# Simple Classifiers: using linear and kNN approaches

Alípio Jorge

February 2021

# Where are we?

- A program **learns** when it improves with experience (Mitchell simplified)
  - improves: makes **better decisions** in a task
  - experience: **data**
- The true decision function is **unknown**
- ML as **function approximation**
  - use the observed data to **induce** a function

# Machine Learning tasks

- Regression
- Classification
- Recommendation
- Clustering
- Outlier detection

# Classification

**Classification** is perhaps the most popular machine learning task.

- Distinguish our enemies from our allies
- Poisonous mushrooms from edible ones
- Good days to go hunting from good days to stay put
- Sort different kinds of seeds for human feeding, animal feeding, sowing and throwing away.

# Classification

In many aspects, it is very similar to **Regression**.

- Both can be used for **understanding** the past
- For **real time** decision and action
- And for **predicting** the future

# Classification

We can also see classification as **function approximation**.

- An underlying unknown function that maps objects to **classes** $C_i$

$$f(\mathbf{X}) : \{input \quad domain\} \rightarrow \{C_1, C_2, ..., C_K\}$$

- The aim is to find a **classifier**: a function that approximates $f(\mathbf{X})$

# Classification

We want to discover a function $\hat{f}(\mathbf{X})$ that approximates the real function.

- The discovery process is done by **learning** from examples
  - observed **data**.

Questions:

- What **kind of function** do we consider?
- What is a **good approximation**?
- How do we **obtain** the function from data?

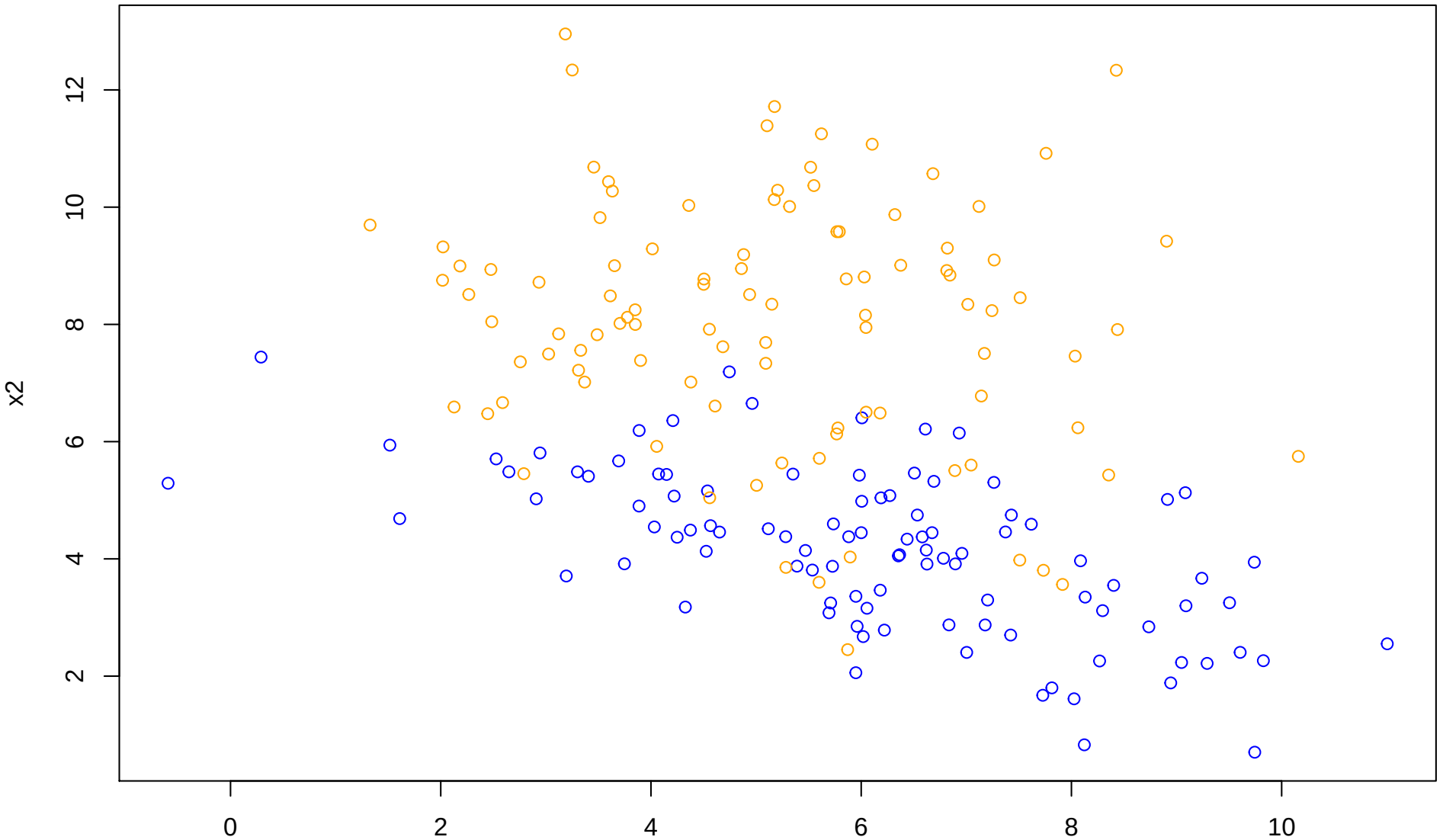
# Classification

## What we will do next

- We simulate a dataset with two classes
- We see how different approaches work
- We observe the **decision boundaries**

# Classification

First we simulate a classification dataset, with **two classes**.



