Similarity-based learning

Part 1: Introduction

Last two week Review

1. Explain what supervised learning is
2. Distinguish it from unsupervised learning and reinforcement learning
3. Describe in detail an algorithm for decision tree induction
4. Demonstrate the application of decision tree induction to a data set
5. List related algorithms
6. Discuss high-level concepts such as choice of hypothesis language, overfitting, underfitting and noise

This week and following week Learning Objectives

After completing this topic 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:

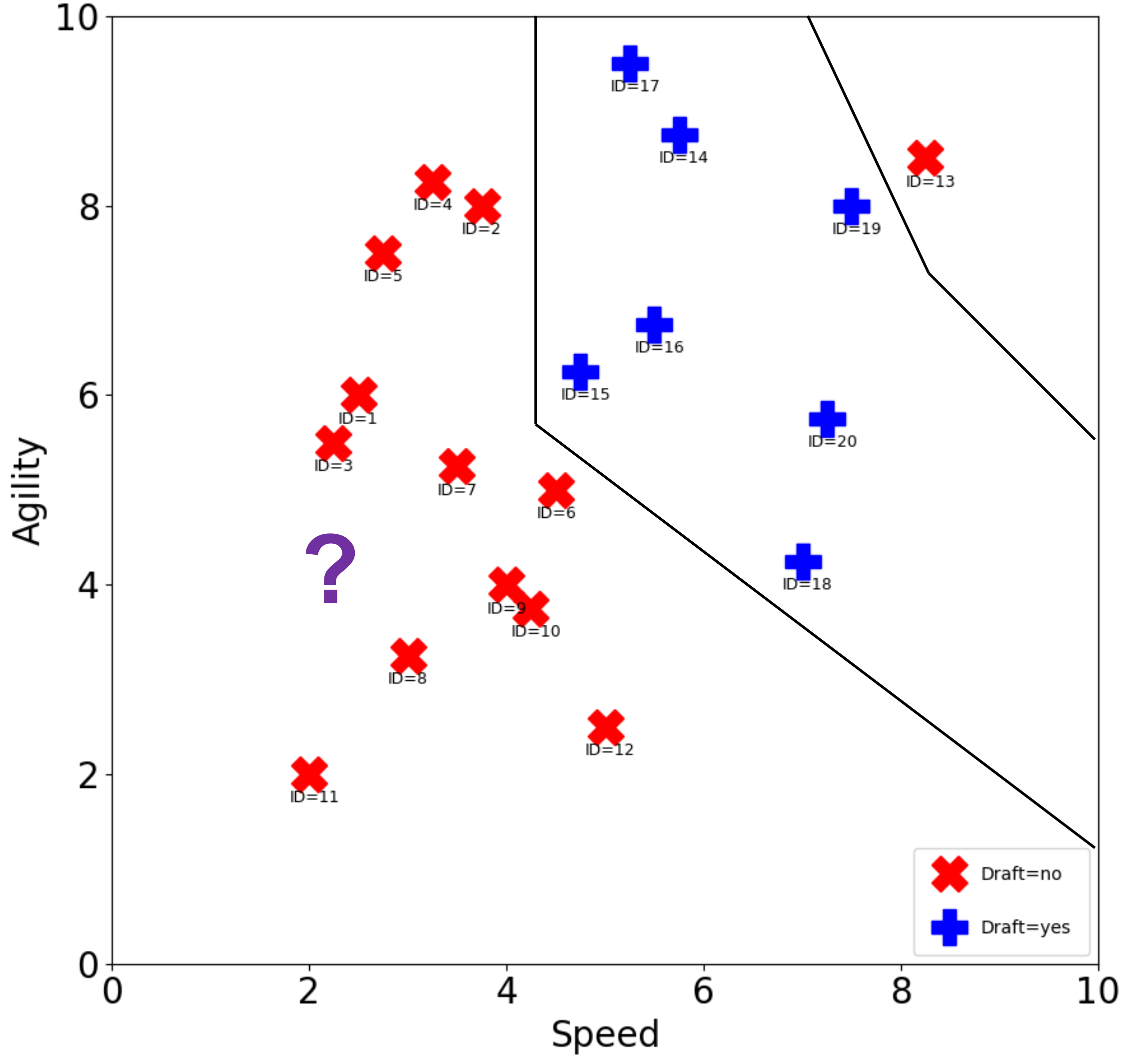
    Speed - continuous variable

    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**: $metric(\boldsymbol{a}, \boldsymbol{b}) = 0 \ \Leftrightarrow \boldsymbol{a} = \boldsymbol{b}$
3. **Symmetry**: $metric(\boldsymbol{a}, \boldsymbol{b}) = metric(\boldsymbol{b}, \boldsymbol{a})$
4. **Triangular Inequality**: $metric(\boldsymbol{a}, \boldsymbol{b}) \leq metric(\boldsymbol{a}, \boldsymbol{c}) + metric(\boldsymbol{b}, \boldsymbol{c})$

