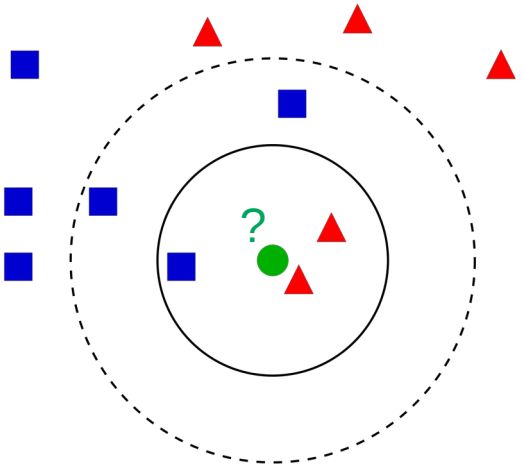
R-course:
Machine Learning using R

K-nearest neighbor algorithm

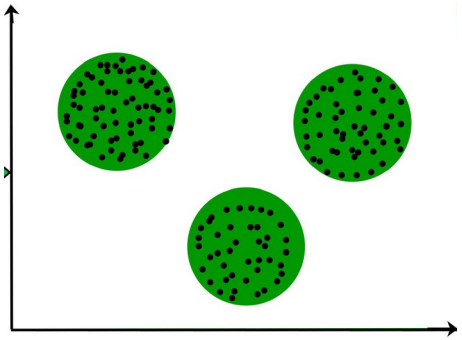


Yannick Rothacher

Zürich, 2021

Recap: k-means clustering

- The k-means algorithm is a **partitional clustering** method
- Often there is not a “correct” solution in k-means clustering
  - The goal is to find out whether there is a cluster-structure present in the data (find hidden patterns)



- Such an approach is generally referred to as **unsupervised** learning
- In contrast, in **supervised** learning we try to predict the correct class/value of a new data point
  - Techniques, which try to predict the correct class of data are referred to as **classifiers**

Supervised learning: Classification

- Classifiers can be used to predict all kinds of data labels
  - Who will win in the next football-worldcup?
  - Will an operation be successful?
  - Which type of tumor is shown on a medical scan?
- Predictions are always based on training data:

Hours studied	Score last year	IQ	**Exam result**
1	20	101	**failed**
20	99	105	**passed**
12	54	99	**passed**
...	...	...	...

Target variable

New observation:
Will this person pass or fail?

Hours studied	Score last year	IQ	**Exam result**
25	70	98	?

