Introduction: What is machine learning?

Alípio Jorge

February 2021

What is Machine Learning?

- Artificial Intelligence?
- Data Mining?
- Data Science?
- Big Data?
- Deep Learning?
- Statistics?

Machine Learning moments

- Legendre, Gauss, Bayes (18th century): fitting
- Alan Turing (1950): "Can Machines Think?"
- John McCarthy, Marvin Minsky (1955): The Dartmouth Workshop on **artificial intelligence**
- Arthur Samuel (1959): coined the term **machine learning**
  - the checkers program

Machine Learning moments

- Warren McCulloch and Walter Pitts (1943): computational model for **neural networks**
- Frank Rosenblatt (1957): The **Perceptron**, a form of ANN
- Marvin Minsky and Seymour Papert (1969): Book
- **Perceptrons: an introduction to computational geometry**
- Ryszard S. Michalski (1983): Book with Jaime Carbonell, Tom Mitchell
    - **Machine learning: An artificial intelligence approach**

Machine Learning moments

- Leo Breiman (1984): book **Classification and regression trees**
- Ross Quinlan (1986): paper **Induction of Decision Trees**
- **ID3** algorithm
- Tom Mitchell (1997): book **Machine Learning**

Machine Learning moments

- Hinton, LeCunn, Bengio,.. (...2000...): **Deep Learning** revolution
- AlphaGo beats the European *go* Champion (2015)
  - Search Trees + Deep Learning + Reinforcement Learning
- AlphaGo Zero beats world champion programs in chess, shogi and go (2017)
  - no human input, only by playing against itself
- Now (2021): **what's next?**

Machine Learning > a classical definition

Tom Mitchell:

"A computer program is said to learn from **experience** E with respect to some class of tasks T and **performance measure** P if its performance at tasks in T, as measured by P, improves with experience E."

The aim of Machine Learning

A program for supporting management decisions

- I manage a club. How much should I spend to be top 6 in the league?

Using Machine Learning

- I use past data with clubs budget and position in the league

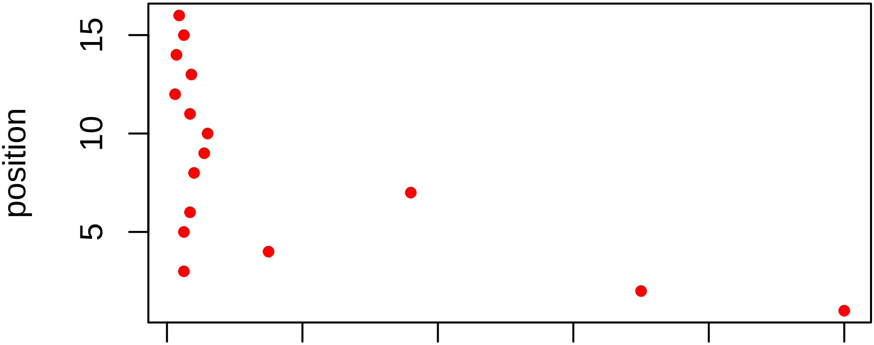
	club	position	budget
1	Porto	1	100.0
2	Benfica	2	70.0
3	Sporting	7	36.0
4	Sp. Braga	4	15.0
5	Marítimo	10	6.0
6	V. Guimarães	9	5.5
7	Nacional	8	4.0
8	Gil Vicente	13	3.6
9	Académica	11	3.4
10	Rio Ave	6	3.4
11	P. Ferreira	3	2.5

The aim of Machine Learning

What is the situation?

- We have observations (experience)
- We assume there is a function that outputs the position given the budget

$$position = f(budget)$$



The aim of Machine Learning

Approximating the function

- The true f is **unknown**
- We use the data to find an **approximation**

What type of function is it?