# Classification > least squares

We now want to obtain a function that is a **good approximation** of the unknown original function.

- If we represent each class as a number, we can treat the **discrete** function as a **continuous** one.
- In our case, the class blue will be represented by 0 and orange by 1.

| | x1 | x2 | y |
|---|---|---|---|
| 1 | 5.535692 | 3.809123 | 0 |
| 2 | 7.618419 | 4.591018 | 0 |
| 101 | 5.791766 | 9.581688 | 1 |
| 102 | 7.733957 | 3.805342 | 1 |

# Classification > least squares

## Ad-hoc approach

We can use the least squares approach to learn a **linear** classifier.

- Learn a regression model that finds a regression plane for the multivariate regression problem.
- Given a point $(x1, x2)$, we use the regression model to estimate $y$
- if $y < 0.5$ the class is **blue**, otherwise the class is **orange**.

# Classification > least squares

## Ad-hoc approach

For example:

- Let $y = -0.4 + 0.01x_1 + 0.14x_2$
- For point $(6, 4)$ we have:
- $y = -0.4 + 0.06 + 0.56 = 0.22$
- Class is **blue**

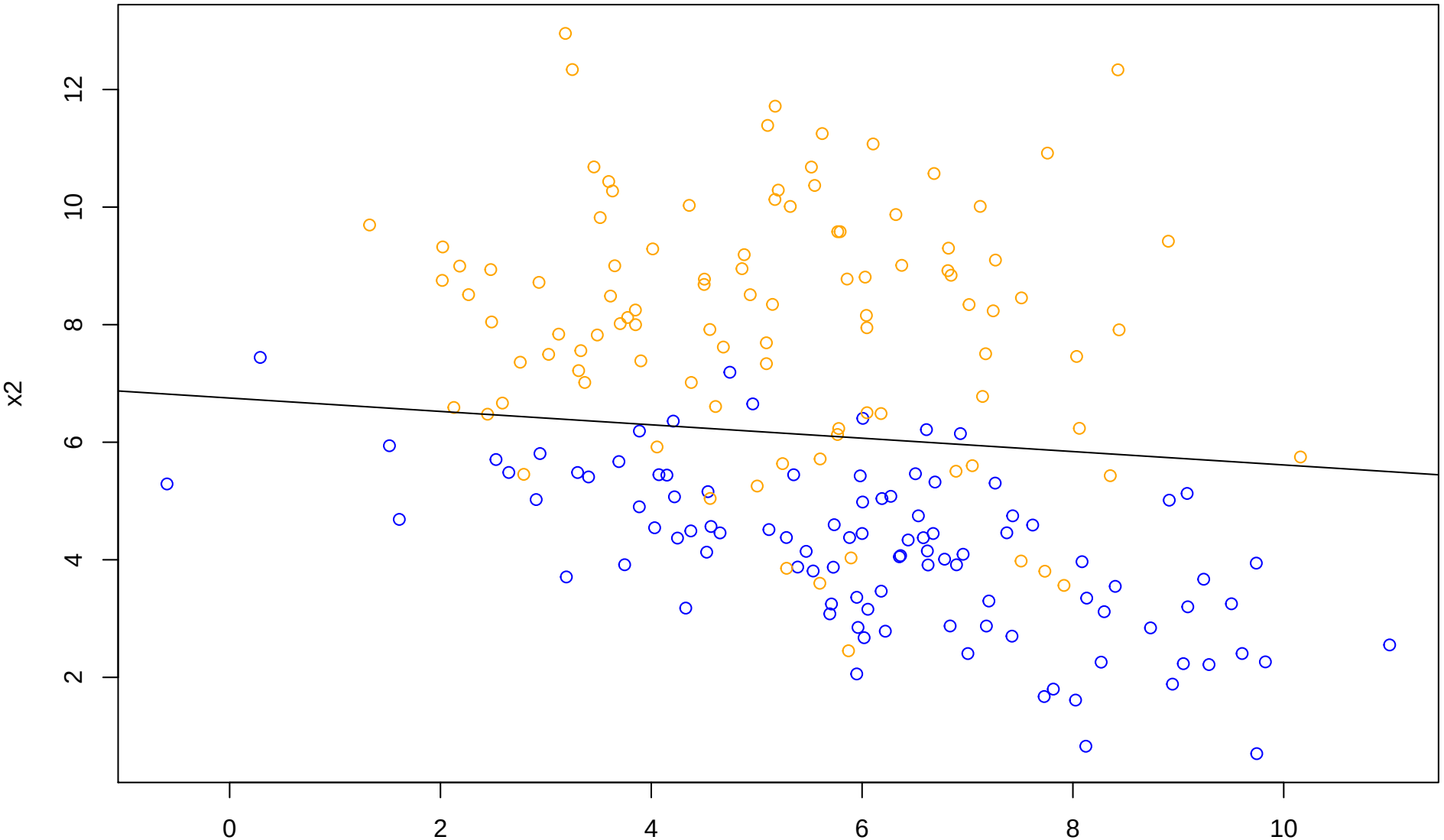
# Classification > least squares

- Visualize the **decision boundary**
  - the line that separates the classes
- How is the decision boundary obtained in this case?
  - The linear model is a plane with equation $y = a_0 + a_1 x_1 + a_2 x_2$.
  - The boundary will be defined by $y = 0.5$.
  - The line of $x_2$ as a function of $x_1$ is:

$$a_0 + a_1 x_1 + a_2 x_2 = 0.5 \equiv x_2 = \frac{0.5 - a_0}{a_2} - \frac{a_1}{a_2} x_1$$

# Classification > least squares

- Plot the data and plot the **decision boudary**.



# Classification > least squares

What is the actual **classification model**?

```
m$coefficients
```

```
(Intercept)          x1          x2
-0.48636124  0.01662712  0.14609943
```

```
lscm <- function(regmodel, observation){
  val <- predict(regmodel,observation)
  sapply(val,function(v) if(v<0.5) 'blue' else 'orange' )
}

lscm(m,data.frame(x1=c(4,4),x2=c(8,5),y=NA))
```

```
       1        2
"orange"   "blue"
```

# Classification

## Some notes

- Linear regression optimizes the **regression problem**
  - Not necessarily the classification problem
- Is the threshold of 0.5 **always** the best option?
- Other linear classifiers **optimize classification**
  - Logistic regression
  - Linear discriminant
  - SVM with a linear kernel
- We will see this in future lectures

# Classification

Can we have a classification model with a better fit?

- We decided that the decision boundary was **linear**
- However, other classes of functions could find a more **accurate** boundary
- But remember, expressive classes of functions can also incur in **overfitting** more easily.

# Classification > Nearest Neighbours

Linear models can be optimal when we have **linear decision boundaries**.

- This is **typically** not the case.
- However, linear models can be very **useful**, even when it is not the case

Can we have a more **flexible** approach?

- The **Nearest Neighbours** approach is radically different
- It can flexibly adapt to decision boundaries with **arbitrary shapes**
- It can be used for **regression** and **classification**
- But **not without** disadvantages.

# Classification > Nearest Neighbours

Given a data set $(\mathbf{X}, \mathbf{y})$, the prediction $y$ of a new point $x$ is calculated as

$$\hat{\mathbf{y}}(x) = \frac{1}{k} \sum_{x_i \in N(x)} y_i$$

- $k$ is the **number of neighbours** used to obtain the estimate.
- $N(x)$ is the set of $k$ nearest neighbours of $x$
- The **distance function** in this case is the Euclidean distance
  - But we can use any distance/similarity function

# Classification > Nearest Neighbours

What about classification?

- A classifier can be obtained by **thresholding**:
  - if $\hat{\mathbf{y}}(x) < 0.5$ then $class = blue$
  - else $class = orange$
- But we can have it **directly** from classification data

$$\hat{\mathbf{y}}(x) = argmax_{C_j} \ \ \#\{ \ x_i \in N(x) \ \ | \ \ y_i = C_j\}$$