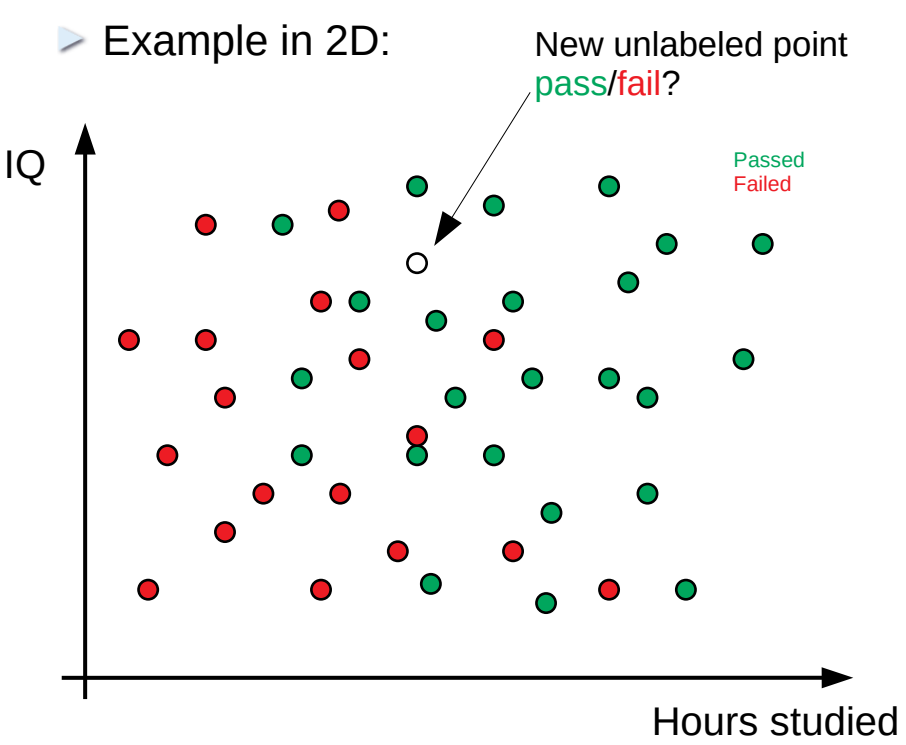
Supervised learning: Classification

- The workflow for classifiers is always similar
- First, a model is fitted to **training** data, for which the true class labels are known
- Afterwards, the model is applied to new observations to predict their (**unknown**) class labels
- Thus, we always first need training data to **train our model**
- There are different statistical models, which can be used as a classifier
  - e.g. **logistic regression** is a simple classifier usually applied when the target label has two levels (e.g. predict sex: male/female)
  - **K-nearest neighbor** is an easy to understand classifier and therefore a good introduction to the topic of classification

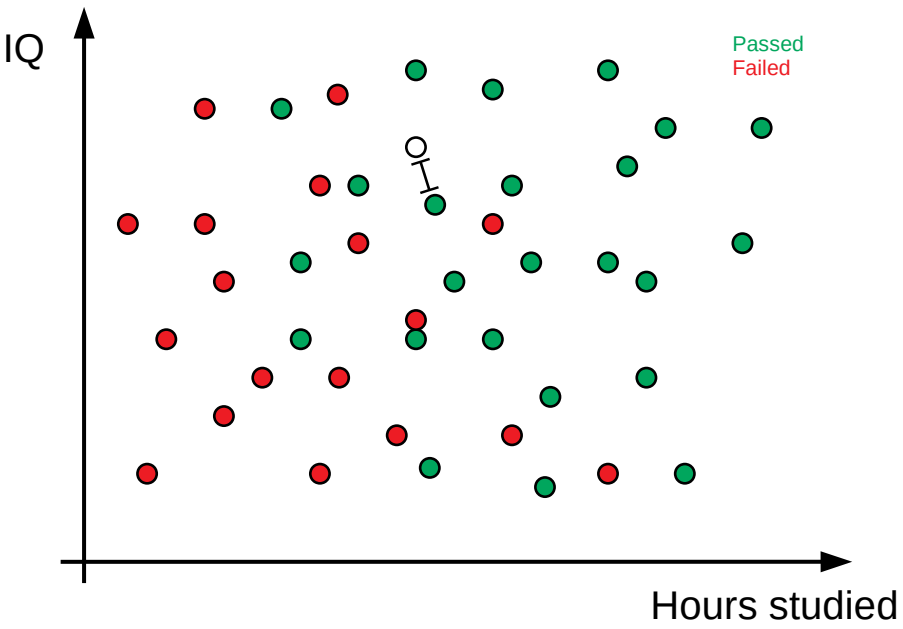
Hours studied	Score last year	IQ	**Exam result**
1	20	101	failed
20	99	105	passed
12	54	99	passed
...	...	...	...

K-nearest neighbor (KNN)

- Not to confuse with k-means algorithm!
- Idea: Classify a new data-point based on the labels of the neighboring training data points
  - Example in 2D:



