COMP47460

Nearest Neighbour Classifiers

Aonghus Lawlor
Derek Greene

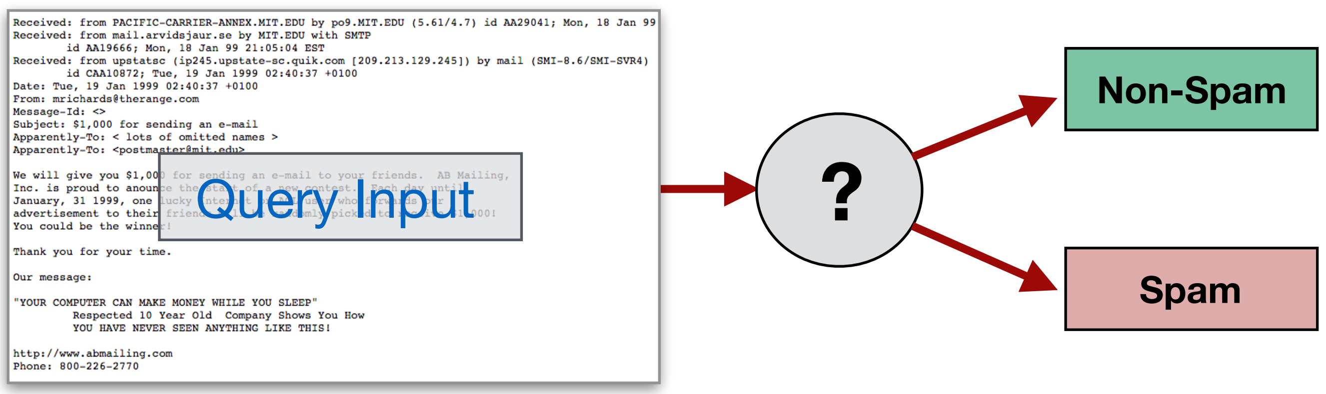
School of Computer Science
Autumn 2018

Overview

- Eager v Lazy Classification Strategies
- Distance-based Models
- Feature Spaces
- Measuring Distance
- Data Normalisation
- Nearest Neighbours
- *k*-Nearest Neighbour Classifier (kNN)
- Weighted kNN
- kNN in Weka

Reminder: Classification

- Supervised Learning: Algorithm that learns a function from manually-labelled training examples.
- Classification: Training examples, usually represented by a set of descriptive features, help decide the *class* to which a new unseen query input belongs.
- Binary Classification: Assign one of two possible target class labels to the new query input.



- Multiclass Classification: Assign one of *M*>2 possible target class labels to the new query input.

Eager v Lazy Classifiers

- Eager Learning Classification Strategy
    - Classifier builds a full model during an initial training phase, to use later when new query examples arrive.
    - More offline setup work, less work at run-time.
    - Generalise before seeing the query example.
- Lazy Learning Classification Strategy
    - Classifier keeps all the training examples for later use.
    - Little work is done offline, wait for new query examples.
    - Focus on the local space around the examples.

- Distance-based Models: Many learning algorithms are based on generalising from training data to unseen data by exploiting the distances (or similarities) between the two.

Example: Athlete Selection

- Dataset of performance ratings for 20 college athletes.
- Describe each athlete by 2 continuous features: *speed*, *agility.* Binary class label indicates whether or not they were *selected* for the college team ('Yes' or 'No').

Athlete	*Speed*	*Agility*	*Selected*
x1	2.50	6.00	No
x2	3.75	8.00	No
x3	2.25	5.50	No
x4	3.25	8.25	No
x5	2.75	7.50	No
x6	4.50	5.00	No
x7	3.50	5.25	No
x8	3.00	3.25	No
x9	4.00	4.00	No
x10	4.25	3.75	No

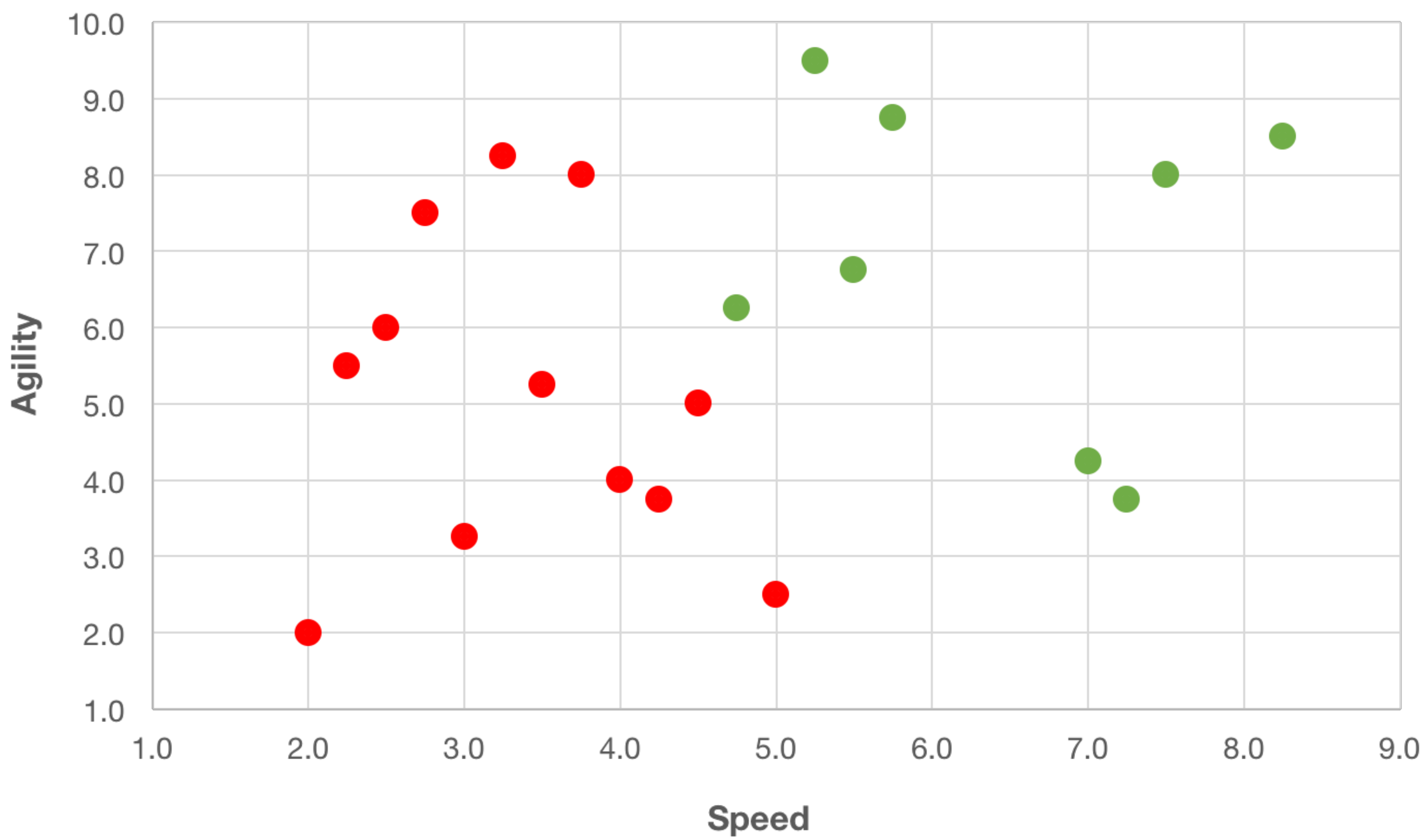
Athlete	*Speed*	*Agility*	*Selected*
x11	2.00	2.00	No
x12	5.00	2.50	No
x13	8.25	8.50	Yes
x14	5.75	8.75	Yes
x15	4.75	6.25	Yes
x16	5.50	6.75	Yes
x17	5.25	9.50	Yes
x18	7.00	4.25	Yes
x19	7.50	8.00	Yes
x20	7.25	3.75	Yes

Q. Will athlete **q** be selected?

Athlete	*Speed*	*Agility*	*Selected*
q	3.00	8.00	???

