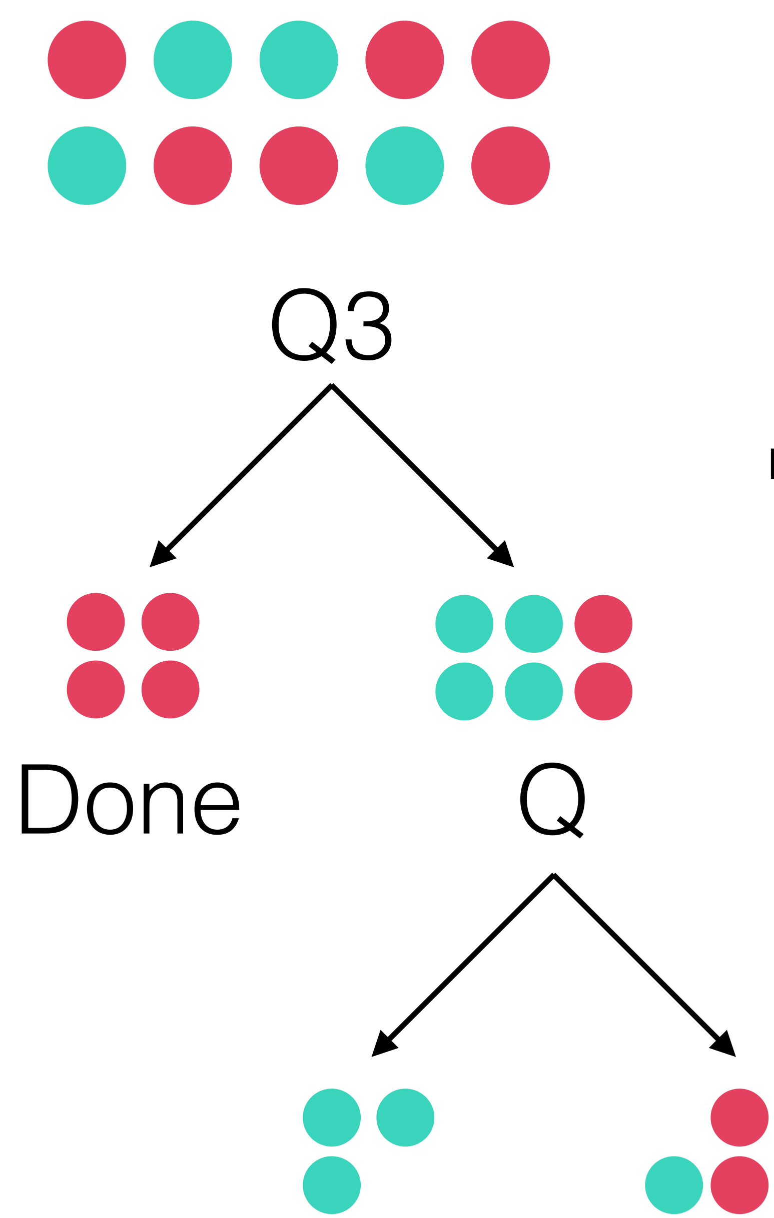
ID3 Training Algorithm

3. Recursively find best split on subset of data generated at each node



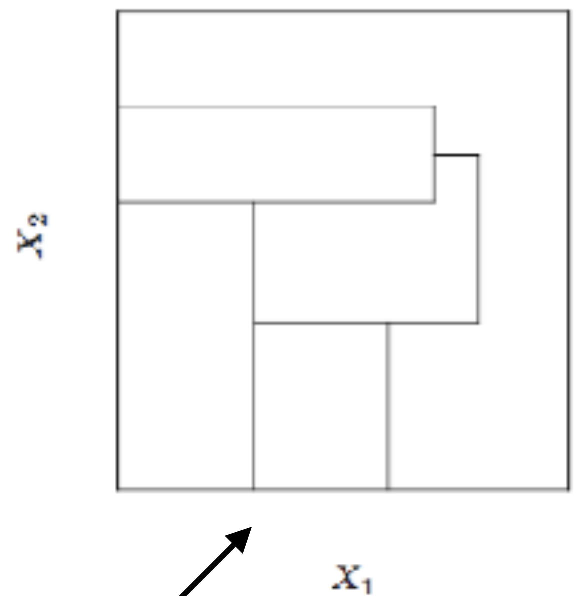
ID3 Training Algorithm

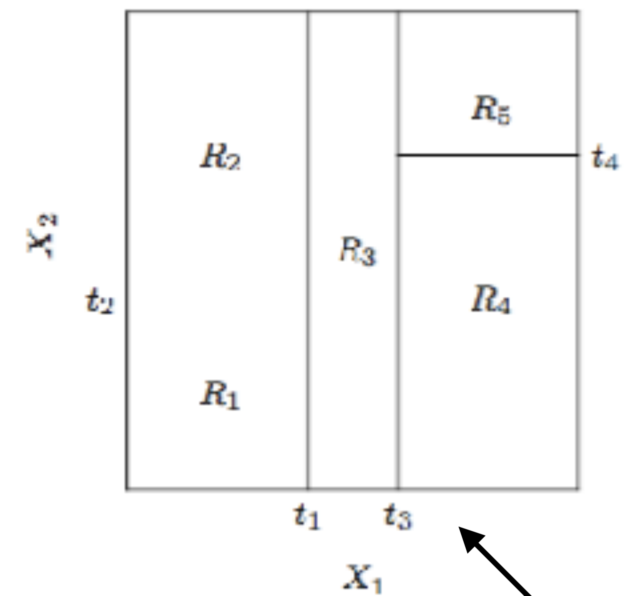
4. Exit when suitable condition is reached



Most stopping conditions are related to the size of the final nodes

Feature Space



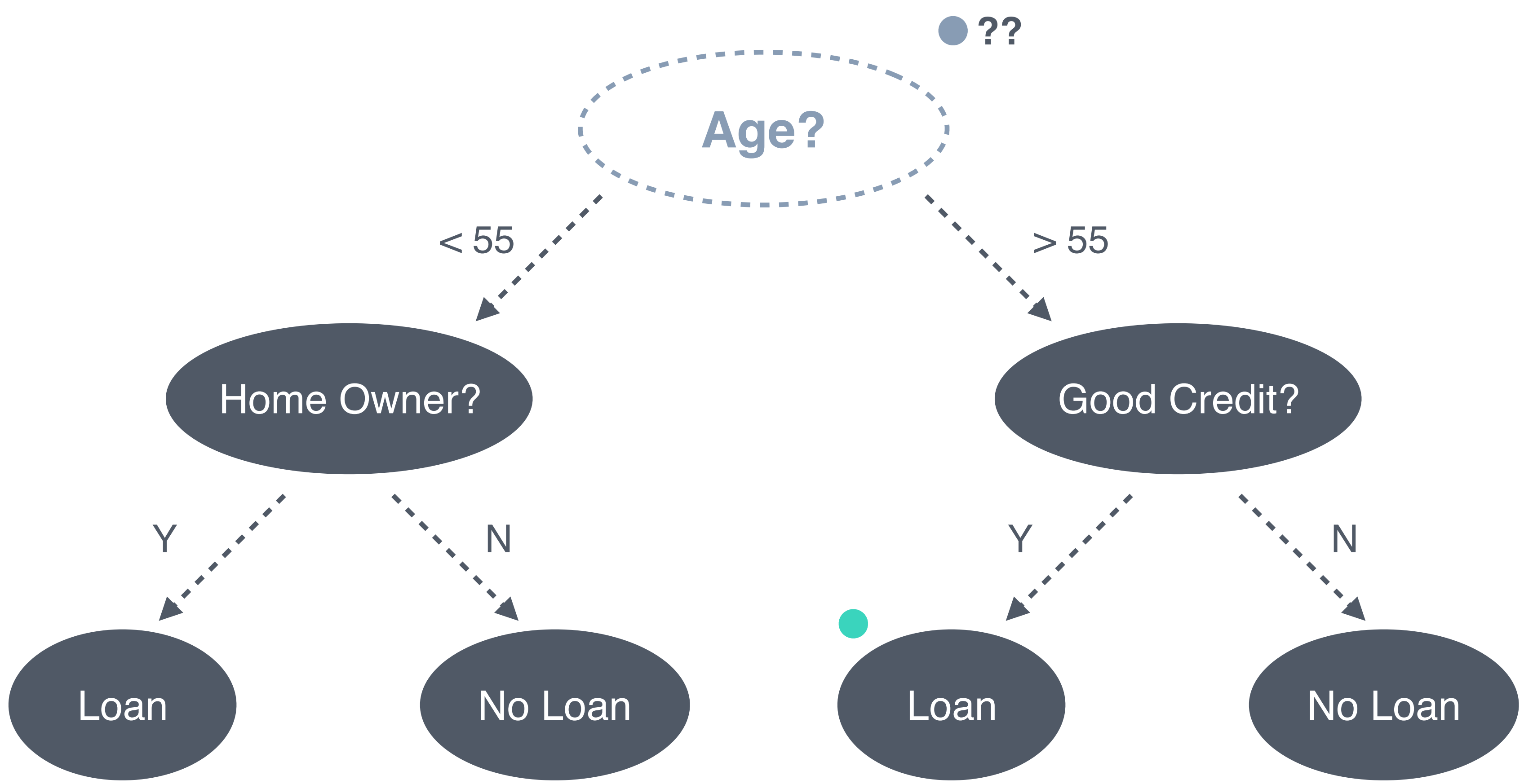


Not the result of
decision tree learning

Possible feature space
partitioning

Decision tree learning only allows axis
perpendicular splits

Visualising Decision Trees



Easy to visualise and explain

Pros and Cons

PROS

- Interpretability
- Require little data prep
- Very fast to classify new data, reasonably fast to train
- Good for data that's not separable by any single boundary

