Week 1: Prediction, Error, and Cross-Validation

Lectures*: 1. Prediction Motivation, 2. What is prediction?, 3. Relative importance of steps, 4. In and out of sample errors, 5. Prediction study design, 6. Types of errors, 7. ROC curves, 8. Cross validation, 9. What data should you use?

Juan David Leongómez

19 March, 2021

Contents

*All lectures by Jeffrey Leek (John Hopkins Bloomberg Scool of Public Health).

---

1 Lecture 1: Prediction Motivation

Overfitting: A model that is too attuned to the noise of a sample, and hence does not perform well on a new sample.

---

2 Lecture 2: What is prediction?

2.1 Components of a predictor

1. **Question**
   - A well-defined question
     - What am I trying to predict?
     - What am I trying to predict it with?
2. **Input data**
   - Collect the best input data that you can
     - From that data you can:
       * Use measured characteristics that you have
       * Use computation to build featured that you think may be useful to meka the prediction
3. **algorithm**
   - Use the ML algorithm
     - E.g.:
       * Random-forest

        * Decision trees
4. **Parameters**
    - Estimate the parameters of those algorithms
    - Use those parameters to apply the algorithm to a new data set
5. **Evaluation**
    - Evaluate the algorithm in that new data

2.2 SPAM example (*super simplified*)

2.2.1 Data set

```
library(kernlab)
data(spam)
library(kableExtra)
library(xtable)
library(magrittr)
sp <- data.frame(variable = names(spam),
           classe = sapply(spam, typeof),
           first_values = sapply(spam, function(x) paste0(head(x),
                                                          collapse = ", ")),
           row.names = NULL)
kable(sp,
      booktabs = TRUE,
      caption = "Variables of the \\textit{spam} dataset") %>%
  kable_styling(latex_options = "HOLD_position",
                font_size = 5)
```

Table 1: Variables of the *spam* dataset

variable	classe	first__values
make	double	0, 0.21, 0.06, 0, 0, 0
address	double	0.64, 0.28, 0, 0, 0, 0
all	double	0.64, 0.5, 0.71, 0, 0, 0
num3d	double	0, 0, 0, 0, 0, 0
our	double	0.32, 0.14, 1.23, 0.63, 0.63, 1.85
over	double	0, 0.28, 0.19, 0, 0, 0
remove	double	0, 0.21, 0.19, 0.31, 0.31, 0
internet	double	0, 0.07, 0.12, 0.63, 0.63, 1.85
order	double	0, 0, 0.64, 0.31, 0.31, 0
mail	double	0, 0.94, 0.25, 0.63, 0.63, 0
receive	double	0, 0.21, 0.38, 0.31, 0.31, 0
will	double	0.64, 0.79, 0.45, 0.31, 0.31, 0
people	double	0, 0.65, 0.12, 0.31, 0.31, 0
report	double	0, 0.21, 0, 0, 0, 0
addresses	double	0, 0.14, 1.75, 0, 0, 0
free	double	0.32, 0.14, 0.06, 0.31, 0.31, 0
business	double	0, 0.07, 0.06, 0, 0, 0
email	double	1.29, 0.28, 1.03, 0, 0, 0
you	double	1.93, 3.47, 1.36, 3.18, 3.18, 0
credit	double	0, 0, 0.32, 0, 0, 0
your	double	0.96, 1.59, 0.51, 0.31, 0.31, 0
font	double	0, 0, 0, 0, 0, 0
num000	double	0, 0.43, 1.16, 0, 0, 0
money	double	0, 0.43, 0.06, 0, 0, 0
hp	double	0, 0, 0, 0, 0, 0
hpl	double	0, 0, 0, 0, 0, 0
george	double	0, 0, 0, 0, 0, 0
num650	double	0, 0, 0, 0, 0, 0
lab	double	0, 0, 0, 0, 0, 0
labs	double	0, 0, 0, 0, 0, 0
telnet	double	0, 0, 0, 0, 0, 0
num857	double	0, 0, 0, 0, 0, 0
data	double	0, 0, 0, 0, 0, 0
num415	double	0, 0, 0, 0, 0, 0
num85	double	0, 0, 0, 0, 0, 0
technology	double	0, 0, 0, 0, 0, 0
num1999	double	0, 0.07, 0, 0, 0, 0
parts	double	0, 0, 0, 0, 0, 0
pm	double	0, 0, 0, 0, 0, 0
direct	double	0, 0, 0.06, 0, 0, 0
cs	double	0, 0, 0, 0, 0, 0
meeting	double	0, 0, 0, 0, 0, 0
original	double	0, 0, 0.12, 0, 0, 0
project	double	0, 0, 0, 0, 0, 0
re	double	0, 0, 0.06, 0, 0, 0
edu	double	0, 0, 0.06, 0, 0, 0
table	double	0, 0, 0, 0, 0, 0
conference	double	0, 0, 0, 0, 0, 0
charSemicolon	double	0, 0, 0.01, 0, 0, 0
charRoundbracket	double	0, 0.132, 0.143, 0.137, 0.135, 0.223
charSquarebracket	double	0, 0, 0, 0, 0, 0
charExclamation	double	0.778, 0.372, 0.276, 0.137, 0.135, 0
charDollar	double	0, 0.18, 0.184, 0, 0, 0
charHash	double	0, 0.048, 0.01, 0, 0, 0
capitalAve	double	3.756, 5.114, 9.821, 3.537, 3.537, 3
capitalLong	double	61, 101, 485, 40, 40, 15
capitalTotal	double	278, 1028, 2259, 191, 191, 54
type	integer	spam, spam, spam, spam, spam, spam

