NUI Galway
OÉ Gaillimh

Similarity-based learning

Part 1: Introduction

Learning Objectives

After completing this successfully, you will be able to …

- Explain what instance-based learning is
- Distinguish between *lazy* and *eager* learning
- Describe operation of k-Nearest Neighbours for classification and regression
- Discuss implications of the *curse of dimensionality*
- Discuss implications of selecting different distance metrics
- Identify suitable applications for kNN and explain how it could be applied

Overview of topic

This week:

1. Learning objectives and overview
2. Problem description
3. Distance-based similarity
4. The nearest neighbour algorithm
5. The k nearest neighbours algorithm

Next week:

6. Alternate similarity measures
7. Predicting continuous targets
8. Feature selection
9. Similarity-based learning considerations
10. Review of topic

Similarity-based learning

Part 2:

Problem description

Example dataset for similarity-based learning

- College athletes dataset
  - Two attributes:

    Speed - continuous variable

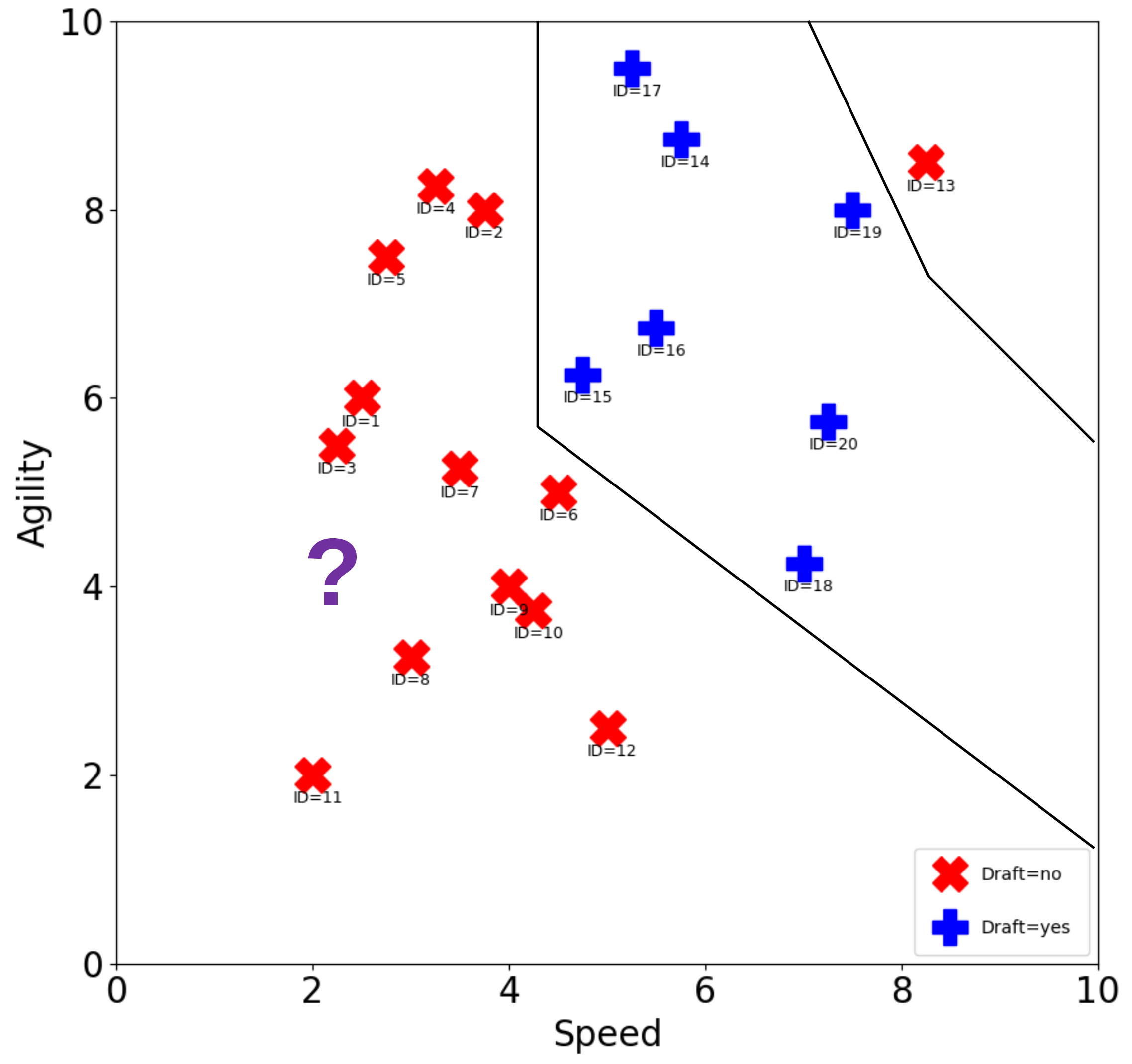
    Agility - continuous variable
  - Data on whether or not each college athlete was drafted to a professional team. Target: **Draft** - yes/no
  - 20 examples in dataset
  - See **college_athletes.xlsx** or **college_athletes.csv**

- Objective:
  - Apply similarity-based learning methods to predict whether an athlete who did not feature in the dataset should be drafted

College Athletes

ID	Speed	Agility	Draft
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

Feature space plot for the college athletes dataset



College Athletes

ID	Speed	Agility	Draft
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

Similarity-based learning

Part 3:
Distance-based similarity

Measuring similarity using distance

- Consider the college athletes dataset from earlier
- How should we measure the similarity between instances in this case? E.g. how similar are datapoints 5 and 12?
- Distance is one option: plot the points in 2D space and draw a straight line between them
- This approach can scale to arbitrarily high dimensions as we will see. We can think of each feature of interest as a dimension in hyperspace

College Athletes

ID	Speed	Agility	Draft
1	2.5	6	no
2	3.75	8	no
3	2.25	5.5	no
4	3.25	8.25	no
5	2.75	7.5	no
6	4.5	5	no
7	3.5	5.25	no
8	3	3.25	no
9	4	4	no
10	4.25	3.75	no
11	2	2	no
12	5	2.5	no
13	8.25	8.5	no
14	5.75	8.75	yes
15	4.75	6.25	yes
16	5.5	6.75	yes
17	5.25	9.5	yes
18	7	4.25	yes
19	7.5	8	yes
20	7.25	5.75	yes

Measuring similarity using distance

- A **metric** or distance function may be used to define the distance between any pair of elements in a set.
- $metric(\boldsymbol{a}, \boldsymbol{b})$ is a function that returns the distance between two instances $\boldsymbol{a}$ and $\boldsymbol{b}$ in a set
- $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors containing the values of the attributes we are interested in for the data points we wish to measure between

Properties of a metric

The function $metric(\boldsymbol{a}, \boldsymbol{b})$ should satisfy the following four conditions:

1. **Non-negativity**: $metric(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
2. **Identity**: $metric(\boldsymbol{a}, \boldsymbol{b}) = 0 \iff \boldsymbol{a} = \boldsymbol{b}$
3. **Symmetry**: $metric(\boldsymbol{a}, \boldsymbol{b}) = metric(\boldsymbol{b}, \boldsymbol{a})$
4. **Triangular Inequality**: $metric(\boldsymbol{a}, \boldsymbol{b}) \leq metric(\boldsymbol{a}, \boldsymbol{c}) + metric(\boldsymbol{b}, \boldsymbol{c})$

Euclidean distance

- Euclidean distance is one of the best-known distance metrics
- Computes the length of a straight line between two points

$$Euclidean(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{m} (\boldsymbol{a}[i] - \boldsymbol{b}[i])^2}$$

- Here m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- Square root of the sum of squared differences for each feature

Manhattan distance

- Manhattan distance (also known as “taxicab distance”)
- Computes the length of a straight line between two points

$$Manhattan(\boldsymbol{a},\boldsymbol{b}) = \sum_{i=1}^{m} abs(\boldsymbol{a}[i] - \boldsymbol{b}[i])$$

- As before m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- abs() returns the absolute value
- Sum of the absolute differences for each feature

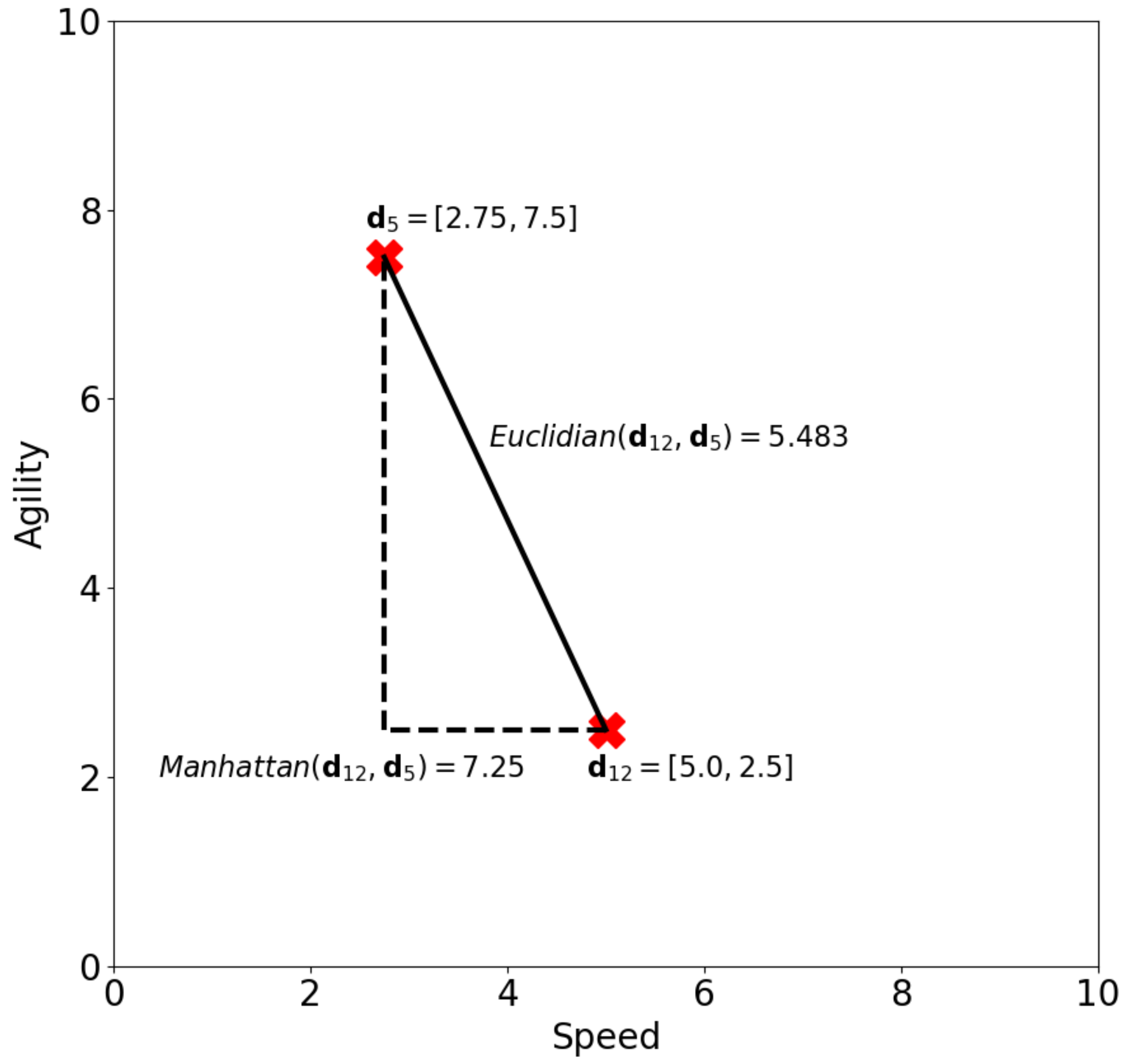
Example: calculating distance

Calculate distance between

$\boldsymbol{d}_{12} = [5.00, 2.50]$ and $\boldsymbol{d}_{\mathbf{5}} = [2.75, 7.50]$

$$Euclidean(\boldsymbol{d}_{12}, \boldsymbol{d}_{\mathbf{5}})$$
$$= \sqrt{(5.00 - 2.75)^2 + (2.50 - 7.50)^2}$$
$$= 5.483$$

$$Manhattan(\boldsymbol{d}_{12}, \boldsymbol{d}_{\mathbf{5}})$$
$$= abs(5.00 - 2.75) + abs(2.50 - 7.50)$$
$$= 7.25$$



Minkowski distance

- The Minkowski distance metric generalises both the Manhattan distance and the Euclidean distance metrics

$$Minkowski(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{m} abs(\boldsymbol{a}[i] - \boldsymbol{b}[i])^p \right)^{\frac{1}{p}}$$

- As before m is the number of features/attributes to be used to calculate the distance (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- $abs()$ returns the absolute value
- Sum of the absolute differences for each feature