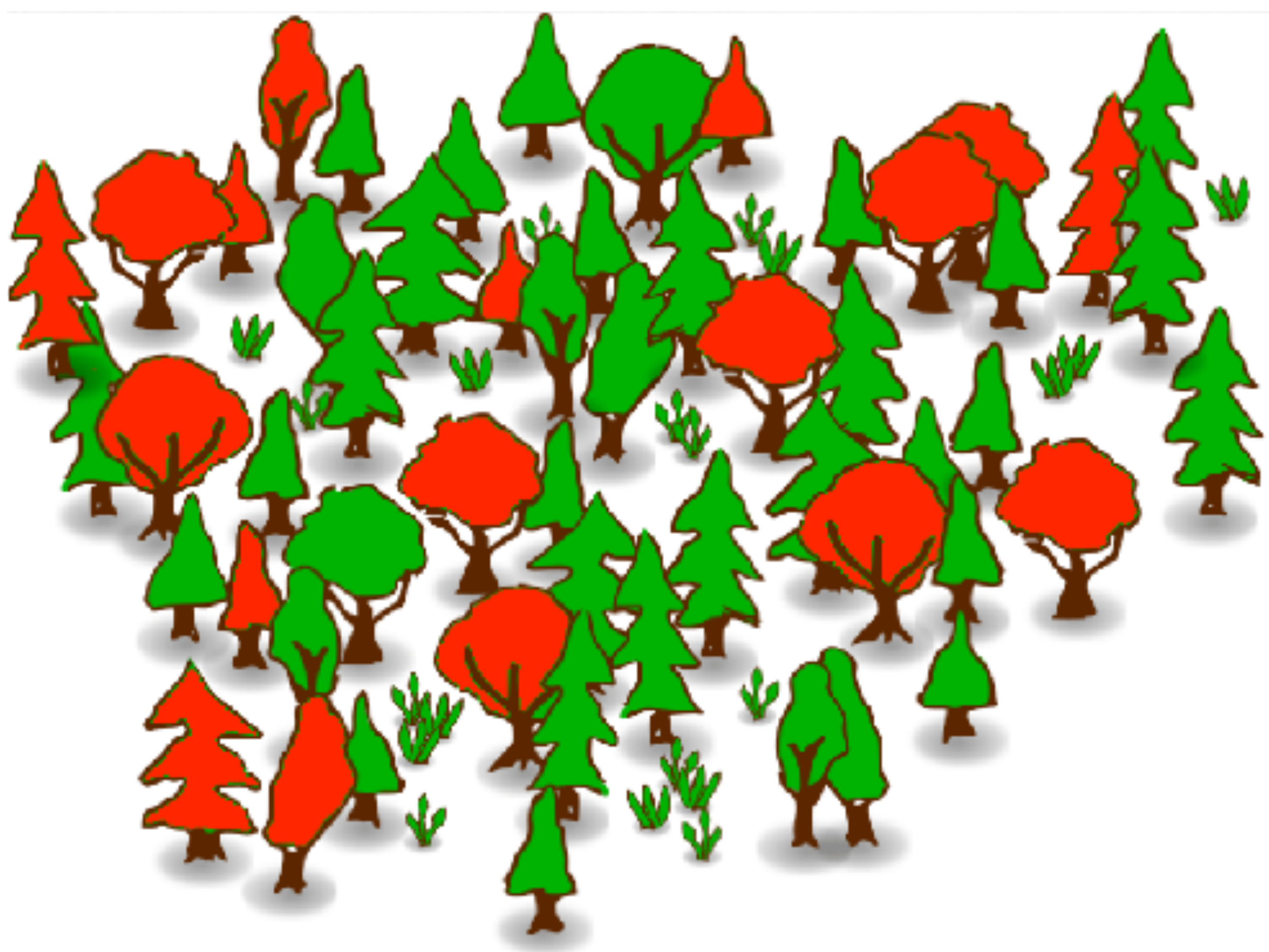
CONS

- Only allows axis-perpendicular splits
- Structure (even if not output) highly susceptible to small changes in data
- Greedy training process means globally optimal solution unlikely to be found

Exercise

Random Forests

(and Ensemble Models in general)



Random Forests

CONS

- Only allows axis-perpendicular splits
- **Structure (even if not output) highly susceptible to small changes in data**
- **Greedy training process means globally optimal solution unlikely to be found**