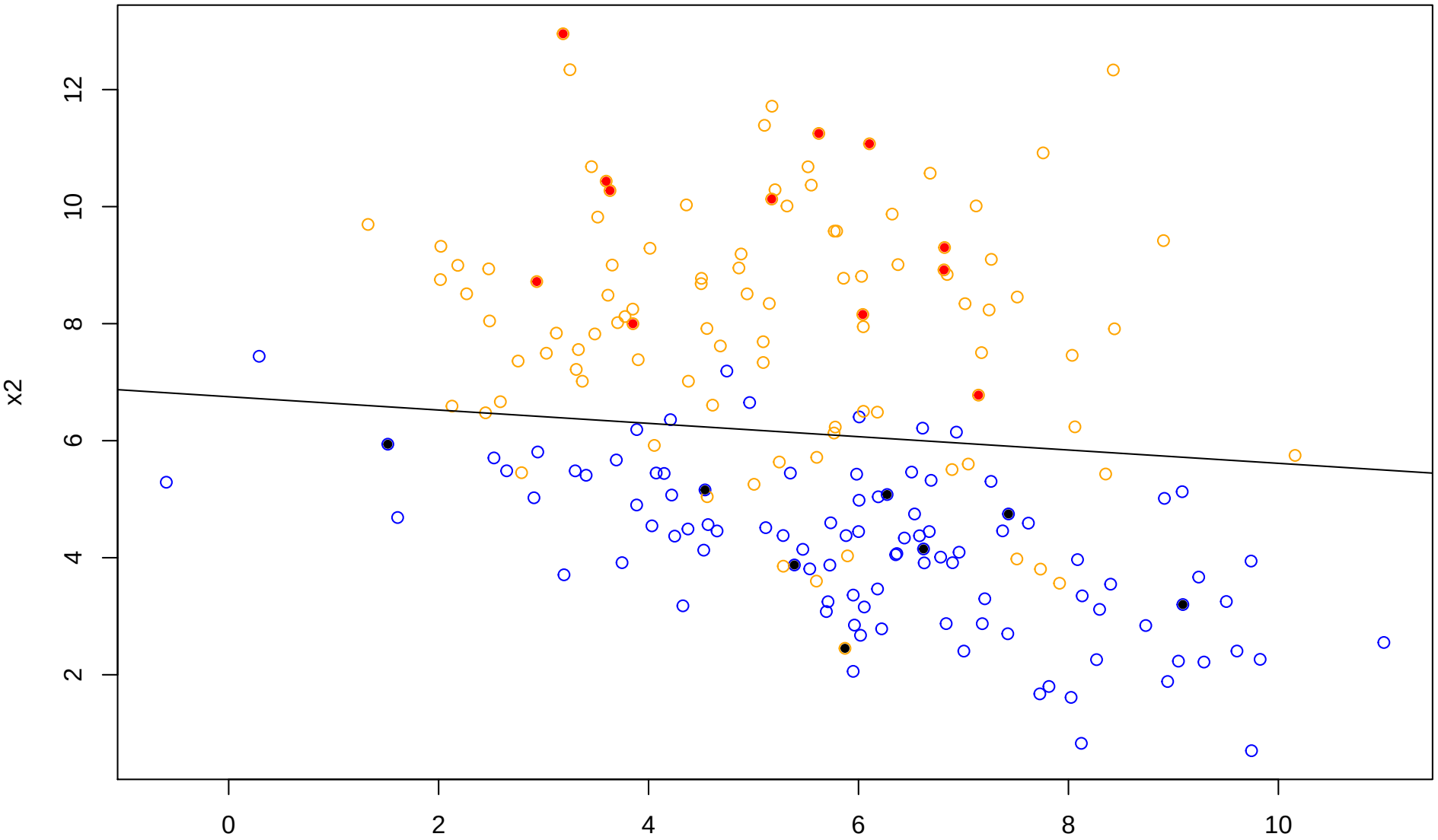
# Classification > Nearest Neighbours

Using function `knn` from the `class` library with 3 neighbours.

| | x1 | x2 | pred.classes |
|---|---|---|---|
| 90 | 6.619307 | 4.150111 | blue |
| 86 | 4.537817 | 5.158983 | blue |
| 115 | 6.040686 | 8.156749 | orange |
| 134 | 3.632576 | 10.274317 | orange |
| 124 | 7.143040 | 6.778603 | orange |
| 105 | 3.185431 | 12.954132 | orange |

# Classification > Nearest Neighbours

Looking at a sample of points we see they **mostly agree** with the linear model



# Classification > Nearest Neighbours

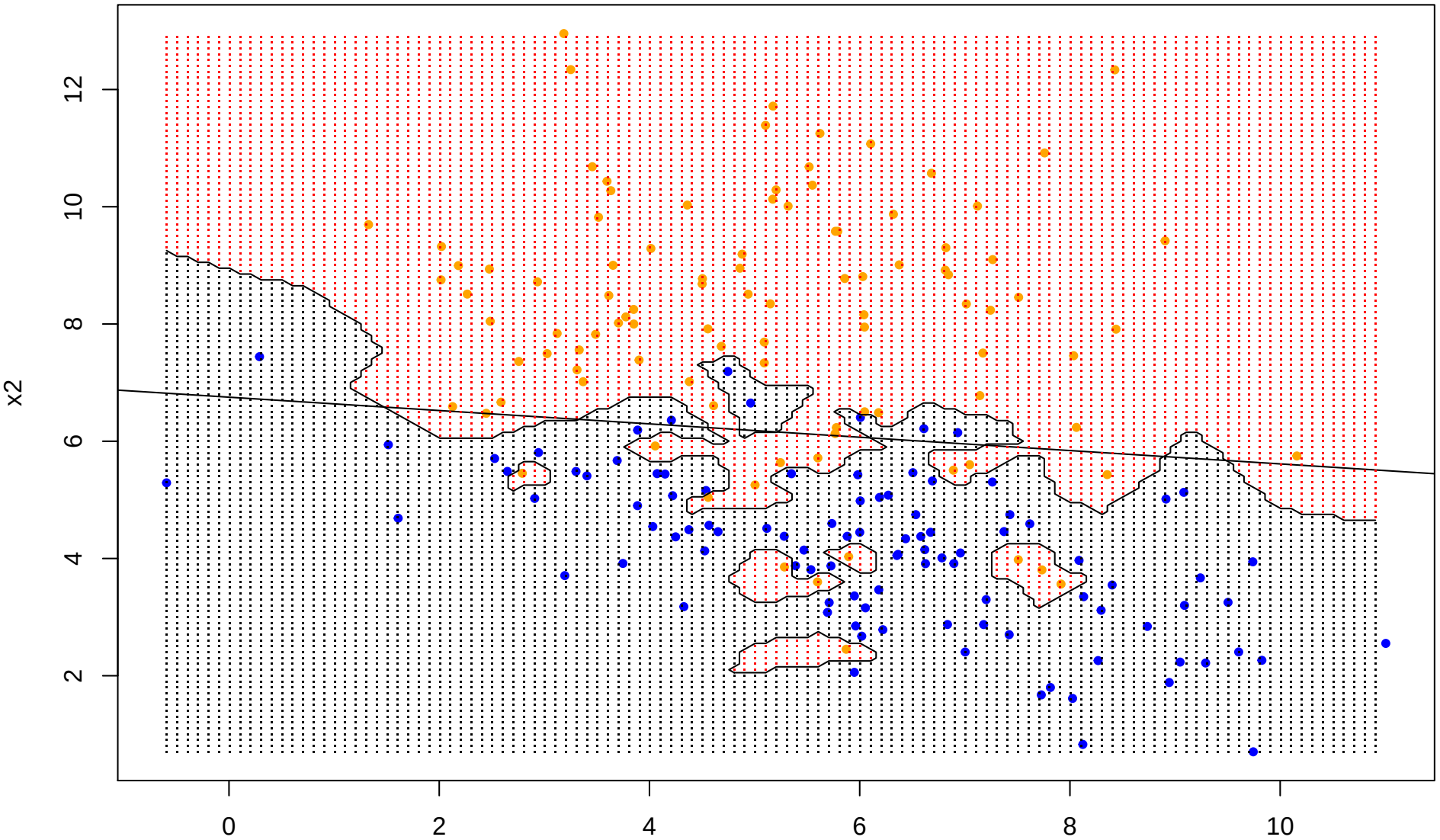
- What is the **precise** decision boundary produced by kNN?
- What is the effect of the **parameter k** on the decision boundary?