Comparison of distance metrics

- Euclidian and Manhattan distance are most commonly used, although it is possible to define infinitely many distance metrics using the Minkowski distance
- Manhattan is cheaper to compute than Euclidean as it is not necessary to compute the squares of differences and a square root, so may be a good choice for very large datasets if computational resources are limited
- It’s worthwhile to try out several different distance metrics to see which is most suitable for the dataset at hand

NUI Galway
OÉ Gaillimh

Similarity-based learning

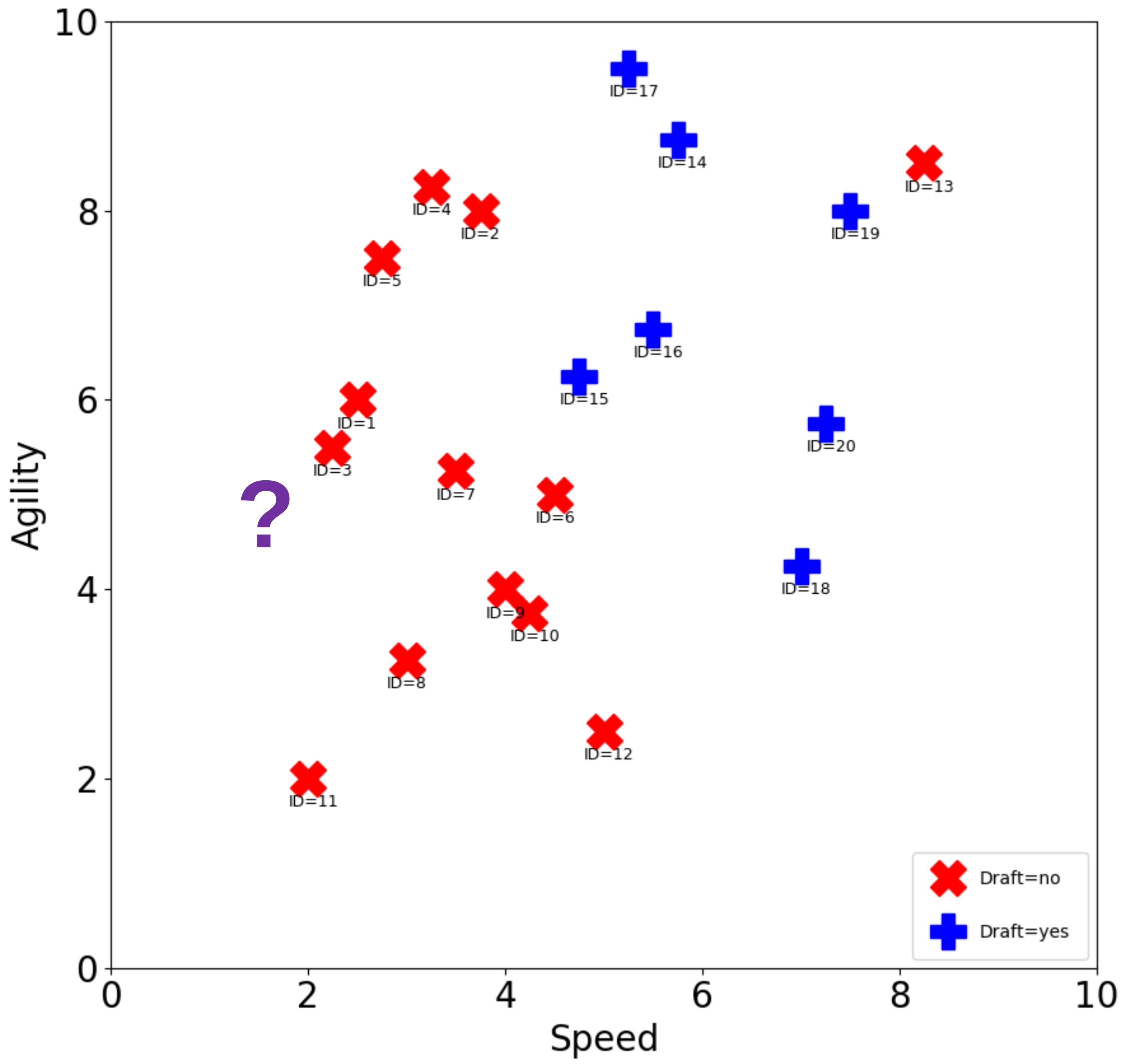
Part 4:
The Nearest Neighbour algorithm

The Nearest Neighbour algorithm

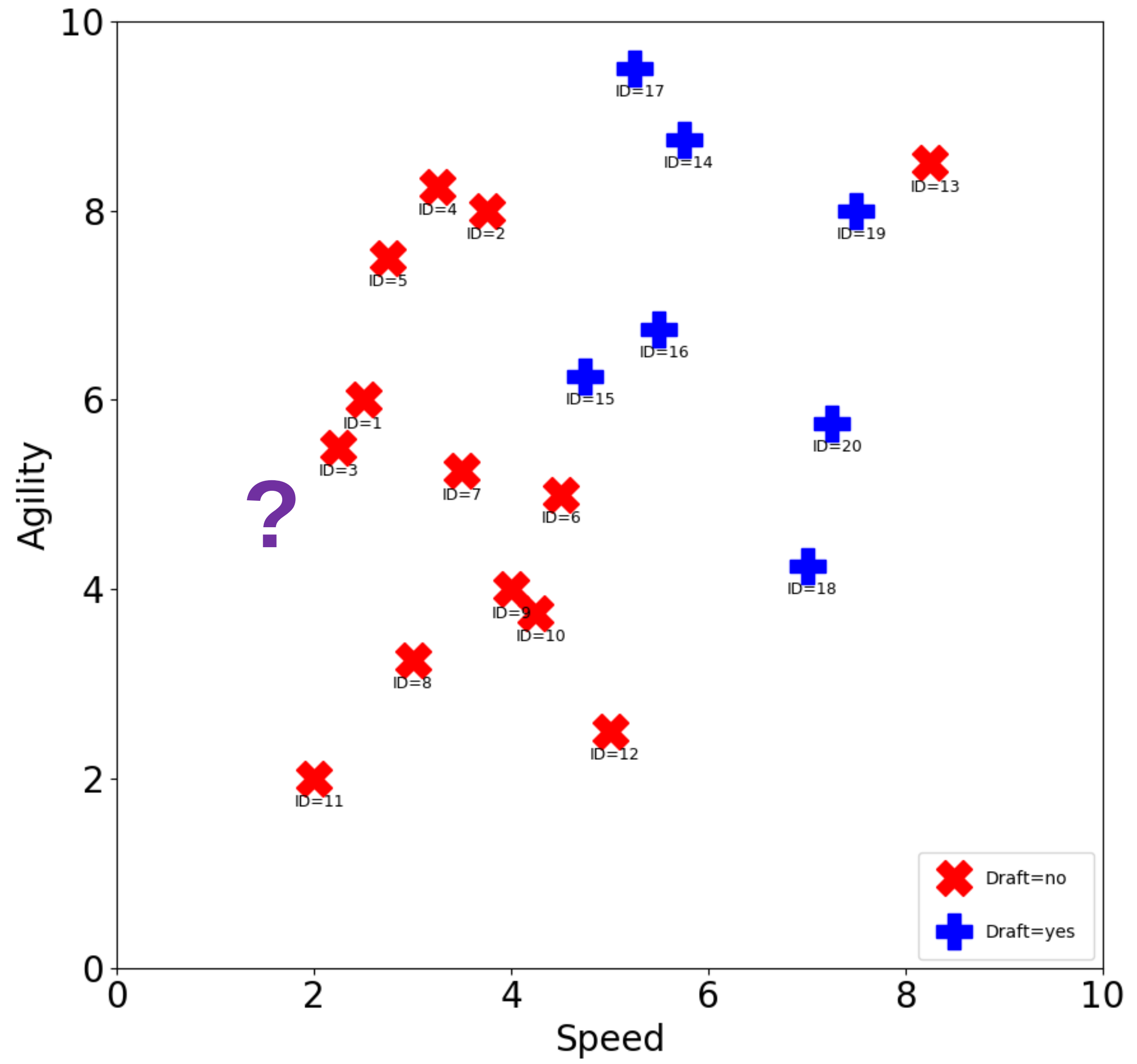
- 1-Nearest Neighbour algorithm:
  - Simplest similarity-based/instance-based method
  - No real training phase: just store the training cases
  - Given a query case with value to be predicted, compute its distance from all stored instances
  - Choose the nearest one; assign the test case to have the same label (class or regression value) as this one
  - Requires a **distance metric**
  - Main problem: susceptibility to noise
- To reduce susceptibility to noise, use more than 1 neighbour (more on this later)

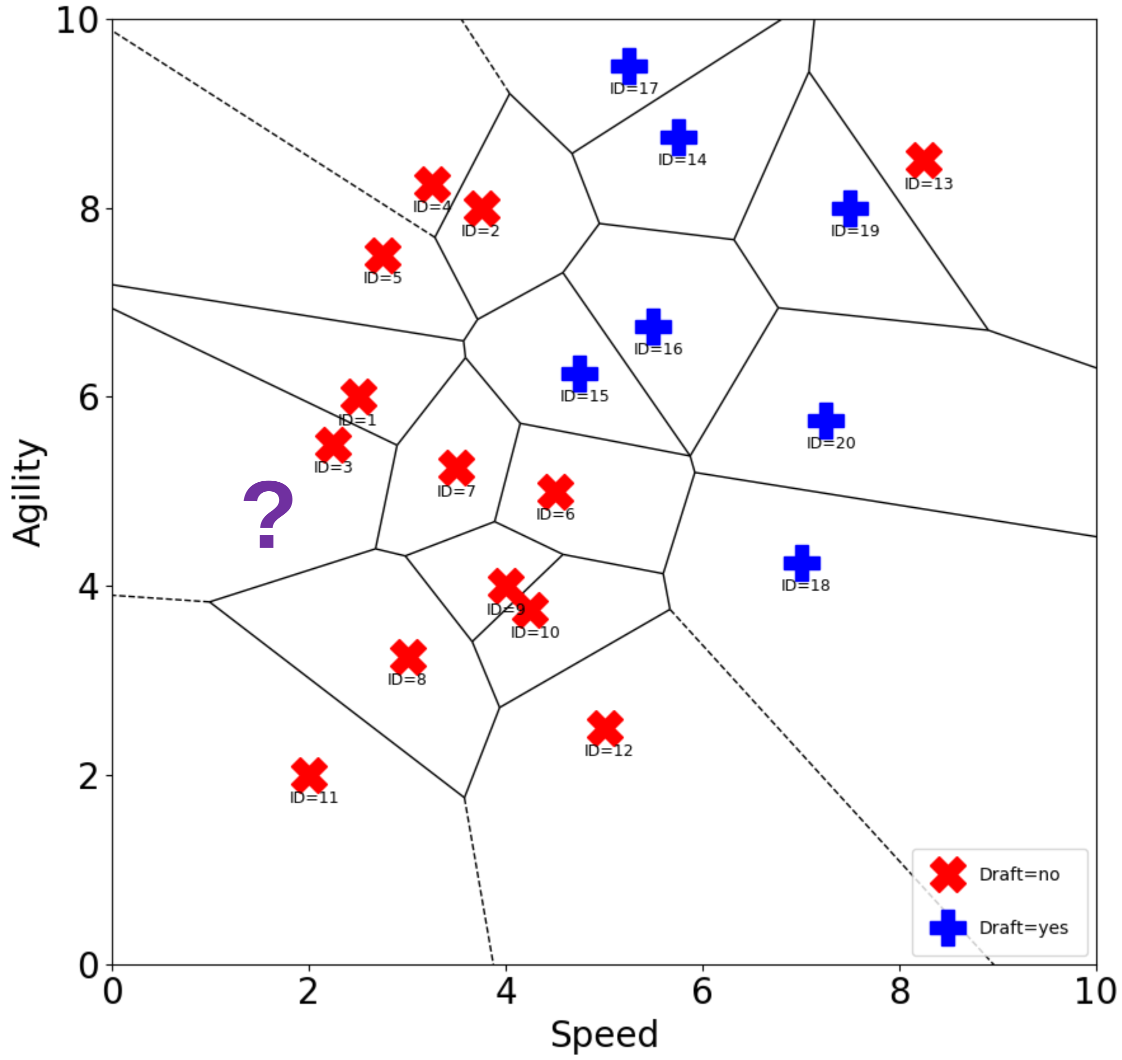
Visualising decision boundaries

- By visually inspecting the feature space plot, we can see that 1 NN will predict the target class as "no"
- 1 NN with Euclidean distance is equivalent to partitioning the feature space into a **Voronoi tessellation**
- Finding the predicted target class is equivalent to finding which **Voronoi region** it occupies
- Note: all visualisations from here on use Euclidean distance