2.2.2 Algorithm based on the frequency of the word '*your*'

```
#plot(density(spam$your[spam$type == "nonspam"]),
#     col = "blue", main = "", xlab = "Frecuency of 'your'")
#lines(density(spam$your[spam$type == "spam"]),
#      col = "red")

library(ggplot2)
ggplot(spam, aes(x = your, color = type, fill = type)) +
  geom_density(alpha = 0.3) +
```

```
  labs(x = "Frecuency of 'your'",
       y = "Density",
       color = "Spam",
       fill = "Spam") +
  scale_color_manual(values = c("blue", "red")) +
  scale_fill_manual(values = c("blue", "red")) +
  theme_light()
```

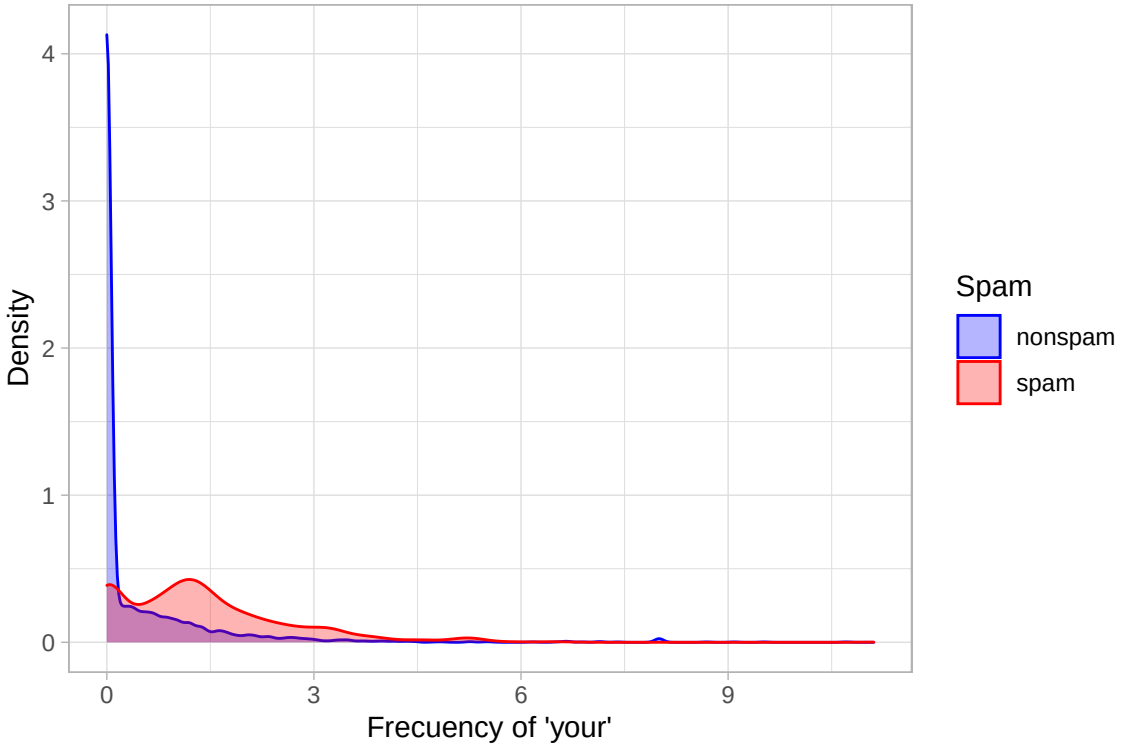

Figure 1: Frequency of the word '*your*' in spam and nonspam emails.

2.2.2.1 The frequency of the word '*your*'

2.2.2.2 Our algorithm

- Find a value C
- Frequency of '*your*' $> c$ predict "spam

```
library(ggplot2)
ggplot(spam, aes(x = your, color = type, fill = type)) +
  geom_density(alpha = 0.3) +
  geom_vline(xintercept = 0.5) +
  labs(x = "Frecuency of 'your'",
       y = "Density",
       color = "Spam",
       fill = "Spam") +
  scale_color_manual(values = c("blue", "red")) +
  scale_fill_manual(values = c("blue", "red")) +
  theme_light()
```

