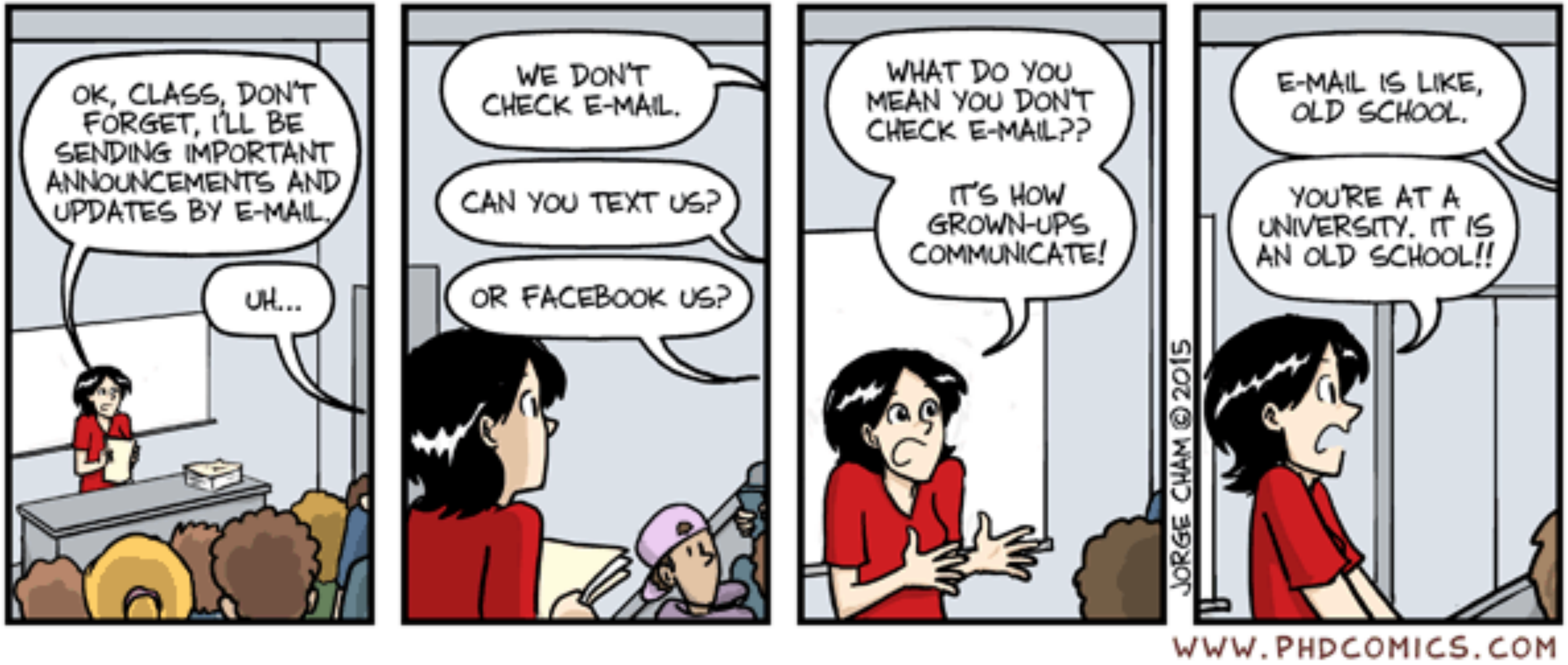

OK, CLASS, DON'T FORGET, I'LL BE SENDING IMPORTANT ANNOUNCEMENTS AND UPDATES BY E-MAIL.
UH...
WE DON'T CHECK E-MAIL.
CAN YOU TEXT US?
OR FACEBOOK US?
WHAT DO YOU MEAN YOU DON'T CHECK E-MAIL??
IT'S HOW GROWN-UPS COMMUNICATE!
JORGE CHAM © 2015
E-MAIL IS LIKE, OLD SCHOOL.
YOU'RE AT A UNIVERSITY. IT IS AN OLD SCHOOL!!
WWW.PHDCOMICS.COM

Machine Learning

Data Science & Computer Science
Centre for Research in Mathematics and Data Science
Artificial Intelligence Research Group

Non-parametric methods
Ref ch8

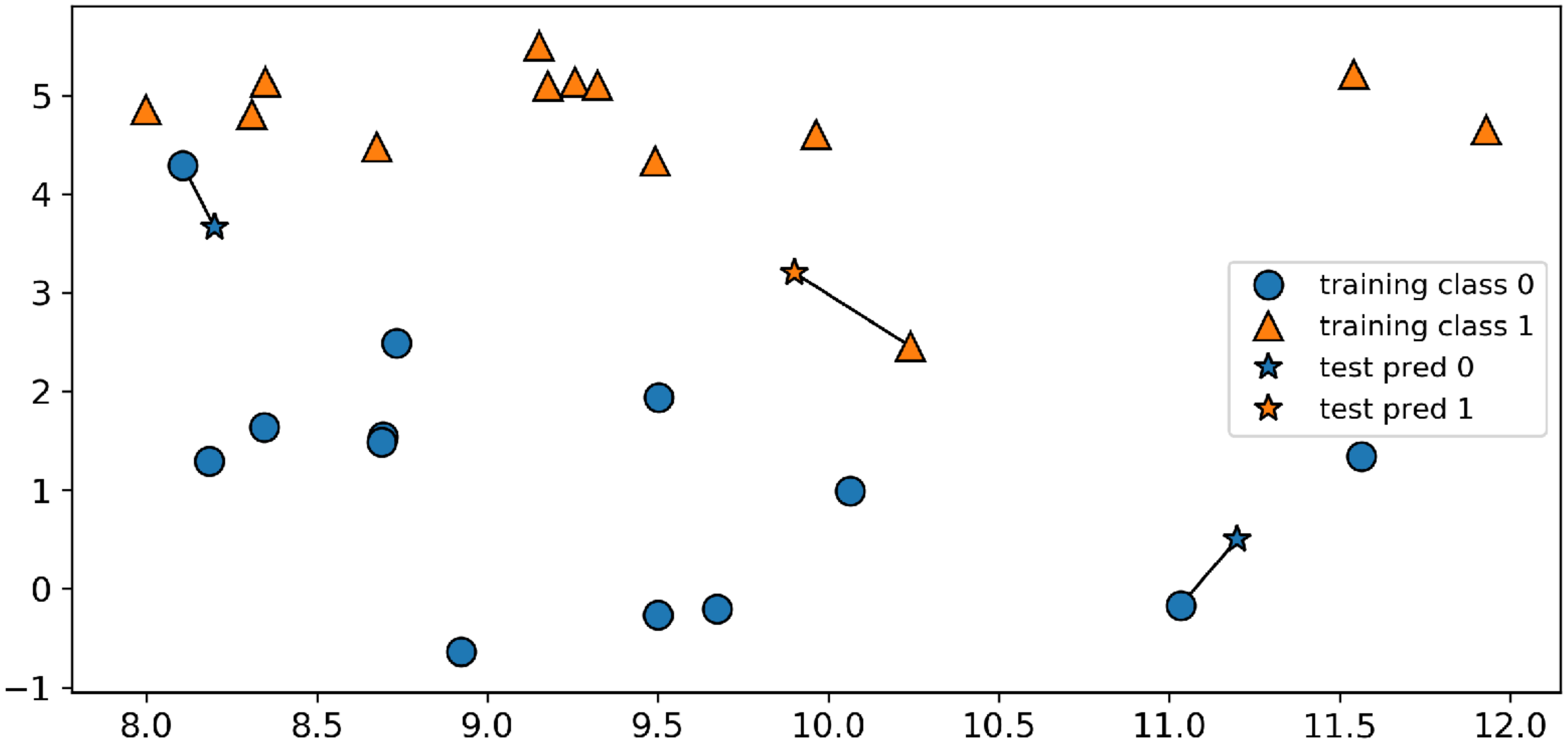
- KNN
- Kernel smoother

k-Nearest Neighbours

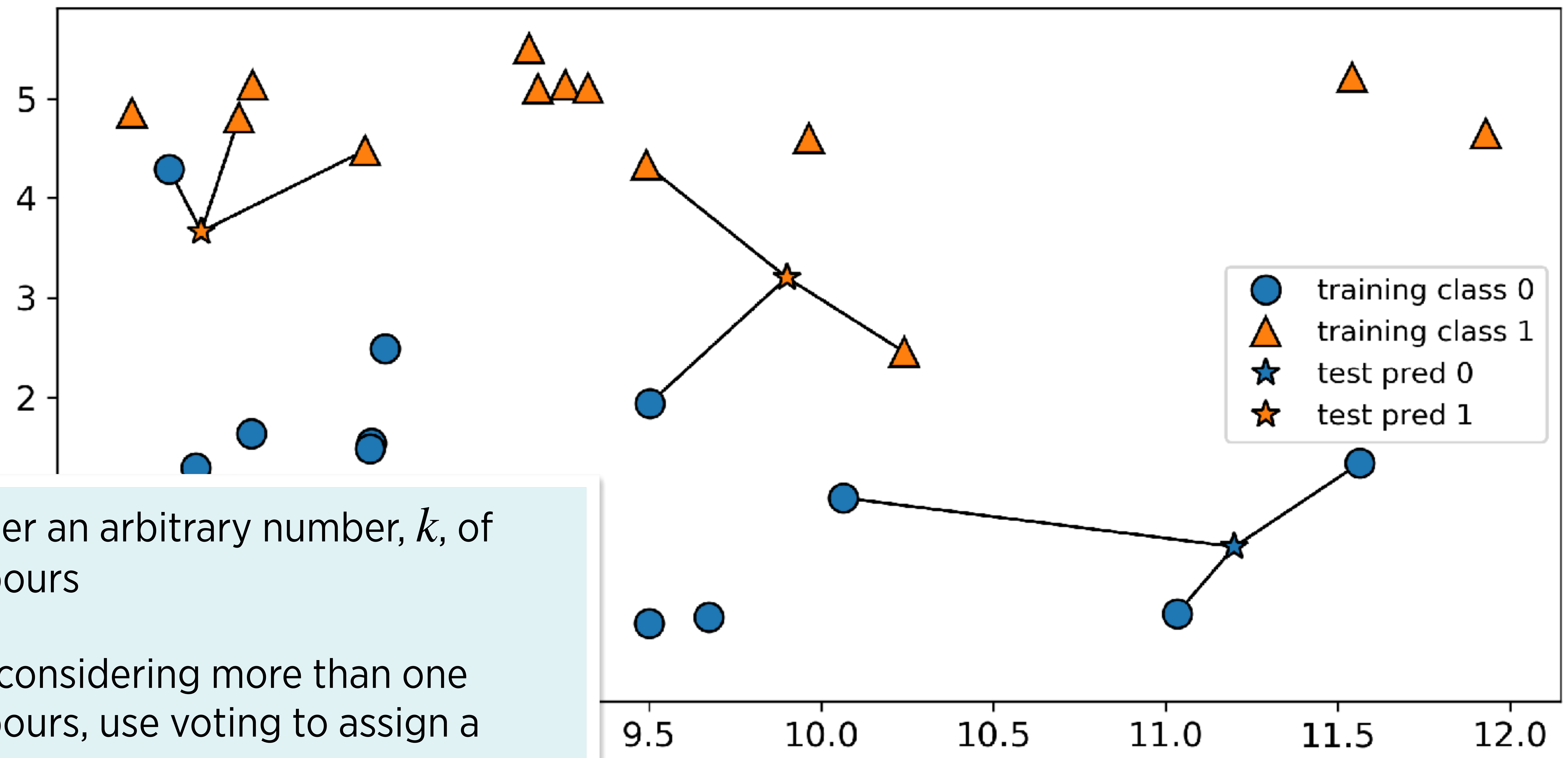
What is kNN?