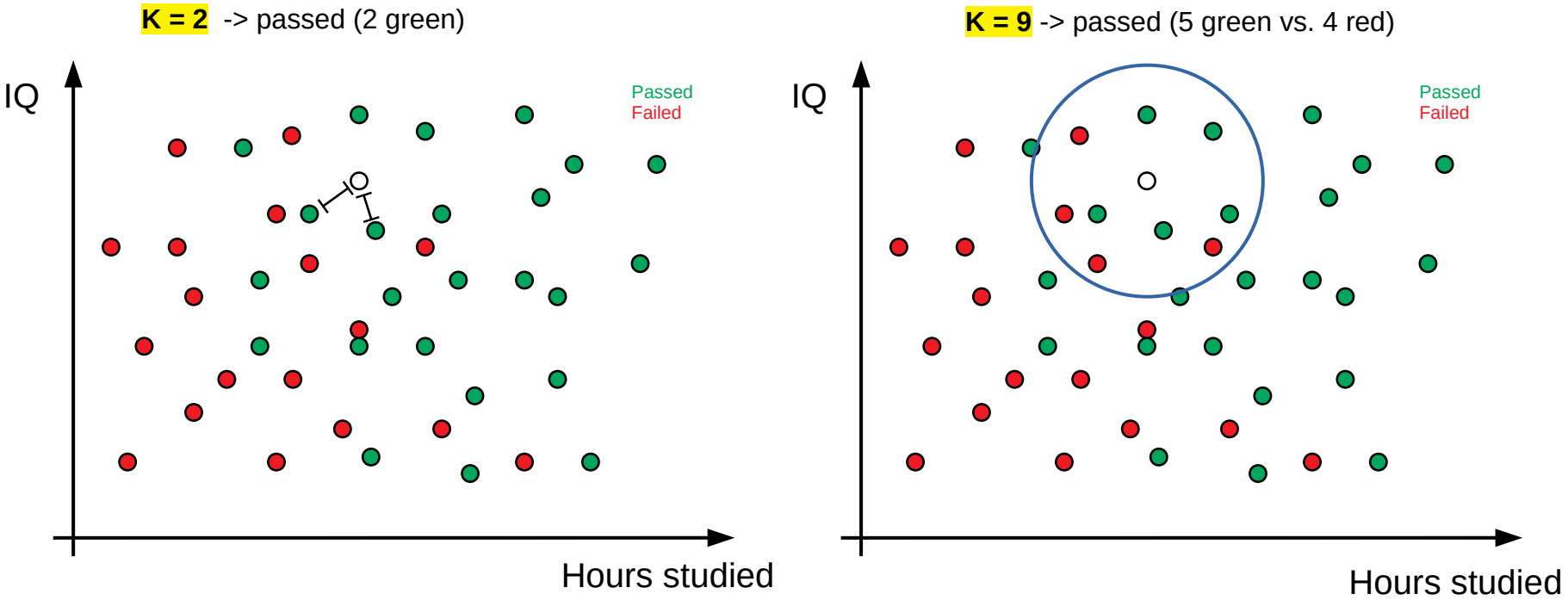
K = 1
"Which label is my **closest neighbor**?"
(e.g. based on Euclidean distance)



- Assign the label "passed" because closest neighbor is "passed" as well

K-nearest neighbor (KNN)

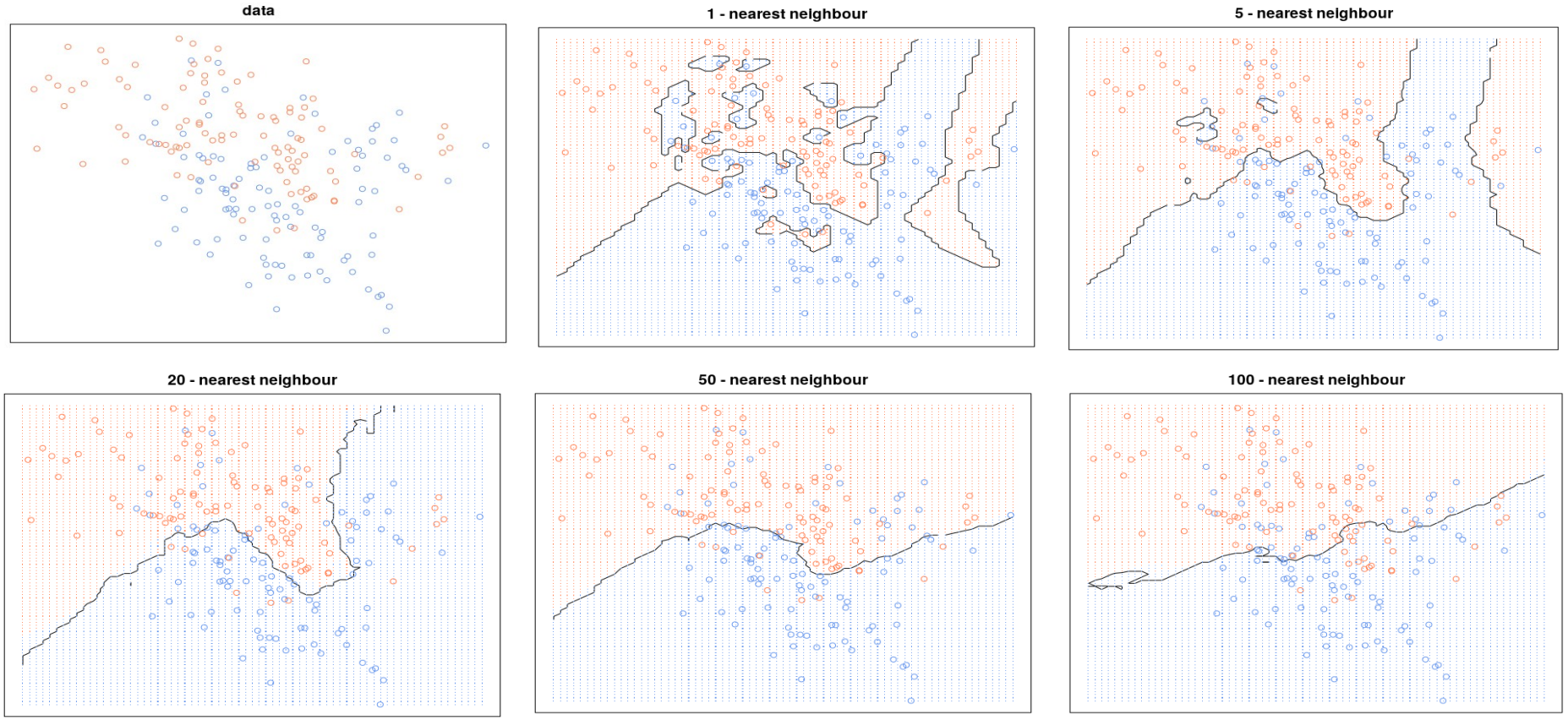
- The parameter k determines how many neighbors are considered (use the majority vote as prediction):