To answer these questions we will plot a fine **grid of points** over the data points.

- We will also compare with the decision boundary given by the linear model

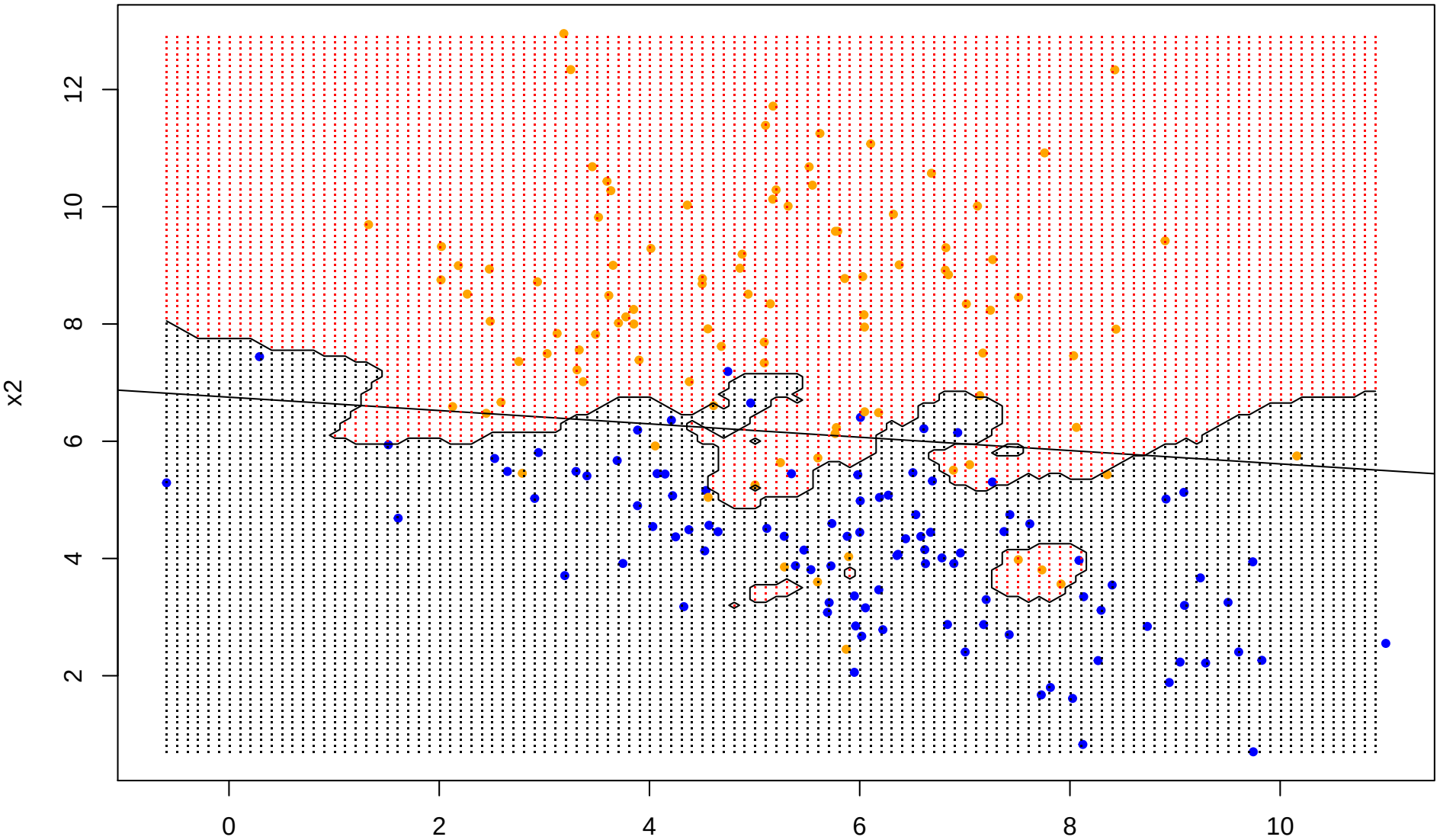
# Classification > Nearest Neighbours > Boundary