Feature Spaces

We can use the feature values to visually position the 20 athletes in a 2-dimensional coordinate space (i.e. *agility* versus *speed*):



2 features describing each example (agility & speed)

➡ 2 coordinate dimensions for measuring similarity/ distance

Features Space: A *D*-dimensional coordinate space used to represent the input examples for a given problem, with one coordinate for each descriptive feature.

Measuring Distance

- Distance function: A suitable function to measure how distant (or similar) two input examples are from one another are in some *D*-dimensional feature space.

- Local distance function: Measure the distance between two examples based on a single feature.

Athlete	*Speed*	*Agility*
x1	2.50	6.00
x2	3.75	8.00

  - e.g. what is distance between **x1** and **x2** in terms of *Speed*?
  - e.g. what is distance between **x1** and **x2** in terms of *Agility*?

- Global distance function: Measure the distance between two examples based on the combination of the local distances across all features.
  - e.g. what is distance between **x1** and **x2** based on both *Speed* and *Agility*?

Measuring Distance

- Overlap function: Simplest local distance measure. Returns 0 if the two values for a feature are equal and 1 otherwise. Generally suitable for categorical data.

Athlete	*Gender*	*Nationality*
x1	Female	Irish
x2	Male	Irish
x3	Male	Italian

For feature *Gender*

$d_g(x1,x2) = 1$
$d_g(x1,x3) = 1$
$d_g(x2,x3) = 0$

For feature *Nationality*

$d_n(x1,x2) = 0$
$d_n(x1,x3) = 1$
$d_n(x2,x3) = 1$

- Hamming distance: Global distance function which is the sum of the overlap differences across all features - i.e. number of features on which two examples disagree.

$d(x1,x2) = 1 + 0 = 1$
$d(x1,x3) = 1 + 1 = 2$
$d(x2,x3) = 0 + 1 = 1$

Overlap distance for *Gender* +
Overlap distance for *Nationality*

Measuring Distance

Athlete	Speed	Agility
x1	2.50	6.00
x2	3.75	8.00
x3	2.25	5.50

- Absolute difference: For numeric data, we can calculate absolute value of the difference between values for a feature.

For feature *Speed*

$d_s(x1,x2) = |2.50-3.75| = 1.25$
$d_s(x1,x3) = |2.50-2.25| = 0.25$
$d_s(x2,x3) = |3.75-2.25| = 1.5$

For feature *Agility*

$d_s(x1,x2) = |6.0-8.0| = 2.0$
$d_s(x1,x3) = |6.0-5.5| = 0.5$
$d_s(x2,x3) = |8.0-5.5| = 2.5$

- Again we can compute a global distance between two examples by summing the local distances over all features.

$d(x1,x2) = 1.25 + 2.0 = 3.25$
$d(x1,x3) = 0.25 + 0.5 = 0.75$
$d(x2,x3) = 1.5 + 2.5 = 4.0$

Absolute difference for *Speed* +
Absolute difference for *Agility*

- For *ordinal features*, calculate the absolute value of the difference between the two positions in the ordered list of possible values.

e.g. Ordinal Feature *Dosage*:
{Low,Medium,High} = {1, 2, 3}

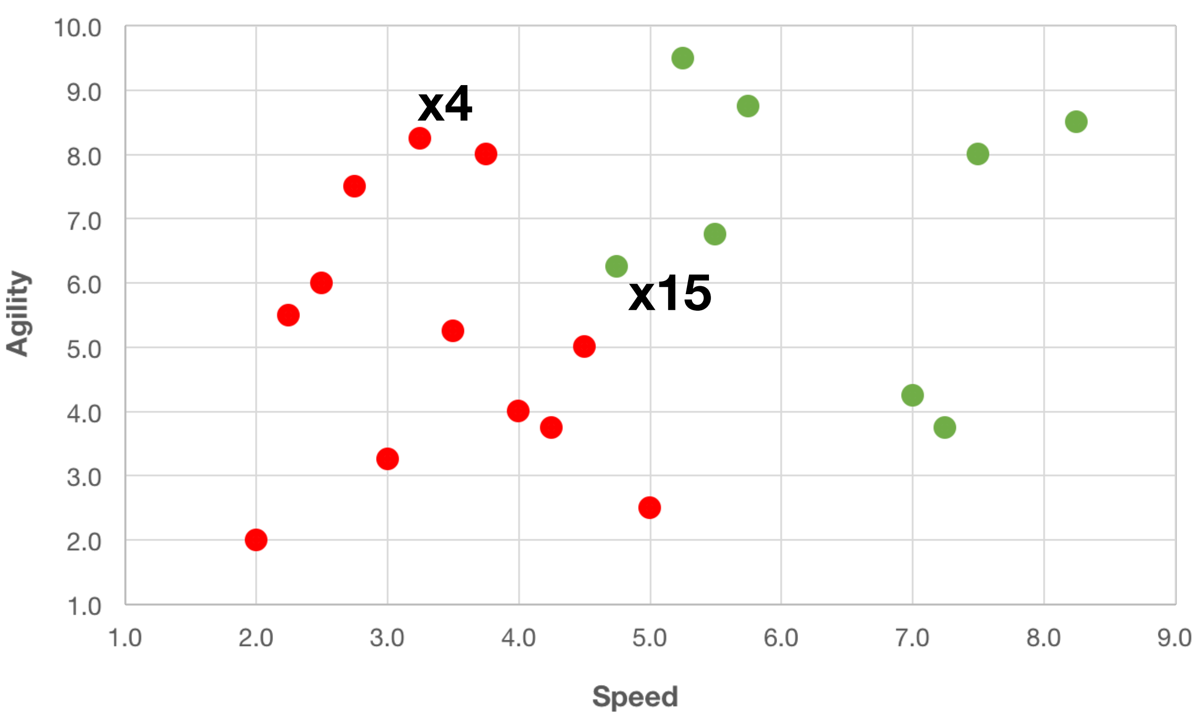
```
diff(Low,High) = |1-3| = 2
diff(Medium,Low) = |2-1| = 1
diff(High,High) = |3-3| = 0
```

Measuring Distance

- Euclidean distance: Most common measure used to quantify distance between two examples with real-valued features.
- The "straight line" distance between two points in a Euclidean coordinate space - e.g. a feature space.
- Calculated as square root of sum of squared differences for each feature *f* representing a pair of examples.

$$\mathrm{ED}(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{f \in F} (q_f - p_f)^2}$$

Athlete	*Speed*	*Agility*
x4	3.25	8.25
x15	4.75	6.25



$$ED(x4, x15) = \sqrt{(3.25 - 4.75)^2 + (8.25 - 6.25)^2} = \sqrt{6.25} = 2.5$$

Heterogeneous Distance Functions

- In many datasets, the features associated with examples will have different types (e.g. continuous, categorical, ordinal etc).
- We can create a global measure from different local distance functions, using an appropriate function for each feature.