- The prediction may change depending on k
- There is the possibility of ties, **what could be done to deal with ties?**

KNN - Decision boundaries

- The decision boundaries for new data points depend on how many neighbors are considered:

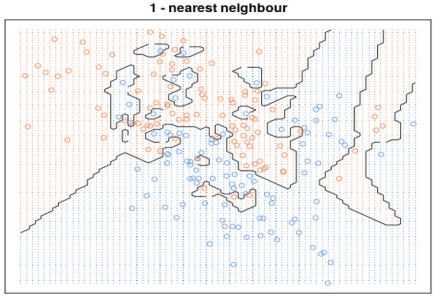


- The decision boundary becomes **simpler** with larger k
  - What is the **best** value of k?

How to choose k

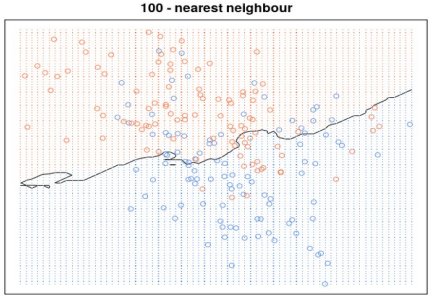
- What is the best value of k?

k = 1



Which is better?

k = 100



- In order to decide what the best value for k is we want to **measure** how "well" the classifier is performing
  - How can we do that?
- Simple idea: Check how correctly the classifier is labeling the training data (**Training error**)
  - Confusion matrices (for k=1 and k=100):

k = 1

KNN prediction	true label blue	true label red
blue	100	0
red	0	100

k = 100

KNN prediction	true label blue	true label red
blue	66	26
red	34	74

Train- and test-error

- Calculate the training error (number of miss-classifications divided by total number of predictions):

k = 1

KNN prediction	true label: blue	true label: red
blue	100	0
red	0	100

→ 0/200 = **0%**

k = 100

KNN prediction	true label: blue	true label: red
blue	66	26
red	34	74

→ 60/200 = **30%**

- Looking at the training error is not that informative. K=1 will always have a 0% training error rate.
- Because we want to use the classifier to predict the class of new observations (not included in the training set), it makes sense to calculate the error rate for unseen data (**Test-error**)