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

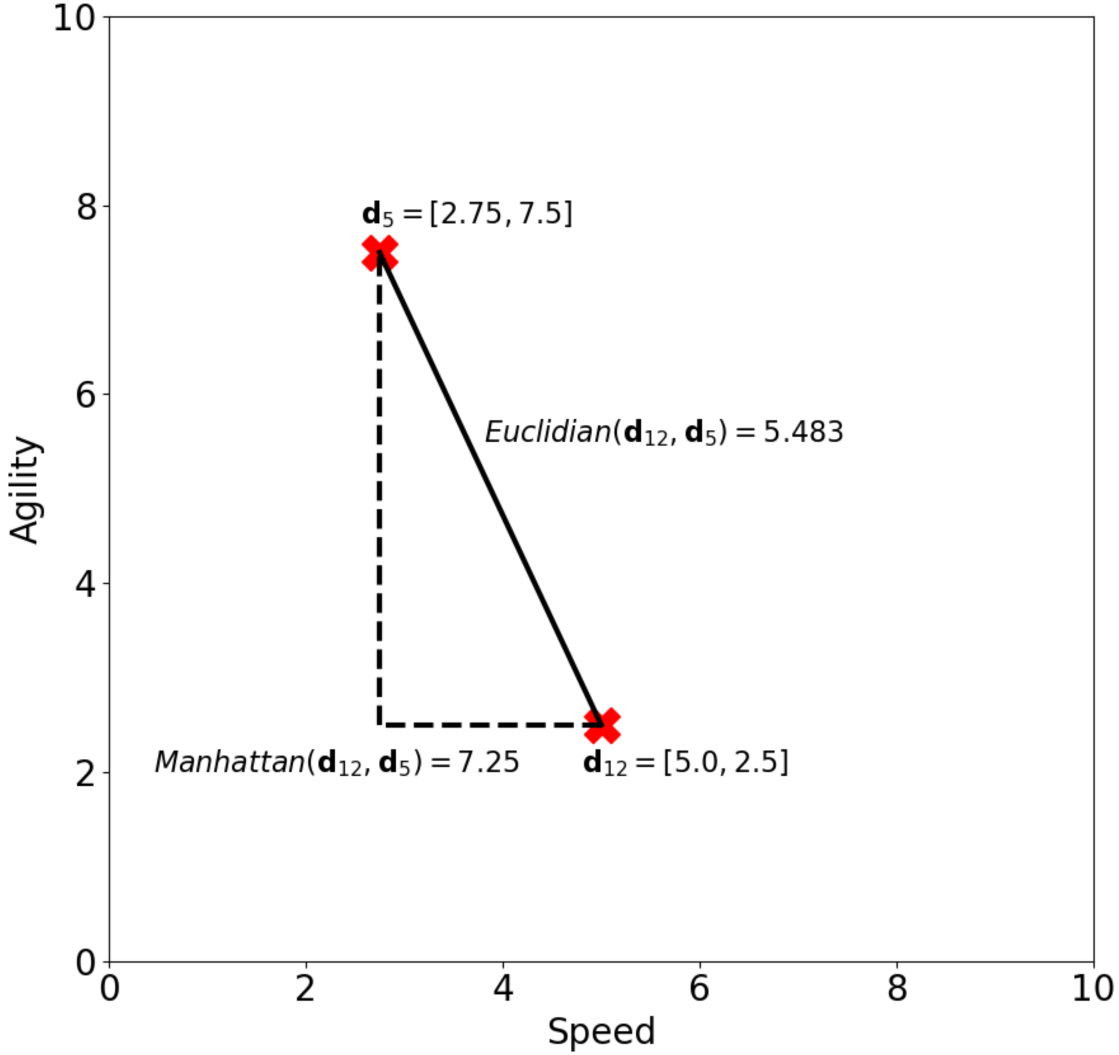
Example: calculating distance

Calculate distance between

$\boldsymbol{d}_{12} = [5.00, 2.50]$ and $\boldsymbol{d}_5 = [2.75, 7.50]$

$$Euclidean(\boldsymbol{d}_{12}, \boldsymbol{d}_5)$$
$$= \sqrt{(5.00 - 2.75)^2 + (2.50 - 7.50)^2}$$
$$= 5.483$$

$$Manhattan(\boldsymbol{d}_{12}, \boldsymbol{d}_5)$$
$$= abs(5.00 - 2.75) + abs(2.50 - 7.50)$$
$$= 7.25$$



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

Similarity-based learning

Part 4:

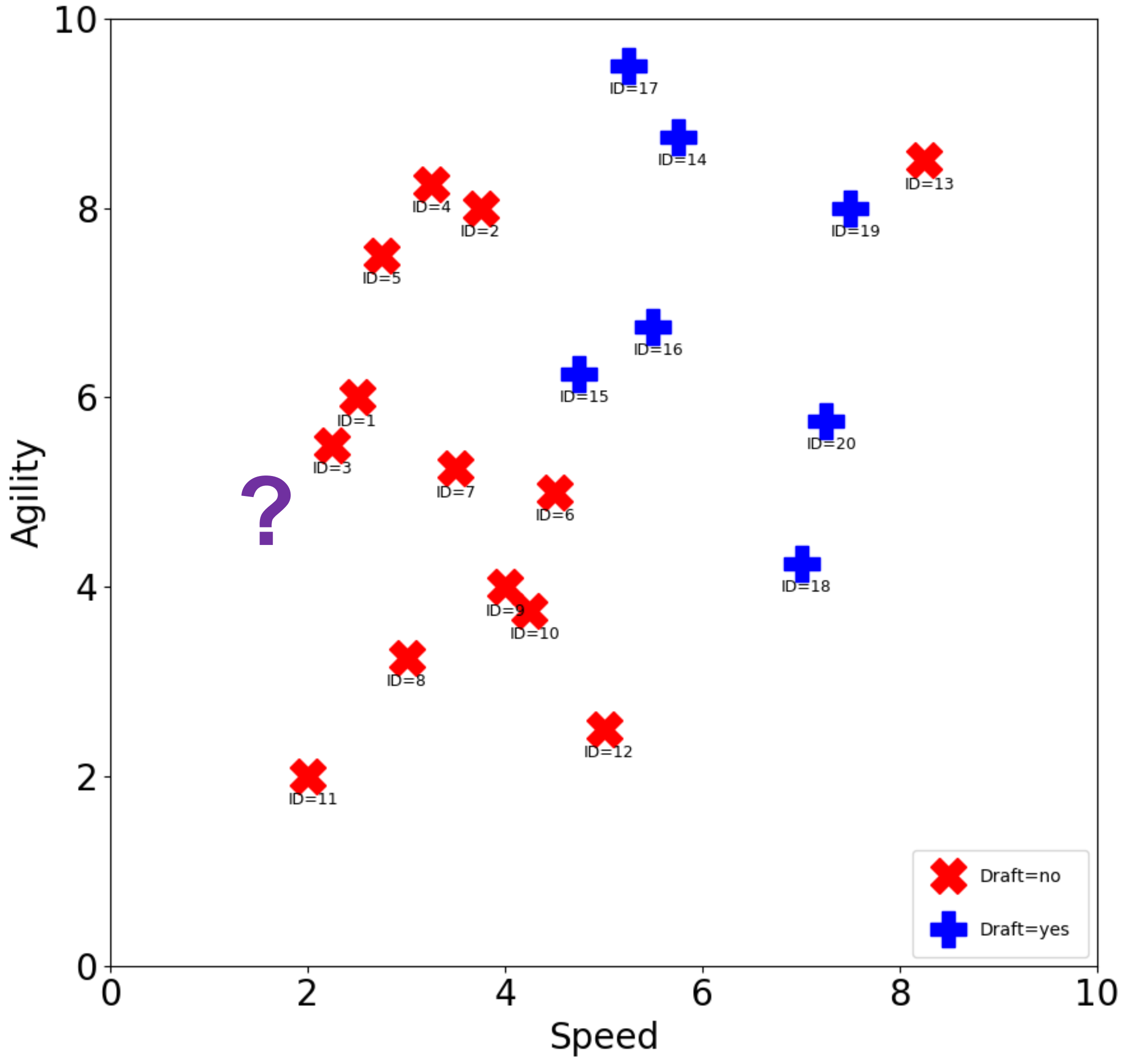
The Nearest Neighbour algorithm

The Nearest Neighbour algorithm

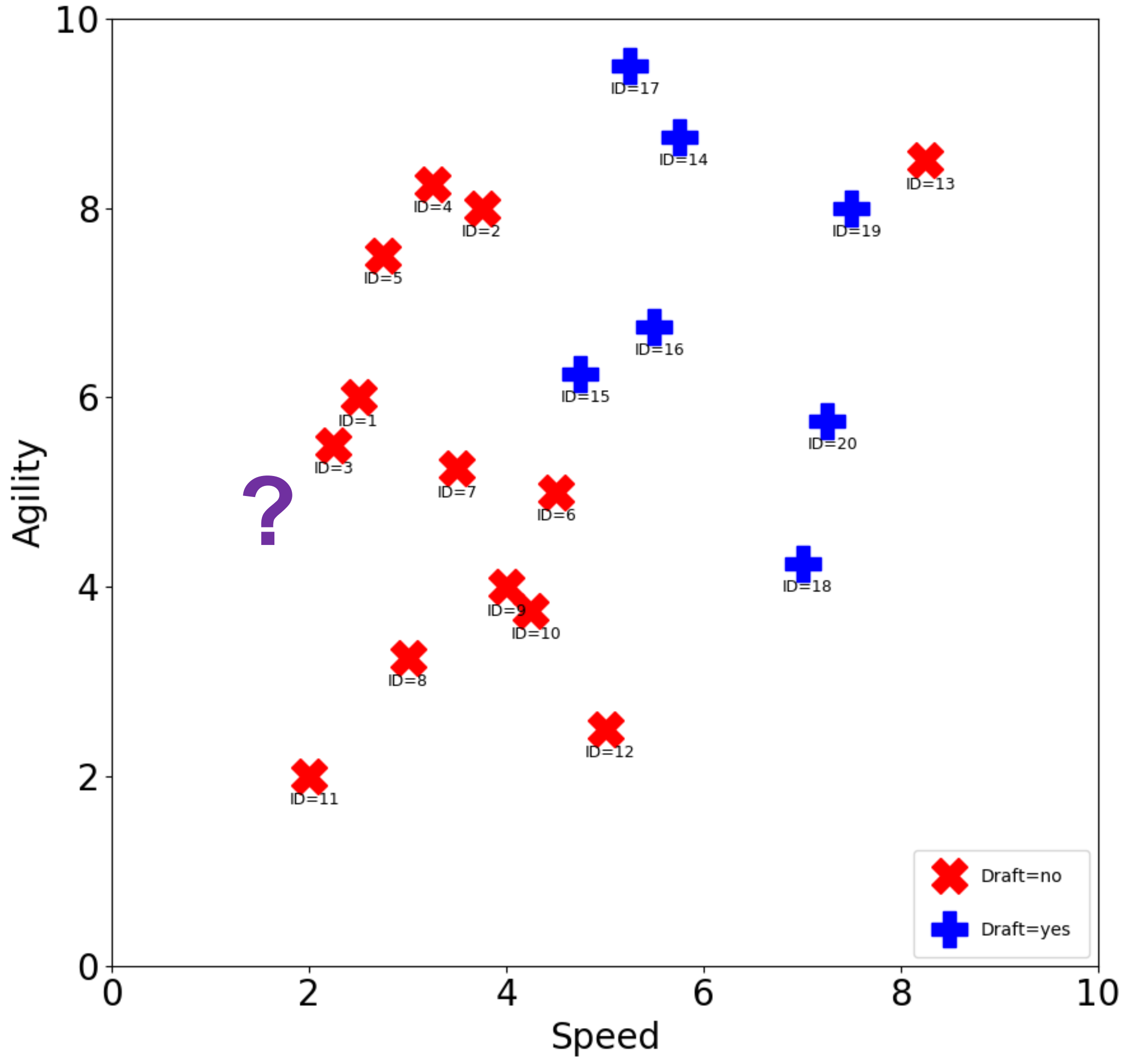
- 1-Nearest Neighbour algorithm:
  - Simplest similarity-based/instance-based method
  - No real training phase: just store the training cases
  - Given a query case with value to be predicted, compute its distance from all stored instances
  - Choose the nearest one; assign the test case to have the 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)