Athlete	*Speed*	*Agility*	*Gender*	*Nationality*
x1	2.50	6.00	Female	Irish
x2	3.75	8.00	Male	Irish
x3	2.25	5.50	Male	Italian

Use absolute difference for continuous features *Speed* & *Agility*

Use overlap for categorical features *Gender* & *Nationality*

```
d(x1,x2) = 1.25 + 2.0 + 1 + 0 = 4.25
d(x1,x3) = 0.25 + 0.5 + 1 + 1 = 2.75
d(x2,x3) = 1.5 + 2.5 + 0 + 1 = 5.0
```

Global distance calculated as sum over individual local distances

- Often domain expertise is required to choose an appropriate distance function for a particular dataset.

Data Normalisation

- Numeric features often have different ranges, which can skew certain distance functions.
- So that all features have similar range, we apply *feature normalisation*.
- Min-max normalisation:
  Use min and max values for a given feature to rescale to the range [0,1]
- Example: Feature *Age*

Example	*Age*
x1	24
x2	19
x3	50
x4	40
x5	23
x6	68
x7	45
x8	33
x9	80
x10	58

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

$$\min(x) = 19$$

$$\max(x) = 80$$

$$\max(x) - \min(x) = 61$$

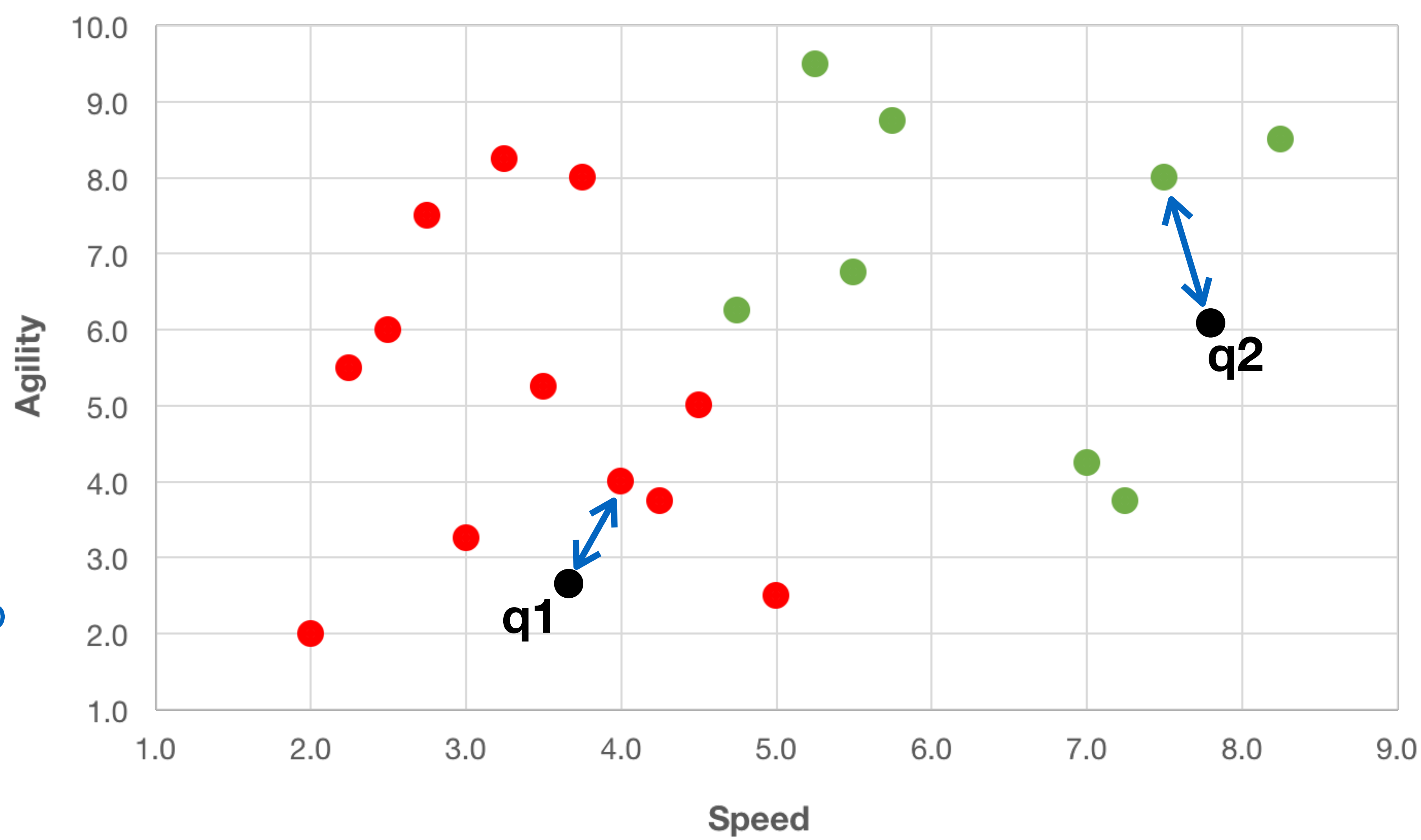
Age (Non-normalised)	24	19	50	40	23	68	45	33	80	58
Age (Normalised)	0.08	0.00	0.51	0.34	0.07	0.80	0.43	0.23	1.00	0.64

Nearest Neighbour Classifier

Lazy Learning approach: Do not build a model for the data. Identify most similar previous example(s) from the training set for which a label has already been assigned, using some distance function.

Nearest neighbour rule (1NN): For a new query input **q**, find a single labelled example **x** closest to **q**, and assign **q** the same label as **x**.