Train- and test-error

- **Test-error** for 40 new observations (for k=1 and k=100)

k = 1

KNN prediction	true label: blue	true label: red
blue	12	8
red	3	17

→ 11/40 = **27.5%**

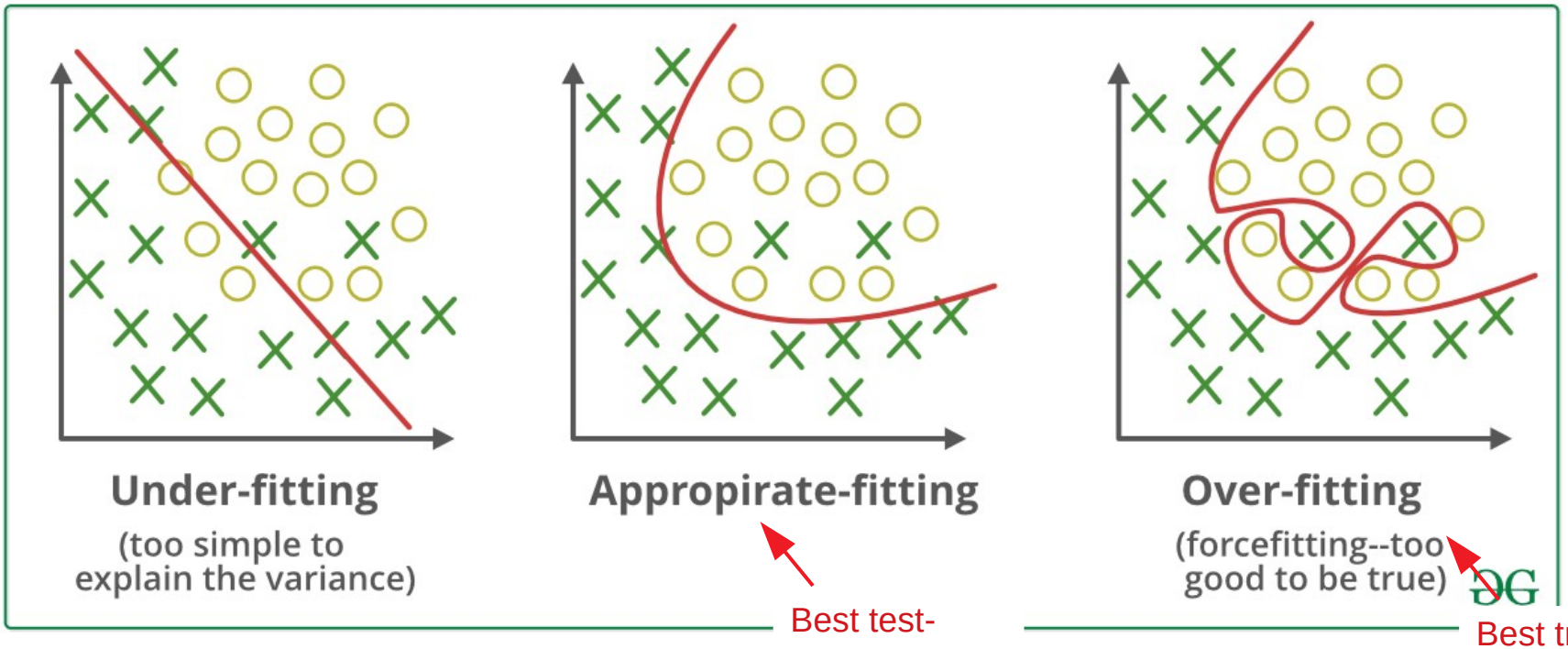
k = 100

KNN prediction	true label: blue	true label: red
blue	13	7
red	2	18

→ 9/40 = **22.5%**

- Now the two classifiers both perform equally well (in this case k=100 even slightly better)
- The goal is to keep the **test-error** as low as possible
- Most probably, setting k=1 and setting k=100 are both not ideal
  - k=1 is too flexible (**over-fitting** the training data) and k=100 is perhaps too simple (**under-fitting** the training data)