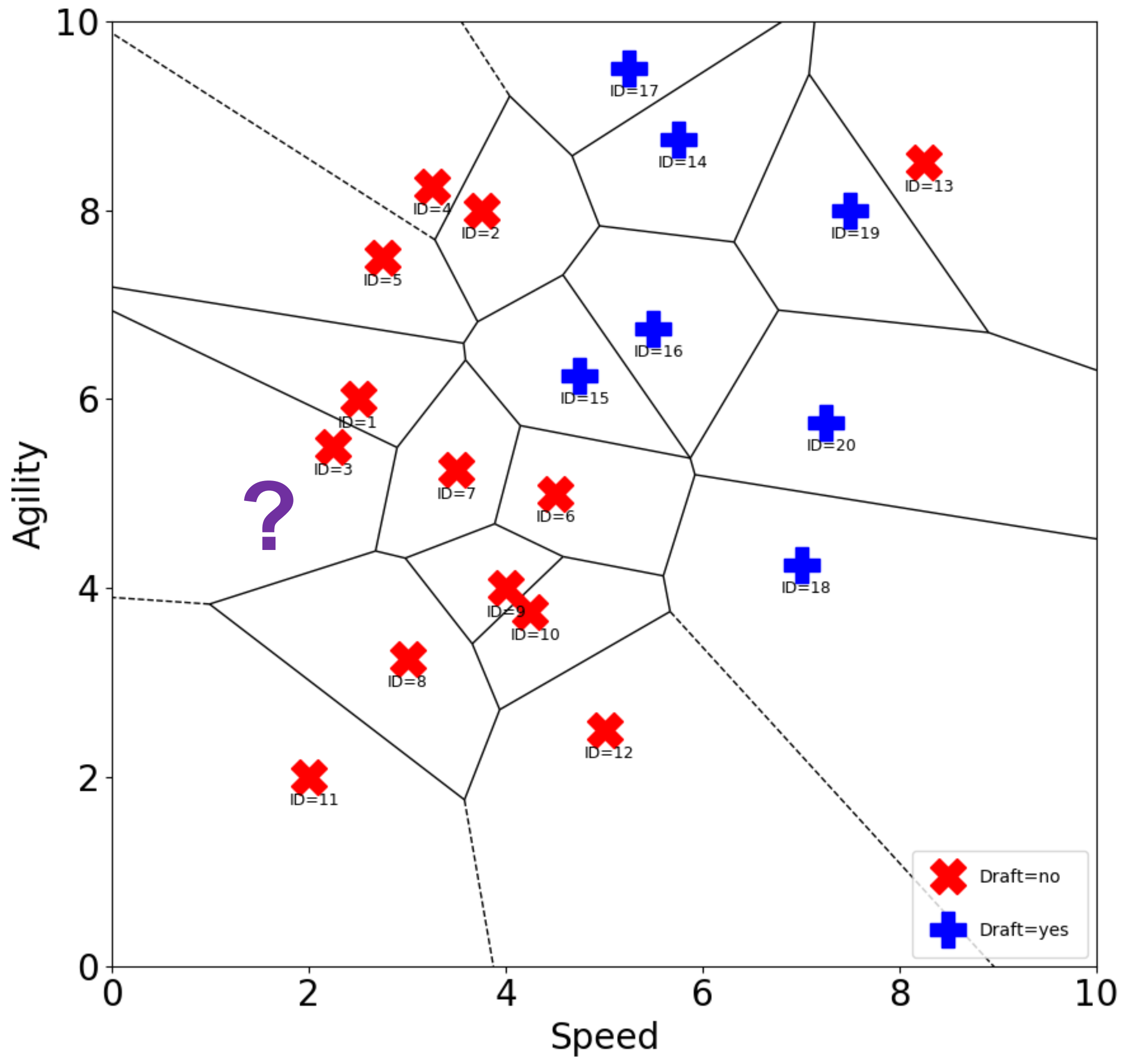
Visualising decision boundaries

- 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

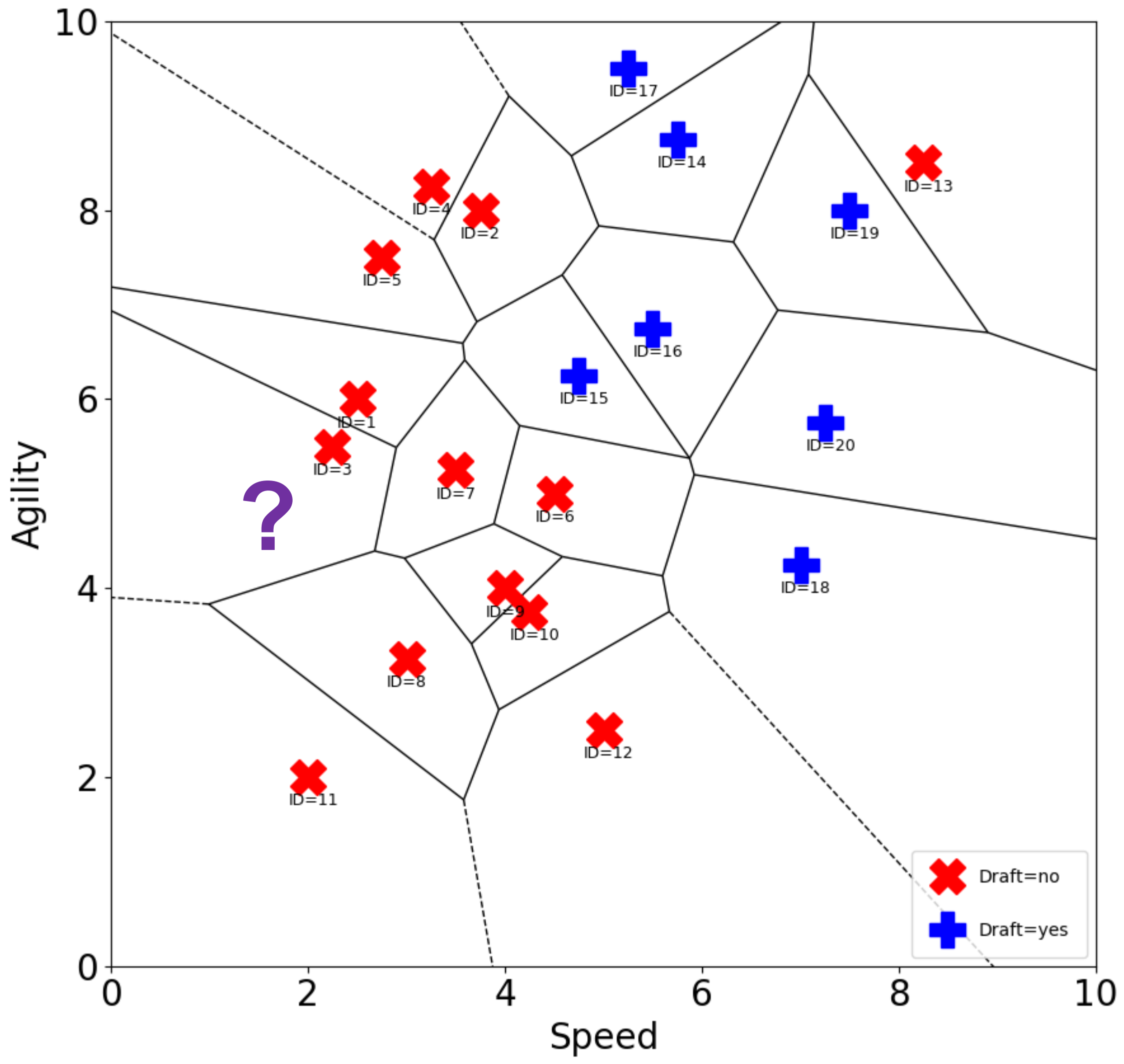