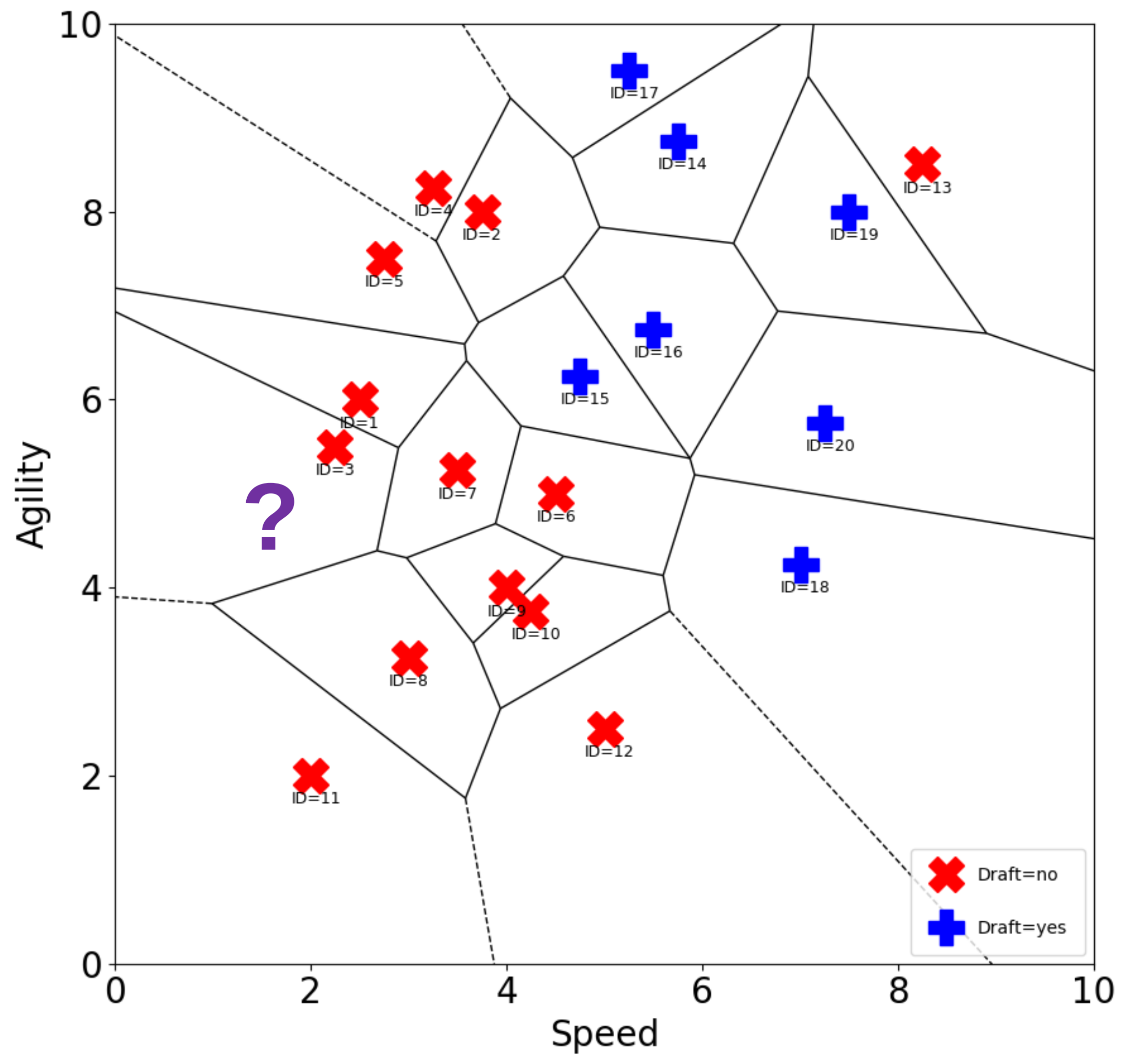


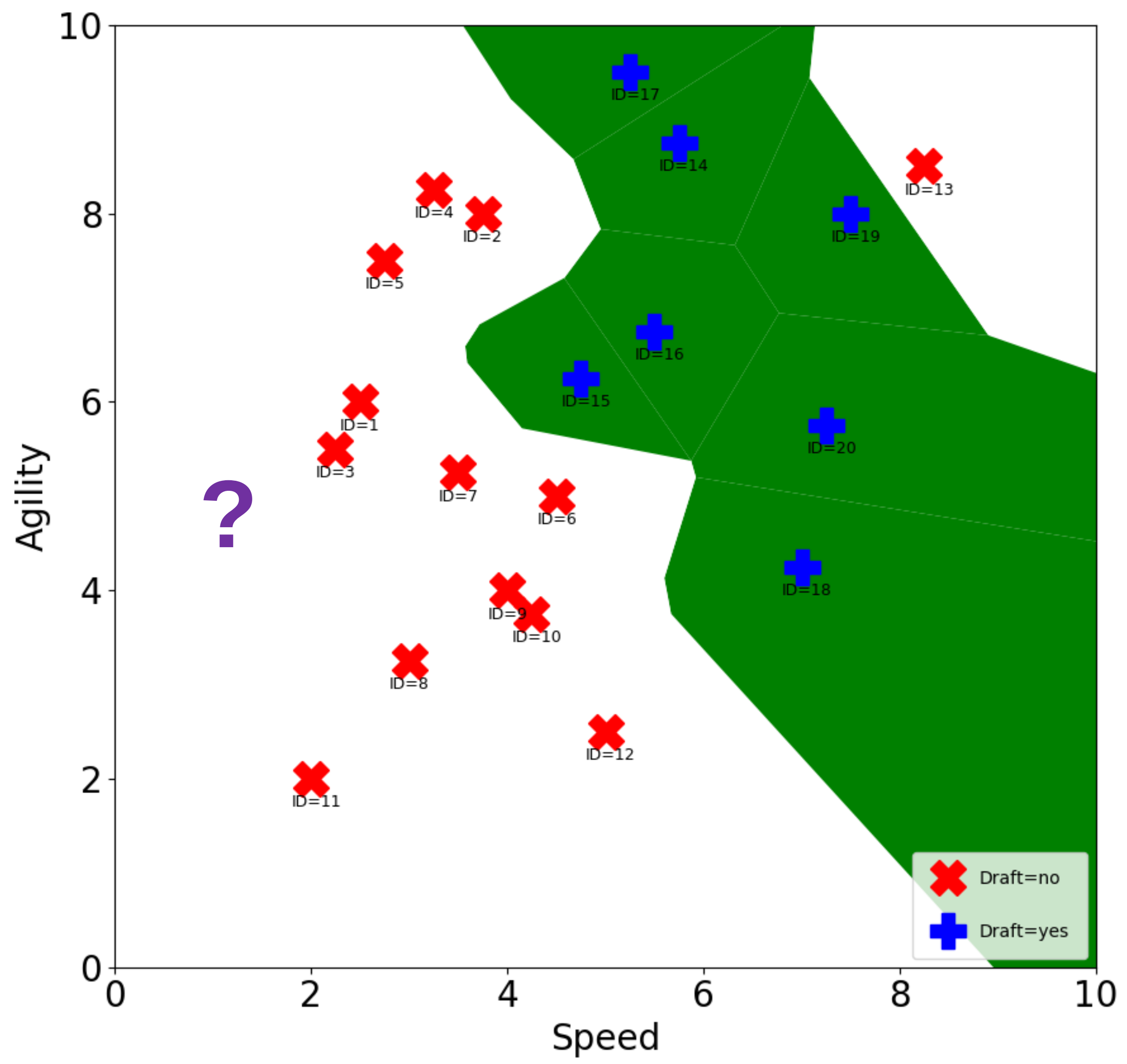
Voronoi tessellation





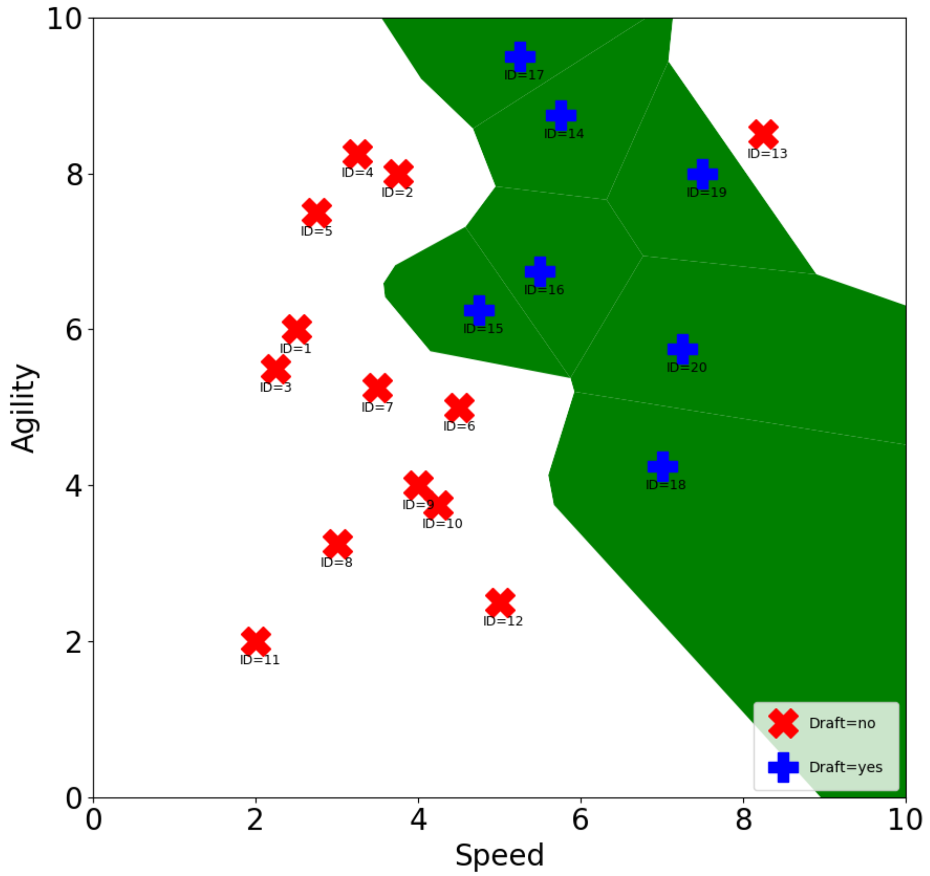
1-NN Decision boundary from Voronoi tessellation

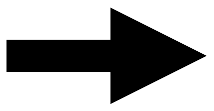


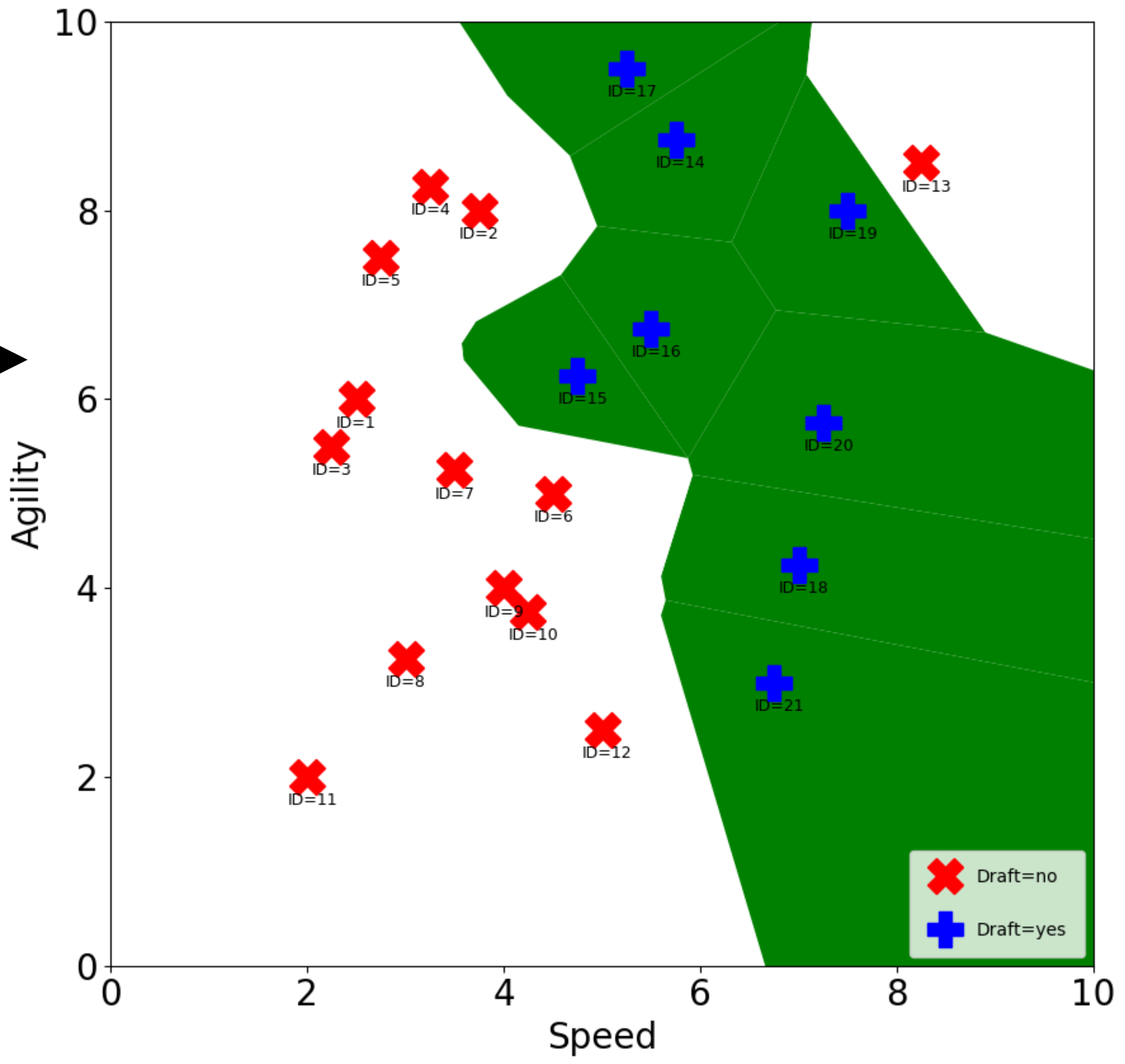


Effect of adding more training data

ID	Speed	Agility	Draft?
21	6.75	3.00	yes







Simple Example of 1-NN

- *HousePrices-1NN.xlsx*
  - Very simple & inflexible 1-NN implementation
  - Cannot handle other datasets directly