- We don't know for sure
- We try simple functions first
  - constant function? too simple
  - linear function? Could be

The aim of Machine Learning

Approximating the function

- We run a **linear regression** (example in R)

```
model <- lm(position ~ budget)
model
```

```
Call:
lm(formula = position ~ budget)

Coefficients:
(Intercept)       budget
    10.1744      -0.1035
```

The aim of Machine Learning

Approximating the function

- The approximated function $\hat{f}(x)$ is:

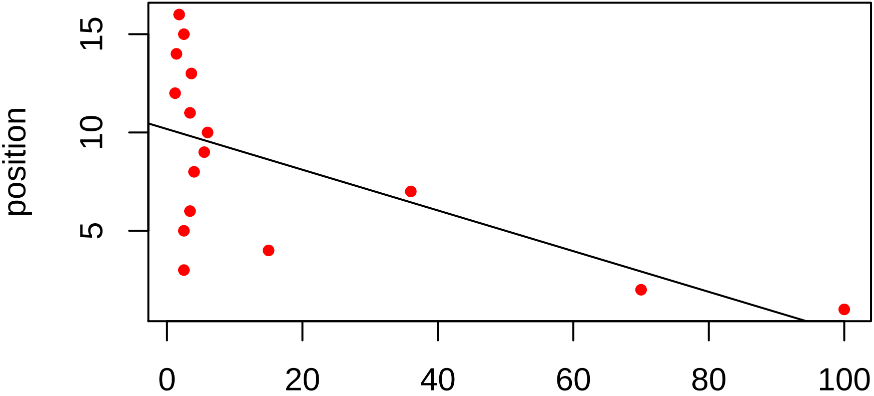
$$position = \hat{f}(budget) = 10.174 - 0.104 \times budget$$

The aim of Machine Learning

How good is the result?

- Let us inspect $\hat{f}(x)$ against the observations

```
plot(budget,position, pch=20, col='red')
abline(model$coefficients[1],model$coefficients[2])
```



The aim of Machine Learning

Did we find a useful approximation?

- We now use the learned function to answer questions like:
  - *"How does the club rank if we invest 10 million?"*

```
predict(model,data.frame(budget=c(0.1,1,5,10,100),position=NA))
```

```
         1          2          3          4          5 
10.1640375 10.0708721  9.6568037  9.1392182 -0.1773217 
```

```
round(
  predict(model,data.frame(
    budget=c(0.1,1,5,10,100),
    position=NA)))
```

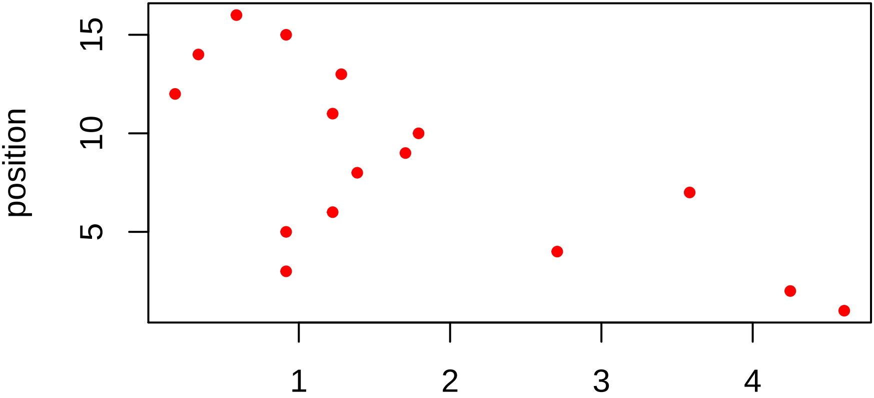
```
 1  2  3  4  5 
10 10 10  9  0 
```

The aim of Machine Learning

Improving the approximation

- Can we improve the model?
- The budget has a very skewed distribution
  - We try a transformation $x \to log(x)$

```
plot(log(budget),position, pch=20, col='red')
```



The aim of Machine Learning

Learn a new model

```r
model <- lm(position ~ log(budget))
model
```

```

Call:
lm(formula = position ~ log(budget))

Coefficients:
(Intercept)  log(budget)
     12.678       -2.421
```