Assign **q1** to 'No' class

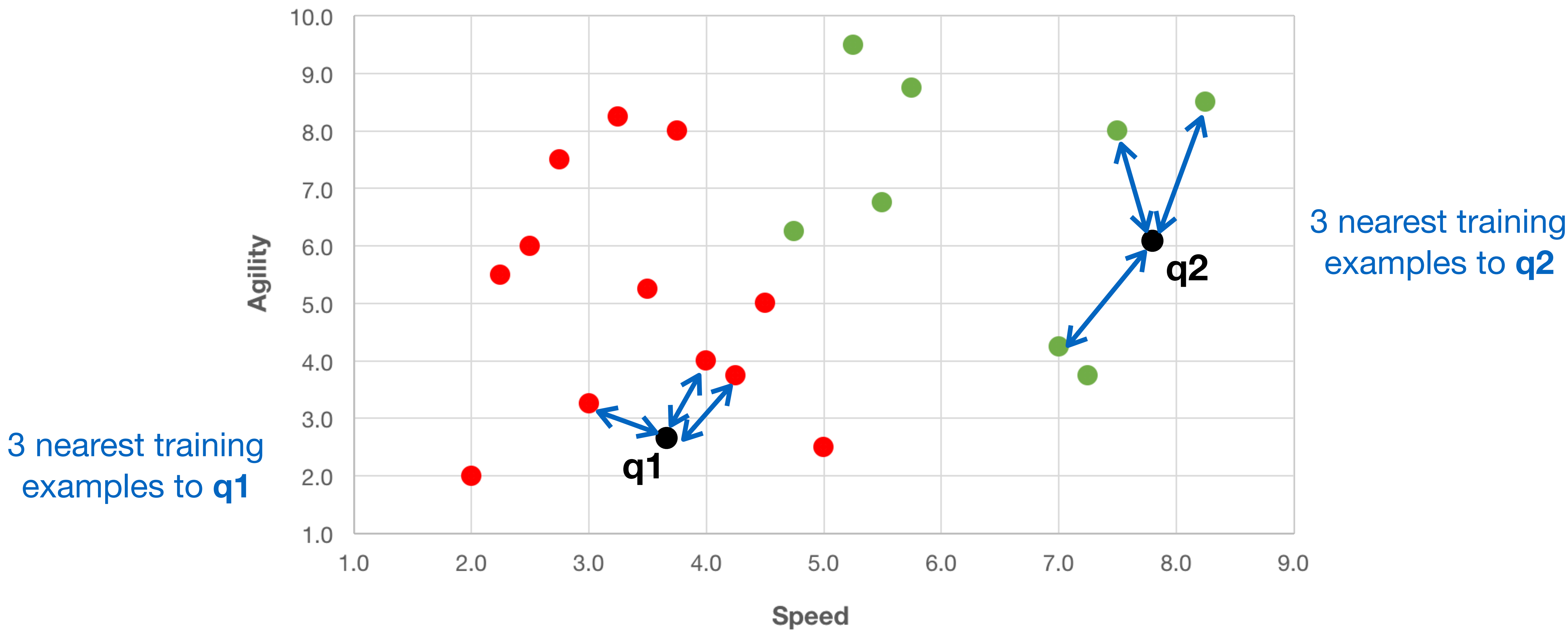


Assign **q2** to 'Yes' class

k-Nearest Neighbour Classifier

k-Nearest neighbours (kNN): The NN approach naturally generalises to the case where we use *k* nearest neighbours from the training set to assign a label to a new query input.

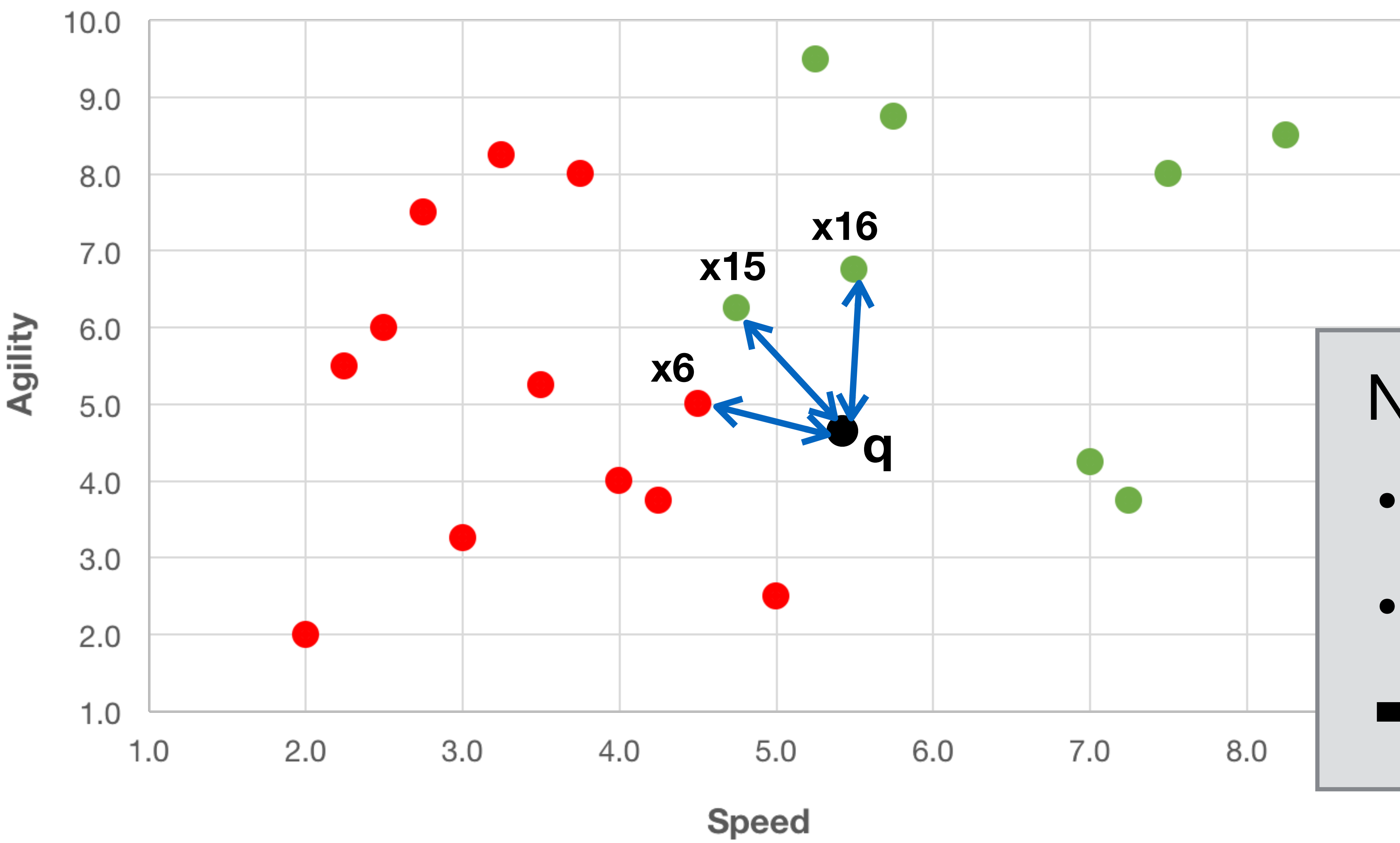
Example: For new query inputs, calculate distance to all training examples. Find $k=3$ nearest examples (i.e. with smallest distances).



k-Nearest Neighbour Classifier

Majority voting: The decision on a label for a new query example is decided based on the "votes" of its *k* nearest neighbours. The label for the query is the majority label of its neighbours.

Example: Measure distance from **q** to all training examples. Find the *k=3* nearest examples, and use their labels as votes.