	Size (m^2)	# Beds	# Floors	Age (yrs)	Price (k€)
A	195	5	1	40	450
B	130	3	2	35	220
C	140	3	2	26	310
D	80	2	1	30	170
E	180	5	2	38	400
Query	130	4	2	30	?

- How it works:
  - Training data scaled using z-normalisation (more on normalisation later)
  - Enter a query: it is scaled using same scale factors
  - Euclidean/Manhattan distances between query and each instance in the training set are computed
  - Minimum distance identified
  - Look up corresponding row to get 1-NN price estimate

Similarity-based learning

Part 5:

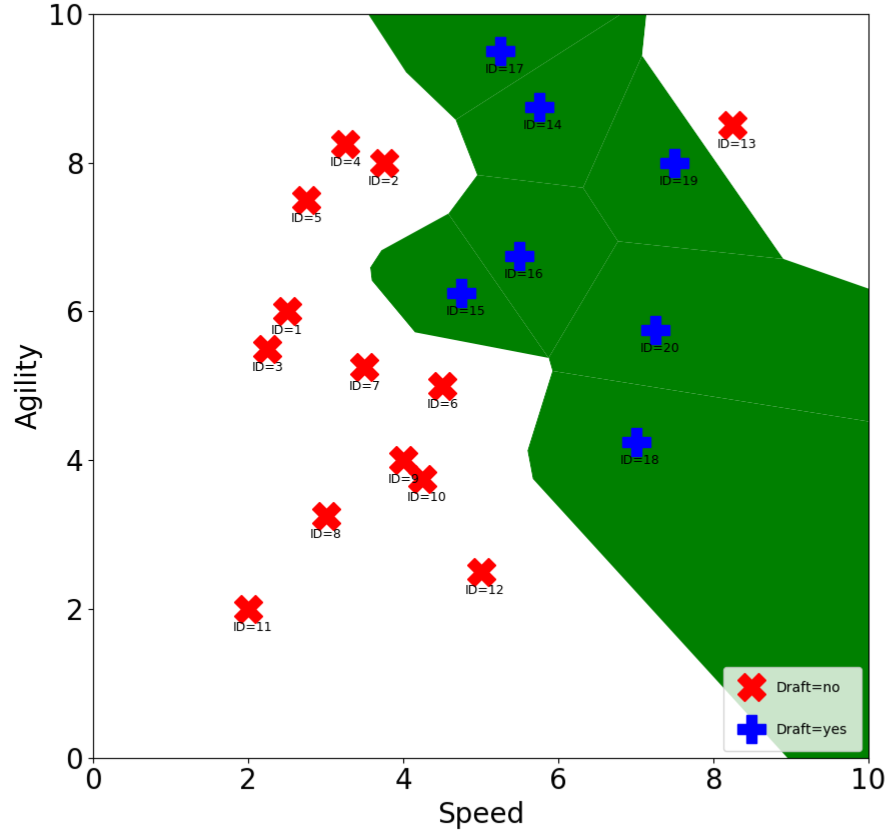
The k Nearest Neighbours algorithm

k Nearest Neighbours Algorithm

- Operation is relatively easy to appreciate
- Key insight:
  - Each example is a point in the feature space
  - If samples are close to each other in feature space, they should be close in their target values
- Related to *Case-based Reasoning*
- How it works:
  - When you want to classify a new **query case**:

    Compare it to the stored set and retrieve the *k* most similar one(s)

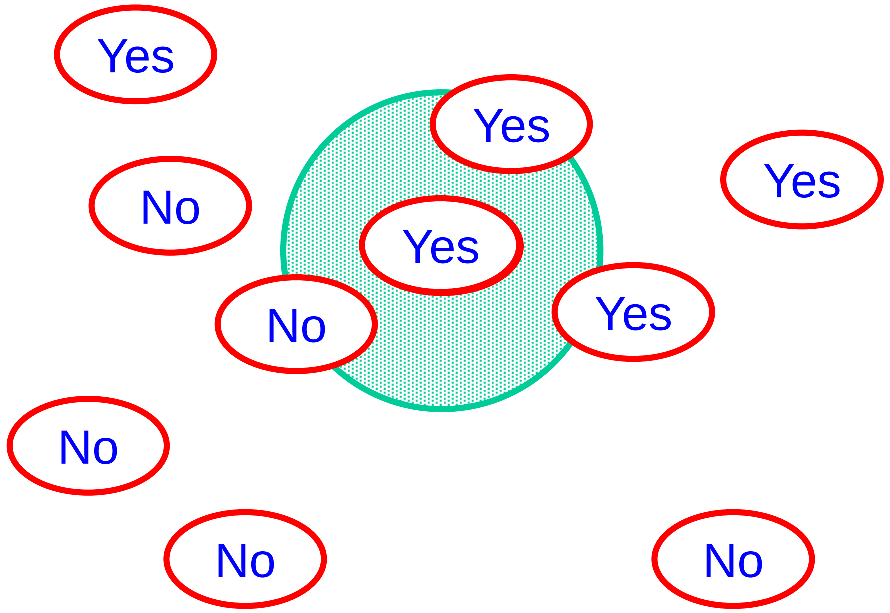
    Give the query case a label based on the similar one(s)



k Nearest Neighbours Algorithm

- *k* Nearest Neighbours algorithm:
  - Base prediction on several (*k*) nearest neighbours
  - Compute distance from query case to all stored cases, and pick the nearest *k* neighbours
  - Simplest way to do this: sort them by distance, pick lowest *k*
  - More efficient: can identify k nearest in a single pass through the list of distances
- Classification with kNN:
  - Neighbours vote on classification of test case
  - Prediction is the majority class

k Nearest Neighbours: Visualisation of Classification Example



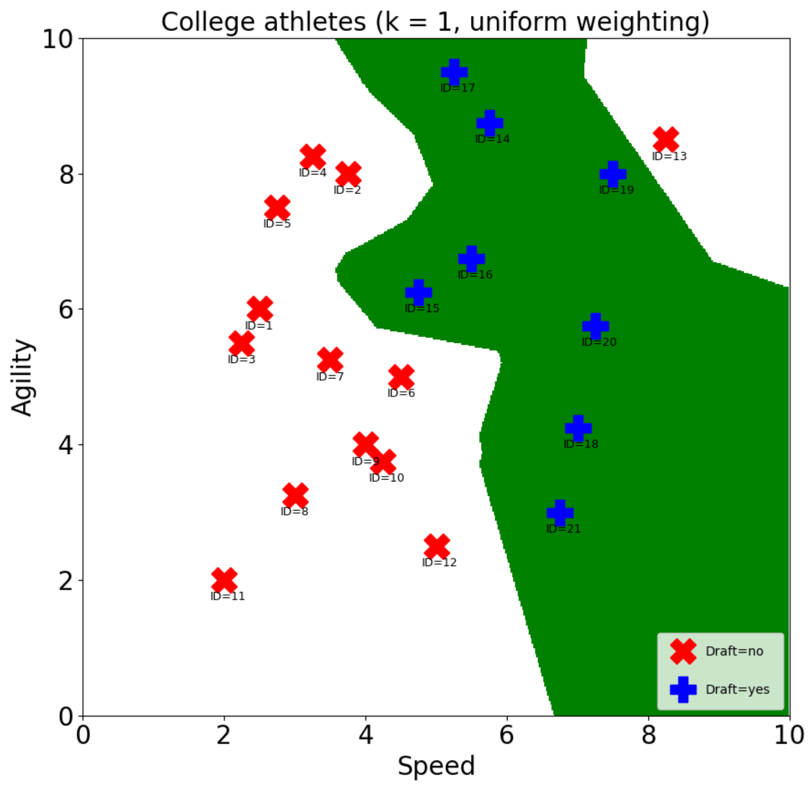
This visualisation assumes Euclidean distance

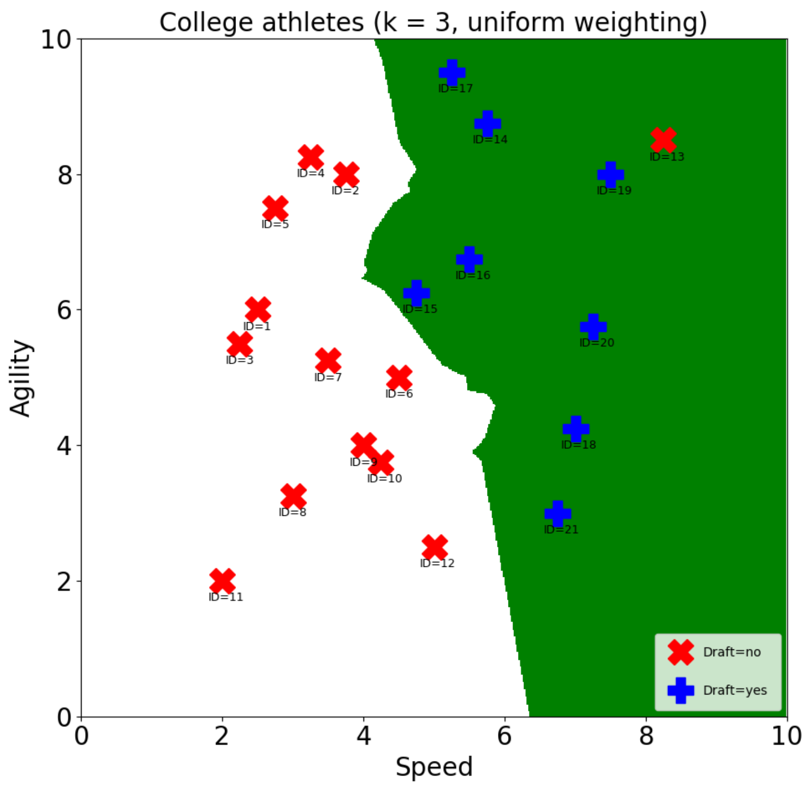
3 nearest neighbours vote: Decision is Yes