The aim of Machine Learning

Approximating the function

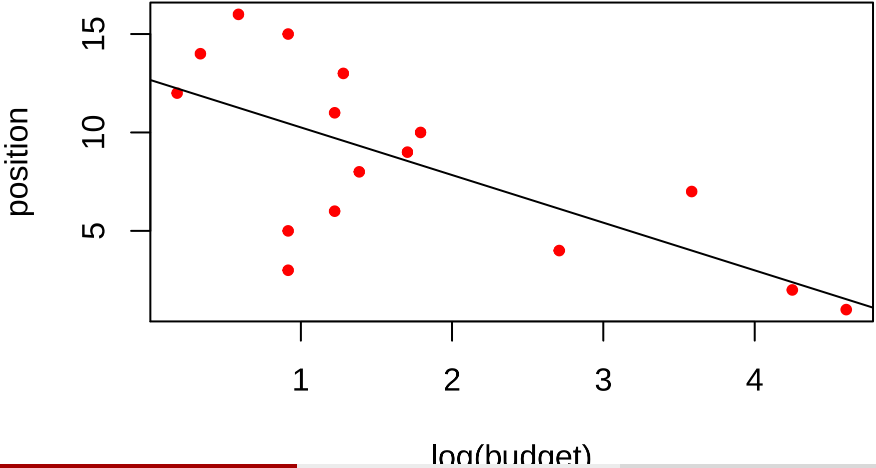
- The approximated function $\hat{f}(x)$ is:

$$position = \hat{f}(budget) = 12.678 - 2.421 \times log(budget)$$

The aim of Machine Learning

Inspecting the result with the log transform

```
plot(log(budget),position, pch=20, col='red')
abline(model$coefficients[1],model$coefficients[2])
```



The aim of Machine Learning

Did we find a useful approximation?

- We use again the learned function to answer questions like:
  - *"How does the club rank if we invest 10 million?"*
  - We see more 'plausible' answers now

```
predict(model,data.frame(budget=c(0.1,1,5,10,100),
                         position=NA))
```

```
        1         2         3         4         5
18.252918 12.678175  8.781597  7.103432  1.528689
```

```
round(
  predict(model,data.frame(
    budget=c(0.1,1,5,10,100),
    position=NA)))
```

```
 1  2  3  4  5
18 13  9  7  2
```

The aim of Machine Learning

Recap

Our goal is to find a **useful approximation** $\hat{f}(x)$ of $f(x)$

- $f(x)$ is the function that generates the phenomenon
- It is **unknown**
- What is the best **approximation**?
- How do we measure the **goodness** of an approximation?
- How do we **find the best approximation** of $f(x)$ (or at least a very good one)?

Linear model

How to find an approximated function?

We start by focusing on a **class of functions**

- The simplest choice would be **constant** functions
  - This is a common **baseline** or **strawman**
- We can be more ambitious and go for **linear functions**

$$\hat{y} = \hat{f}(x) = \beta_0 + \beta_1 x$$

Linear model

How to find an approximated function?

- Now we have to look for one linear function that suits us
  - All we have, all we know, is our **data**
  - There are **many** different ways to do that
  - **One** is the **least squares** method

Linear model > The Least Squares Method

What is the best linear function for our data?

- We define a **loss** function to measure how wrong our approximation is
- For a number of reasons, the **residual sum of squares** is a good choice

$$RSS = \sum_{i}^{N} (f(x_i) - \hat{f}(x_i))^2 = \sum_{i}^{N} (y_i - (\beta_0 + x_i.\beta_1))^2$$

- We now have to find the parameters β that **minimize** RSS given the data
- This is 'easily' solved analytically (no **search** required)

How are the values for the coefficients determined?

The multidimensional definition of RSS is

$$RSS(\beta) = (\mathbf{y} - \mathbf{X}\beta))^T(\mathbf{y} - \mathbf{X}\beta)$$

where $\mathbf{X}$ is the $N \times p$ **data matrix**

If $\mathbf{X}^T\mathbf{X}$ is nonsingular (meaning **no redundant dimensions**) then we have a unique solution for the equation

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

And that is how we **learn** the parameters

How are the values for the coefficients determined?

- The regression model is found **analytically**
  - $\hat{\beta} = [\ \beta_0, \beta_1, \ldots, \beta_m\ ]$
  - the i^{th} case is $x_i = [1, x_1^i, \ldots, x_m^i]$
  - then, we can use the dot product for estimating y_i

$$\hat{y}_i = \overrightarrow{\beta}.x_i$$

  - X is the $n \times (m+1)$ matrix of independent variables with a left column of 1s
  - Y is the $n \times 1$ matrix of target/dependent values

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Where does this equation come from?