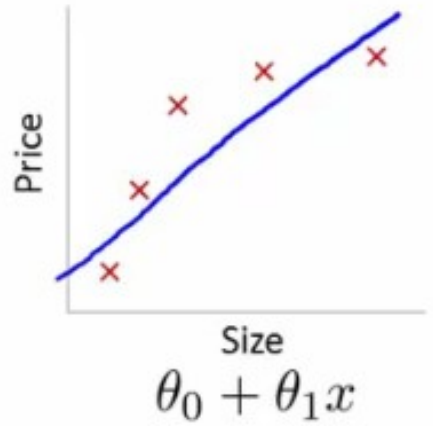
Over- vs. under-fitting

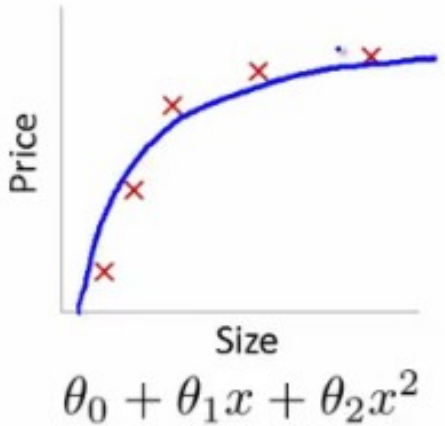


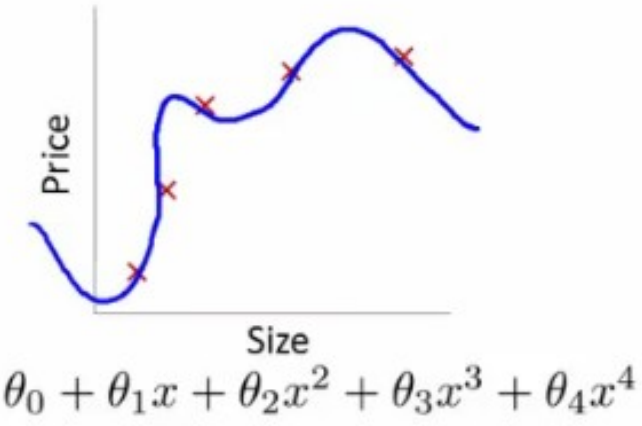
- Imagine you apply repeatedly an over- or under-fitting model to collected data:
  - Under-fitting will show **high bias** and **low variance**
  - Over-fitting will show **low bias** and **high variance**

https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf

Over- vs. under-fitting (in regression)





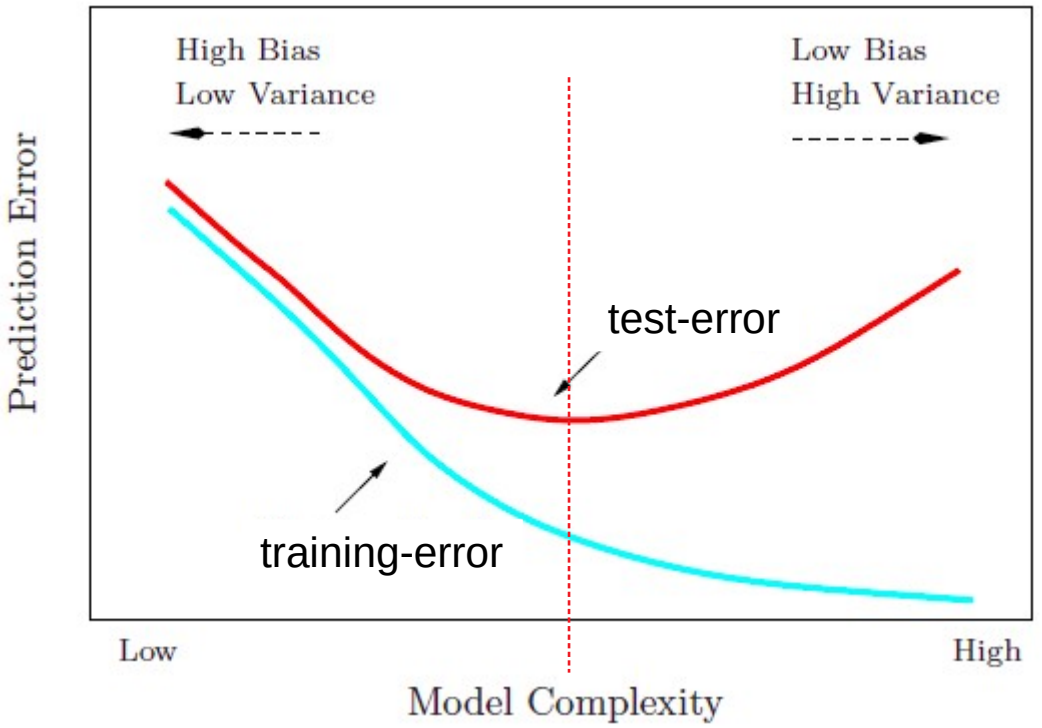


- Imagine you apply repeatedly an over- or under-fitting model to collected data:
  - Under-fitting will show **high bias** and **low variance**
  - Over-fitting will show **low bias** and **high variance**

https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf

Where is the optimum between over- and under-fitting?