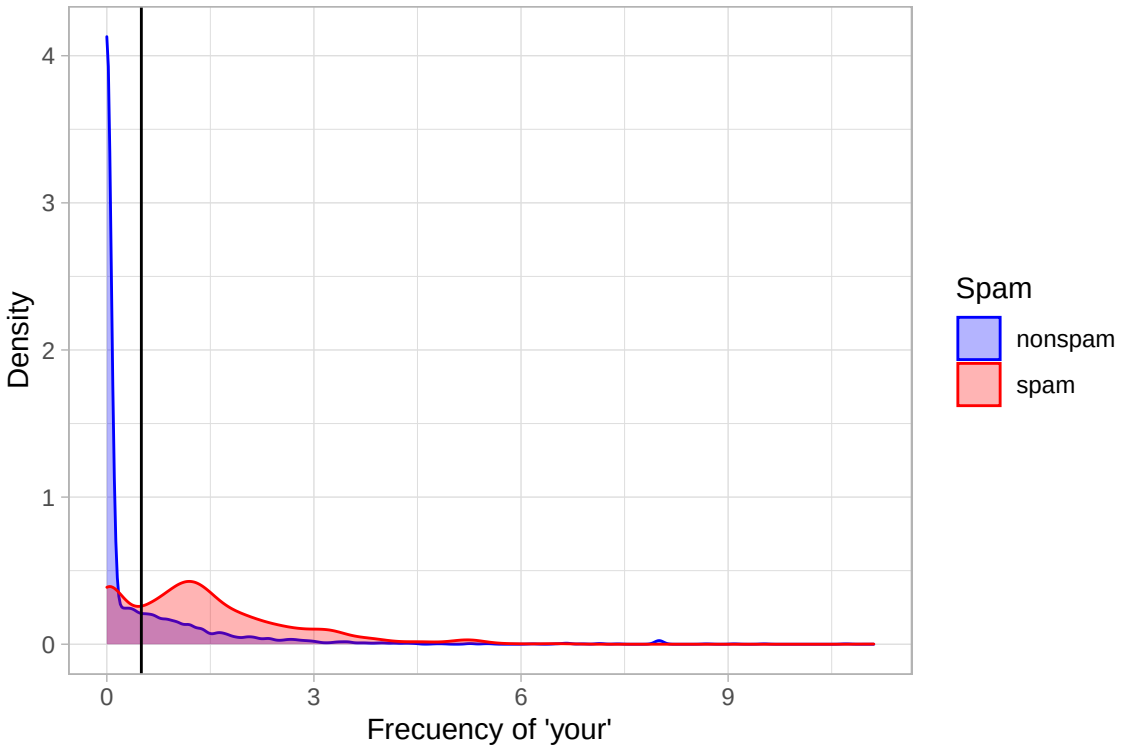

Figure 2: Frequency of the word '*your*' in spam and nonspam emails. The black, vertical line represents a cutoff value (C) of 0.5, which may work as most of the spike in nonspam is below that value, and one of the spikes in spam is above that value.

2.2.2.3 The C cutoff (in this case, 0.5)

2.2.3 Evaluate algorithm

We will calculate a prediction for each of the different emails we have.

```
prediction <- ifelse(spam$your > 0.5, "spam", "nonspam")
tab1 <- table(prediction, spam$type)/length(spam$type)
kable(tab1,
      booktabs = TRUE,
      caption = "Classification of spam and nonspam emails based on the frequency of
      the word '*your*'") %>%
  kable_styling(latex_options = "HOLD_position") %>%
  footnote(general = paste0("Nonspam is correctly cathegorised on about ",
                            round(tab1[1,1], 2),
                            " of the cases,
                            while spam is correctly identified on about ",
                            round(tab1[2,2], 2),
                            " of the cases. This gives us an accuracy of about ",
                            round(tab1[1,1], 2), " + ", round(tab1[2,2], 2),
                            " = ", round(tab1[1,1] + tab1[2,2], 2),
                          " (or ", round((tab1[1,1] + tab1[2,2])*100, 0),
                          "\\\\%).",
                          threeparttable = TRUE,
                          escape = FALSE,
                          footnote_as_chunk = TRUE)
```

Table 2: Classification of spam and nonspam emails based on the frequency of the word '*your*'

	nonspam	spam
nonspam	0.4590306	0.1017170
spam	0.1469246	0.2923278

Note: Nonspam is correctly cathegorised on about 0.46 of the cases, while spam is correctly identified on about 0.29 of the cases. This gives us an accuracy of about 0.46 + 0.29 = 0.75 (or 75%).

This, however, was tested on the same data set from which we calculated the prediction function. As we will see, this provides an optimistic estimate of the overall error rate.

---

3 Lecture 3: Relative importance of steps

3.1 The importance of good (and abundant) data

"***The combination of some data and and aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data.***" *John Tukey*

In other words, is important to know when to give up. When the data available is insufficient to answer the question you are trying to answer.

Garbage in → garbage out

In most cases, a lot more data can often beat out getting much better statistical or machine learning models in almost every case.

3.2 Features matter!

3.2.1 Properties of good features

- Lead to data compression
- Retain relevant information
  - The important properties of good features are:
    - They compress the data in a way that make it possible to compute your prediction, or your machine learning algorithm, in a very simple way.
    - Maybe are more interpretable than the data that you've collected (the raw data that might be complicated)
- Are created based on expert application of knowledge
  - Is it better to create features automatically?
  - Or is it better to use expert domain knowledge?

3.2.2 Common mistakes

- Trying to automate future selection
- Automate feature selection in a way that does not allow for you to understand how those features are actually applied to make good predictions
- Black box predictions can be very useful, they can be very accurate but they can also change on a dime if we're not paying attention to how those features actually do predict the outcome

- Not paying attention to data-specific quirks
  - For example not understanding how outliers affect prediction or weird behaviours associated with specific features
  - Not understanding those can cause problems
- Throwing away information unnecessarily

3.2.3 Features may be automated with care

This is semi-supervised learning or deep-learning.

3.3 Algorithms matter less than you think

Using a sensible method (e.g. a statistical model) will get you a very large weight of solving the problem. And then, getting the absolute best method can improve, but it often doesn't improve that much over most good sensible methods.