- Aim is to find β_i that **minimize** the squares of the residuals
  - **least squares** approach

$$\min_{\hat{\beta}} \sum_{i=1}^{n} (\hat{\beta}.x_i - y_i)^2$$

- by **deriving** and equaling to zero we get to the $\hat{\beta}$ equation

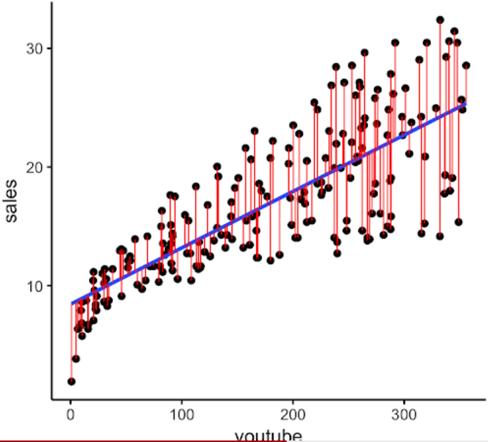
Is this learning process computationally heavy?

- **Computational complexity** analysis
- How **hard** is it to compute the β_i ?
  - matrix multiplications can be $O(n.m^2)$
  - matrix **inversion** can be $O(m^3)$
- **not so bad**
  - linear with the number of cases (great)
  - problematic with many predictors (**usually** not a problem)

How good is the found approximation?

How can we measure the intrinsic quality of the model?

- **How to measure** how much the predicted $\hat{y}$ are close to the actual y
- The difference $e_i = \hat{y}_i - y_i$ is a **residual** or error
  - Best fit has $e_i = 0$ for all i



Linear model > R squared

How can we measure the intrinsic quality of the model?

- The **sum of the squares** of the residuals is a measure of total **error**

$$RSS = \sum_{i=1}^{n} e_i^2$$

- We **normalize** this error with the error predicting the mean $TSS = \sum_{i=1}^{n}(y_i - \overline{y})^2$
- Subtract to 1
- Obtain a **problem independent** measure: R^2

$$R^2 = 1 - \frac{RSS}{TSS}$$

Linear model > R squared

How can we measure the intrinsic quality of the model?

- R^2:
  - is problem independent
  - ranges between 0 and 1
    - 0 if the model is as good as predicting average
    - 1 if the model has zero error
  - is not a safe indicator for approaches that can overfit

In summary?

"searching for" the function (**learning**)

- is done analytically
- is unique (if...)
- is efficient: **order of complexity** at most $\mathcal{O}(p^2N)$
  - if $N > p$
  - Complexity is dominated by the matrix multiplication operations

Mathematically?

- This approach allows the demonstration of
  - **relevant theoretical properties**
- These can be useful to determine
  - the **quality** of the result

With respect to learning?

- We focus on a **simple class** of functions (the linear functions)
- We lose **expressiveness**
- We **generalise** in a safer way (avoid **overfitting**)

What can we have beyond linear models?

- Quadratic functions
  - These are more expressive
  - But can also overfit more easily
  - higher computational learning costs
- Constant functions . . .

Back to our question

##What is machine learning?

Ingredients:

- We have an objective (regression in this case)
- We compressed data, with some information **loss** (learning)
  - Compression enables **generalisation**
  - The result of compression is often called a **model**
  - We also **pre-processed** the data to improve compression
- We can apply the obtained compression operation to new data (new **cases**)
- This allows us to make **inference**
  - prediction
- And to apply the model efficiently in real situations

Activities

(1) Generate an artificial dataset using a linear function with known parameters and some added noise. Study **the effect of the number of examples and of noise dispersion** on the coefficient estimates, on their quality and on rsquare. Produce plots with the variation of the rsquared and of the parameters with the number of examples (up to 5000) and with the dispersion of the noise (`sd` between 0 and 5).

```
d<- data.frame(x1 = runif(10,min = 0,max = 5), y = NA)
d$y <- 1+5*d$x1+rnorm(nrow(d),0,1)
plot(d$x,d$y)
m<-lm(d$y~d$x1)
sm <- summary(m)
sm$r.squared
```

Activities

(2) Perform a similar study using a **random** function $f(x)$.

```
d<- data.frame(x1 = runif(10,min = 0,max = 5), y = NA)
d$y <- rnorm(nrow(d),0,1)
```

The End

Bibliography

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