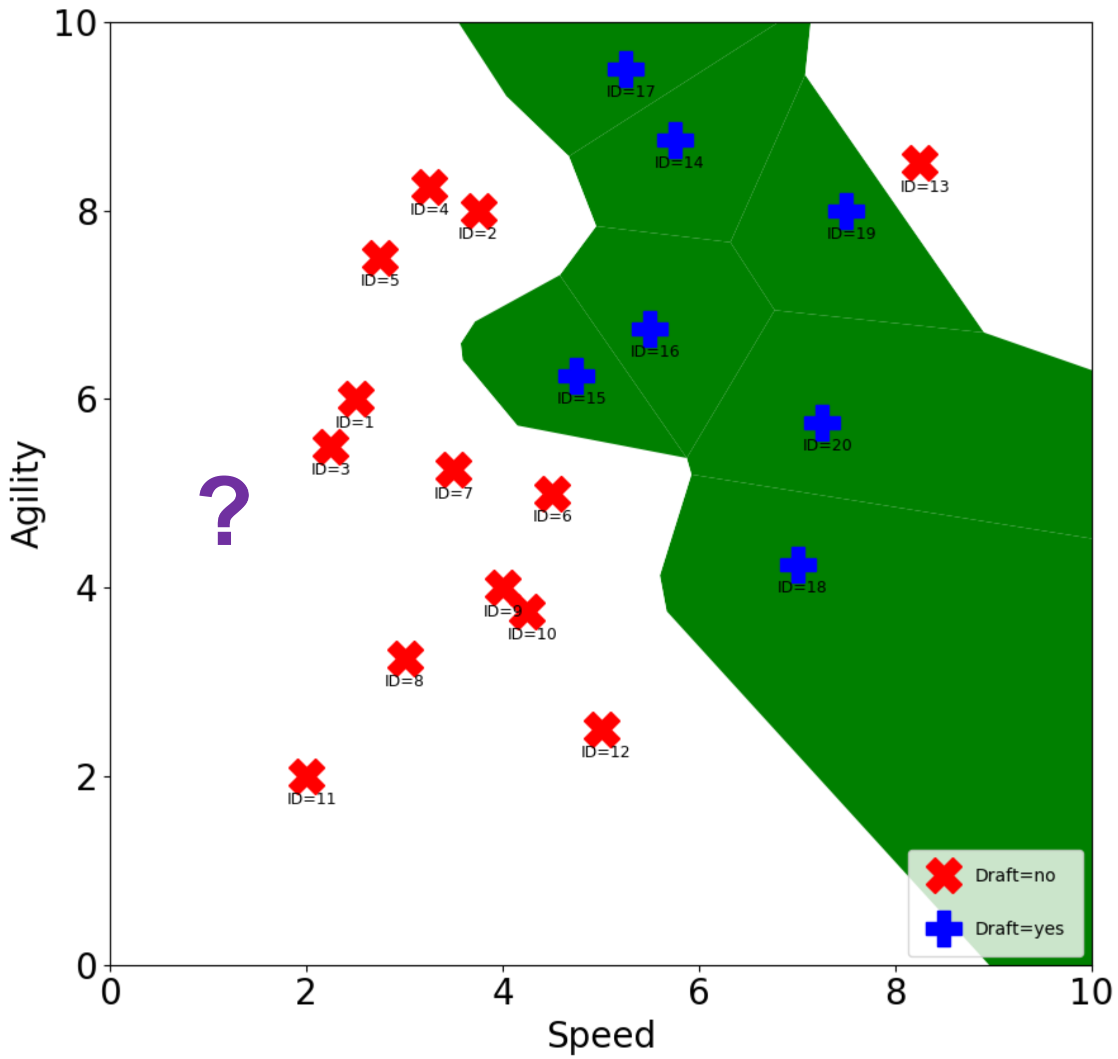
Voronoi tessellation





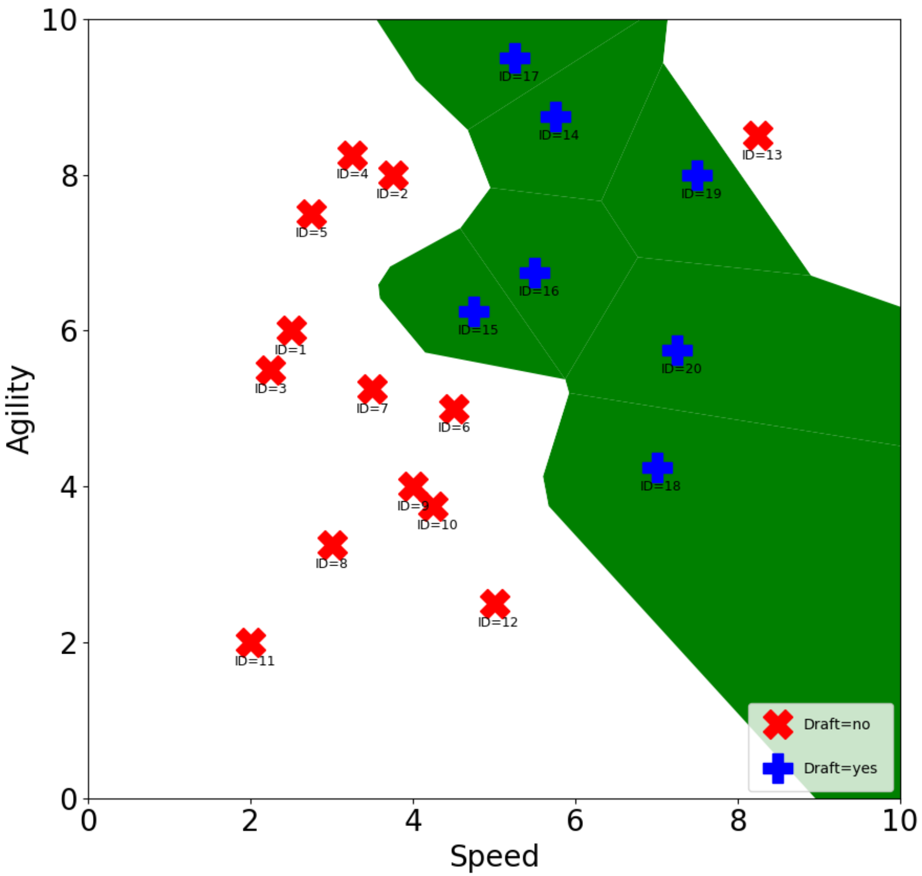
1-NN Decision boundary from Voronoi tessellation