- $k = 1$



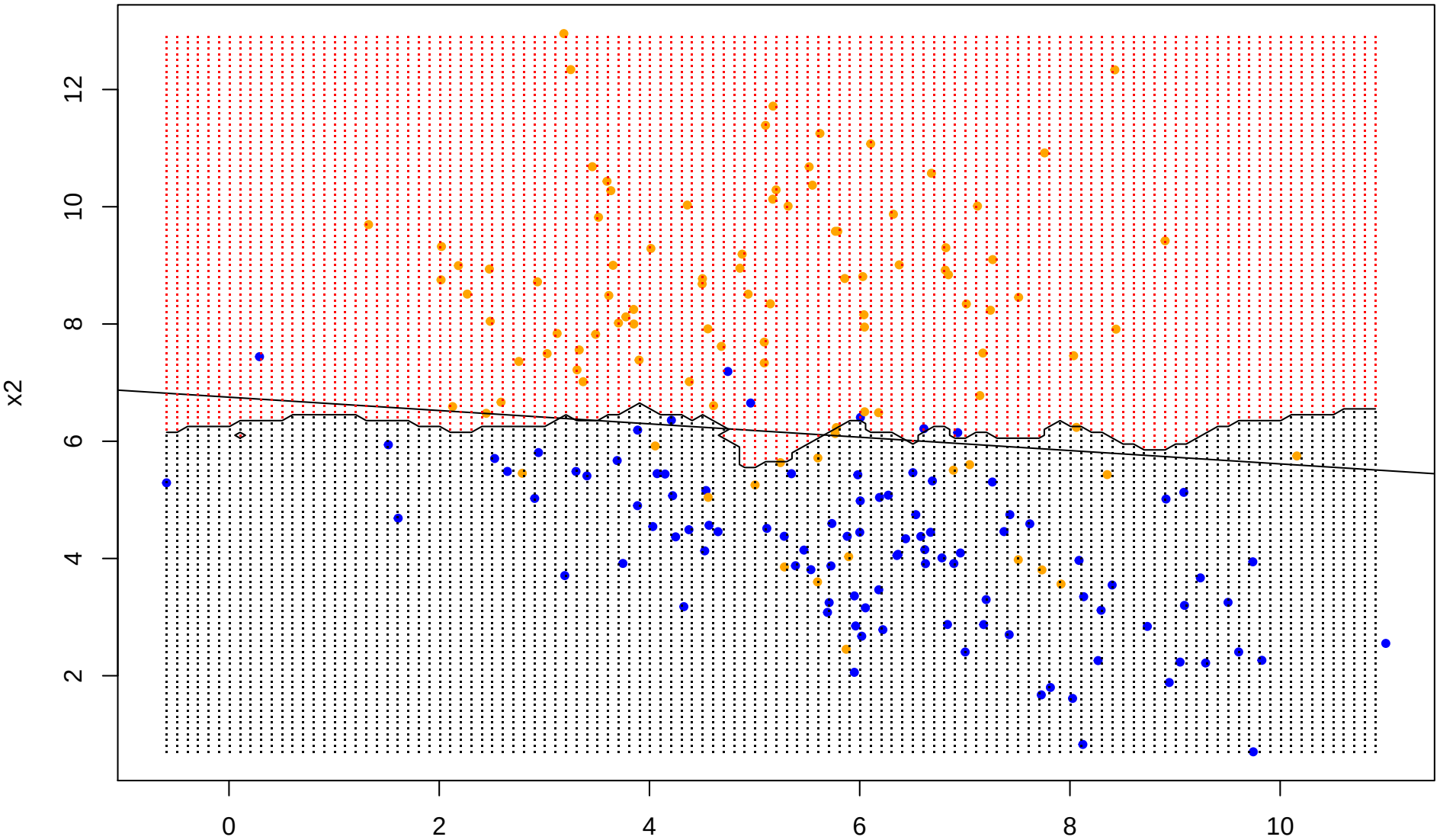
# Classification > Nearest Neighbours > Boundary