- Test-error should be as small as possible



- Dashed, red line shows the **ideal model flexibility**
  - Everything to the right of dashed line is over-fitting
  - Everything to the left of dashed line is under-fitting

Outlook: Cross-validation

- It is important to **estimate the test-error** of a classifier to judge its performance
- Ideally, one uses a **training data set** to fit the model and separate **test data set** to assess its performance
  - Often a separate test data set is not available
- Later we will see what methods are used to estimate the test-error using **one single data set** (cross-validation)

KNN in R

```
library(class)
knn.pred <- knn(train = iris[,1:4],  # use the iris data as training data
                test = iris[, 1:4],  # knn always needs a specified test set
                cl = iris$Species,   # correct labels of training set
                K = 5)               # parameter k
# Confusion matrix:
confT <- table(knn.pred, iris$Species)
confT
knn.pred     setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         2
  Virginica       0          3        48

# Training error:
missT <- confT
diag(missT) <- 0  # turn diagonal values to zeros
missCount <- sum(missT)   # count the miss-classifications (5)
trainErr <- missCount/nrow(iris)
trainErr
[1] 0.03333333
```

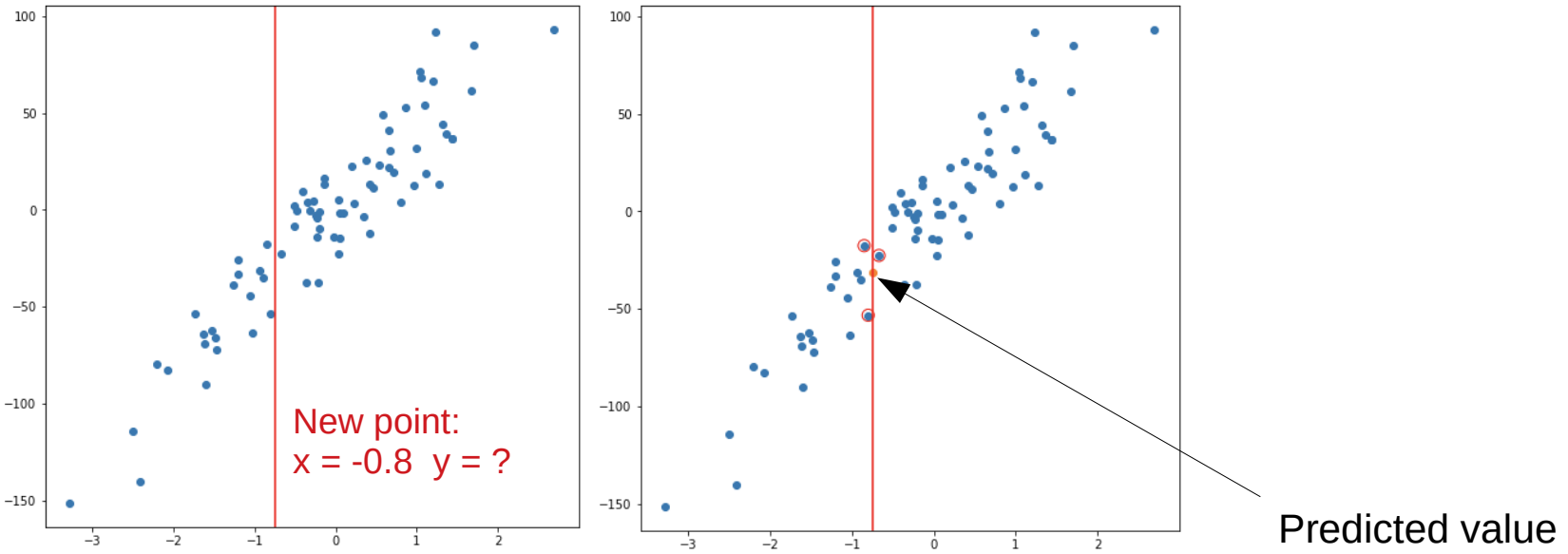
Because we do not have a test set, we reuse the training data as test data

Standardization in KNN

- Similar to the case of PCA it often makes sense to standardize the data before applying KNN classification
  - Why could standardization be important?
- **Example**: If one variable has a much larger variance than all the others (e.g. ranging from 0 - 100’000, other variables: 0 - 0.1), it will be much more important in determining the Euclidean distance between two data points.
  - Other variables are neglected!
- Through standardization, all variables have equal variance
  - In R: use the scale() function to standardize numeric vectors