- Arguably the simplest machine learning algorithm
- Building the model consists only of storing the training data set
  - a “lazy” algorithm — does not learn a function from the training data
  - making a prediction for a new data point by finding the “nearest neighbours”
- A non-parametric method
  - do not assume a fixed model structure.
  - often, model size grows with the training set.
- All methods we’ve discussed so far have fixed model structure, with parameters learnt and we then no longer need the training data.

One nearest neighbour



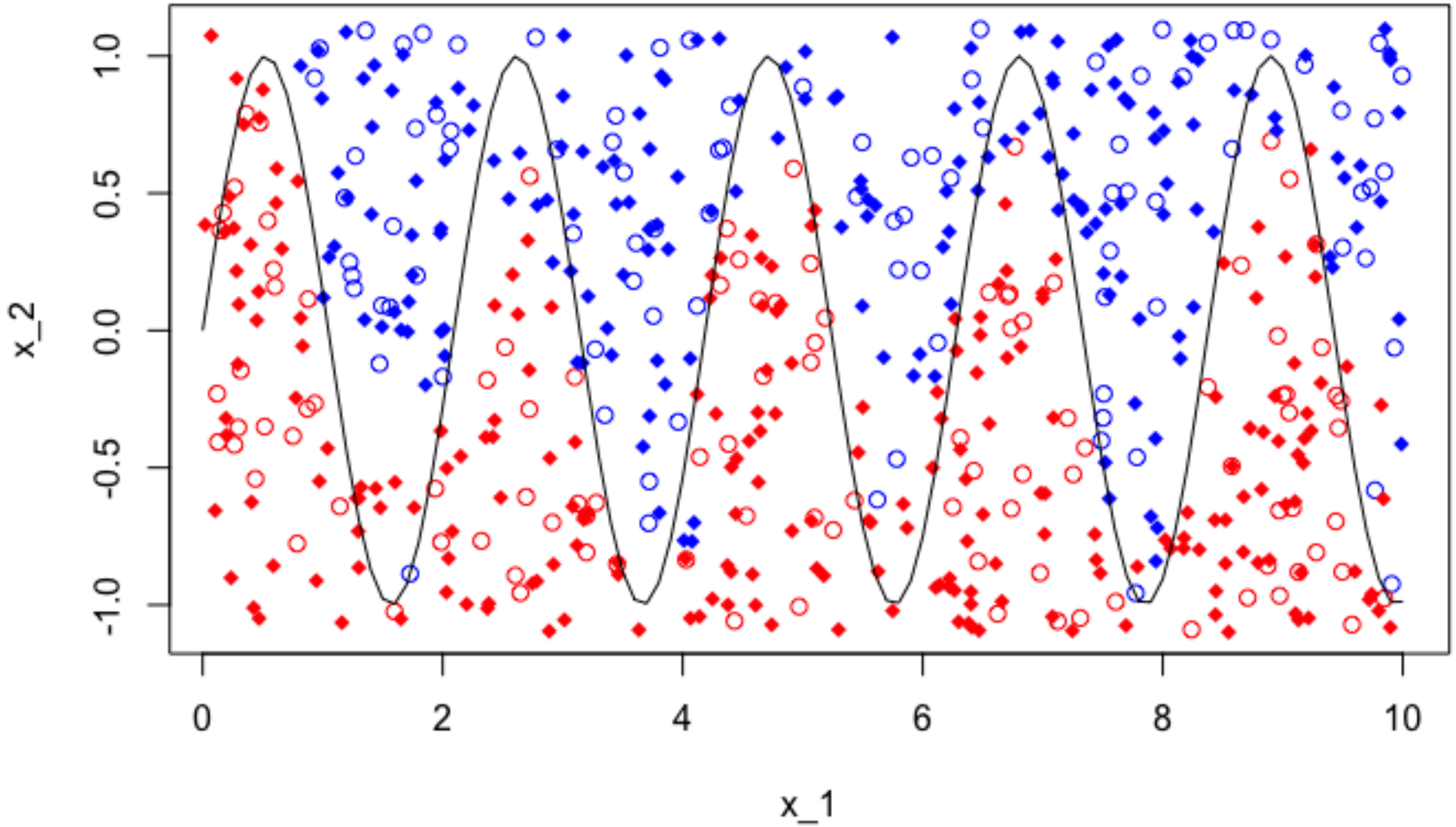
k nearest neighbours (kNN)



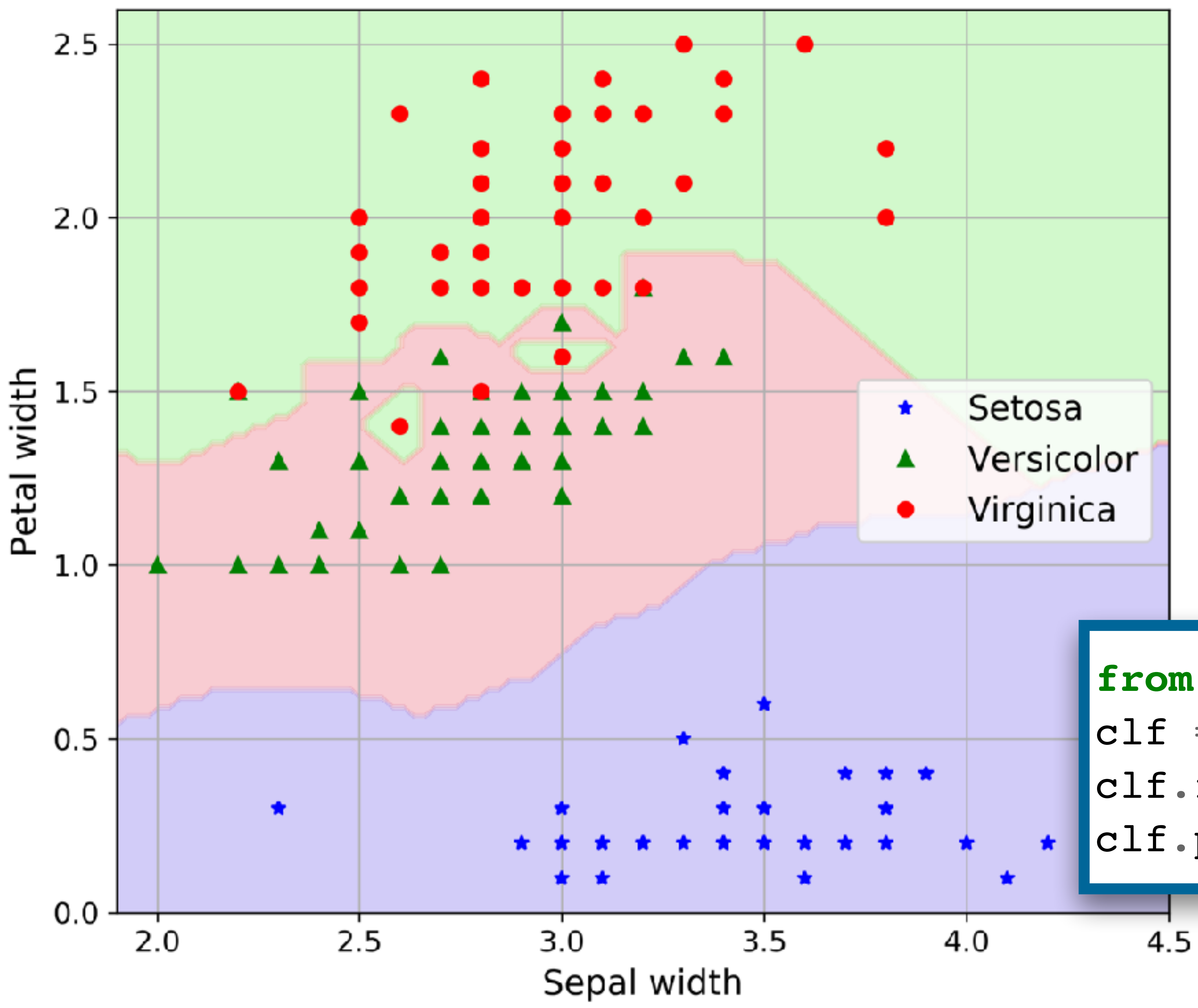
- Consider an arbitrary number, k, of neighbours
- When considering more than one neighbours, use voting to assign a label.

Advantage of kNN classifier

- Can handle problems where the decision boundary is not linear.



Advantage of kNN classifier



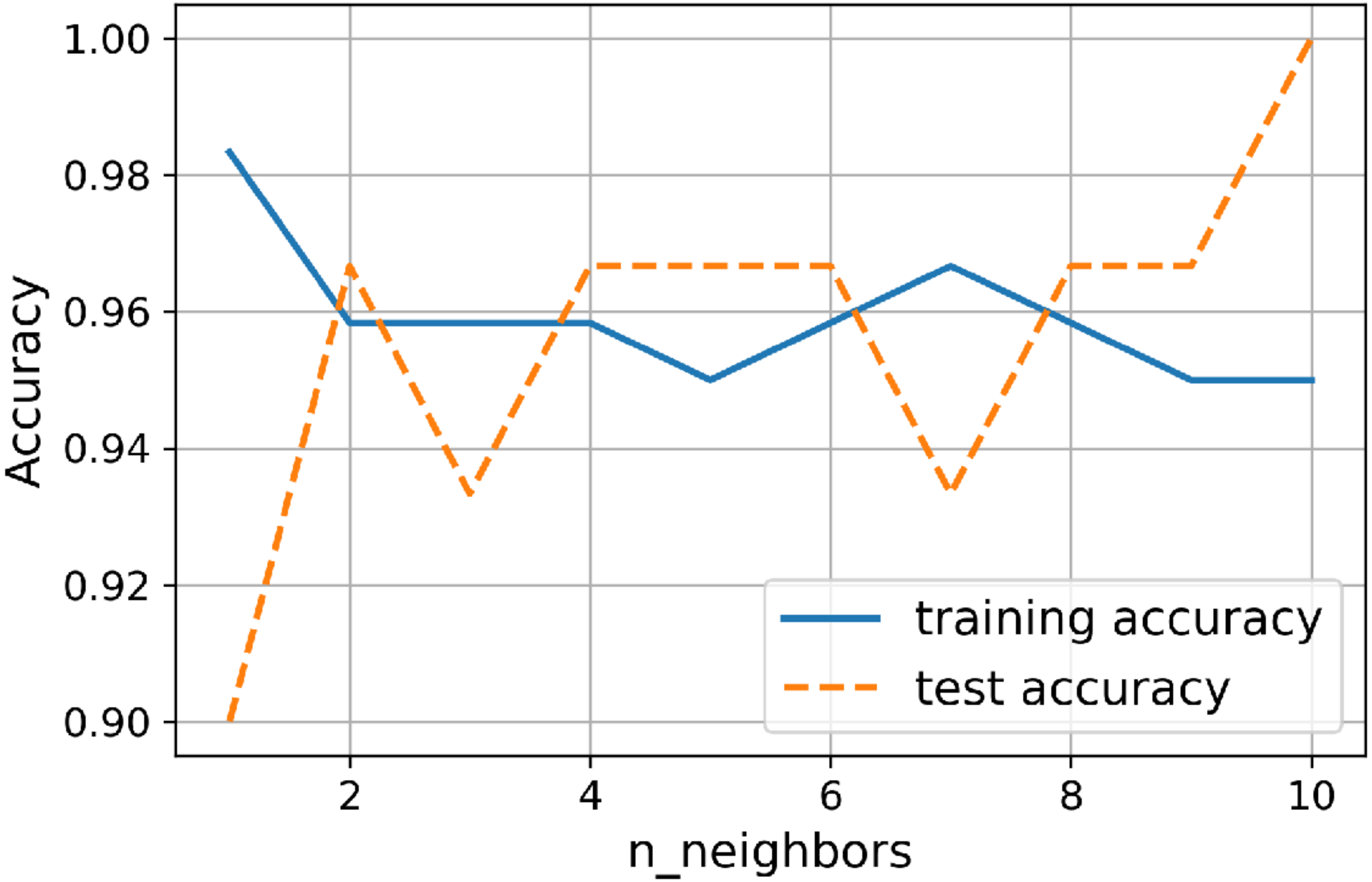
- kNN can handle multiple classes with ease
- For more classes,
  - count how many neighbours belong to each class
  - predict the most common class