3.3.1 Issues to consider

The best Machine Learning Method:

- Interpretable
- Tradeoffs between being
  - Simple
  - Accurate
- Fast (both to train and to test)
  - Easy to build and test the model in a small data set
- Scalable
  - To a larger data set

3.4 Prediction

Prediction is about accuracy tradeoffs:

- Interpretability versus accuracy
- Speed versus accuracy
- Simplicity versus accuracy
- Scalability versus accuracy

The idea is to fine a balance between those components. In some cases, it may be better to have very clear, interpretable results. For example:

> **if** total cholesterol ≥ 160 **and** smoke **then** *10 year CHD risk* $\geq 5\%$
> **else if** smoke **and** systolic blood pressure ≥ 140 **then** *10 year CHD risk* $\geq 5\%$
> **else** *10 year CHD risk* $< 5\%$ (from http://www.cs.cornell.edu/~chenhao/pub/mldg-0815.pdf)

---

4 Lecture 4: In and out of sample errors

- **In Sample Error:** Error rate you get on the same data set used to build your predictor (also called resubstitution error)
  - Always a bit optimistic, because the algorithm tunes to the noise in the data set
  - On a different data set, the noise will be different and the accuracy will decrease a bit
- **Out of Sample Error:** Error rate you get on a new data set (also called generalisation error)

4.1 Key ideas

- Out of Sample Error is what you care about (and what should be reported; otherwise, you know accuracy is optimistic)
- In sample error < out of sample error
- Due to overfitting:
  - Matching the algorithm to the data you have

4.2 A simple axample

```
#library(kernlab)
#data(spam)
RNGkind(sample.kind = "Rounding") #to make it equivalent to older R versions
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(333)
smallSpam <- spam[sample(dim(spam)[1], size = 10),]
spamLabel <- (smallSpam$type == "spam")*1 + 1

#plot(smallSpam$capitalAve, col = spamLabel)

smallSpam <- tibble::rownames_to_column(smallSpam)

ggplot(smallSpam, aes(x = as.numeric(row.names(smallSpam)),
                      y = capitalAve,
                      color = type)) +
  geom_point(alpha = 0.3) +
  labs(x = "Index",
       y = "Average number of capital letters",
       color = "Spam") +
  scale_color_manual(values = c("blue", "red")) +
  #ylim(c(0,20)) +
  theme_light()
```

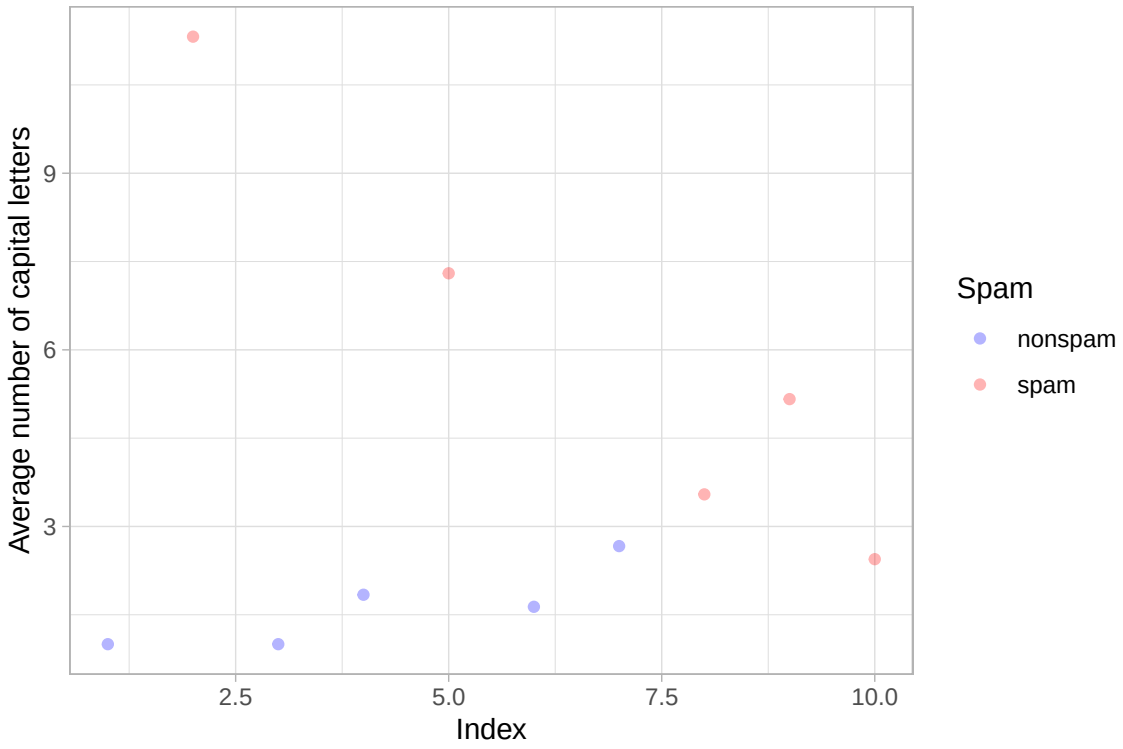

Figure 3: Average number of capital letters in a subset of the *Spam* data set.