Random Forests

1. Use subset of the data
2. Use subset of the features
3. Train multiple (as many as practical) trees
4. Use majority vote

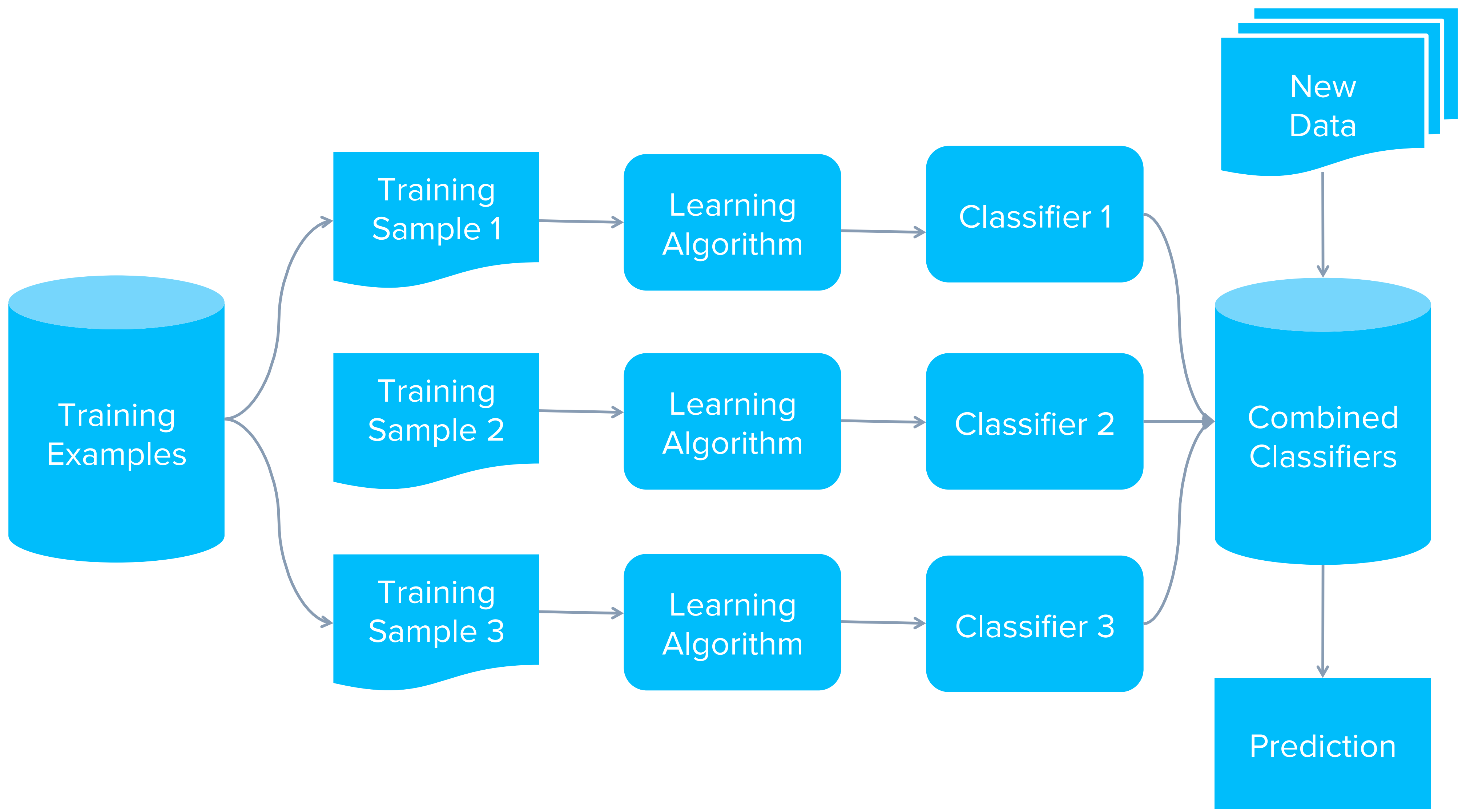
Random Forests

1. Use subset of the data
2. Use subset of the features
3. Train multiple (as many as practical) trees
4. Use majority vote

General Philosophy

All models are overfit in some way, averaging the predictions of multiple models tends to average out these biases and improve performance

Random Forests



Can bootstrap sample or sample without replacement

Random Forests

Rule of thumb:

Do your initial analysis with a single tree. If it's working reasonably well, expect using a random forest to boost your performance. If a single tree is performing poorly, don't expect a random forest to magically work.

Also, remember you might gain in prediction power, but you will lose human interpretability

Other Forest Based Techniques

- Balanced Random Forest
- Ada Boost
- XGBoost