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train, y_train)
clf.predict(X_test)
```

Analysing kNN classifier

Using more neighbours leads to smoother decision boundary,
i.e. a simpler model (less complexity).

kNN Complexity



- Bias-variance
  - less neighbours → more complex model
- Overfitting when k=1

Distance measures and Other considerations

Distance measure

- We impose four requirements on any distance measure we use.

- For the distance $\mathrm{dist}(X, Y)$ between two points X and Y

  1. The distance must never be negative.
  2. The distance of any point from itself is zeros, i.e. $\mathrm{dist}(A, A) = 0$.
  3. The distance from A to B is the same as the distance from B to A, i.e. $\mathrm{dist}(A, B) = \mathrm{dist}(B, A)$. (the symmetry condition)
  4. For any points A, B and Z: $\mathrm{dist}(A, B) \leq \mathrm{dist}(A, Z) + \mathrm{dist}(Z, B)$, if Z is the same point as A or B or is on the direct route between them. (the triangle inequality)

Distance measures

- There are many distance measures
  - Euclidean: dist$(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_p - b_p)^2}$
  - Manhattan: dist$(A, B) = |a_1 - b_1| + |a_2 - b_b|$
  - Maximum dimension: dist$(A, B) = \arg\max_i\,(a_i - b_i)$ for $i = [1,p]$
  - Mahalanobis: dist$(A, B) = \sqrt{(\mathbf{A} - \mathbf{B})^T \mathbf{S} (\mathbf{A} - \mathbf{B})}$, where $\mathbf{S}$ is the covariance matrix of the distribution that A and B belong to.
  - correlation based distance
  - binary metrics
  - etc.
- For most applications, Euclidean distance is the most natural measure to use.

Normalisation

- A major problem when using distance measures is that large values frequently swamp the small ones
  - if one features has values range from [0, 1000] and another has values range from [0, 1], then the first feature will overwhelm the distance measure by one million to one.
- Therefore, we generally *normalise* the value of continuous attributes
- One possible method is to normalise the data so that all attributes have values from 0 to 1:

$$x_i' = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

  this is sometimes called the min-max scaling.

Importance of different attributes

- Sometimes not all attributes are equal
- Some are irrelevant, while others are more important.
- In these cases, we add *weights* to the attributes.
- For example, in Euclidean case:

  $$\text{dist}(A, B) = \sqrt{w_1(a_1 - b_1)^2 + w_2(a_2 - b_2)^2 + \ldots + w_p(a_p - b_p)^2}$$

  where w_i are the weights
- It's customary to scale the weight values so that $\sum_i w_i = 1$

Dealing with categorical attributes

- There is no entirely satisfactory way of dealing with categorical attributes

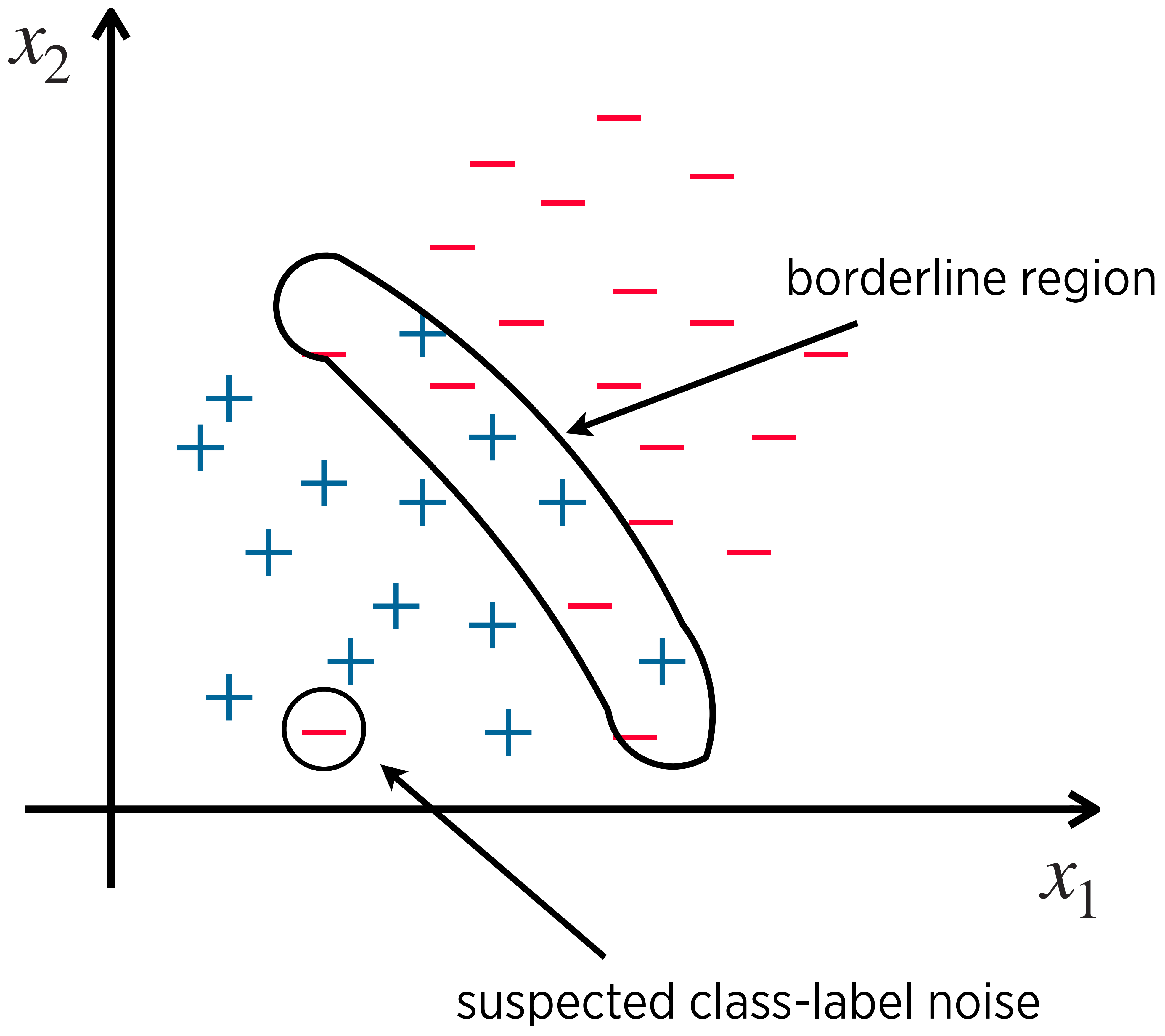
- One possibility is to say difference between any two identical values of the attribute is 0, and any different values is 1.
  - e.g. red-red = 0, red-blue = 1, blue - purple = 1, etc.

- If there is some ordering to the values, then we could put values to them.
  - e.g. {good, average, bad} = {1, 0.5, 0}

Distance can be misleading

- The distance measures should not be applied mechanically, ignoring specific aspects of the given domain
- Case study: data with attributes {`size`, `price`, `season`}. Observations x_1 = (2, 1.5, `summer`) and x_2 = (1, 0.5, `winter`):
  - How do we measure the distances?
- Concrete choice of class values will depend on specific needs of the given application.

Dangerous Examples



- $\mathbf{x}$ and $\mathbf{y}$ forms a *Tomek Link*:
  1. $\mathbf{x}$ is the NN of $\mathbf{y}$
  2. $\mathbf{y}$ is the NN of $\mathbf{x}$
  3. $\mathbf{x}$ and $\mathbf{y}$ have different classes
- Can remove from training set all such pairs
- Sometimes this process need to be repeated

Advantage: Can reduce the value of k once Tomek Links have been removed.

Redundant Examples

- Modern data sets are large: to classify one thousand samples using a training data set of 10^6 observations with 10^4 attributes will require $10^6 \times 10^4 \times 10^3 = 10^{13}$ operations. That is *A LOT!*