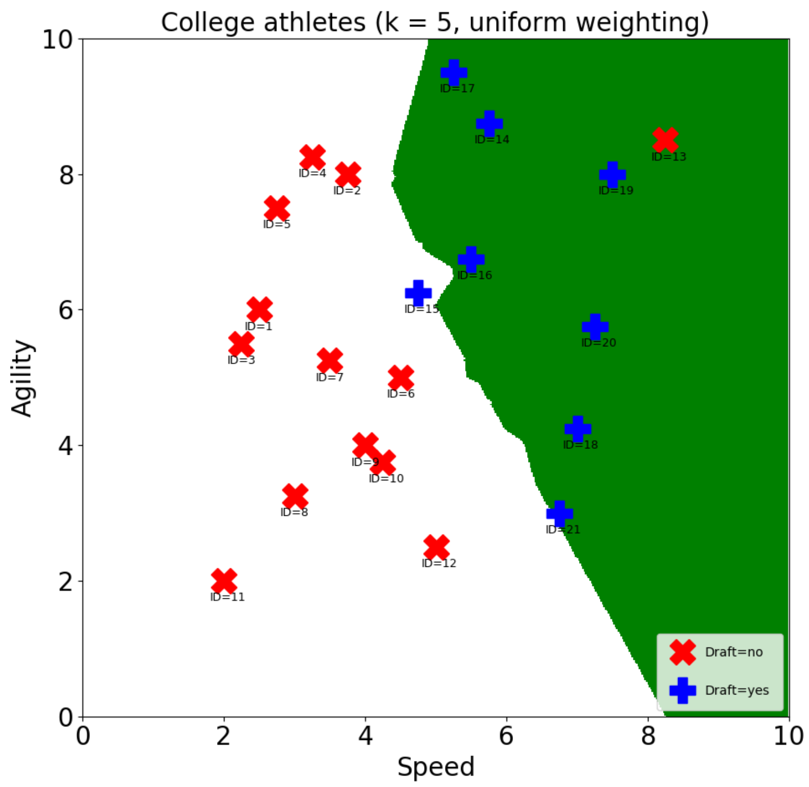
Choosing a value for *k*

- What value for *k*?
  - At least 3 (if not using 1-NN); often in range 5-21
  - Application dependent: need to experiment to find optimum
  - Can use all cases with **distance-weighted kNN (later)**
- Increasing *k* has a smoothing effect
  - Too-low: tends to overfit if data is noisy
  - Too-high: tends to underfit
  - In imbalanced datasets, the majority target class tends to dominate for larger *k*
- Note: *k* does not affect computational cost much
  - Most cost of computation is in calculating distances from the query to all stored instances (linear in #cases and #attributes)