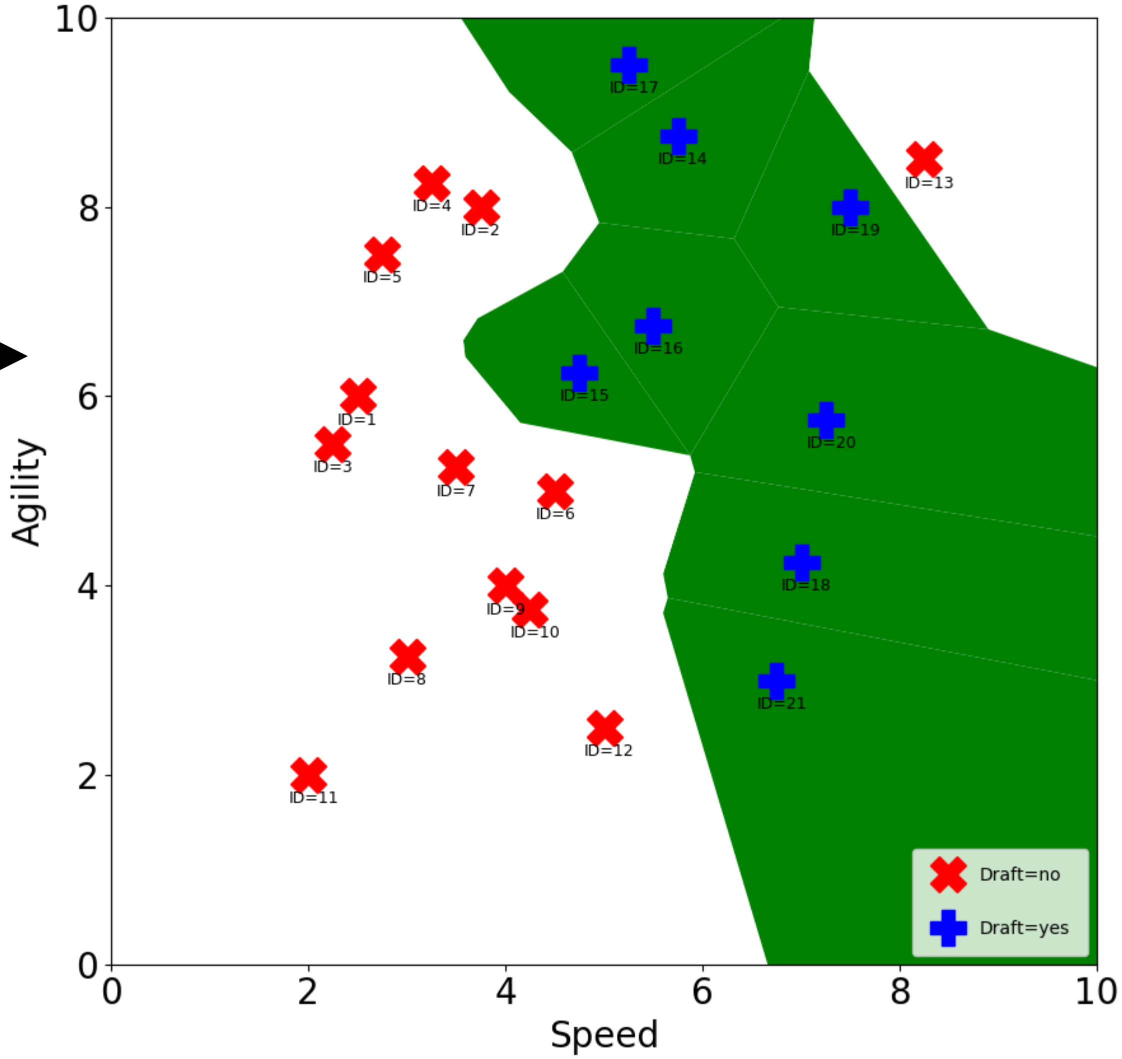




Effect of adding more training data

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

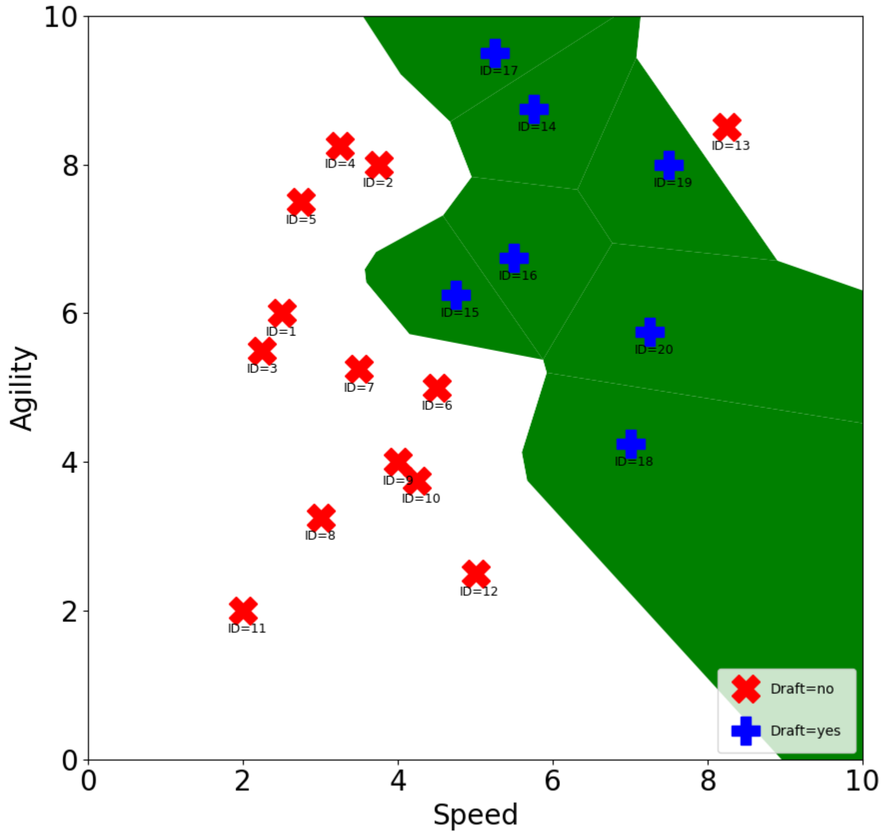
Part 5:
The k Nearest Neighbours algorithm

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)

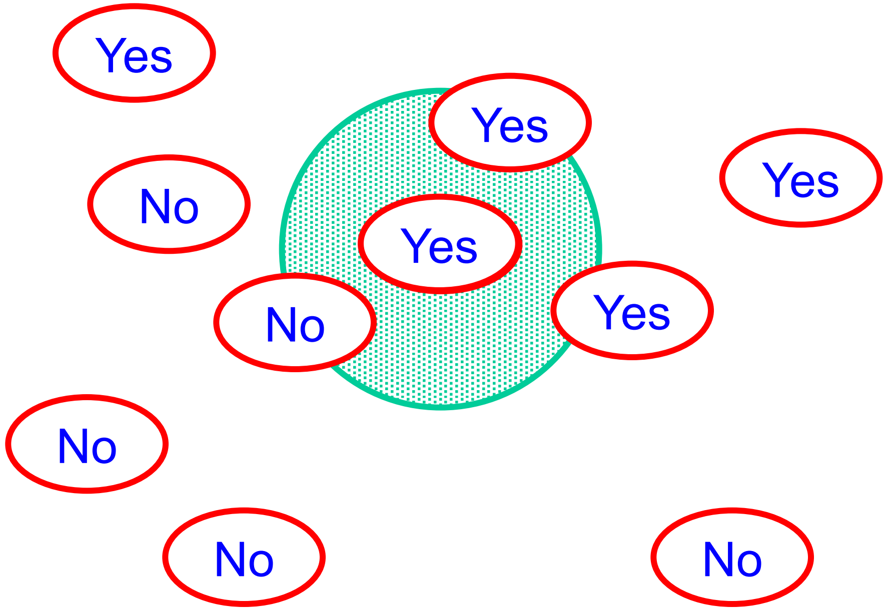
    Give the query case a label based on the similar one(s)