Neighbour counts

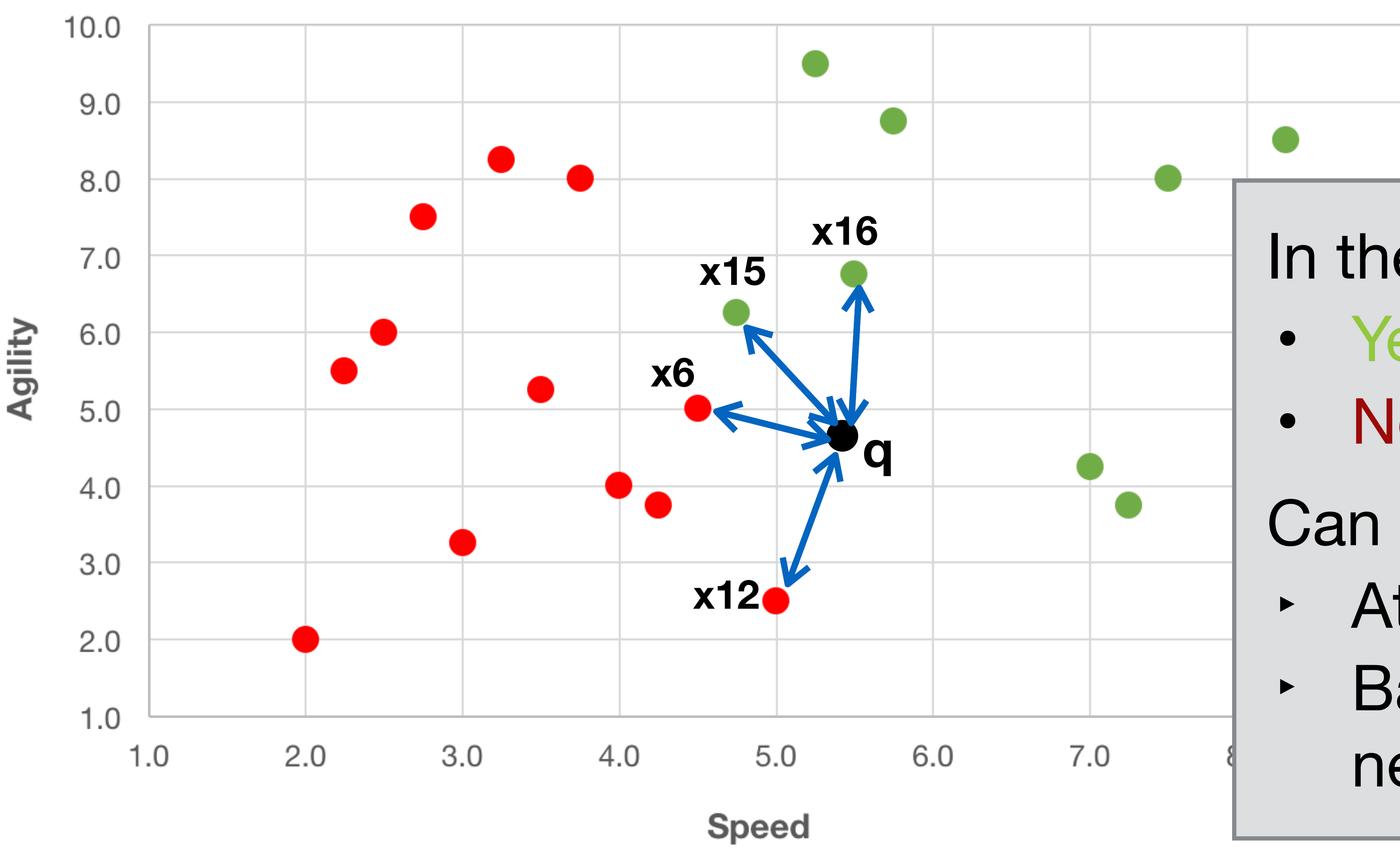
- Yes = 2 votes
- No = 1 vote

➡ Majority says Yes!

k-Nearest Neighbour Classifier

Majority voting: The decision on a label for a new query example is decided based on the “votes” of its *k* nearest neighbours. The label for the query is the majority label of its neighbours.

Example: Measure distance from **q** to all training examples. Find the *k=4* nearest examples, and use their labels as votes.



In the case that…

- Yes = 2 votes
- No = 2 votes

Can break ties…

- At random
- Based on sum of neighbour distances

Example: kNN Classification (k=3)

- Training set of 20 athletes - 8 labelled as 'Yes', 12 as 'No'.
- Each athlete described by 2 continuous features: *Speed*, *Agility* Euclidean distance would be an appropriate distance function.

Athlete	*Speed*	*Agility*	*Selected*
x1	2.50	6.00	No
x2	3.75	8.00	No
x3	2.25	5.50	No
x4	3.25	8.25	No
x5	2.75	7.50	No
x6	4.50	5.00	No
x7	3.50	5.25	No
x8	3.00	3.25	No
x9	4.00	4.00	No
x10	4.25	3.75	No

Athlete	*Speed*	*Agility*	*Selected*
x11	2.00	2.00	No
x12	5.00	2.50	No
x13	8.25	8.50	Yes
x14	5.75	8.75	Yes
x15	4.75	6.25	Yes
x16	5.50	6.75	Yes
x17	5.25	9.50	Yes
x18	7.00	4.25	Yes
x19	7.50	8.00	Yes
x20	7.25	3.75	Yes

Will a new input example **q** be labelled as 'Yes' or 'No'?

Athlete	*Speed*	*Agility*	*Selected*
q	5.00	8.00	???

Example: kNN Classification (k=3)

- Measure distance between **q** and all 20 training examples.

Athlete	*Speed*	*Agility*	*Selected*	*Distance*
x1	2.50	6.00	No	3.201562
x2	3.75	8.00	No	1.250000
x3	2.25	5.50	No	3.716517
x4	3.25	8.25	No	1.767767
x5	2.75	7.50	No	2.304886
x6	4.50	5.00	No	3.041381
x7	3.50	5.25	No	3.132491
x8	3.00	3.25	No	5.153882
x9	4.00	4.00	No	4.123106
x10	4.25	3.75	No	4.315669

Athlete	*Speed*	*Agility*	*Selected*	*Distance*
x11	2.00	2.00	No	6.708204
x12	5.00	2.50	No	5.500000
x13	8.25	8.50	Yes	3.288237
x14	5.75	8.75	Yes	1.060660
x15	4.75	6.25	Yes	1.767767
x16	5.50	6.75	Yes	1.346291
x17	5.25	9.50	Yes	1.520691
x18	7.00	4.25	Yes	4.250000
x19	7.50	8.00	Yes	2.500000
x20	7.25	3.75	Yes	4.808846

- Rank the training examples and identify set of 3 examples with the smallest distances.

Athlete	*Speed*	*Agility*	*Selected*	*Distance*
x14	5.75	8.75	Yes	1.060660
x2	3.75	8.00	No	1.250000
x16	5.50	6.75	Yes	1.346291

- Yes = 2 votes
- No = 1 vote

➡ Majority says Yes,
so assign label Yes to **q**

Weighted kNN

- Weighted voting: In this approach, some training examples have a higher weight than others.
- Instead of using a binary vote of 1 for each nearest neighbour, typically closer neighbours get higher votes when deciding on the predicted label for a query example.
- Inverse distance-weighted voting: Simplest strategy is to take a neighbour's vote to be the inverse of their distance from the query (i.e. 1/Distance). We then sum over the weights for each class.

$$d(q, x14) = 1.060660$$

$$\rightarrow \; weight(x14) = \frac{1}{d(q, x14)} = \frac{1}{1.060660} = 0.942809$$

$$d(q, x2) = 1.250000$$

$$\rightarrow \; weight(x2) = \frac{1}{d(q, x2)} = \frac{1}{1.250000} = 0.8$$

Example: Weighted kNN (k=3)

- Measure distance between **q** and all 20 training examples.