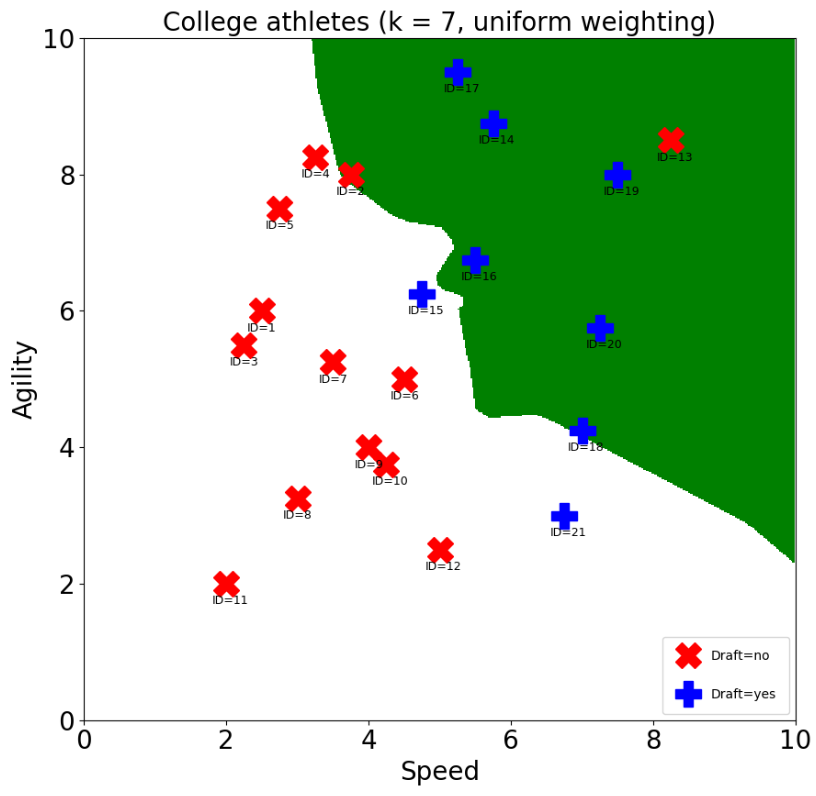
Effect of increasing k

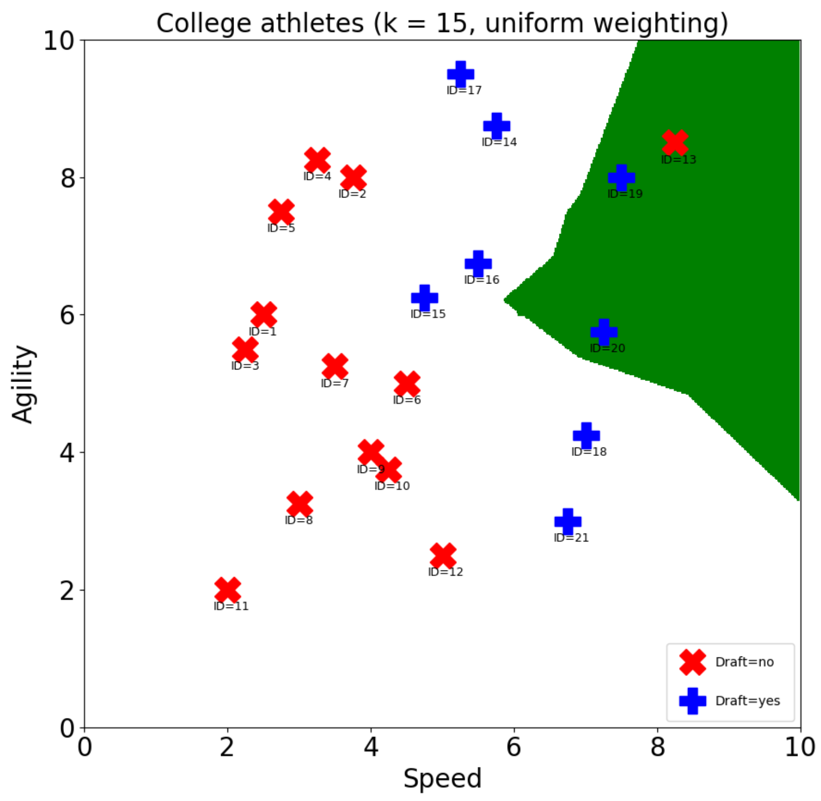


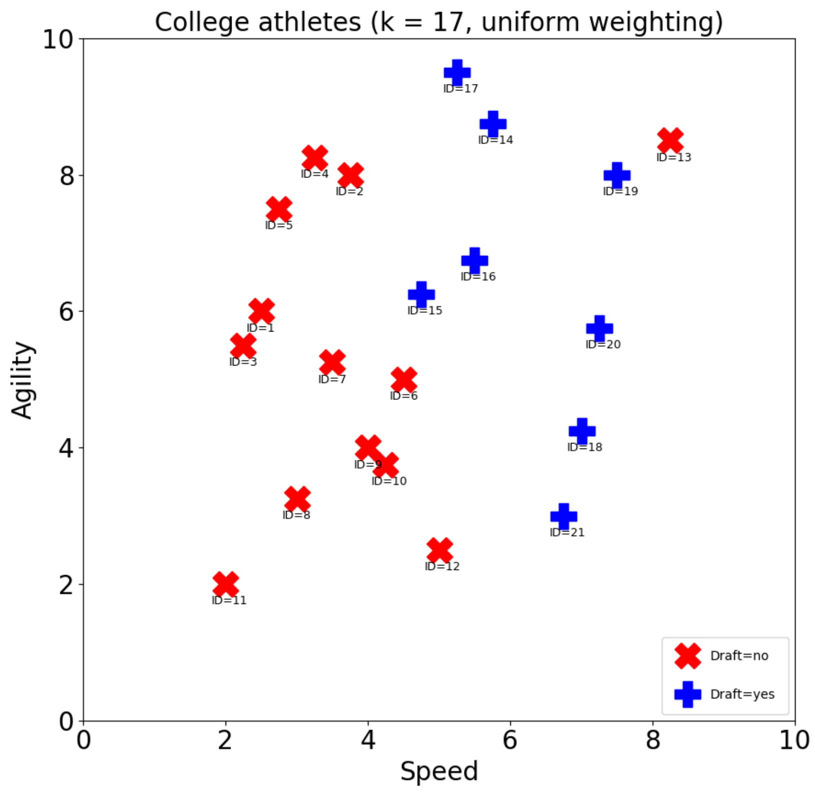




Effect of increasing *k*

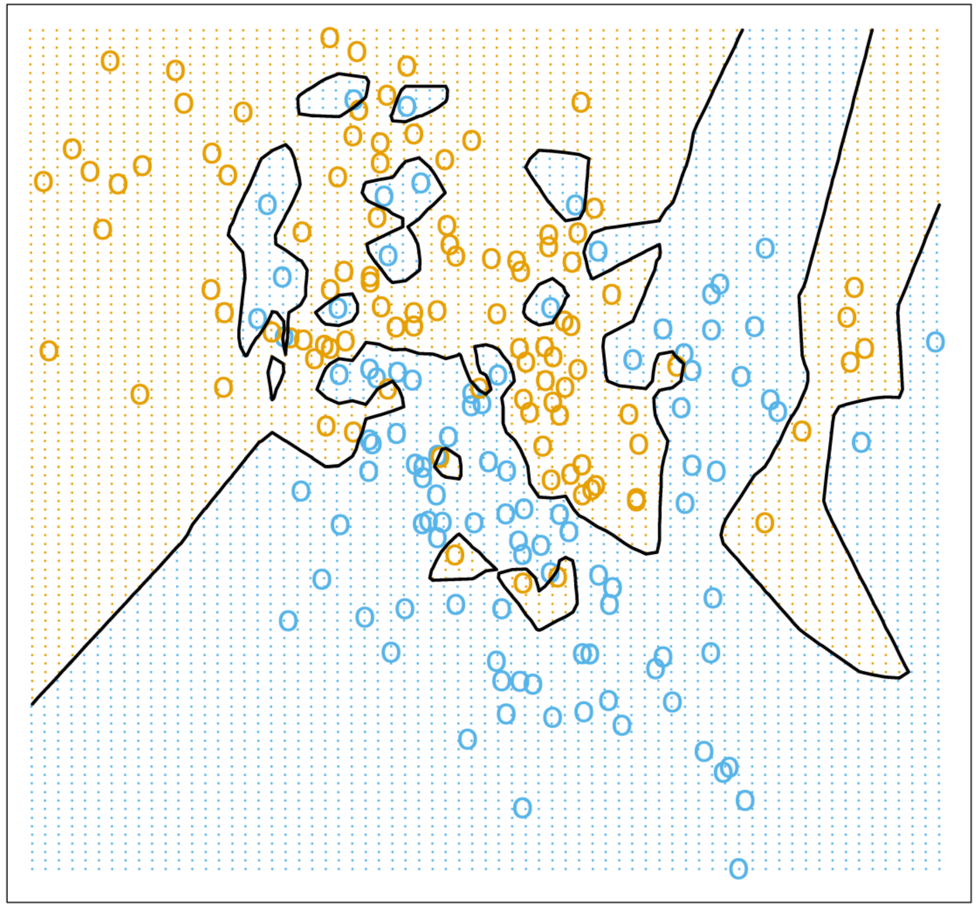




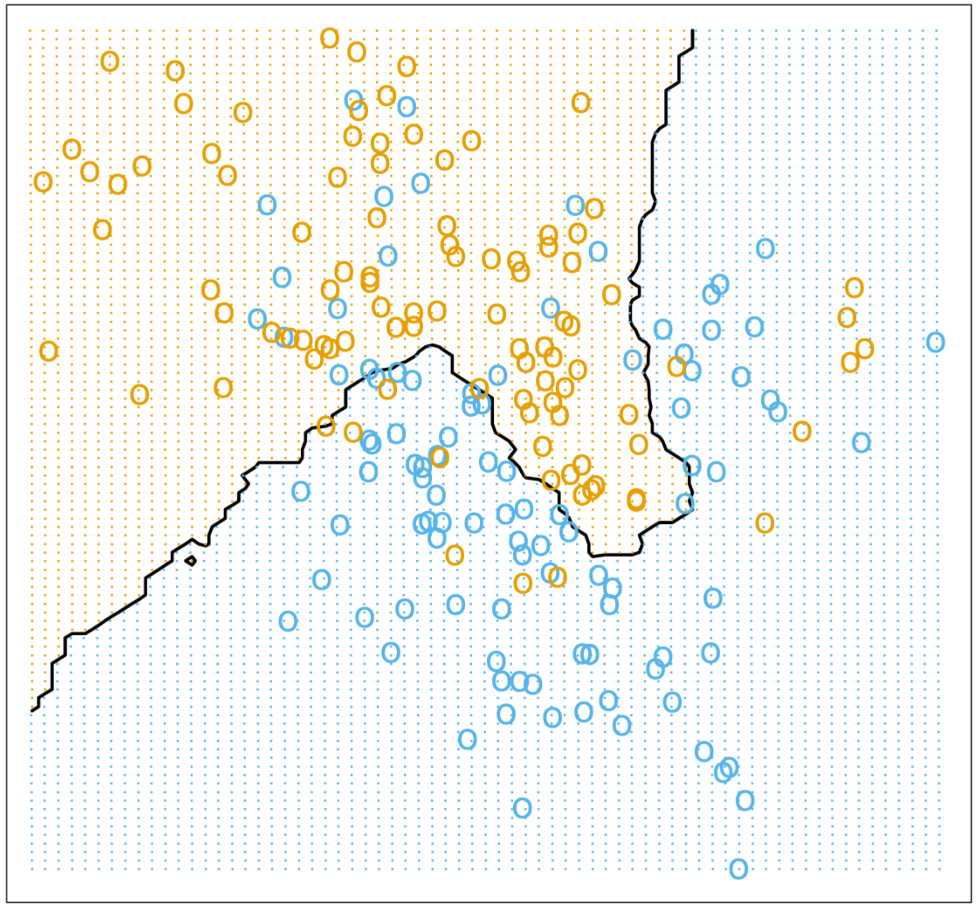


Smoothing Effect of *k*

1-Nearest Neighbor Classifier

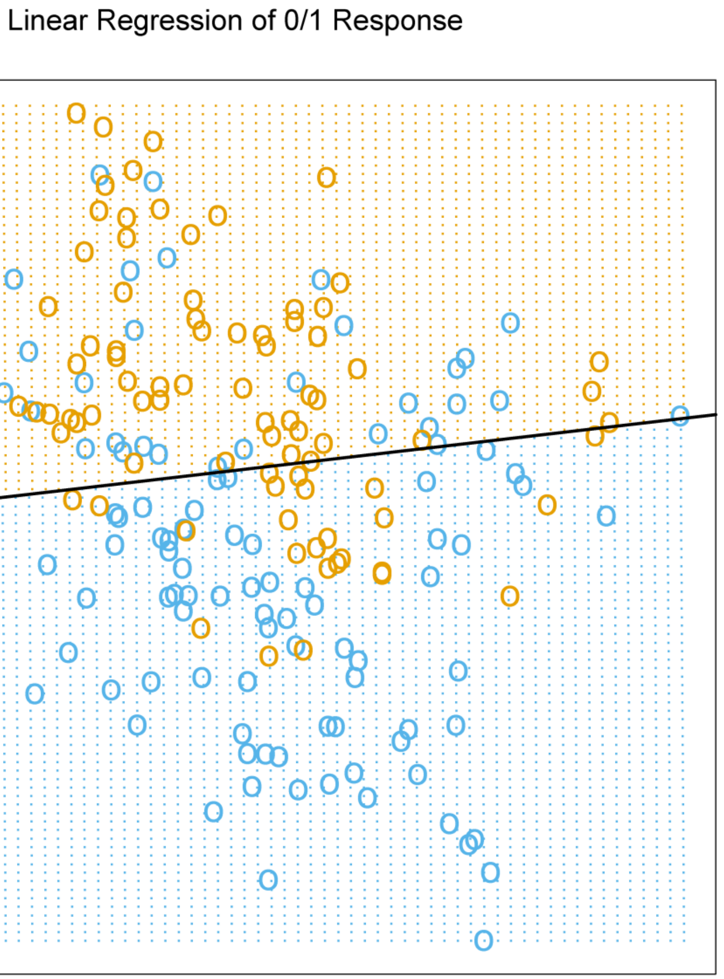


15-Nearest Neighbor Classifier



Hastie, Tibshirani & Friedman, Figs 2.3 and 2.2. Recall discussion of overfitting in Topic 2.

Aside: Underfitting on Same Data

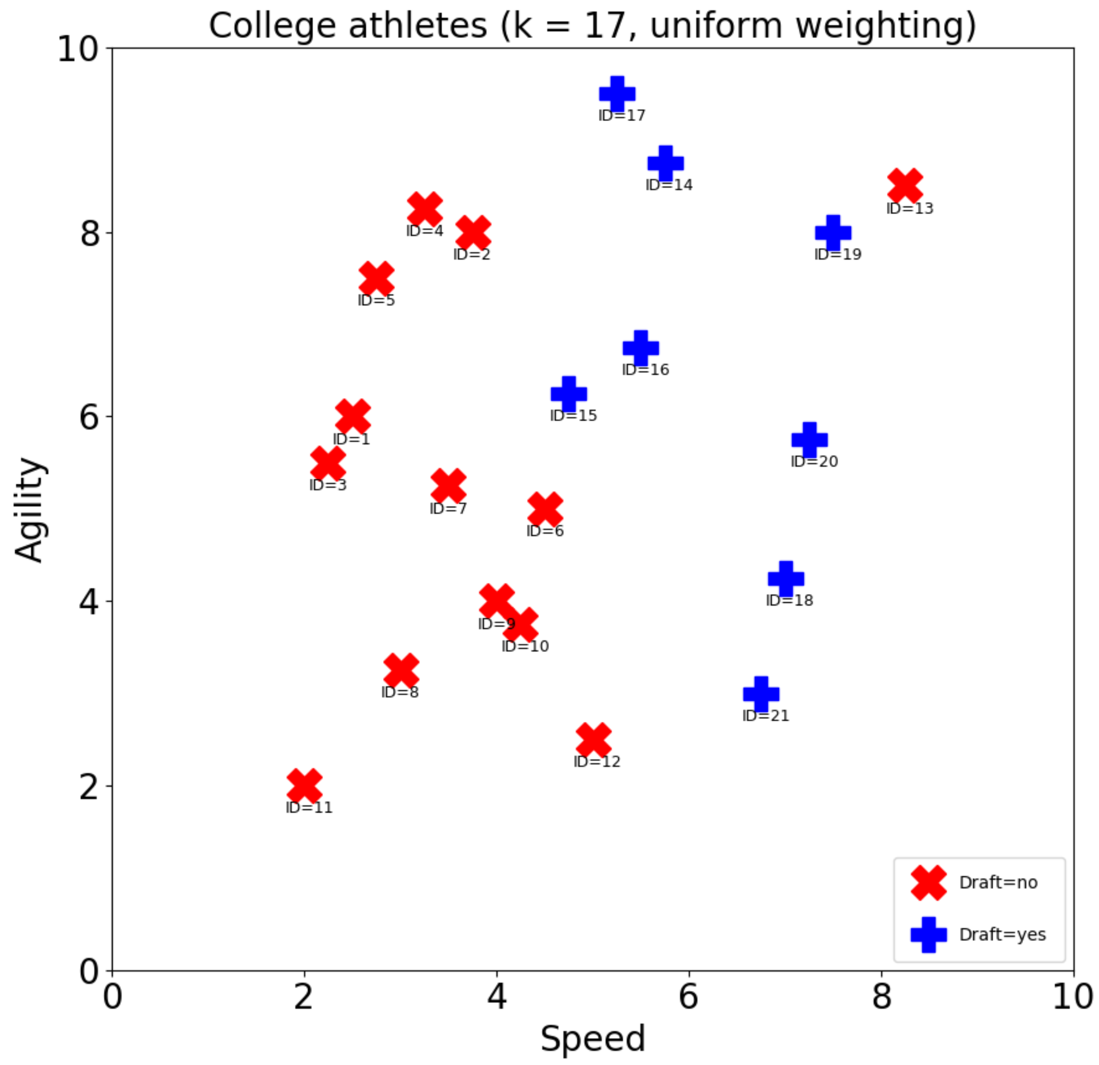


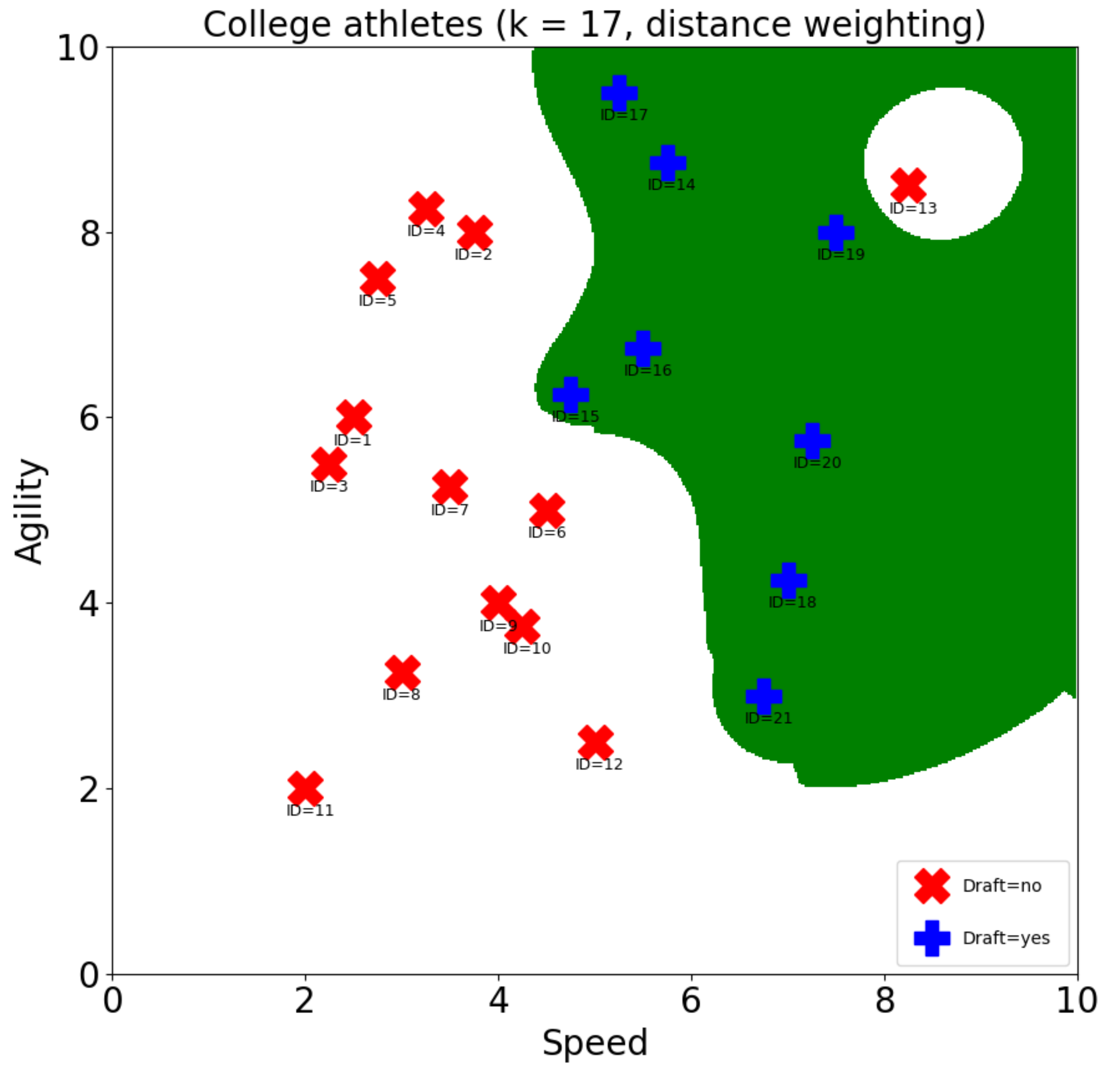
Hastie, Tibshirani & Friedman, Figs 2.1. Will cover linear regression in a future topic.