- Training sets are often redundant in that kNN behaviour will not change even if some samples were removed.

  - Often majority of the examples can be removed as they add to computational cost but not classification performance.

- We want to replace the training data, T, with its *consistent subset*, S, such that it will not affect the what class labels are returned by kNN.

Creating a consistent subset

1. Let S contain one example of each class from the training set, T.

2. Using examples from S, re-classify the examples in T with the 1-NN classifier. Let M be the set of those examples that have in this manner received the wrong class.

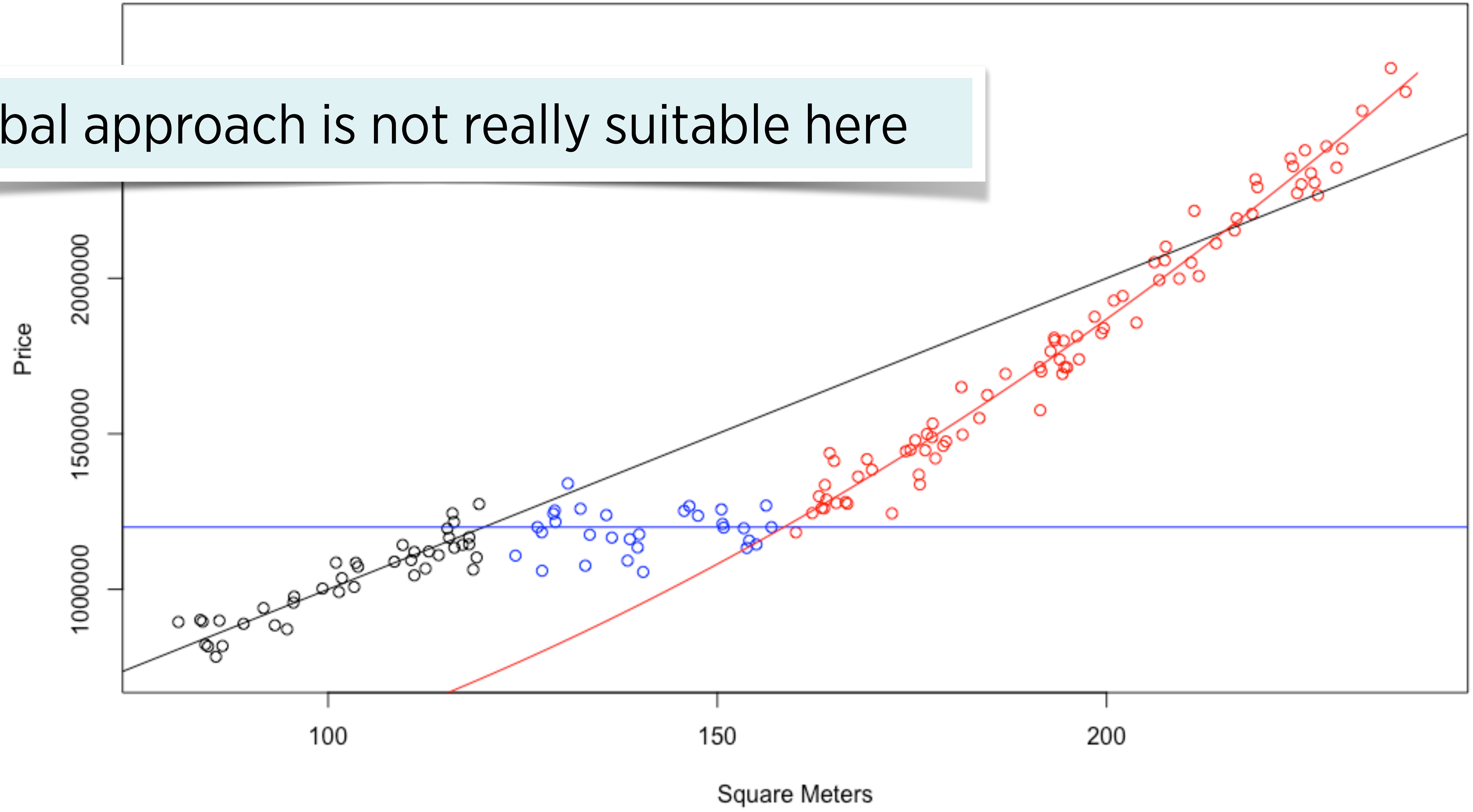
3. Copy to S all examples from M.

4. If the content of S have not changed, in the previous step, then *stop*; otherwise go to step 1.

kNN Regression

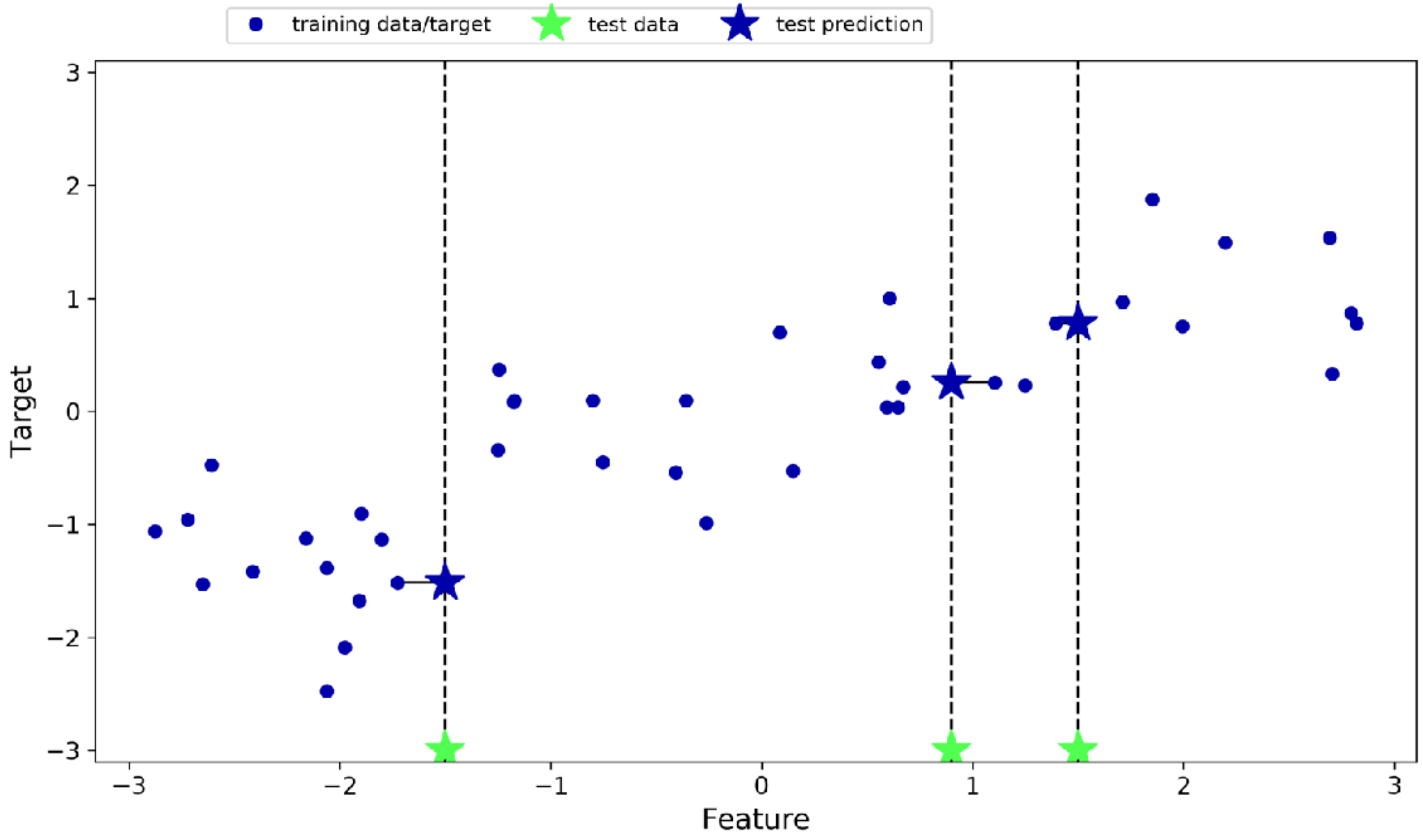
kNN regression

Global approach is not really suitable here



1-NN regression

- The predicted value $\hat{y}_j$ is equal to the value y_i of the closest x_i.



1-nearest neighbours with 2 inputs

- Example:
  prediction of exam score, based on Quiz results and time spent for class.

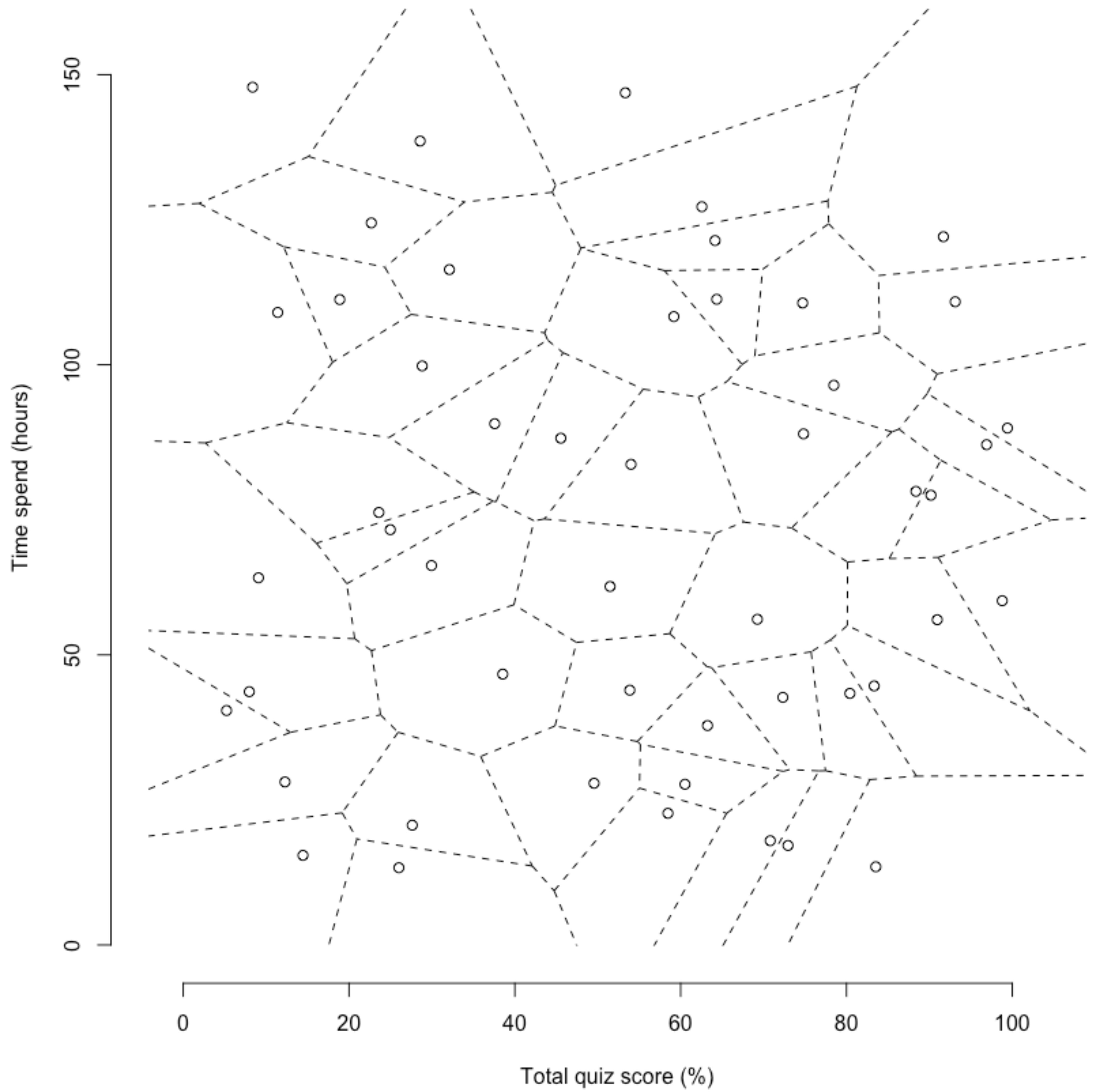
- Voronoi diagram divides up the space:

  - This will create a region for each of the N data points.

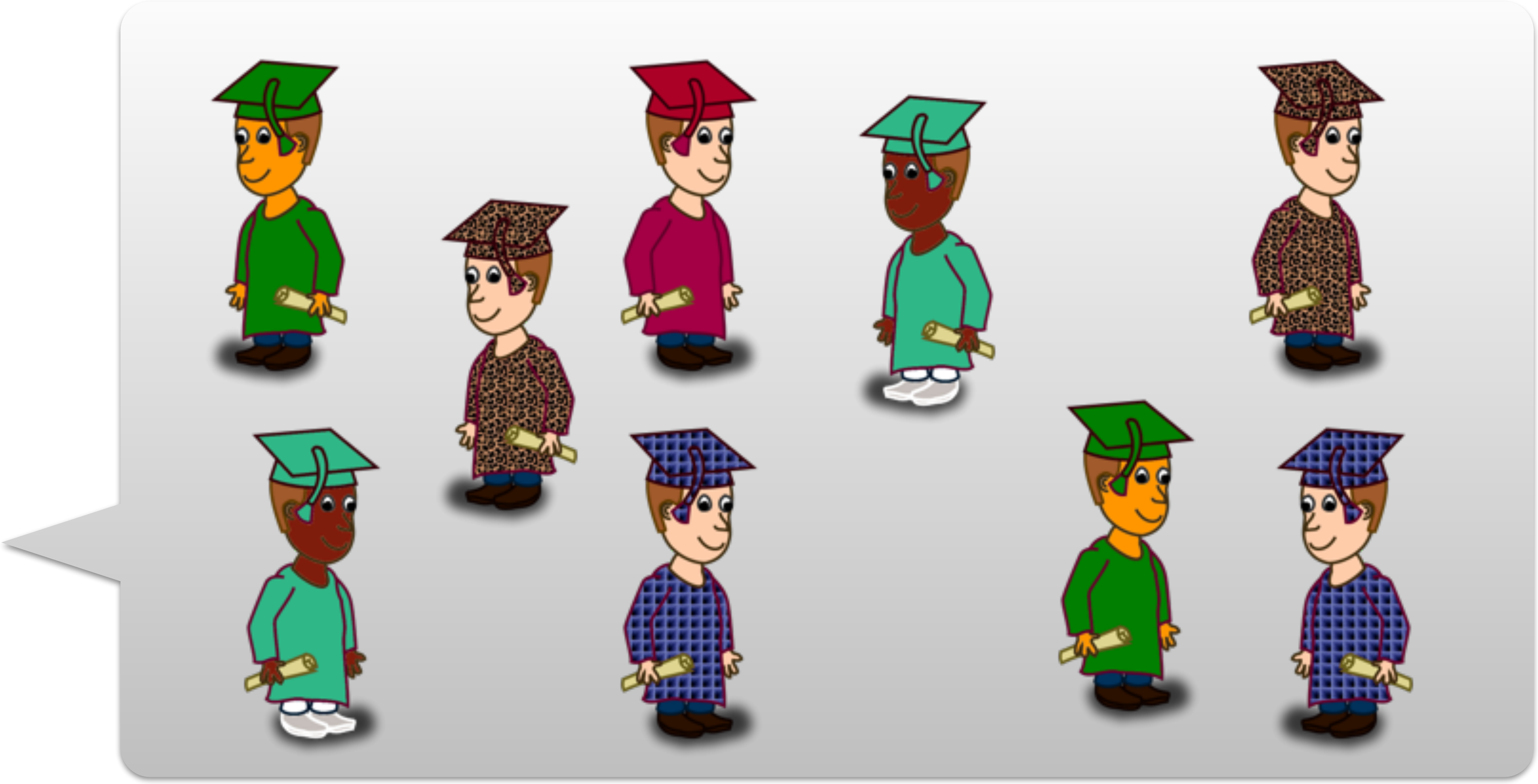
  - Any point in a region is closest to the data point in that region – distances to any other data point are larger.

- This is just for visualisation - we don't actually need to create the Voronoi diagram.

- Same idea for higher dimensions
  (just not possible to visualise)



1-NN search

- Example: Exam results, based on quiz performance
- Query: performance of a student 
- "Training" data set - previous performance 
- Select a distance metric
- Output: most similar student 

1-NN search algorithm

number of all students
in training set

Set closestStudent  = NA;   distanceToNN = ∞

for i in 1:N

    compute: d ← distance( i, )

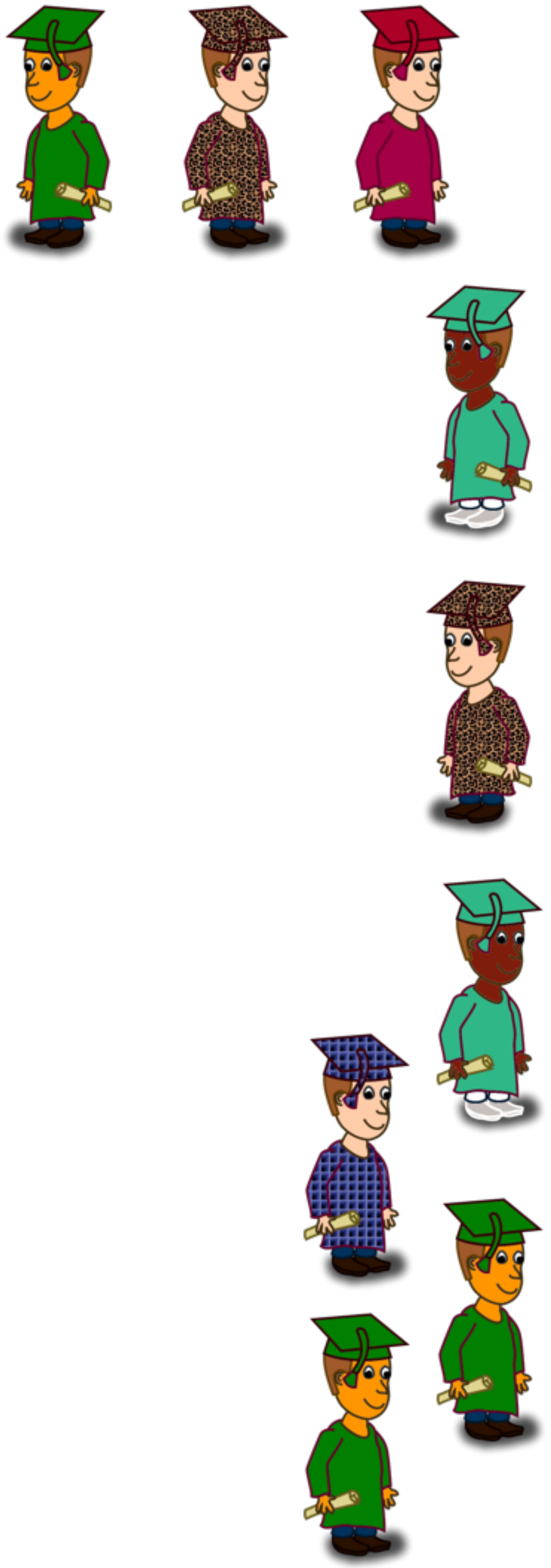