Athlete	*Speed*	*Agility*	*Selected*	*Distance*
x1	2.50	6.00	No	3.201562
x2	3.75	8.00	No	1.250000
x3	2.25	5.50	No	3.716517
x4	3.25	8.25	No	1.767767
x5	2.75	7.50	No	2.304886
x6	4.50	5.00	No	3.041381
x7	3.50	5.25	No	3.132491
x8	3.00	3.25	No	5.153882
x9	4.00	4.00	No	4.123106
x10	4.25	3.75	No	4.315669

Athlete	*Speed*	*Agility*	*Selected*	*Distance*
x11	2.00	2.00	No	6.708204
x12	5.00	2.50	No	5.500000
x13	8.25	8.50	Yes	3.288237
x14	5.75	8.75	Yes	1.060660
x15	4.75	6.25	Yes	1.767767
x16	5.50	6.75	Yes	1.346291
x17	5.25	9.50	Yes	1.520691
x18	7.00	4.25	Yes	4.250000
x19	7.50	8.00	Yes	2.500000
x20	7.25	3.75	Yes	4.808846

- Rank the training examples and identify set of 3 examples with the smallest distances. Assign weights based on 1/Distance, and sum weights for each class.

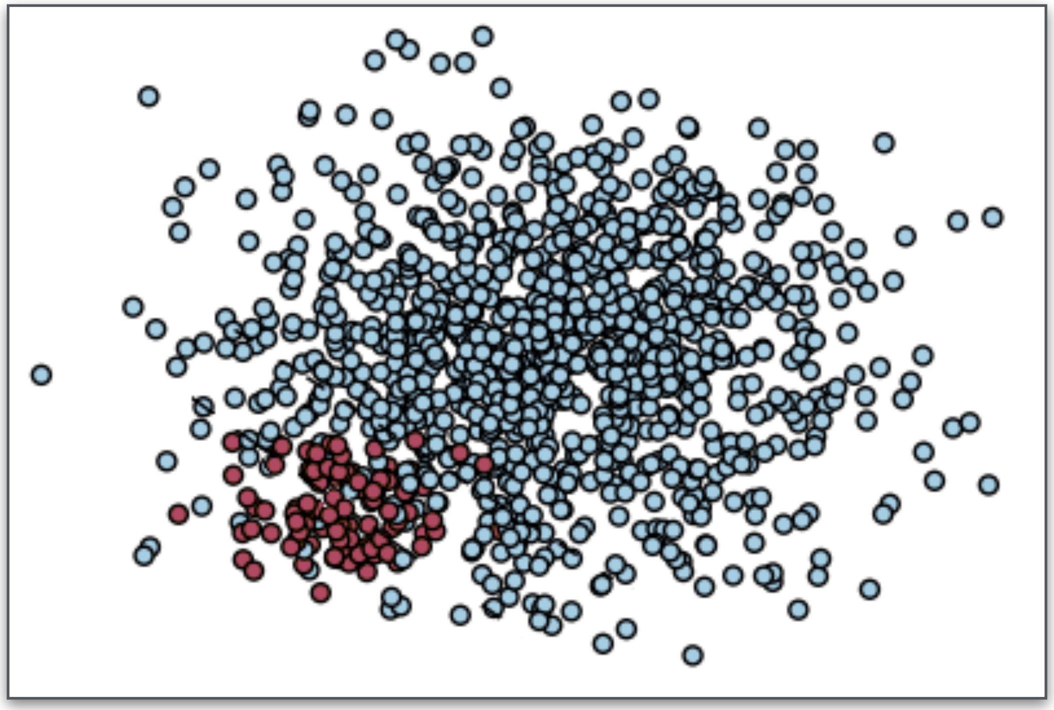
Athlete	*Speed*	*Agility*	*Selected*	*Distance*	*Weight*
x14	5.75	8.75	Yes	1.060660	0.942809
x2	3.75	8.00	No	1.250000	0.800000
x16	5.50	6.75	Yes	1.346291	0.742781