- $k = 3$



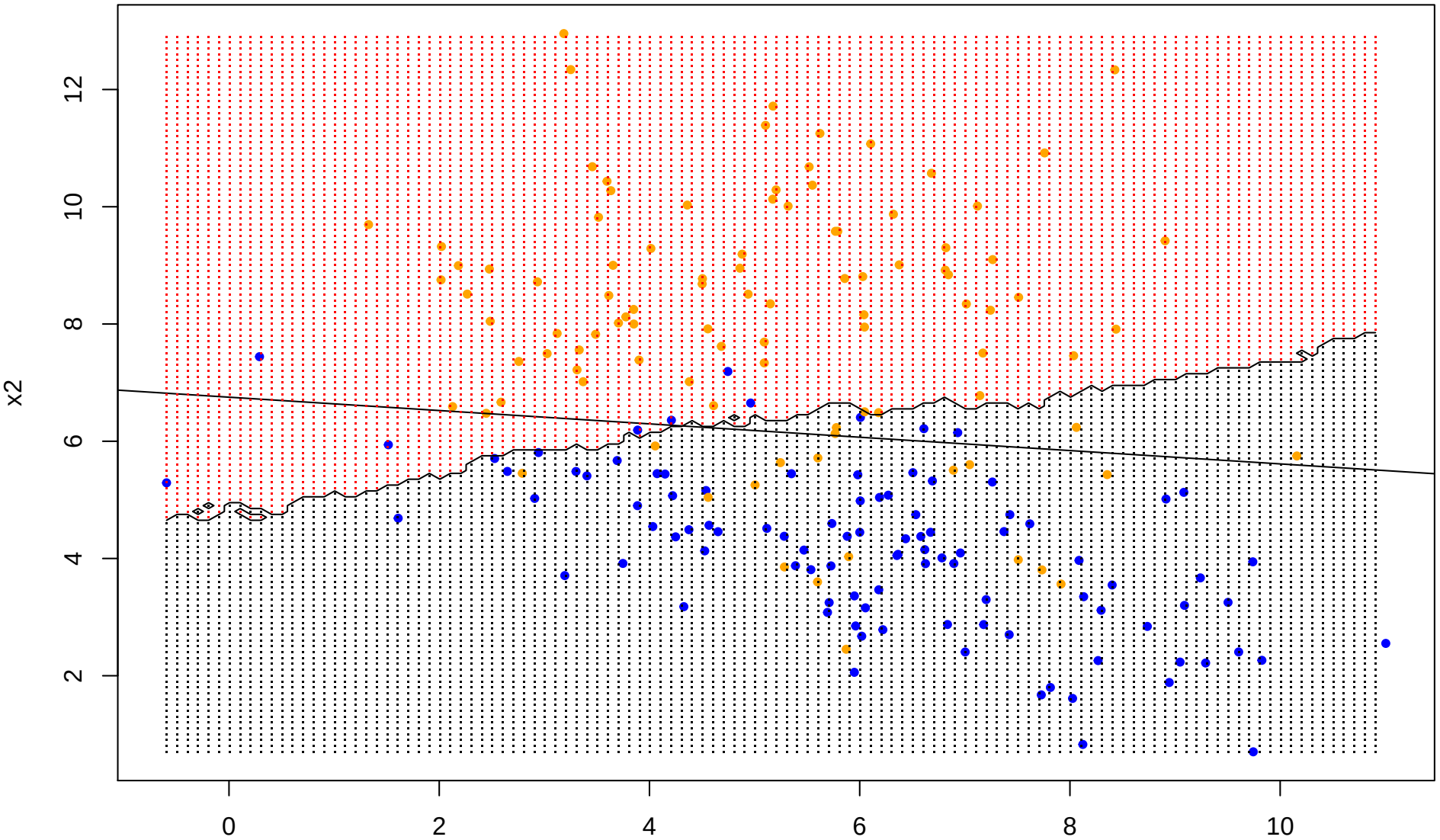
# Classification > Nearest Neighbours > Boundary

- $k = 15$



# Classification > Nearest Neighbours > Boundary

- $k = 100$



# Classification > Nearest Neighbours

Observations

- The boundary given by kNN is more detailed than the linear one
  - The linear boundary is **smooth**
  - The kNN approach has **fewer assumptions**
- In other words
  - The linear model approach has **high bias** and **low variance**
  - The kNN approach has **low bias** and **high variance**
    - Especially if $k$ **is low**

# Classification > Nearest Neighbours

- There is no model in kNN, we have to **store** the data.
  - This can be inefficient.
  - Sampling can be used to reduce the load.
  - Stored data points can be selected.
  - But there is **search effort** (*lazy learning*)
- Is this **learning**?
  - Where is **compression**?

# Classification > Nearest Neighbours

- Linear regression
  - coefficients are the parameters to **learn**
- What are the parameters of **kNN**?
  - $k$ is a **hyperparameter**
  - it is not learned, but it can be **tuned**

## The parameters of kNN

- Imagine the simpler scenario
  - $N$ points and $k$ neighbors
  - no neighborhood overlap
- How many different values?
  - $p{=}N/k$
- Our approximate function is defined with $p$ values
  - These values **would be the parameters**
- These parameters are not **learned**
  - They are estimated at **prediction time**
- But we do not have to worry about that with kNN

## kNN and Least Squares

- Can we choose the $k$ that **minimizes** RSS?
  - that would always be $k = 1$
  - Why?

## Summary

- ML can be seen as function approximation
- How do we find a **good approximation**?
  - **Linear Regression**:
    - Parameter estimation
    - Least Squares
  - We can adapt linear regression to binary classification
    - But this is in general not a good idea
  - **k Nearest Neighbors**:
    - Lazy learning
    - Value estimation in a neighborhood
- There are **other strategies**
- Why do they work?
  - We will soon look into **statistical decision theory**

# Activities

(1) Use the data set **iris** that comes with R (`data(iris)`). Consider the two variables 'Petal.Length' and 'Sepal.Length' and use them to plot the 3 classes in 2 dimensions. Apply kNN to predict the Species. Plot the decision boundaries for different values of $k$.