Distance-weighted kNN

- Distance-Weighted kNN
  - Give each neighbour weight: inverse of distance from target
  - Use weighted vote or weighted average
  - Reasonable to use $k = |\text{all training cases}|$

Effect of distance weighting





NUI Galway
OÉ Gaillimh

Similarity-based learning

Part 6:

Alternate similarity measures

Recap: measuring similarity

- So far, we have measured similarity using distance metrics only
- Each independent variable/attribute is treated as a dimension in hyperspace
- We discussed the well-known Euclidean and Manhattan distance metrics
- We also considered the Minkowski distance, which generalises both Euclidean and Manhattan distances
- In this section we will discuss some issues which can be encountered when measuring similarity and introduce some alternate ways to measure similarity (e.g. similarity indices)
- Note: difference between a similarity index and a distance metric

Data normalisation

- Problem — Scaling:
  - Attribute 1 has range 0-10,
    Attribute 2 has range 0-1000
  - Attribute 2 will dominate calculations
- Solution:
  - Rescale all dimensions independently
    - Mean=0, Std deviation=1 [Z-Normalisation] (HousePrices-1NN.xlsx has a worked example)

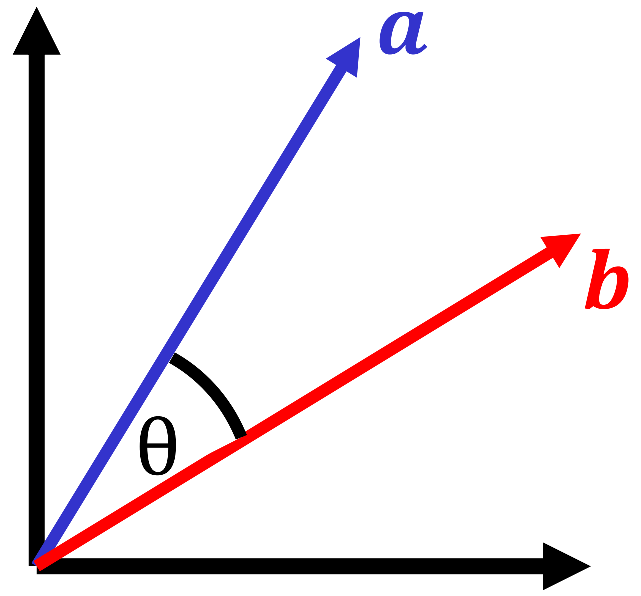
      D ← (D - Mean) / StDev
    - Min=0, Max=1 [0-1 Normalisation] (also referred to as “range normalisation”)

      D ← (D – Min) / (Max – Min) (also utopia = max, nadir=min)

Normalisation is important in many other areas of machine learning and optimisation

Cosine similarity

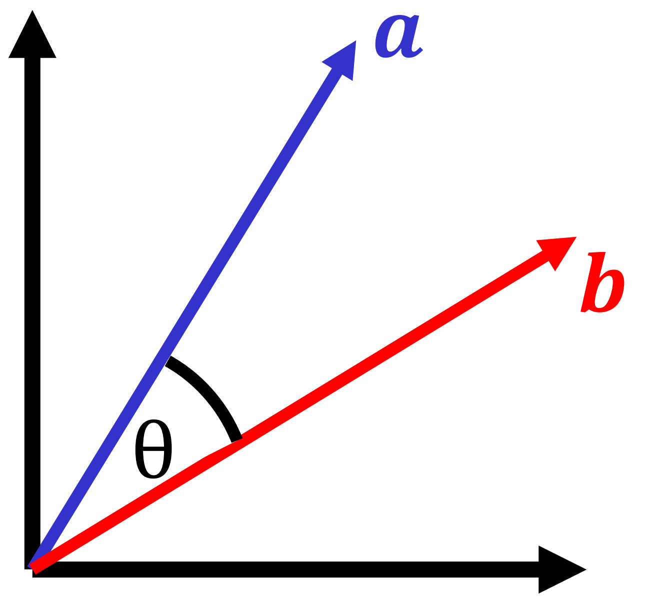
- Cosine similarity is an **index** that may be used to measure the similarity of two instances with continuous attributes. It is the **cosine** of the inner angle between the two vectors that extend from the origin to the points of interest in the feature space
- As before m is the number of features/attributes (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- $\boldsymbol{a}\cdot\boldsymbol{b}$ is the vector dot product, and $|\boldsymbol{a}|$ is the magnitude of the vector $\boldsymbol{a}$



$$Cosine(\boldsymbol{a},\boldsymbol{b}) = cos(\theta) = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|} = \frac{\sum_{i=1}^{m}(\boldsymbol{a}[i]\times\boldsymbol{b}[i])}{\sqrt{\sum_{i=1}^{m}(\boldsymbol{a}[i])^2}\times\sqrt{\sum_{i=1}^{m}(\boldsymbol{b}[i])^2}}$$

Cosine similarity

- The values of $Cosine(\boldsymbol{a},\boldsymbol{b})$ will be in the range 0 to 1 if all attribute values are non-negative, or -1 to +1 if negative values are present. Can use 0-1 range normalisation to ensure $Cosine(\boldsymbol{a},\boldsymbol{b})$ in 0-1 range.
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ = 1 -> most similar (vectors are parallel)
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ = 0 -> least similar (vectors at 90 deg)
- **Note:** for this index higher values indicate greater similarity (for distance metrics the opposite is true)
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ allows **scale-invariant** comparisons between instances



cos(0) = 1.0

cos(90) = 0.0

Similarity for discrete attributes

- So far, we have considered similarity measures that only apply to continuous attributes
- Do not confuse discrete/continuous attributes with classification/regression!
- Many datasets have attributes that have a finite number of discrete values (e.g. Yes/No or True/False, survey responses, ratings)
- Once approach to handling discrete attributes is the **Hamming distance**
- Hamming distance is calculated 0 for each attribute where both cases have the same value, 1 for each where they are different
- E.g. Hamming distance between “Stephen" and “Stefann" is 3.

Similarity for discrete attributes

Similarity measures for binary (e.g. yes/no) attributes include:

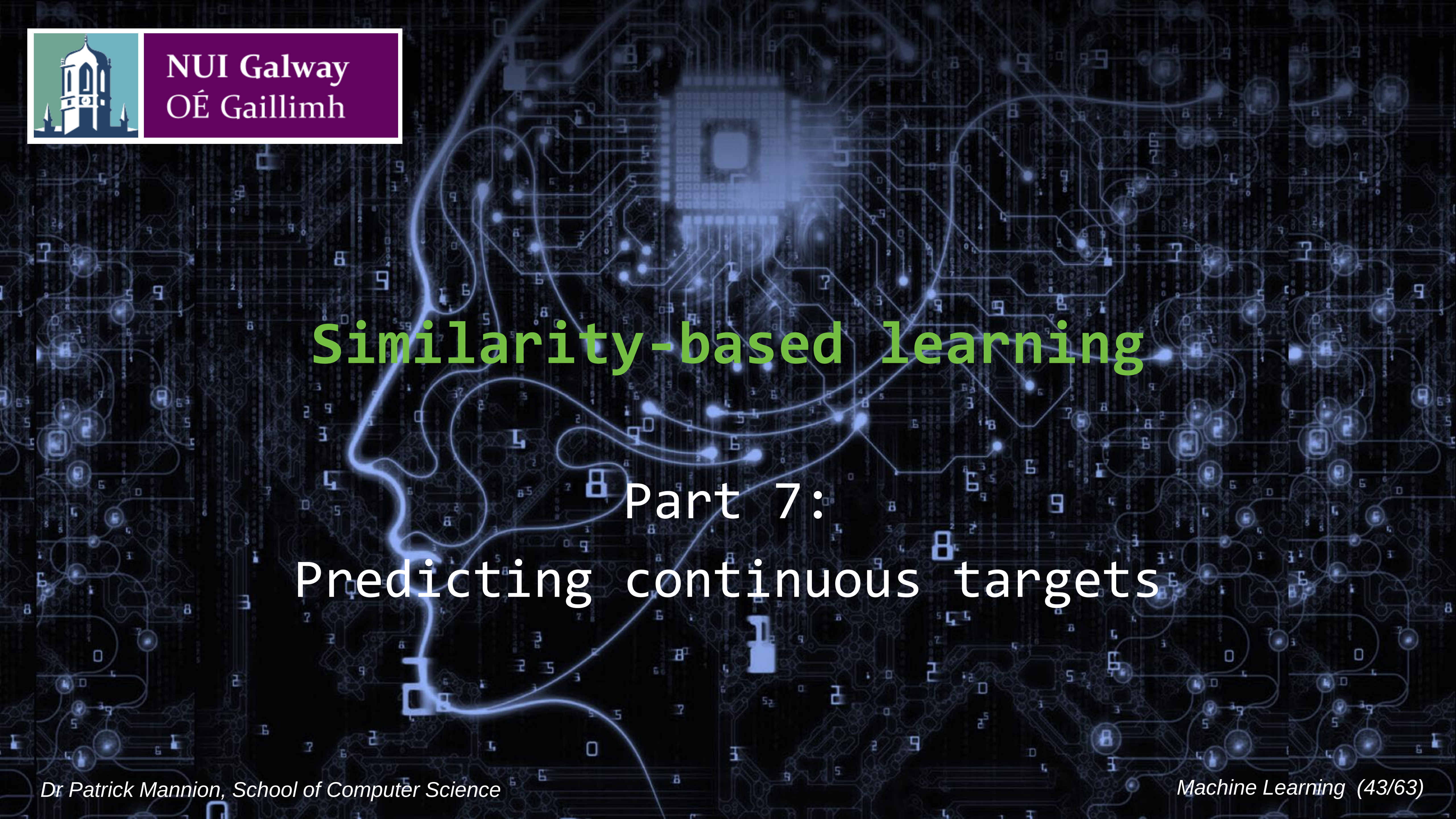
- The **Russel-Rao** similarity index
  - Focuses on measuring "co-presences", e.g. when comparing users of a web store, compare which pages/products they have clicked on (the attributes are for each product/page, whether that product/page was visited before)
  - The similarity score of two users is based on how many co-presences they share
- The **Sokal-Michener** similarity index
  - Measures similarity using co-presences as well as co-absences
  - Co-absences of attributes may be just as important as co-presences in some domains, e.g. in medical applications, it may be important to compare which symptoms each patient has, as well as which symptoms each patient does not have

Choosing a Distance Metric (1)

- Choices will be limited by attribute data type, e.g. continuous or discrete
- In general, want a metric that:
  - Reflects differences between cases that are important
  - Downplays differences that are irrelevant
  - Application-dependent: work needed here
- E.g. Cosine similarity based on angle, not absolute value
  - May be good to compare objects if we care that they are the same shape but not the same scale
  - Less good if the scale is important

Choosing a Distance Metric (2)

- Possible to design distance metrics of our own
  - Given our understanding of what is important in a domain, can formulate a metric that emphasises what is important and de-emphasises what is not
- Remember that distance metrics must obey **Metric Axioms**:
  1. **Non-negativity**: $metric(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
  2. **Identity**: $metric(\boldsymbol{a}, \boldsymbol{b}) = 0 \Leftrightarrow \boldsymbol{a} = \boldsymbol{b}$
  3. **Symmetry**: $metric(\boldsymbol{a}, \boldsymbol{b}) = metric(\boldsymbol{b}, \boldsymbol{a})$
  4. **Triangular Inequality**: $metric(\boldsymbol{a}, \boldsymbol{b}) \leq metric(\boldsymbol{a}, \boldsymbol{c}) + metric(\boldsymbol{b}, \boldsymbol{c})$
- These axioms may occasionally be violated in similarity measures, but only with good reason!
  - E.g. “A contains B” is non-symmetric



Similarity-based learning

Part 7:

Predicting continuous targets

kNN - recap

- *k*-Nearest Neighbour algorithm:
  - Base prediction on several (*k*) nearest neighbours
  - Compute distance from query case to all stored cases, and pick the nearest *k* neighbours

- Classification with kNN:
  - Neighbours vote on classification of test case
- **Regression:**
  - **Average the value of the neighbours**

kNN regression – uniform weighting

$$prediction(\boldsymbol{q}) = \frac{1}{k}\sum_{i=1}^{k} t_i$$

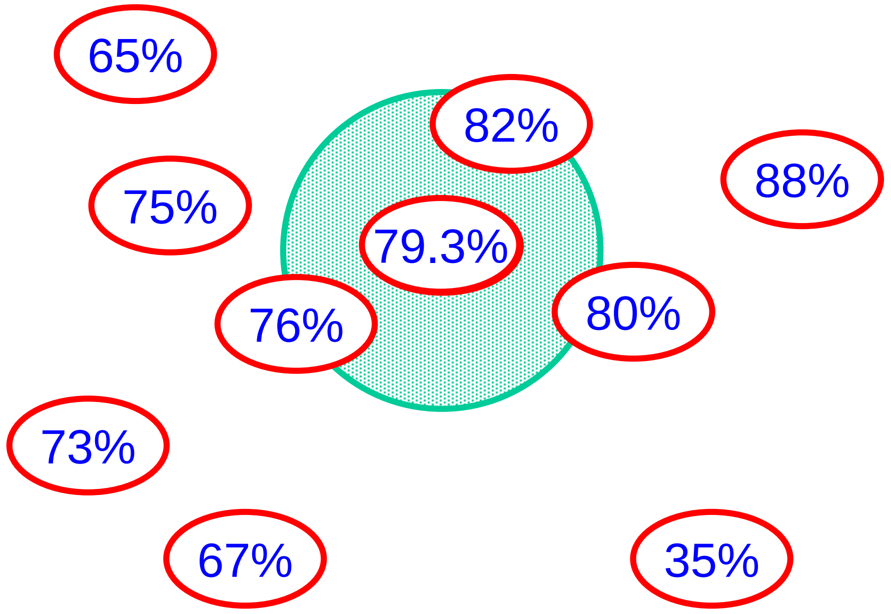
- $\boldsymbol{q}$ is a vector containing the attribute values for the query instance
- *k* is the number of neighbours as before
- t_i is the target value for neighbour *i*
- This assumes that each neighbour is given equal weighting

kNN regression – distance weighting

$$prediction(\boldsymbol{q}) = \frac{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2} \times t_i\right)}{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2}\right)}$$