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



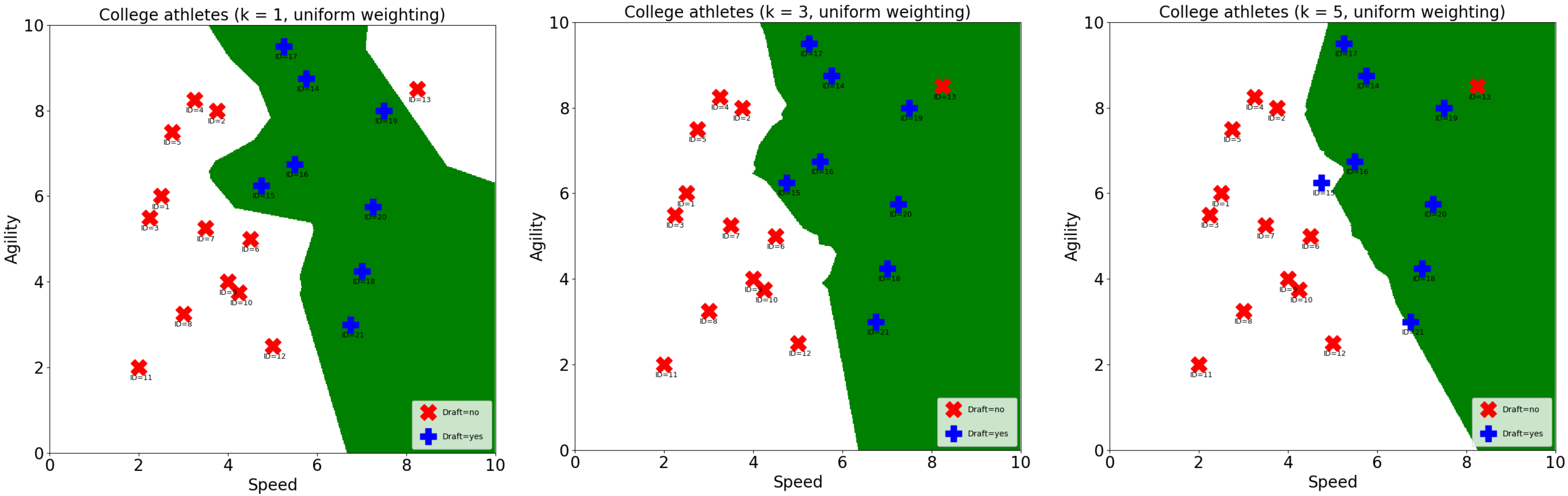
This visualisation assumes Euclidean distance

3 nearest neighbours vote:
Decision is Yes

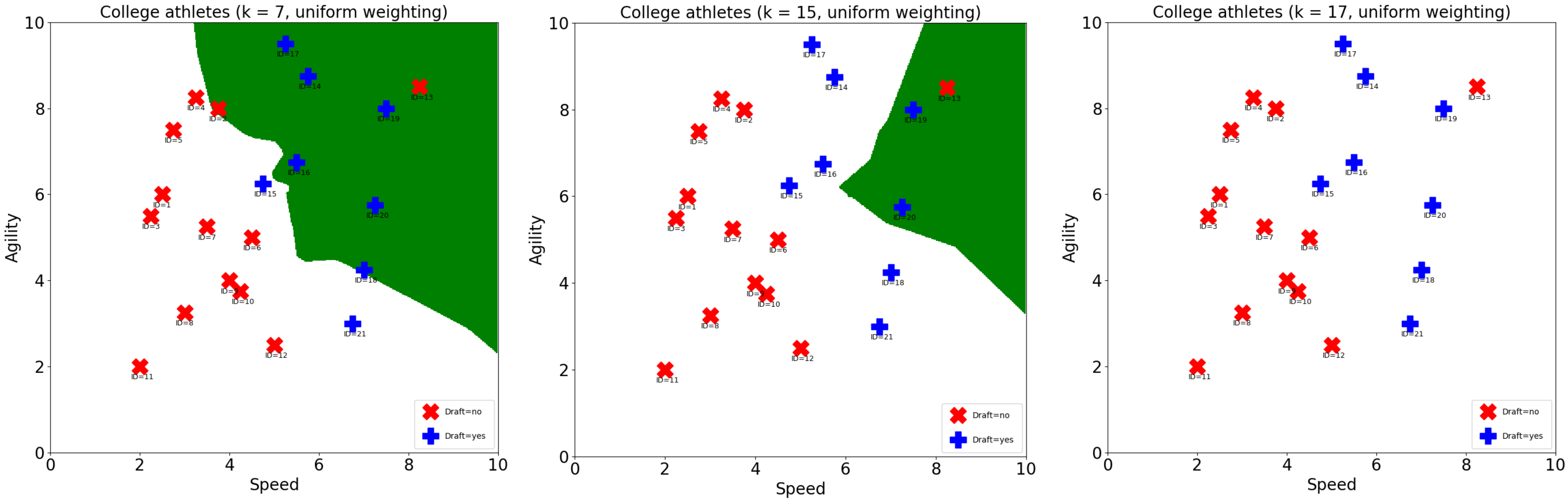
Choosing a value for *k*

- 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)