KNN for regression

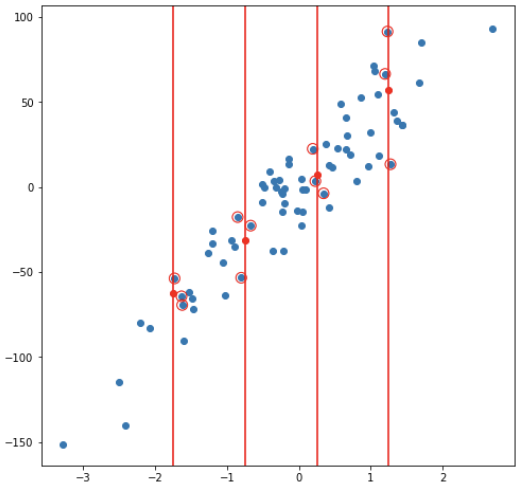
- KNN cannot only be used for classification
  - Can also predict **numerical values**
  - Instead of taking the majority vote of neighbors, take e.g. the **mean of k neighbors**

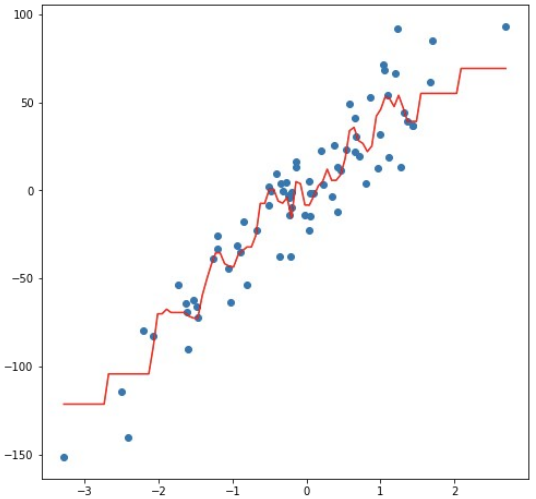


- E.g. univariate data (figures above): Use the average of **k=3** closest neighbors (with respect to x value) as a prediction for y value
  - Can even use the distances as **weights** for the mean

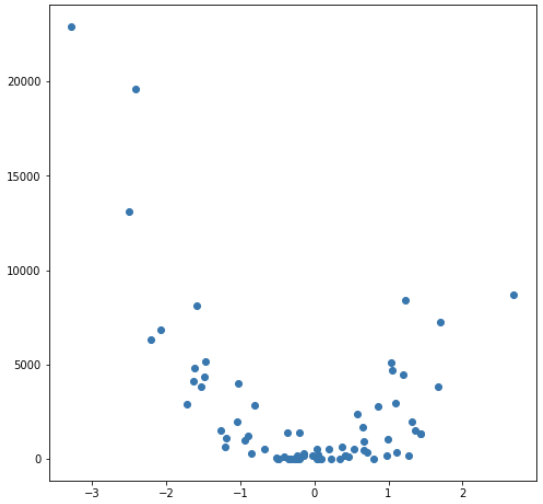
https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955

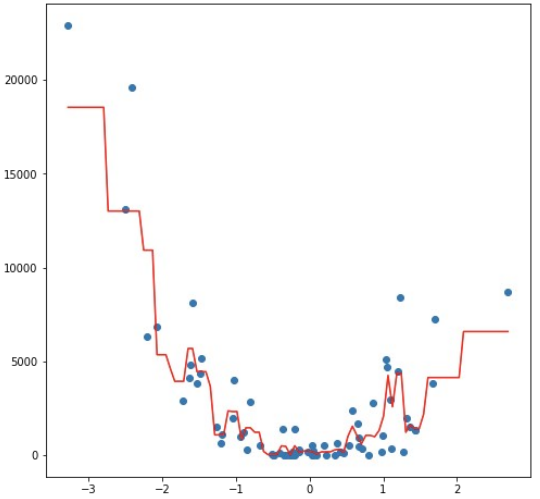
KNN for regression





- KNN for linear relation
- Result is not a clean line but still general pattern is captured





- Advantage of KNN for regression:

  Simple algorithm which can capture non-linearity quite well

https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955