- $\boldsymbol{q}$ is a vector containing the attribute values for the query instance
- $dist(\boldsymbol{q},\boldsymbol{d}_i)$ returns the distance between the query and neighbour *i*
- This assumes that each neighbour is given a weighting based on the inverse square of its distance from the query

k Nearest Neighbours: Visualisation of Regression Example

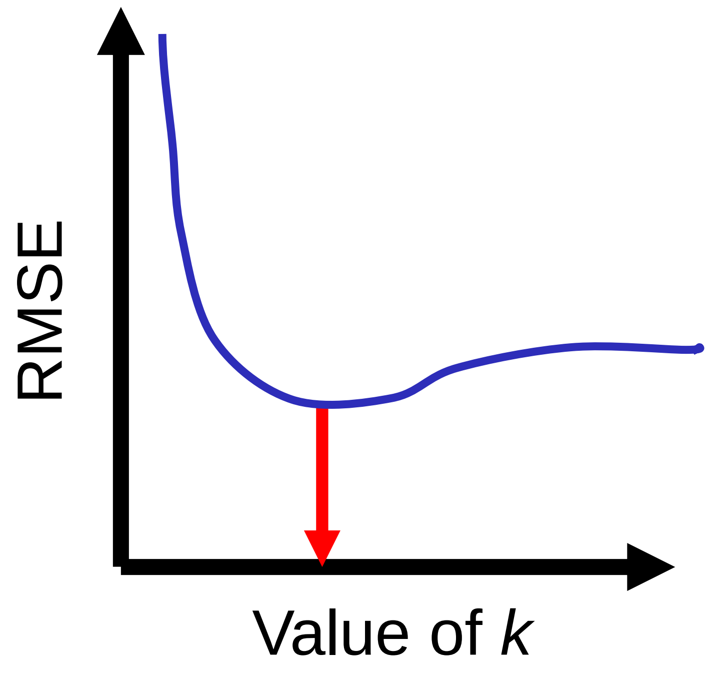


This visualisation assumes Euclidean distance and uniform weighting

Estimate: 79.3%

Selecting *k* for regression tasks

- As with *k*NN for classification, selecting an appropriate value of *k* is important for regression tasks. How should we set it?
- One solution: test a range of values of *k* and select the one that gives the best score on your chosen evaluation metric, e.g. Root Mean Square Error (RMSE). Should incorporate cross validation when computing the RMSE for each value of *k*
- Note: even values of *k* aren't a problem for regression tasks as majority voting isn't used



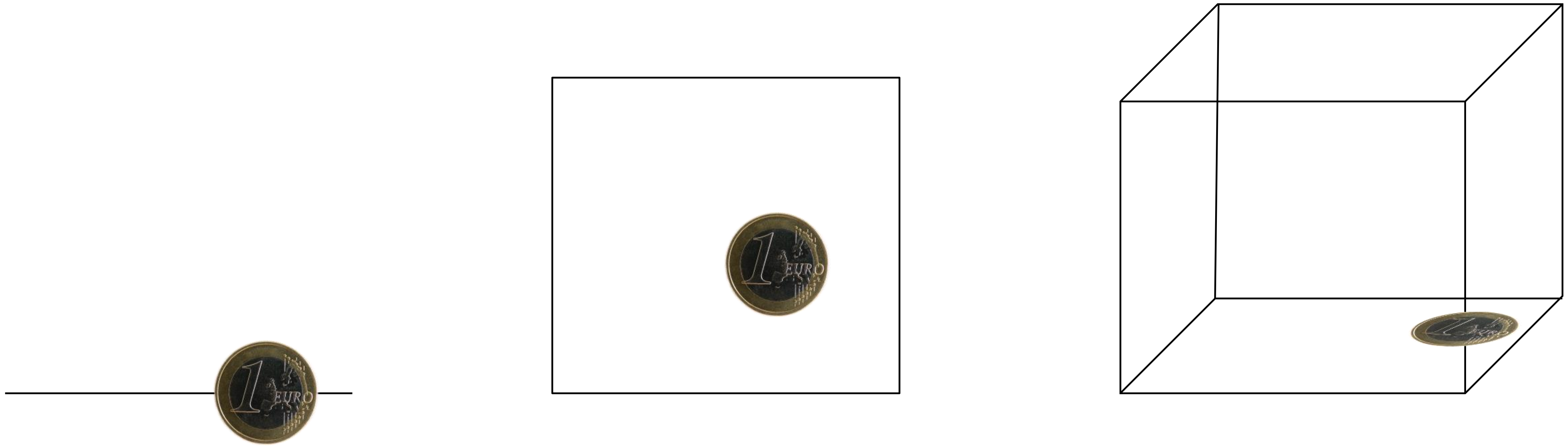
Pick the value of k that Gives the lowest RMSE

NUI Galway
OÉ Gaillimh

Similarity-based learning

Part 8:
Feature selection

The Curse of Dimensionality [1]



The Curse of Dimensionality [2]

- Problem — Curse of Dimensionality:
  - Some attributes are much more significant than others
  - All considered equally in distance metric => bad predictions
  - With many attributes, everything becomes 'distant' [see next]
- Solution a:
  - Assign weighting to each dimension (**not** same as distance-weighted kNN!)
  - Optimise weighting to minimise error
- Solution b:
  - Give some dimensions 0 weight: **Feature Subset Selection**

Any algorithm that considers all attributes in a high-dimensional space equally has this problem, not just kNN + Euclidean Distance!

The Curse of Dimensionality [3]

- Russell & Norvig:

  Consider N cases with d dimensions, in hypercube of **unit** volume

  Assume neighbourhoods are hypercubes, length b: volume is b^d

  To contain k points, average neighbourhood must occupy k/N of entire volume

  => $b^d = k/N$

  => $b = (k/N)^{1/d}$

  High dimensions:

  $k = 10$; $N = 1{,}000{,}000$; $d = 100$ => $b = 0.89$
  i.e. neighbourhood spans nearly 90% of each dimension of space!

  Low dimensions:

  k and N unchanged; $d = 2$ => $b = 0.003$ [OK]

- High-D spaces are generally very sparse: all neighbours far away

Feature Selection

- Fortunately, some algorithms partially mitigate the effects of the curse of dimensionality (e.g. decision tree learning). This is not true for all algorithms however, and heuristics for search can sometimes be misleading!
- kNN and many other algorithms use all attributes when making a prediction
- Acquiring more data is not (always) a realistic option
- The best way to avoid the curse is to use only the most useful features during learning, this process is known as feature selection

Types of features

We may wish to distinguish between different types of descriptive features:

- **Predictive:** provides information that is useful when estimating the correct target value
- **Interacting:** provides useful information only when considered in conjunction with other features
- **Redundant:** features that have a strong correlation with another feature
- **Irrelevant:** doesn't provide any useful information for estimating the target value

Ideally, a good feature selection approach should identify the smallest subset of features that maintain prediction performance

Feature Selection Approaches

- **Rank and prune:**
  - rank features according to their predictive power and keep only the top X%
  - A **filter** is a measure of predictive power used during ranking, e.g. information gain
  - Drawback: features evaluated in isolation, so we will miss useful interacting features
- **Search for useful feature subsets:**
  - We can pick out useful interacting features by evaluating feature subsets
  - Could generate, evaluate and rank all possible feature subsets then pick best (essentially a brute force approach, computationally expensive/infeasible?)
  - Better approach: **greedy local search**, build feature subset iteratively by starting out with an empty selection, then trying to add additional features incrementally. Requires evaluation experiments along the way. Stop trying to add more features to the selection once termination conditions are met.

Similarity-based learning

Part 9:

Similarity-based learning considerations

Lazy vs Eager Learning [1]

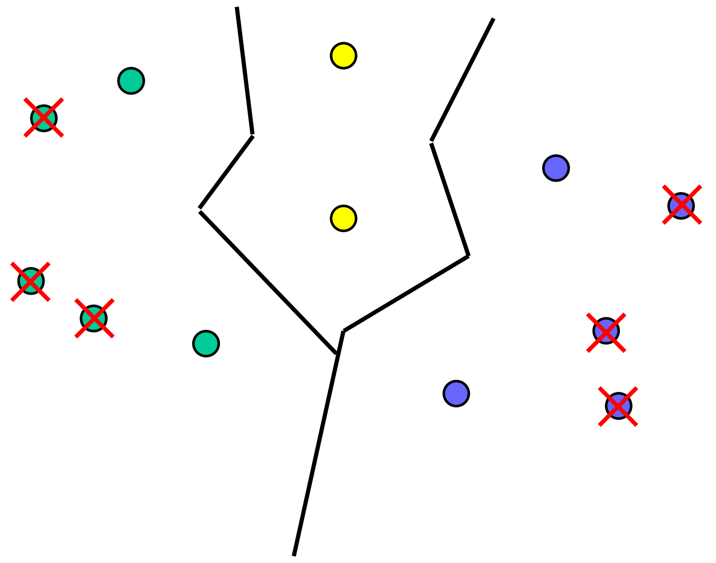
- Eager Learning
  - When given training data, construct model for future use in prediction that summarises the data
  - Analogy: compilation in programming language
  - Slow in model construction,
    quicker in subsequent use
  - Model itself may be useful/informative
- Eager Learning Algorithms:
  - Decision Tree, Artificial Neural Network, …
- On the other hand …

Lazy vs Eager Learning [2]

- Lazy Learning
  - No explicit global model constructed
  - Calculations deferred until new case to be classified
- Creates many local approximations
  - Whereas eager learner must create a global approximation
  - For same form of hypothesis, lazy learner can represent more complex functions
  - E.g. fitting a line to neighbours rather than fitting a line to all of the data
- Lazy Learning Algorithms:
  - 1NN, k Nearest Neighbours, CBR, …
- Concept drift

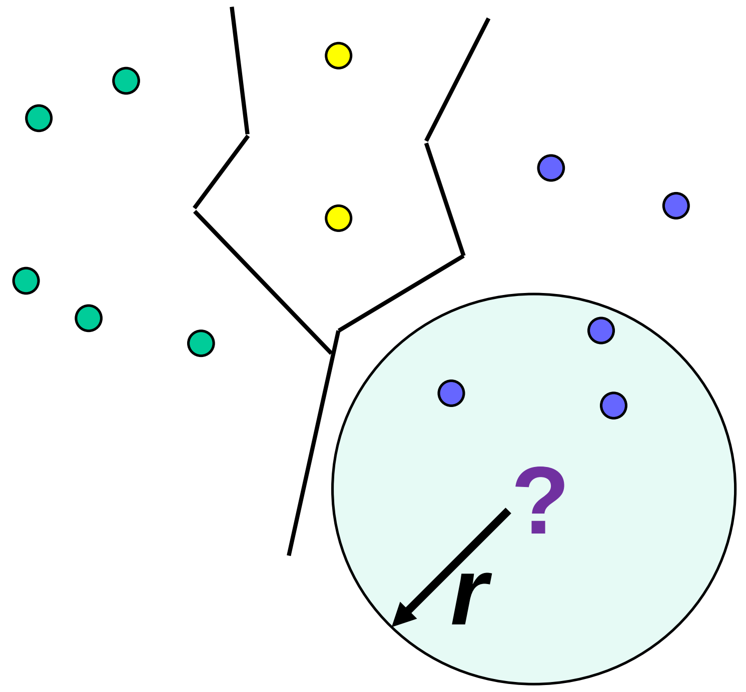
Other Nearest Neighbour variants [1]

- Condensed kNN
  - Classification time proportional to:
    - number of cases
    - number of attributes per case
  - To speed up calculation,
    eliminate unnecessary ones:
    Away from decision boundary



Other Nearest Neighbour variants [2]

- Radius-based Nearest Neighbours
  - Instead of selecting *k* nearest neighbours, could also select all neighbours within a certain radius *r*
  - e.g. RadiusNeighborsClassifier in scikit-learn
  - *r* should be chosen carefully
  - Normalisation is very important!
  - Can also specify a default class for instances that have no neighbours within *r* – potentially useful for flagging outliers



When to use kNN

- Consider using when:
  - Plenty of training cases
  - Moderate number of attributes per case (e.g. < 20)
- Benefits:
  - Easy to implement, few parameters to tune (*k* + similarity measure)
  - Comprehensibility: easy to justify decisions
  - Complex target functions => good for **non-coherent concepts** (e.g. “toxic/non-toxic”). Classes don’t have to be linearly separable.
  - No summarisation => information not lost
- Drawbacks:
  - Problems with irrelevant attributes, many attributes
  - May be slow in classification/regression (no summarisation)

NUI Galway
OÉ Gaillimh

Similarity-based learning

Part 10:
Review of topic

Learning Objectives: Review

After completing this successfully, you are now able to …

- Explain what instance-based learning is
- Distinguish between *lazy* and *eager* learning
- Describe operation of k-Nearest Neighbours for classification and regression
- Discuss implications of the *curse of dimensionality*
- Discuss implications of selecting different distance metrics
- Identify suitable applications for kNN and explain how it could be applied