(2) Consider the data set **iris** and make the classes **setosa** and **virginica** as a single class **setinica**. Compare the application of the linear approach and of the kNN approach to this two-class data set.

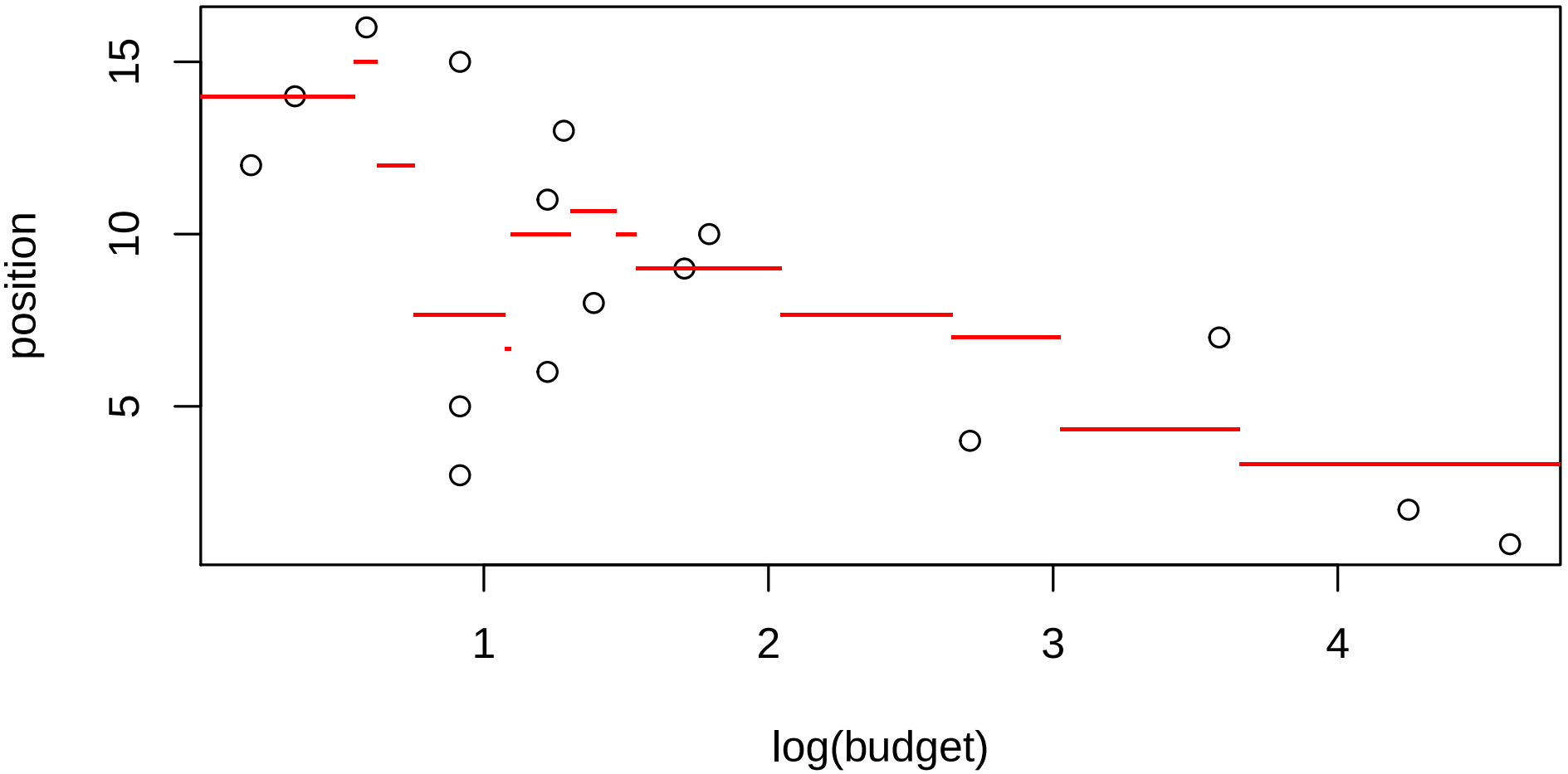
# Activities

(3) Visualize the result of kNN for the regression problem of predicting the final position of a football team given its budget. Use the function `knn.reg` from the `FNN` package to make the predictions using the *log* transformation. Plot the regression line produced by kNN.

```
budget <- c(100, 70, 36, 15, 6, 5.5, 4, 3.6, 3.4, 3.4, 2.5, 2.
position <- c(1,2,7,4,10,9,8,13,11,6,3,5,15,16,14,12)
```

# Activities

(4) Produce the following plot that "draws" the function discovered by *knn*.



# End of lesson

## Bibliography


- Hastie, T., Tibshirani, R.,, Friedman, J. (2008). The Elements of Statistical Learning, Second Edition. New York, NY, USA: Springer New York Inc. (Chapter 2)