4.2.1 Algorithms based on the average number of capital letters

4.2.1.1 Rule 1

- Average number of Capital Letters
  - capitalAve > 2.7 = "spam"
  - capitalAve < 2.4 = "nonspam"

In the example in class (which is different from the one I have unless I add code before seting the seed), there is only one value of spam that is below some values of nonspam. Because of this, and to train the algorithm perfectly, another, specific set of rules is added:

- capitalAve between 2.40 and 2.45 = "spam"
- capitalAve between 2.45 and 2.7 = "nonspam"

Apply these rules

```
rule1 <- function(x){
  prediction <- rep(NA, length(x))
  prediction[x > 2.7] <- "spam"
  prediction[x < 2.4] <- "nonspam"
  prediction[(x >= 2.40 & x <= 2.45)] <- "spam"
  prediction[(x > 2.45 & x <= 2.70)] <- "nonspam"
  return(prediction)
}

tab2 <- table(rule1(smallSpam$capitalAve), smallSpam$type)
kable(tab2,
      booktabs = TRUE,
      caption = "Classification of spam and nonspam emails based on the
```

```
      average number of capital letters") %>%
  kable_styling(latex_options = "HOLD_position") %>%
  footnote(general = paste0("Nonspam is correctly cathegorised on about ",
                            round(tab2[1,1]/10, 2),
                            " of the cases,
                            while spam is correctly identified on about ",
                            round(tab2[2,2]/10, 2),
                            " of the cases. This gives us an accuracy of about ",
                            round(tab2[1,1]/10, 2), " + ", round(tab2[2,2]/10, 2),
                            " = ", round(tab2[1,1]/10 + tab2[2,2]/10, 2),
                          " (or ", round((((tab2[1,1]/10) + (tab2[2,2])/10))*100, 0),
                          "\\\\%)."),
                          threeparttable = TRUE,
                          escape = FALSE,
                          footnote_as_chunk = TRUE)
```

Table 3: Classification of spam and nonspam emails based on the average number of capital letters

	nonspam	spam
nonspam	5	0
spam	0	5

Note:
Nonspam is correctly cathegorised on about 0.5 of the cases, while spam is correctly identified on about 0.5 of the cases. This gives us an accuracy of about 0.5 + 0.5 = 1 (or 100%).

4.2.1.2 Rule 2

- Find a value C
- Average number of Capital Letters $> c$ predict "spam
  - capitalAve > 2.8 = “spam”
  - capitalAve <= 2.8 = “nonspam”

```
rule2 <- function(x){
  prediction <- rep(NA, length(x))
  prediction[x > 2.8] <- "spam"
  prediction[x <= 2.8] <- "nonspam"
  return(prediction)
}

tab3 <- table(rule2(smallSpam$capitalAve), smallSpam$type)
kable(tab3,
      booktabs = TRUE,
      caption = "Classification of spam and nonspam emails based on the
      average number of capital letters") %>%
  kable_styling(latex_options = "HOLD_position") %>%
  footnote(general = paste0("Nonspam is correctly cathegorised on about ",
                            round(tab3[1,1]/10, 2),
```

```
                                " of the cases,
                                while spam is correctly identified on about ",
                                round(tab3[2,2]/10, 2),
                                " of the cases. This gives us an accuracy of about ",
                                round(tab3[1,1]/10, 2), " + ", round(tab3[2,2]/10, 2),
                                " = ", round(tab3[1,1]/10 + tab3[2,2]/10, 2),
                              " (or ", round((((tab3[1,1]/10) + (tab3[2,2])/10))*100, 0),
                              "\\\\%)."),
                              threeparttable = TRUE,
                              escape = FALSE,
                              footnote_as_chunk = TRUE)
```

Table 4: Classification of spam and nonspam emails based on the average number of capital letters

	nonspam	spam
nonspam	5	1
spam	0	4

Note:
Nonspam is correctly cathegorised on about 0.5 of the cases, while spam is correctly identified on about 0.4 of the cases. This gives us an accuracy of about $0.5 + 0.4 = 0.9$ (or 90%).

4.2.2 Apply rures to the complete set of data

```
tab4 <- table(rule1(spam$capitalAve), spam$type)
tab5 <- table(rule2(spam$capitalAve), spam$type)

tab4F <- cbind(tab4, tab5)
kable(tab4F,
      booktabs = TRUE,
      caption = "Classification of spam and nonspam emails based on the
      average number of capital letters") %>%
  kable_styling(latex_options = "HOLD_position") %>%
  add_header_above(c(" " = 1,
                     "Rule1" = 2,
                     "Rule2" = 2),
                   escape = FALSE) %>%
  footnote(number = c(paste0("For Rule1, nonspam is correctly cathegorised on about ",
                             round(tab4[1,1]/length(spam$make), 2),
                             " of the cases,
                             while spam is correctly identified on about ",
                             round(tab4[2,2]/length(spam$make), 2),
                             " of the cases. This gives us an accuracy of about ",
                             round(tab4[1,1]/length(spam$make), 2), " + ",
                           round(tab4[2,2]/length(spam$make), 2),
                             " = ", round(tab4[1,1]/length(spam$make) +
                                          tab4[2,2]/length(spam$make), 2),
```

```
                        " (or ", round((((tab4[1,1]/length(spam$make)) +
                                         (tab4[2,2])/length(spam$make)))*100, 0),
                        "\\\\%)."),
                  paste0("For Rule2, nonspam is correctly cathegorised on about ",
                         round(tab5[1,1]/length(spam$make), 2),
                         " of the cases,
                         while spam is correctly identified on about ",
                         round(tab5[2,2]/length(spam$make), 2),
                         " of the cases. This gives us an accuracy of about ",
                         round(tab5[1,1]/length(spam$make), 2), " + ",
                        round(tab5[2,2]/length(spam$make), 2),
                         " = ", round(tab5[1,1]/length(spam$make) +
                                      tab5[2,2]/length(spam$make), 2),
                        " (or ", round((((tab5[1,1]/length(spam$make)) +
                                         (tab5[2,2])/length(spam$make)))*100, 0),
                        "\\\\%)")),
      threeparttable = TRUE,
      escape = FALSE)
```

Table 5: Classification of spam and nonspam emails based on the average number of capital letters

	Rule1		Rule2	
	nonspam	spam	nonspam	spam
nonspam	2141	588	2224	642
spam	647	1225	564	1171

[1] For Rule1, nonspam is correctly cathegorised on about 0.47 of the cases, while spam is correctly identified on about 0.27 of the cases. This gives us an accuracy of about 0.47 + 0.27 = 0.73 (or 73%).