- Weights for Yes = 0.942809 + 0.742781 = 1.68559
- Weights for No = 0.8

➡ Majority says Yes

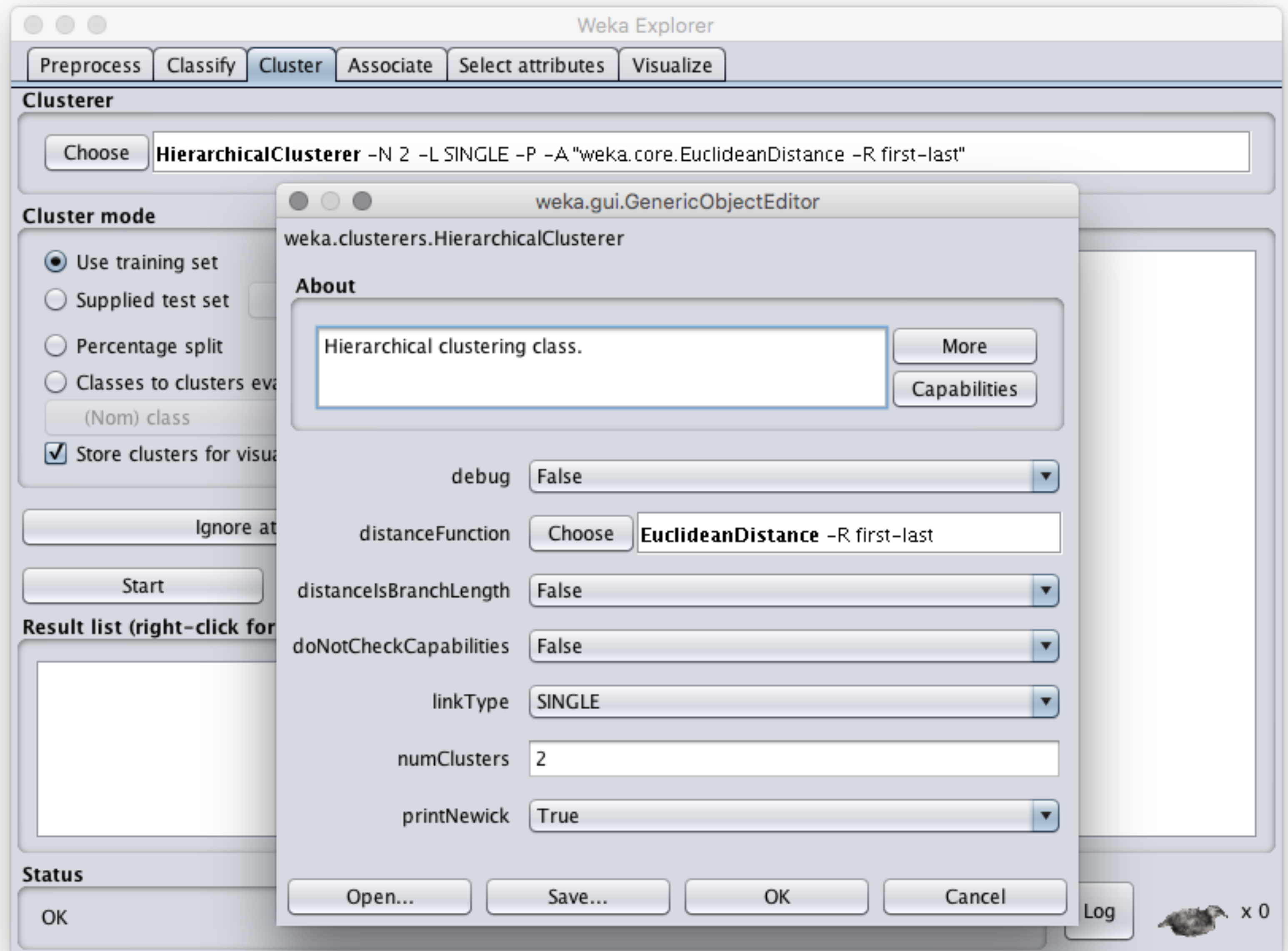
Noisy Data

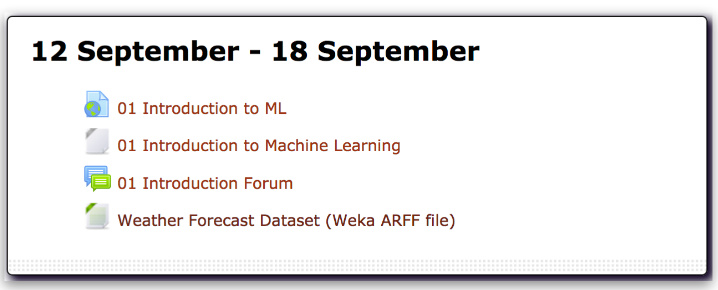
- A simple 1-NN classifier is easy to implement.
- But it will be susceptible to “noise” in the data.
  - ➡ A misclassification will occur every time a single noisy example is retrieved.
- Using a larger neighbourhood size (e.g. $k > 2$) can sometimes make the classifier more robust and overcome this problem.
- But when k is large ($k \rightarrow N$) and classes are *unbalanced*, we always predict the majority class.



k-NN in Weka

Download and install the Java *Weka Toolkit* - Version 3.8 Stable





http://www.cs.waikato.ac.nz/ml/weka

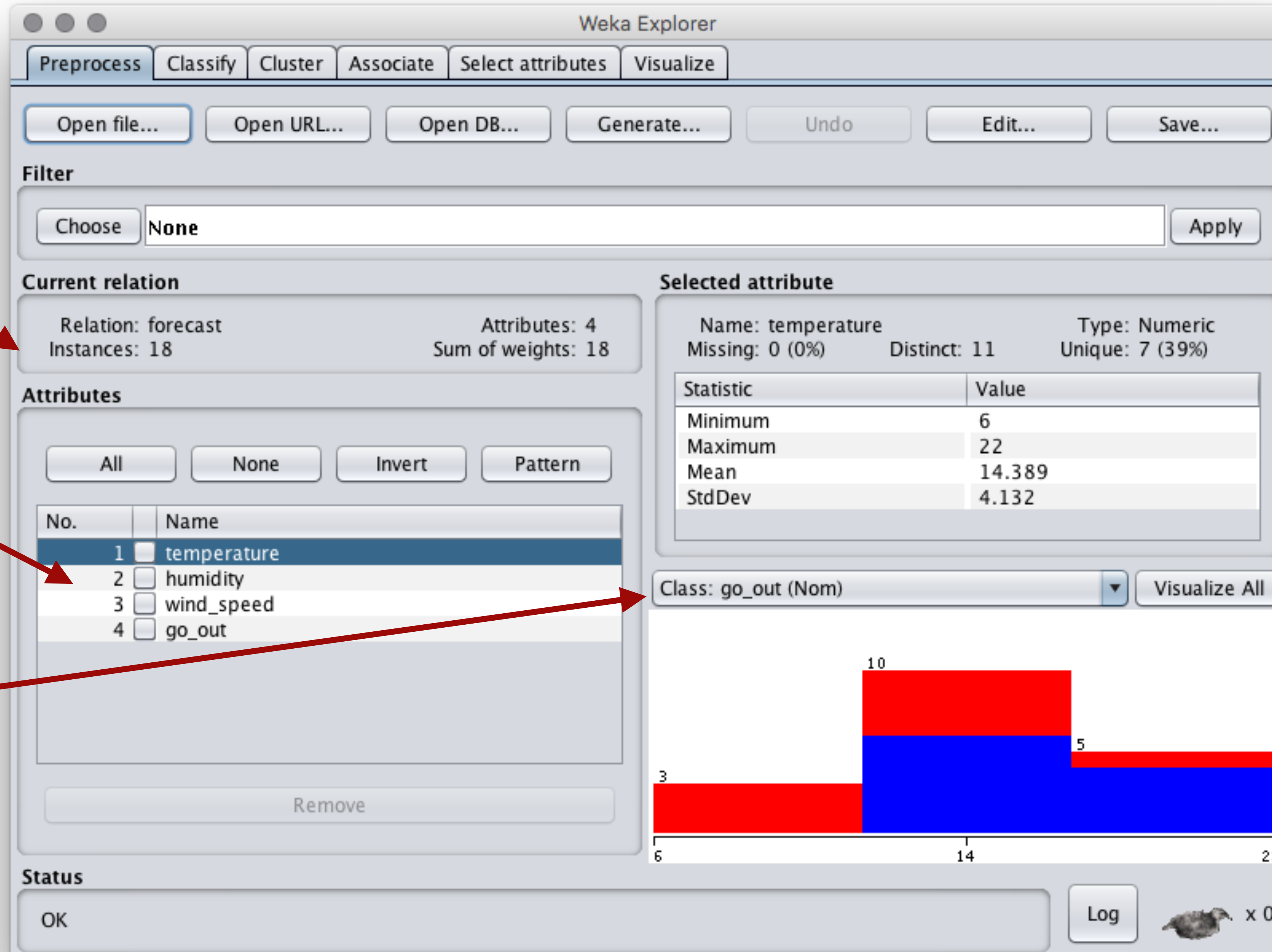
k-NN in Weka

1. Launch the WEKA application and click on the *Explorer* button.
2. Click *Open File* - e.g. forecast.arff (WEKA ARFF dataset format)

18 examples
("instances")

3 numeric features
("attributes")