the query student

    if d < distanceToNN

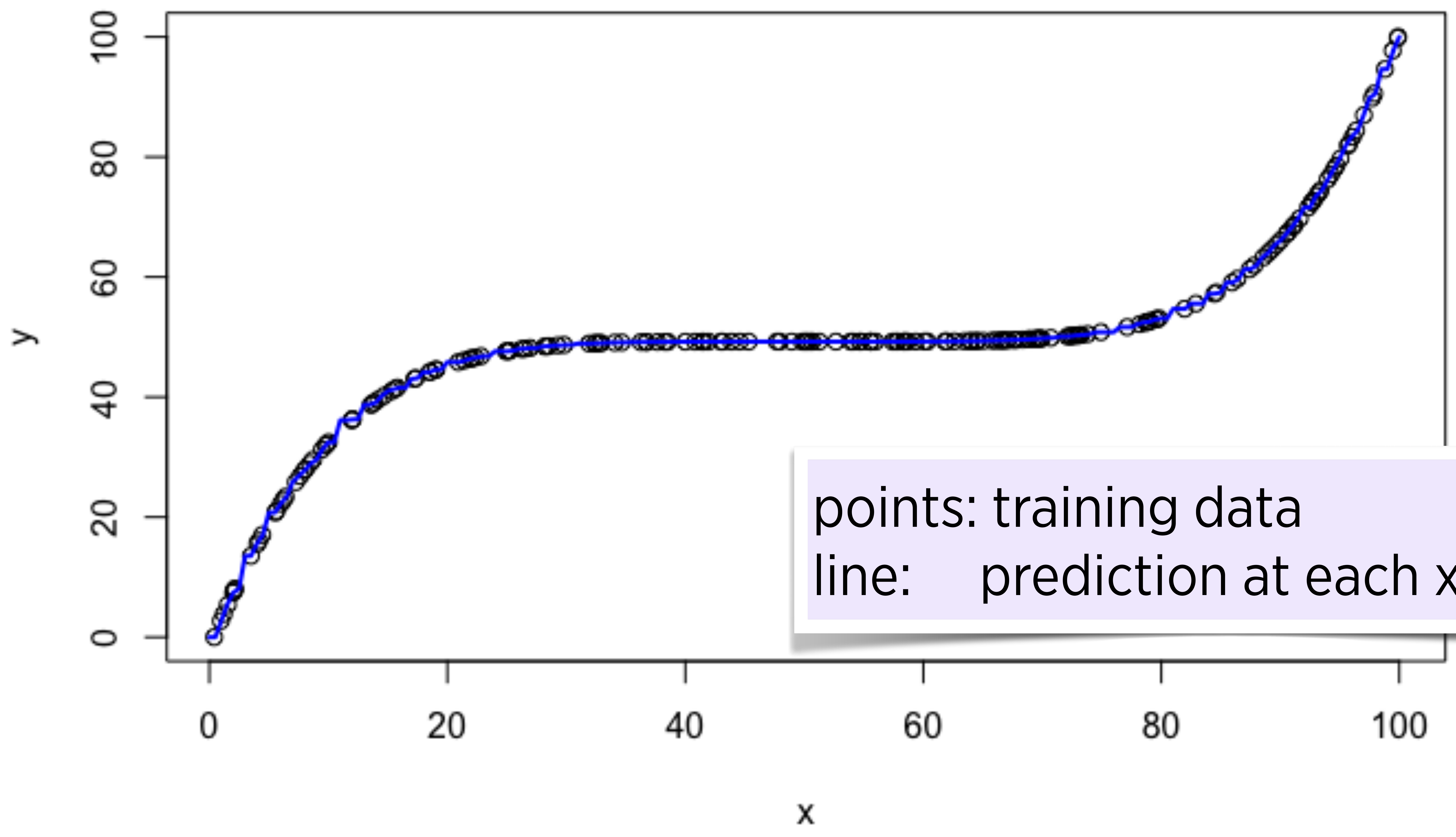
        set  to  i

        set distanceToNN ← d

return most similar student 



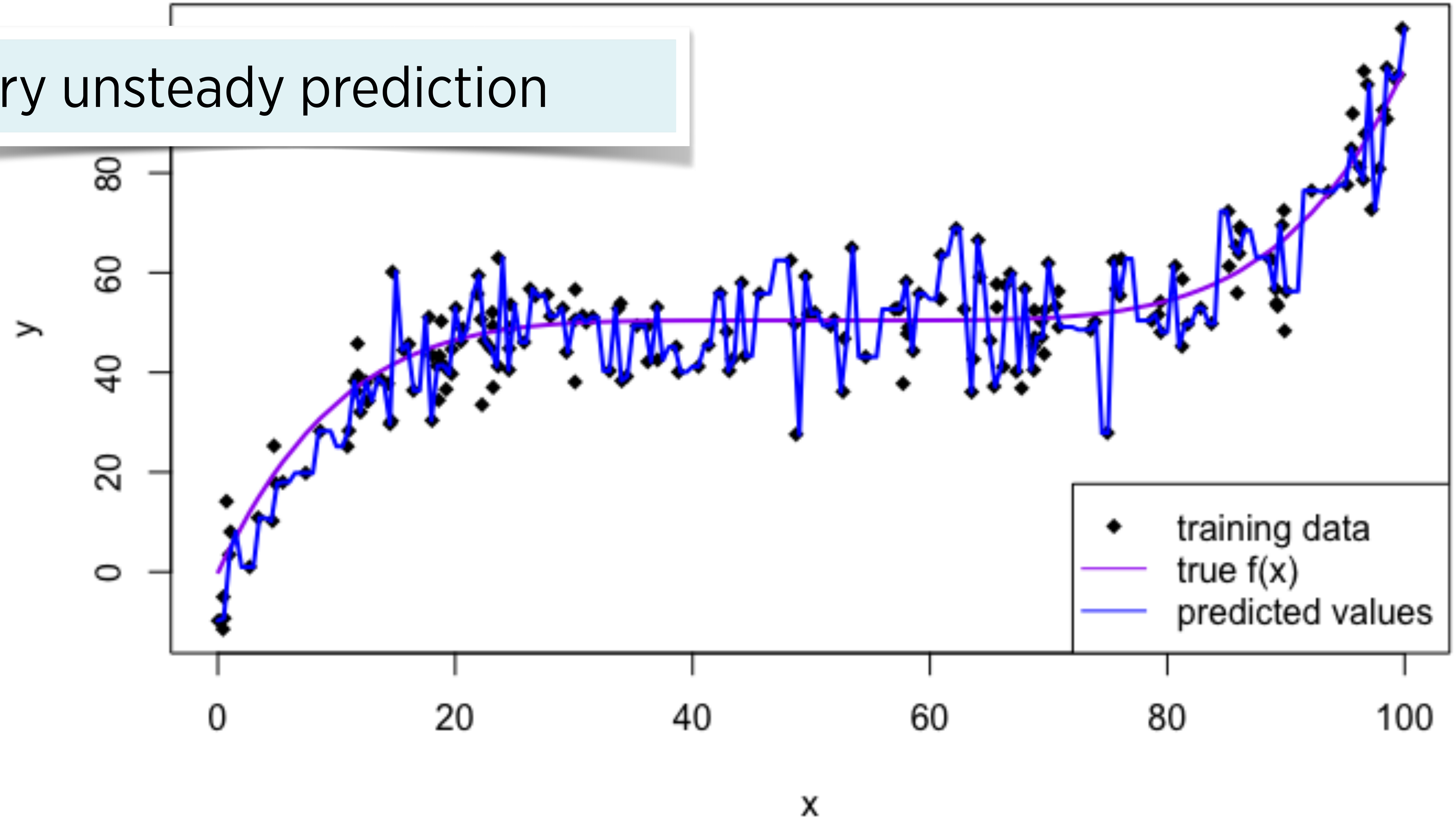
1-NN with dense data



points: training data
line: prediction at each x-value

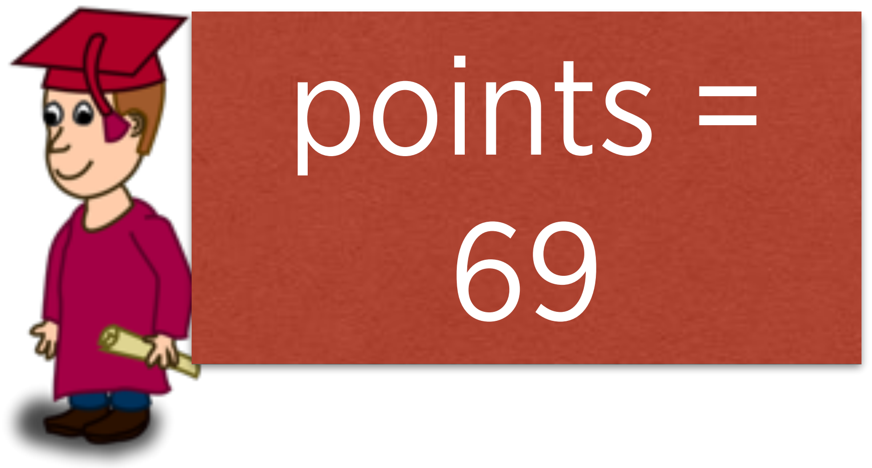
1-NN with noisy data

Very unsteady prediction



k-nearest neighbours

- “Training data” (students, exam points): $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
- Given a new student, x_j, what exam can we expect?

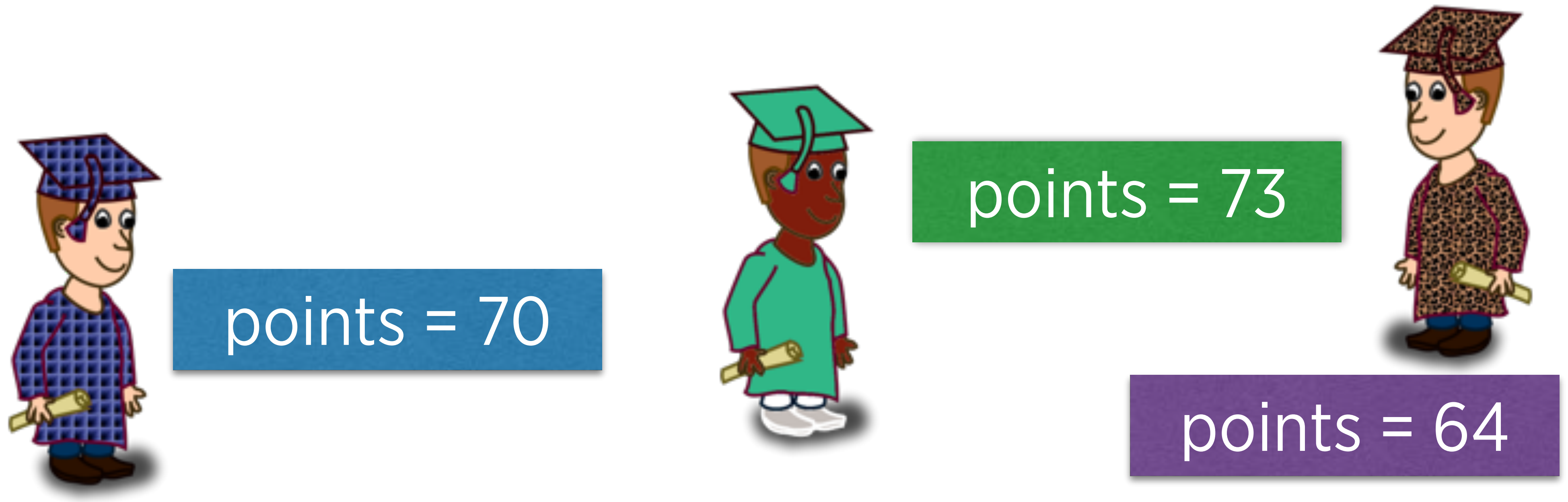


1. Find the k closest x_i in our data set:
   $(x_{NN1}, \ldots, x_{NNk})$ is the ordered list of nearest neighbours (x_{NN1} closest).
   For any other x_i in the data set: distance$(x_i, x_j) \geq$ distance(x_{NNk}, x_j).

2. Predict exam points as:

$$\hat{y}_j = \frac{1}{k}(y_{\mathrm{NN}_1} + \ldots + y_{\mathrm{NN}_k})$$

$$= \frac{1}{k}\sum_{i=1}^{k} y_{\mathrm{NN}_i}$$



k-NN search

- Query: performance of a student

- “Training” data set - previous performance

- Select a distance metric

- Output: most similar students

the query student

k = 3

Placeholder for k students

List of all students (there are N students)

k-NN search algorithm

Set `distanceToNN = sort(d1, …, dk)`

take first k students from data set, sort by distance to query student

list of sorted students

compute: d ← distance(i,)

if `d < distanceToNN[k]`

find j so that `d > distanceToNN[j-1]` but `d < distanceToNN[j]`

\# remove furthest student from , shift queue

[(j+1):k] = [j:(k-1)]

distanceToNN[(j+1):k] = distanceToNN[j:(k-1)]

set distanceToNN[j] ← d and set [j] ←

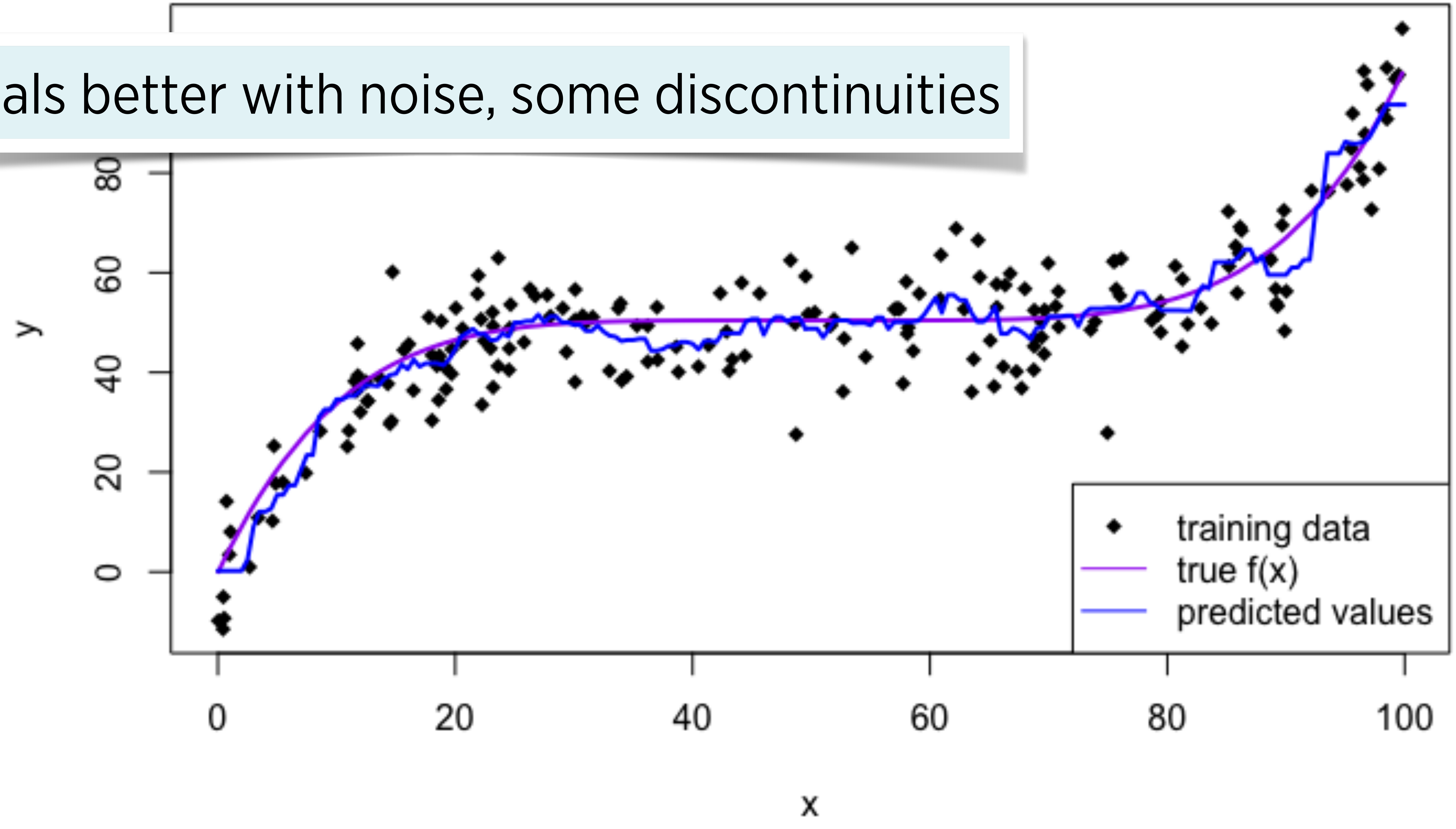
return k most similar students

k-NN search algorithm

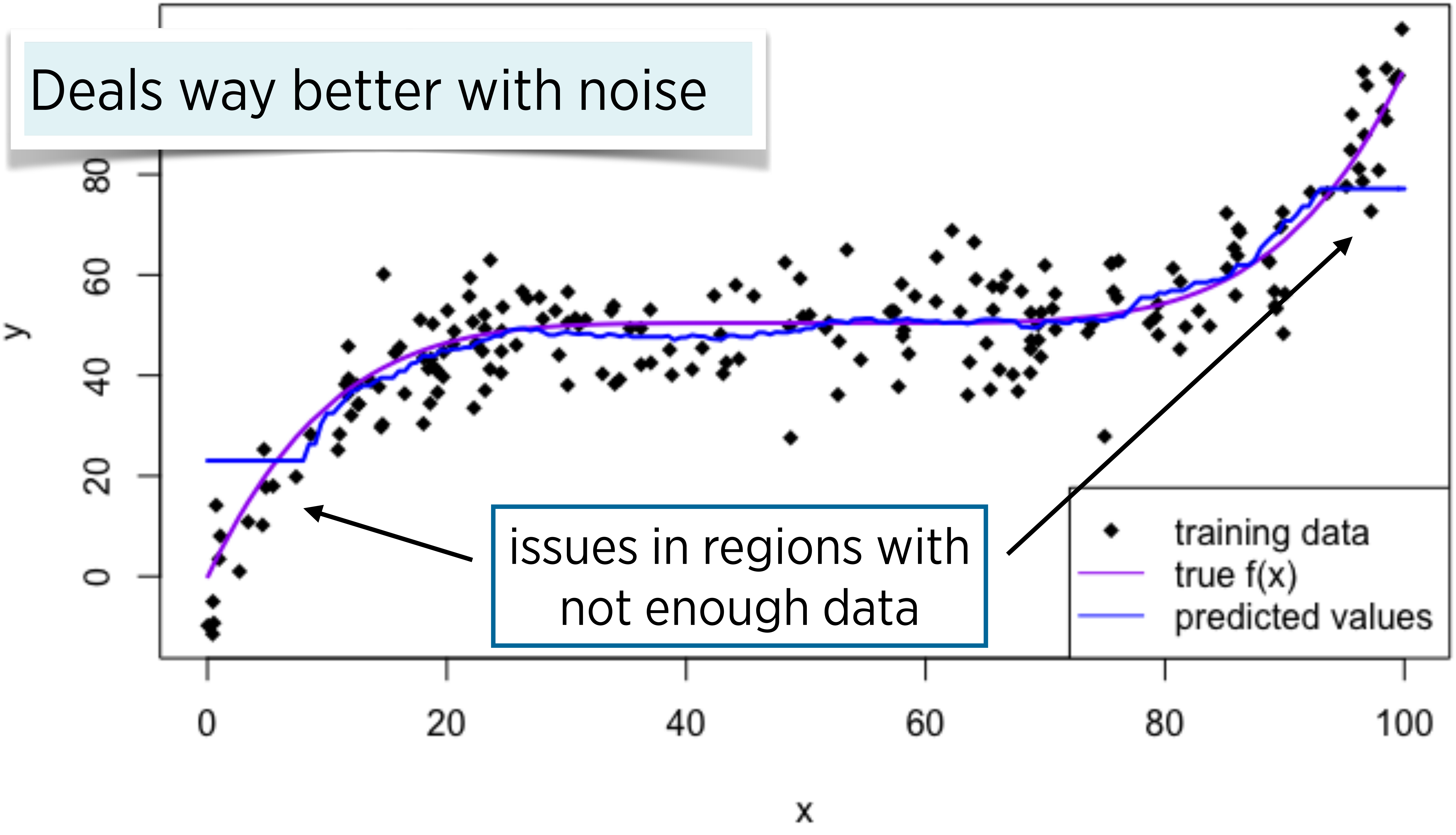
Actually you can just sort the distances and take the k smallest!

k-NN with noisy data: k=9

Deals better with noise, some discontinuities



k-NN with noisy data: k=31



Weighted k-NN