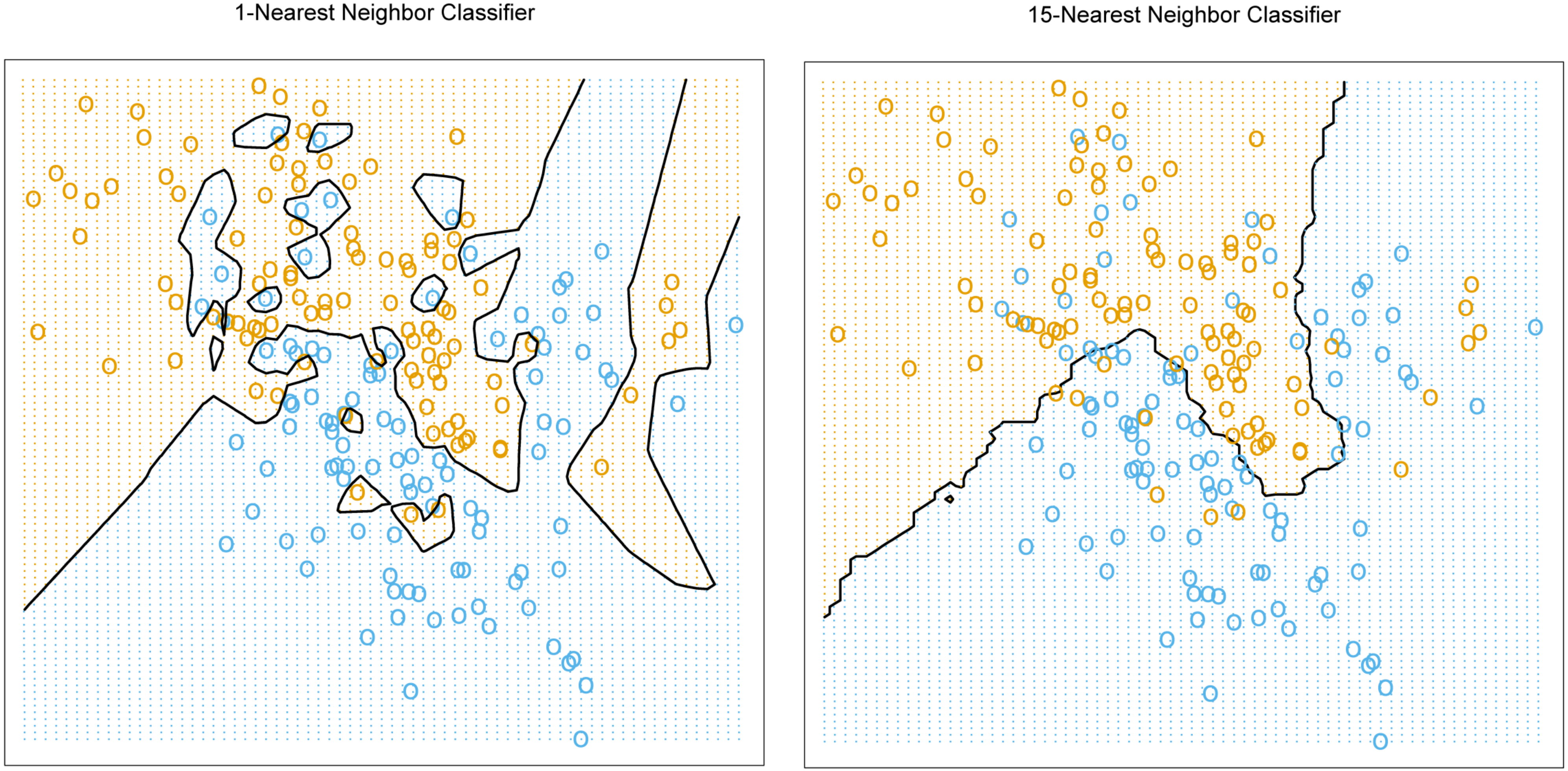
Effect of increasing k



Effect of increasing k



Smoothing Effect of k



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

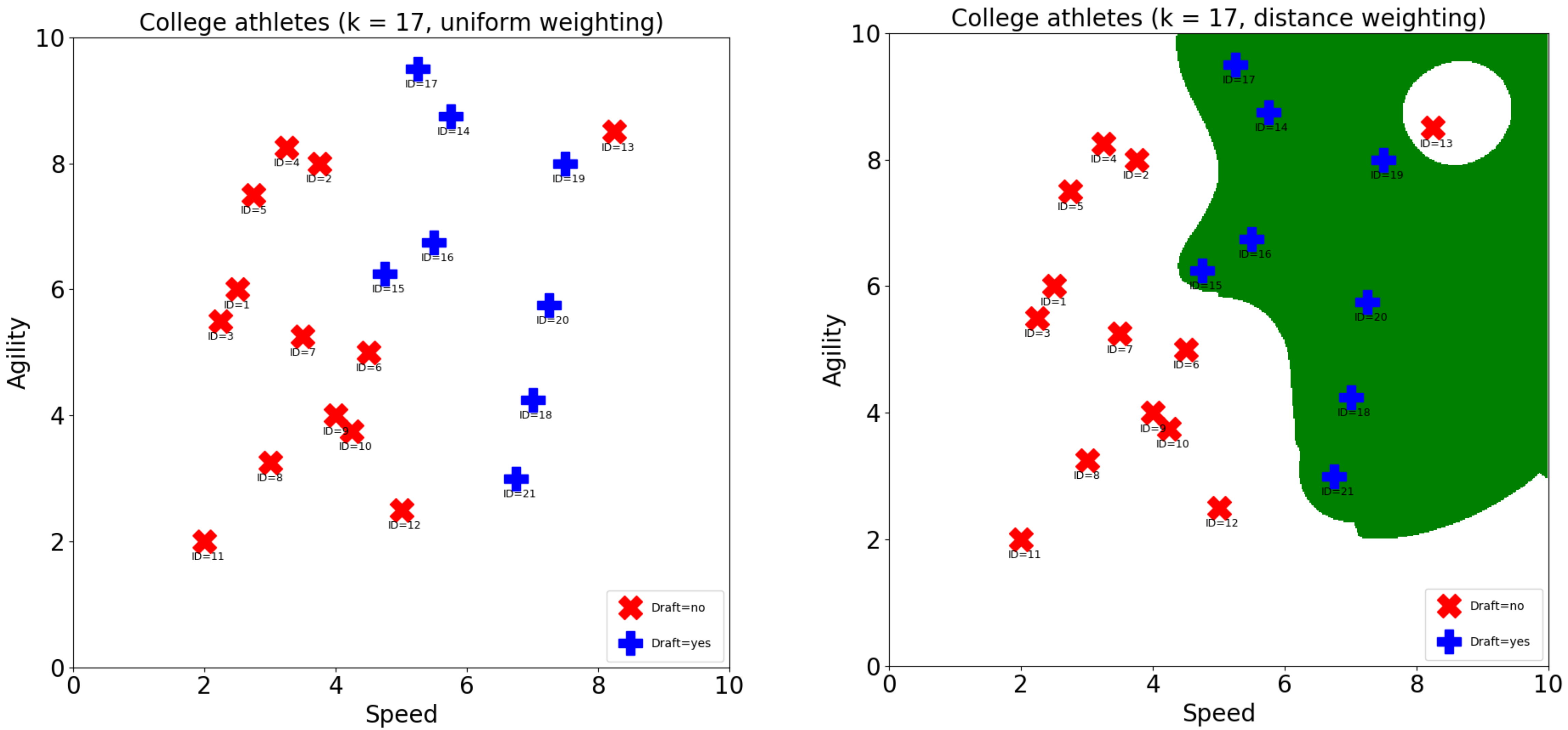
Aside: Underfitting on Same Data

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 $k = |\textit{all training cases}|$

Effect of distance weighting



Recap of todays lecture

- Explained what instance-based learning is
- Distinguished between *lazy* and *eager* learning
- Described operation of k-Nearest Neighbours for classification and regression