[2] For Rule2, nonspam is correctly cathegorised on about 0.48 of the cases, while spam is correctly identified on about 0.25 of the cases. This gives us an accuracy of about 0.48 + 0.25 = 0.74 (or 74%).

Rule2 performs better because in the case of Rule1 (not that much in my case, but in the example in the lecture) there is overfitting.

- Data has two parts
  - **Signal** (the part we are trying to use to predict)
  - **Noise** (random variation in the data set)
- The goal of a predictor is to find **signal**
- You can always design a perfect *in.sample* predictor
  - In a small data set you can capture every single quirk of that data set
  - If you do that, you capture both **signal** + **noise**
  - In the example in the lecture (which I could not replicate), rule1 was tuned too tightly to the observed training set
- Predictor will not perform as well on new samples

To remember. We want to build models that are simple and robust enough that they don't actually capture the noise, while they do capture all of the signal.

5 Lecture 5: Prediction study design

5.1 Prediction study design

1. Define error rate
2. Split data into:
   - Training, Testing, Validation (optional)
3. On the training set pick features
   - Use cross-validation
4. On the training set pick prediction function
   - Use cross-validation
5. If there is no validation set
   - Apply 1x to test set. In other words, **you should apply the prediction model to the test set exactly one time**[1]
6. If validation
   - Apply to test set and refine[2]
   - Apply 1x to validation. In other words, **you should apply the prediction model (tes best one) to the validation set ONLY one time. This gives you a good estimation of your out of sample error rates**

5.2 Avoid small sample sizes

- Suppose you are predicting a binary outcome. E.g.:
  - Diseased/healthy
  - Click on add/not click on add
- One classifier is flipping a coin
- Probability of perfect classification is approximately:
  - $(\frac{1}{2})^n$
  - $n = 1$ flipping coin 50% chance of 100% accuracy
  - $n = 2$ flipping coin 25% chance of 100% accuracy
  - $n = 10$ flipping coin 0.10% chance of 100% accuracy

5.3 Rules of thimb for prediction study design

- If you have a large sample size (assuming test and validation are not too small)
  - 60% training
  - 20% testing
  - 20% validation
- If you have a medium sample size (to insure that testing set is of sufficient size)
  - 60% training
  - 40% testing
- If you have a small sample size (first consider if it is enough to to be able to build a prediction algorithm)
  - Do cross-validation
  - Report caveat of small sample size, and the fact that you never got to predict this in an out of sample or a testing data set

5.4 Some principles to remember

- Set the test/validation set(s) aside and *never look at it*(*them*)

[1] **NOTE:** Only one time because if we applied multiple models to our testing set, and pick the best one, then we are using the test set to train the model, and hence would get an optimistic view on what the error would be on a new data set.

[2] **NOTE:** You can apply all your best prediciton models to your test set and refine them a little bit. You may find that some features do not work so well for out of sample prediction.

- In general, *randomly* sample training and test sets
- Your data sets must reflect structure of the problem
  - If predictions evolve with the split train/test in time chunks (called *backtesting* in finance)
- All subsets should reflect as much diversity as possible
  - Random assignment does this (or *tends to*)
  - You can also try to balance this by features - but this is tricky (although often a good idea)

---

6 Lecture 6: Types of errors

This lecture is about the types of errors and the ways that you will evaluate the prediction functions that you generate during this class.

6.1 Basic terms

- In general, **positive** = identified, and **negative** = rejected. Therefore:
  - **True positive** = correctly identified
  - **False positive** = incorrectly identified
  - **True negative** = correctly rejected
  - **False negative** = incorrectly rejected

```
library(ggpubr)
library(png)

img1 <- readPNG("sens2.png")
img2 <- readPNG("sens.png")

im_A <- ggplot() + background_image(img1)
im_B <- ggplot() + background_image(img2)
ggarrange(im_A, im_B,
          ncol = 2,
          widths = c(1.06, 1),
          labels = "AUTO")
```