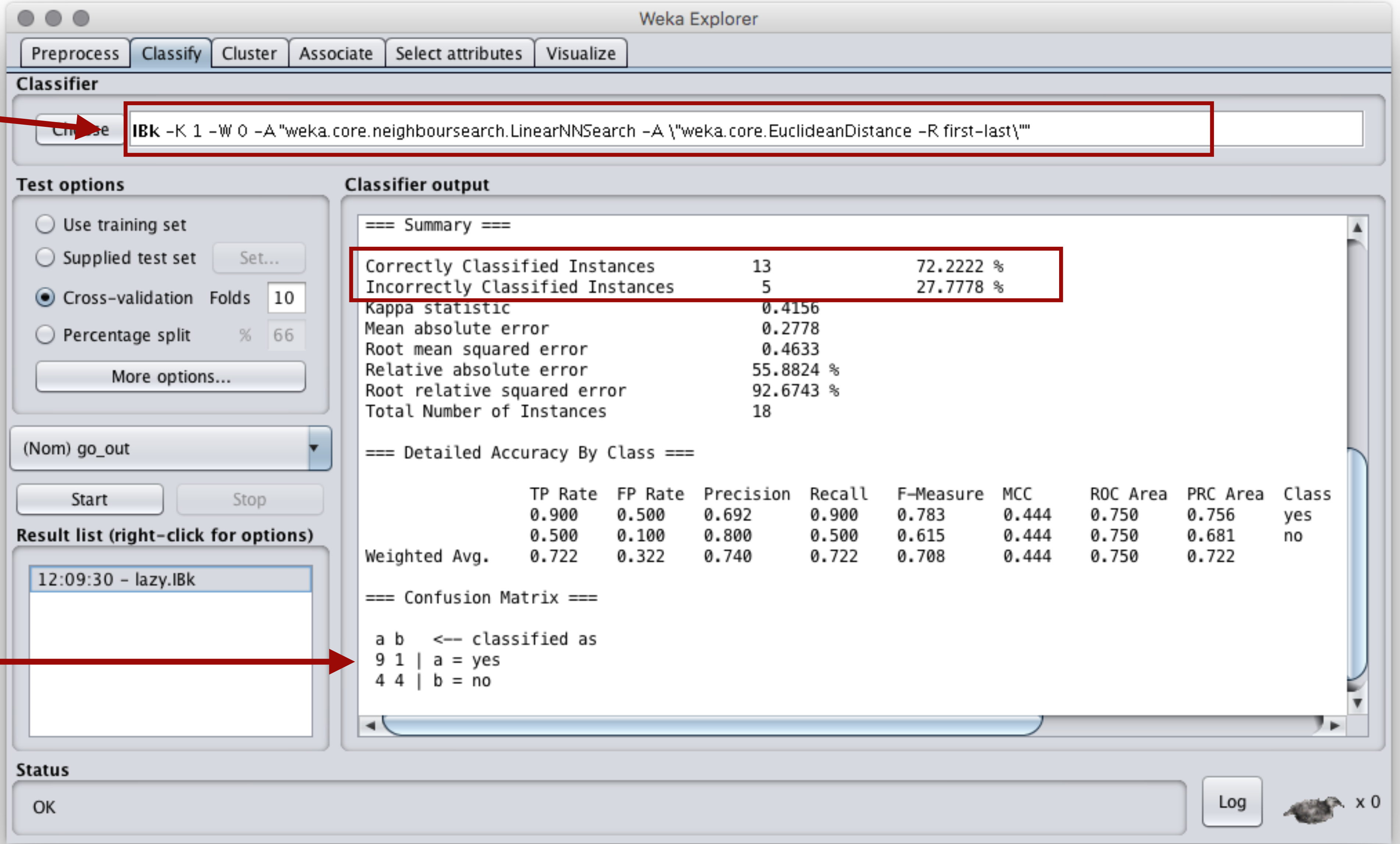
Class label
go_out={yes,no}



k-NN in Weka

3. In *Classify* tab, click *Choose* and find *Lazy→IBk* on the list.
4. Choose *(Nom) go_out* as class label from drop-down list.
5. Click *Start*.

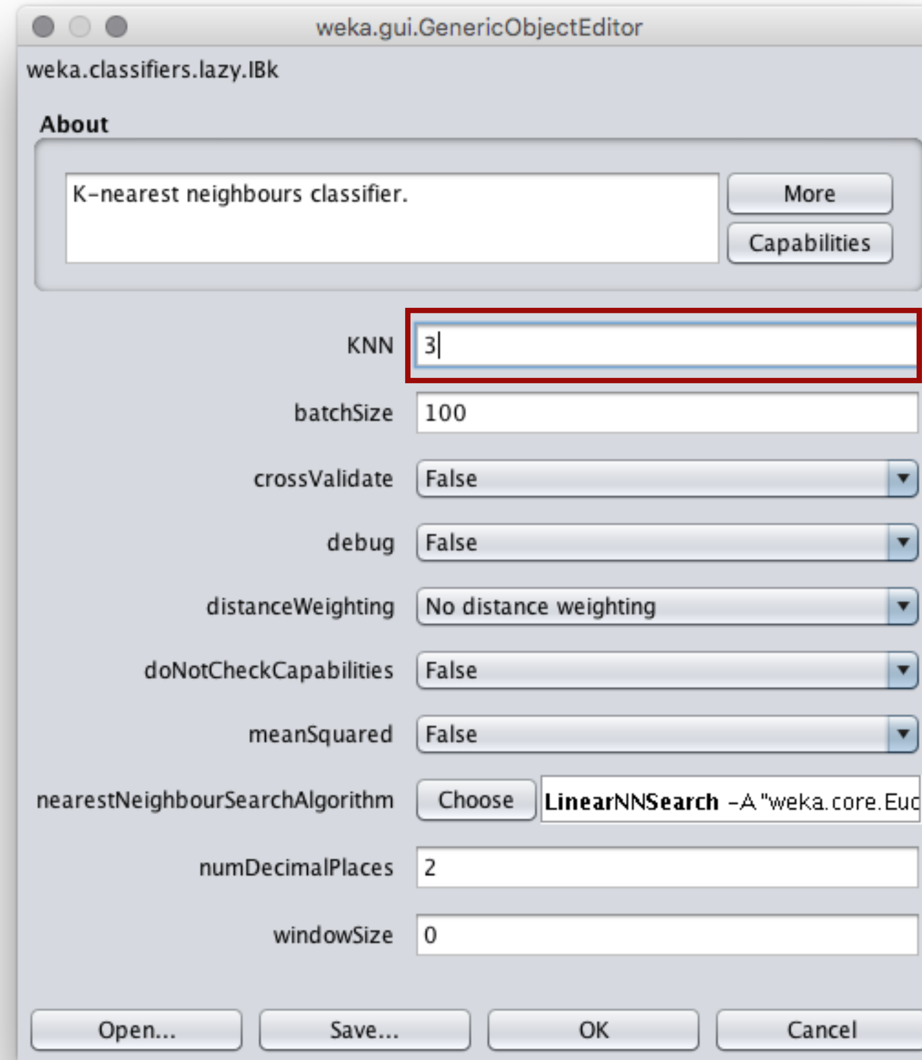
Parameter set:
By default
K=1 neighbours



Output of
classification
process

k-NN in Weka

- To change algorithm parameter values:
  1. Click the parameter set
  2. Enter new value for number of neighbours (KNN) - e.g 3
  3. Click *OK* and re-run process.




```
=== Summary ===

Correctly Classified Instances          15               83.3333 %
Incorrectly Classified Instances         3               16.6667 %
Kappa statistic                          0.6582
Mean absolute error                      0.2156
Root mean squared error                  0.3575
Relative absolute error                 43.3647 %
Root relative squared error             71.5158 %
Total Number of Instances               18

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.900    0.250    0.818      0.900    0.857      0.663    0.900     0.956     yes
                 0.750    0.100    0.857      0.750    0.800      0.663    0.900     0.817     no
Weighted Avg.    0.833    0.183    0.835      0.833    0.832      0.663    0.900     0.894

=== Confusion Matrix ===

 a b   <-- classified as
 9 1 | a = yes
 2 6 | b = no
```