- "Training data" (students, exam points): $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
- Given a new student, x_j, what exam can we expect?

points = ?

1. Find the k closest x_i in our data set:
   $(x_{NN1}, \ldots, x_{NNk})$ is the ordered list of nearest neighbours (x_{NN1} closest).
   For any other x_i in the data set: $\text{distance}(x_i, x_j) \geq \text{distance}(x_{NNk}, x_j)$.

2. Predict exam points as:

$$\hat{y}_j = \frac{c_{j_{\mathrm{NN1}}} y_{\mathrm{NN}_1} + \ldots + c_{j_{\mathrm{NNk}}} y_{\mathrm{NN}_k}}{\sum_{i=1}^{k} c_{j_{\mathrm{NNi}}}}$$

points = 70

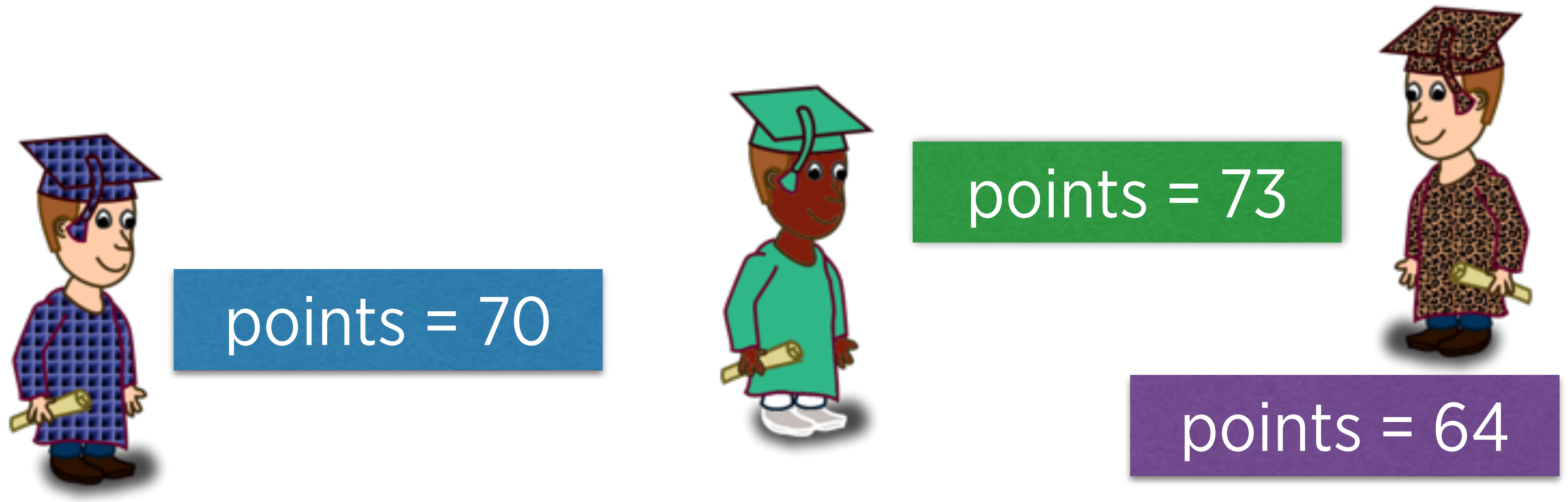
points = 73

points = 64

Setting the weights

- We want to set a weight c_{jNNi} to be small when distance(x_{NNi}, x_j) – the distance between query and example – is large.
- We want to set a weight c_{jNNi} to be large when distance(x_{NNi}, x_j) is small.
- e.g.,

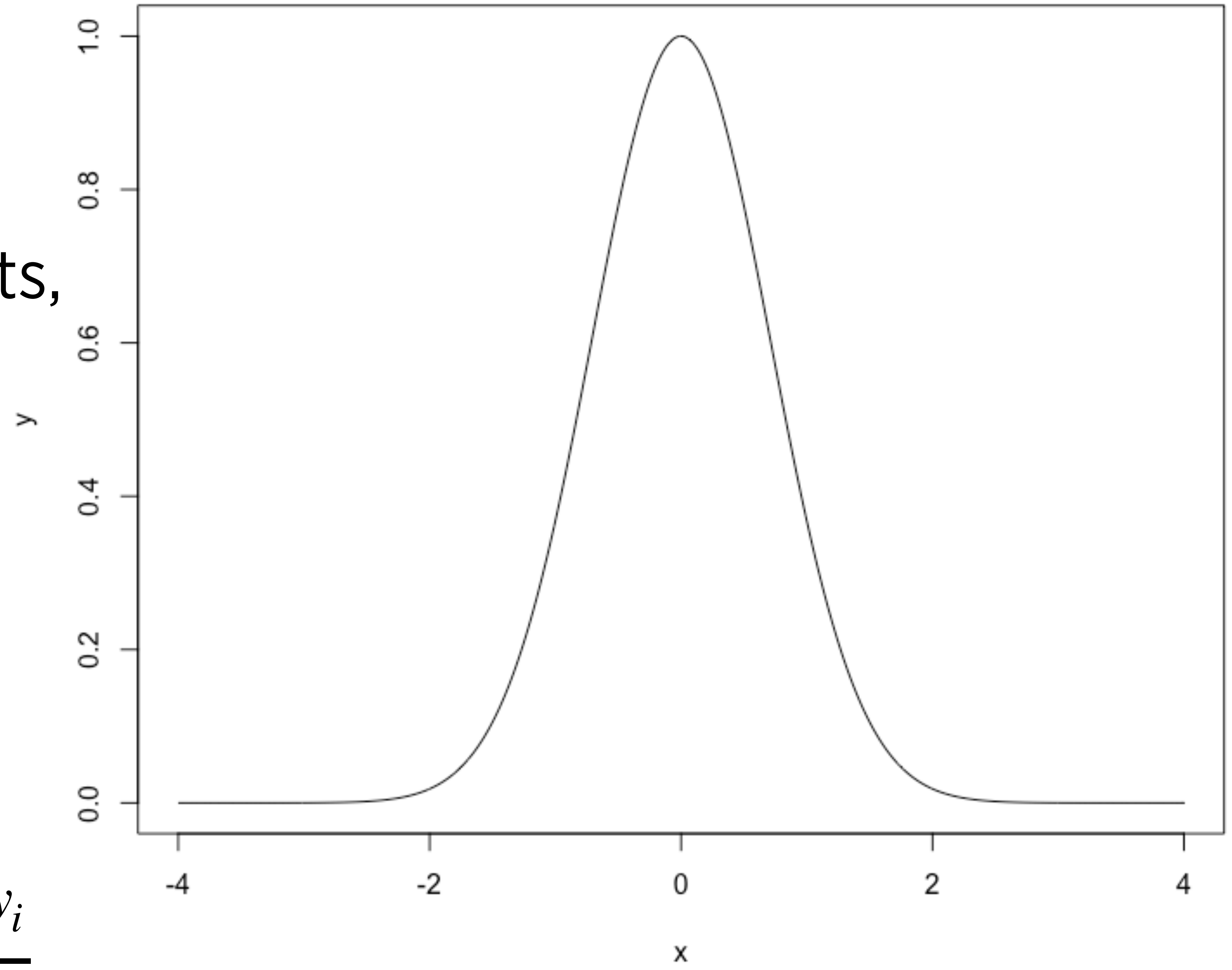
$$c_{j_{\mathrm{NNi}}} = \frac{1}{\mathrm{distance}(\mathbf{x}_i, \mathbf{x}_j)}$$



Kernel regression

- If we place smaller weights on less similar examples …
- … we could actually use all points in the training set.
- We use a function to compute similarities and serve as weights, then calculate weighted average.
- This is called kernel regression.
- Different kernels possible - but Gaussian kernel is a popular choice: $k_h\left(\mathbf{x}_i, \mathbf{x}_j\right) = e^{-h\|\mathbf{x}_i - \mathbf{x}_j\|^2}$
- Nadaraya–Watson kernel regression: $\widehat{m}_h(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) y_i}{\sum_{i=1}^{n} K_h(x - x_i)}$

where $\widehat{m}_h(x) = E(y \mid X = x)$.



e^{-x^2}

Characteristics of nonparametric approaches

- k nearest neighbours (and kernel regression) are nonparametric approaches to regression.
- Characteristics of nonparametric approaches are
  - their flexibility, and that they make only few assumptions about f(x)
  - Complexity of computation can grow with number of observations
- basic kNN has no cost in training – the costs are all in the query, though.