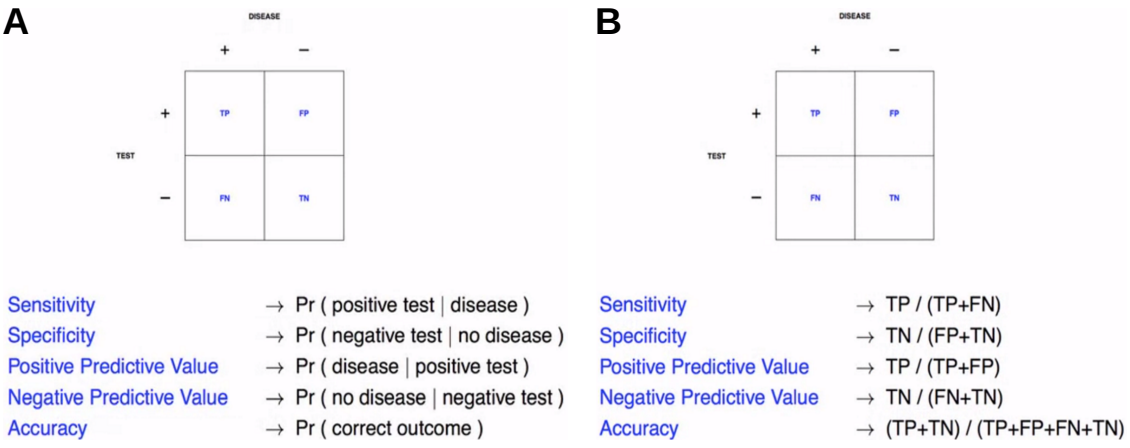

Figure 4: Sensitivity and specificity diagram. **A.** General deffinitions. **B.** Diagram with formulas. *Sensitivity* is the probaility of being classified as disease if you are actually diseaded. *Specificity* is the probaility of being classified as healthy if you are healthy. *Accuracy* is the probability that you are correctly classified.

- Medical testing example:

- **True positive** = sick people correctly identified as sick
- **False positive** = healthy people incorrectly identified as sick
- **True negative** = healthy people correctly identified as healthy
- **False negative** = sick people incorrectly identified as healthy

6.2 Good sensitivity and specificity are not enough

However, it is important to consider how **Positive Predictive Values** change depending on the proportion of people that, for example, actually have the disease.

```
library(ggpubr)
library(png)

img3 <- readPNG("9.png")
img4 <- readPNG("92.png")

im2_A <- ggplot() + background_image(img3)
im2_B <- ggplot() + background_image(img4)
ggarrange(im2_A, im2_B,
          ncol = 2,
          widths = c(1, 1.04),
          labels = "AUTO")
```

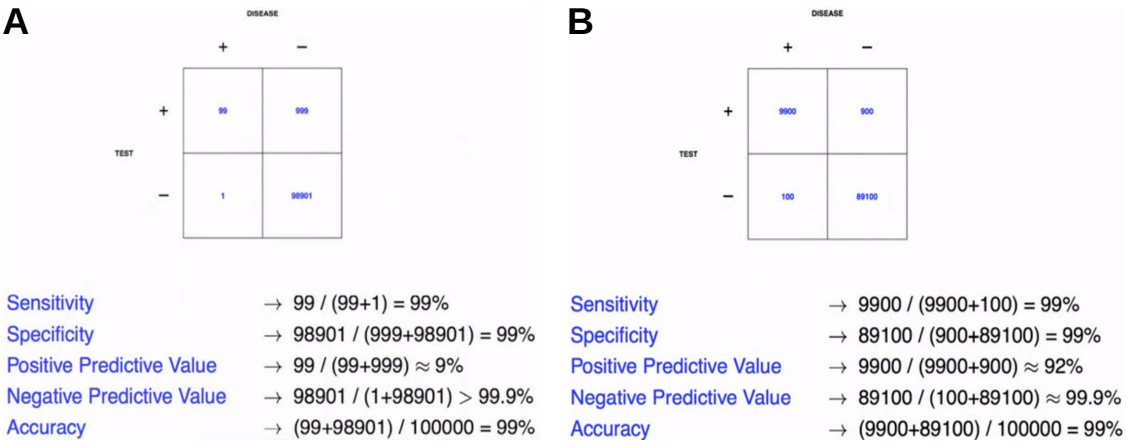

Figure 5: Sensitivity and specificity depending on the proportion of people who have the disease, with a hypothetical population of 100,000, with sensitivity and specificity of 99%. **A.** Example when 0.1% of the population has the disease. **B.** Example when 10% of the population has the disease. In the first case, because only 99 out of about 1100 people who were classified as diseased actualy had the disease, the **Positive Predictive Value** is 9%. In the second case, this value is 92%.

6.3 Error in continuous data

For continuous data the scenario is not so simple. The goal here is to estimate how close you are to the truth (the *true* value).

6.3.1 Mean squared error (MSE):

MSE is the average (squared, hence positive) *distance* between each prediction and its true value.

$$\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2$$

6.3.2 Root mean squared error (RMSE):

Because in **MSE** *distances* are squared, the squared root of the **MSE** (or **RMSE**) is commonly used.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2}$$

6.3.3 Median absolute deviation (MAD)

Because **MSE** and **RMSE** are means, they are affected by outliers. The **MAD**, being a median (of the absolute differences between *predicted* and *true* values), tends to be more robust.

$$median(|Prediction_i - Truth_i|)$$

6.3.4 Common error measures

MSE and **RMSE** are common, but they often do not normally work when there are a lot of outliers, or the values of the variables have very different scales. **MSE** and **RMSE** are sensitive to those outliers; for example, one really high value may raise the mean. For this reason, we can use the **Median absolute deviation**.

Continuous data. Which one you choose may depend on whether there are many outliers.

1. **Mean squared error (MSE)**, or **Root mean squared error (RMSE)**
   - Continuous data, sensitive to outliers
2. **Median absolute deviation (MAD)**
   - Continuous data, often more robust

Categorical data. Which one you choose may depend on what error is more important to you.