- There are many other nonparametric approaches.

Complexity of k nearest neighbours

- The naïve approach is to search through all data points
- For a single prediction of x_j: go through all data points $x_1, \ldots, x_N$.
- for 1-NN: O(N) distance computations per prediction
- for k-NN: O(N log k) distance computations per k-NN query

Summary

Strength, Weakness & Parameters

- Strength
  - the model is very easy to understand
  - often gives reasonable performance
  - a good baseline method to try before more advanced techniques
  - no training time

- Weakness
  - prediction is slow
  - inability to handle many features
  - need to carefully preprocess the data

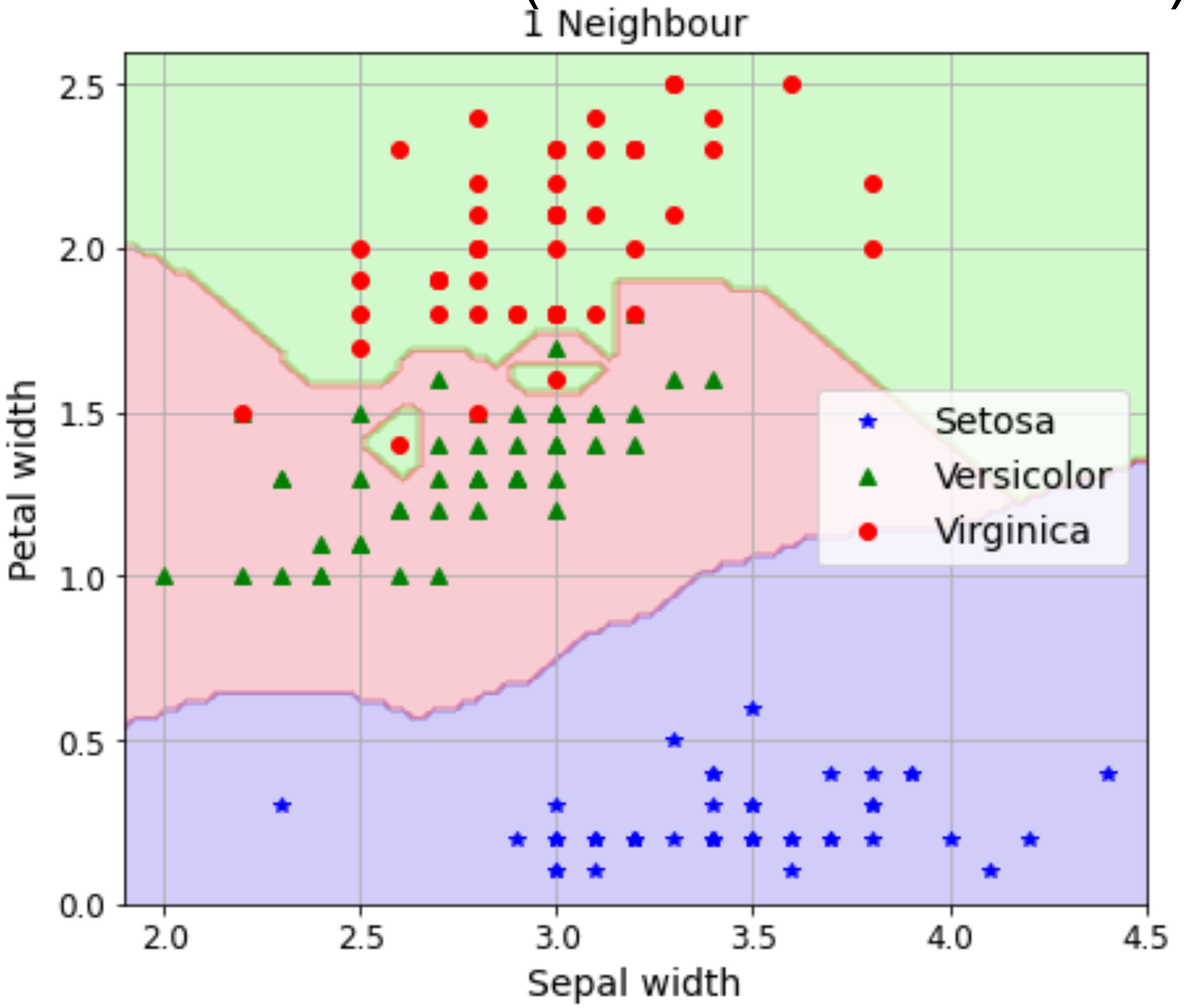
- Parameters
  - number of neighbours
  - how to measure distance

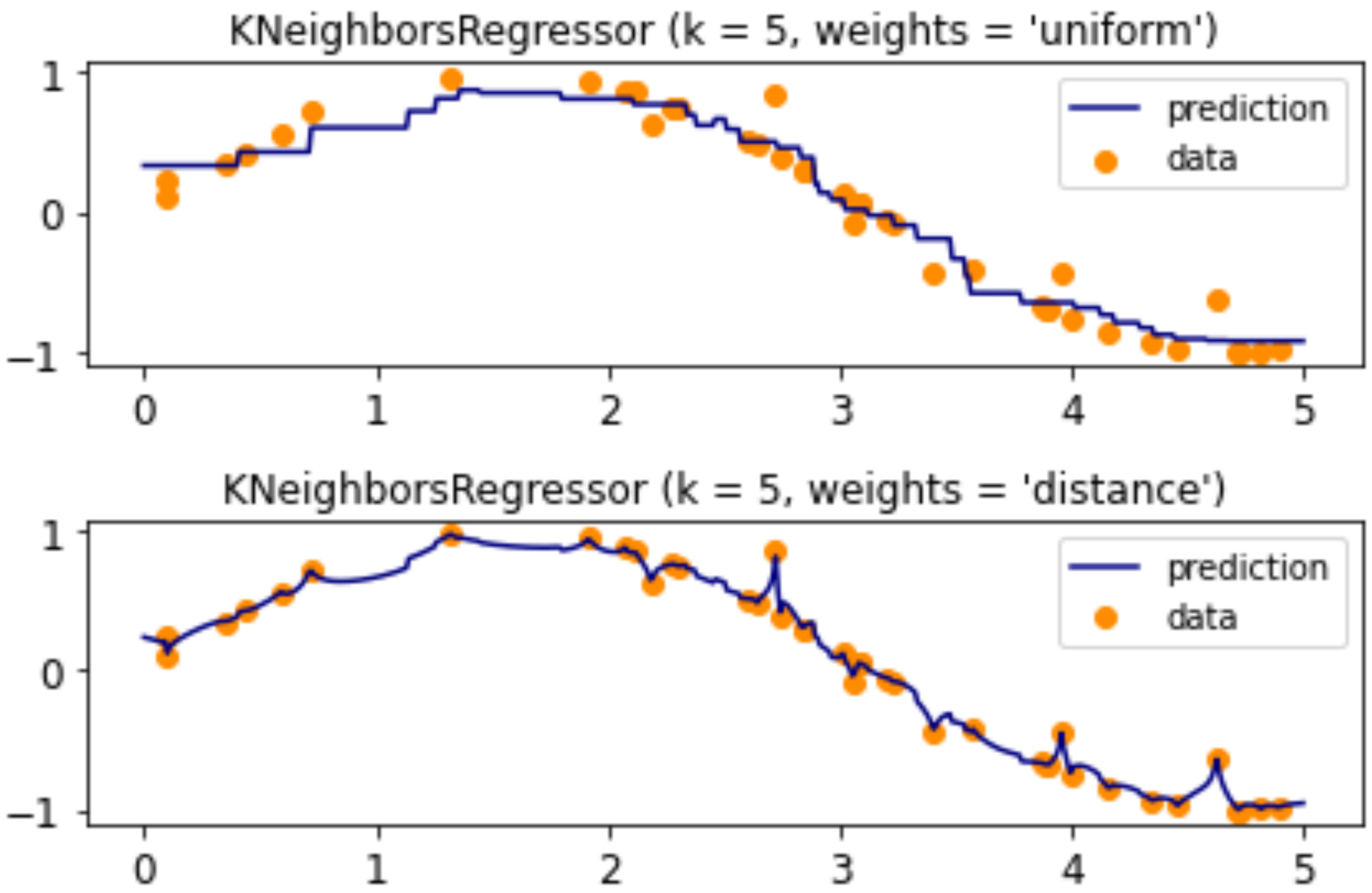
Summary

- Motivation and characteristics of k nearest neighbours
- How do kNN classification and regression work?
- What are distance measures and how do they affect prediction
- 1D cases and higher dimensions
- How to choose the best k value?
- Weighted NN and kernel regression idea
- What makes an approach non-parametric?

Module 07 Tutorial - nonparametric methods

The code is available in vUWS named: *tutorial - module07.py*
Section 1&2 (code demonstration): use KNN for classification and regression.





Module 07 Tutorial - nonparametric methods

Section 3: Based on the exercises above, do the following task.

Task: use KNN to classify handwritten digits:

(a) Create subsets of 2 digits from both the training and test data (pick two digits, e.g. 3 and 8). Classify the test data using the training set, with $k = 3$.

(b) Pick a misclassified example. How do the nearest neighbours look like?

(c) Use the training data to create training and validation sets. Use the validation set to find the best value for k. Use the k you selected, compute the error on the test set. How does it compare with results from part (a)?

(d) Apply the selected k to classify all digits of the test set.