3. **Sensitivity** (recall)
   - If you want few missed positives
4. **Specificity**
   - If you want few negatives called positives
5. **Accuracy**
   - Weights false positives/negatives equally
6. **Concordance**
   - For multi-class cases, there are many options. One example is kappa

---

7 Lecture 7: ROC curves

This lecture's about ROC curves, or Receiver Operating characteristic curves. These are very commonly used techniques to measure the quality or goodness of a prediction algorithm.

7.1 Why a curve

- In binary classification you are predicting one of two categories
  - Alive/dead
  - Click on ad/don't click
- But your predictions are often quantitative
  - Probability of being alive
  - Prediction on a scale from 1 to 10
- The *cutoff* you choose gives different results

```
knitr::include_graphics("ROC.png")
```

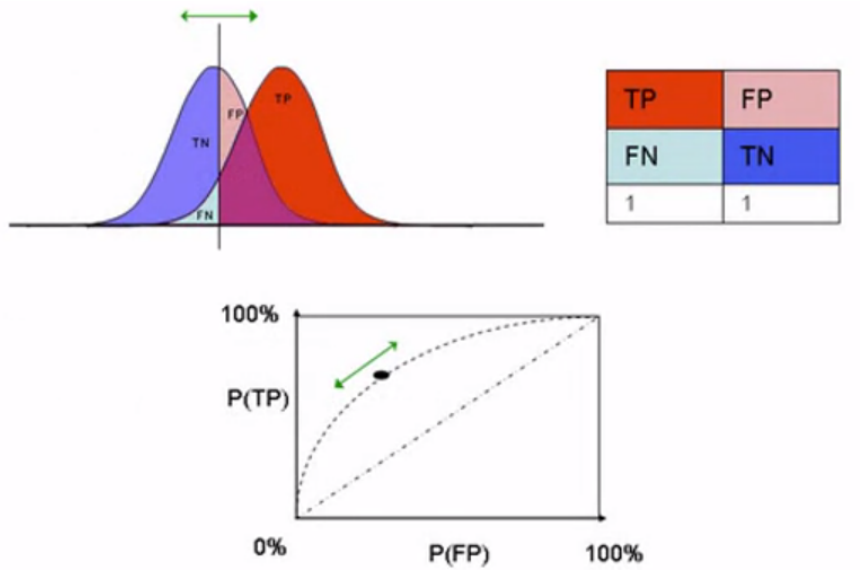

Figure 6: ROC curve. The X axis is usually $1 - specificity$ (the probability of a false positive), and the Y axis usually is the *sensitivity* (the probability of a true positive). Different $cutoffs$ will have different X and Y values, as represented by the curve.

7.1.1 A real example

Example from Wikipedia (https://en.wikipedia.org/wiki/Receiver_operating_characteristic).

```
knitr::include_graphics("ROC2.png")
```

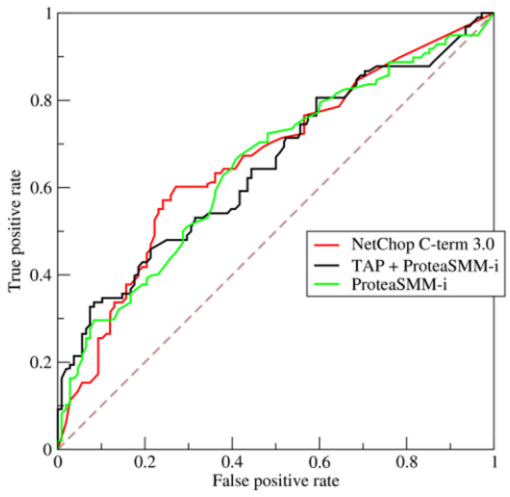

Figure 7: ROC curve examples from different algorithms. To calculate how good a prediction algorithm is, the usual is to calculate the *area under the curve*.

7.2 Area under the curve (AUC)

To calculate how good a prediction algorithm is, the usual is to calculate the **AUC**.

- **AUC = 0.5** → Random guess, which is a 45° line (as represented in Figs 6-8)
- **AUC = 1** → Perfect classifier
- In general, an **AUC > 0.8** is considered "good". However, it depends on the field and the problem being studied

```
knitr::include_graphics("ROC3.png")
```

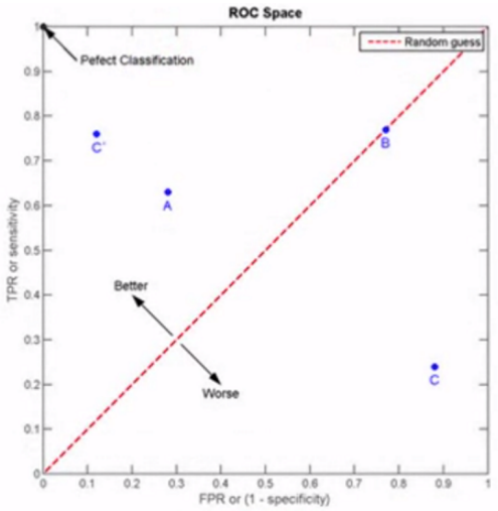

Figure 8: The idea is to maximise both *sensitivity* and *specificity* (the top left area).

---

8 Lecture 8: Cross validation

This lecture is about **cross validation**: one of the most widely used tools, for detecting relevant features, building models, and estimating their parameters when doing machine learning.

8.1 Key idea

1. Accuracy on the training set (also called *resubstitution accuracy*) is **optimistic**
2. A better estimate comes from an independent set (**test set accuracy**)
3. But we cannot use the test set when building the model or it becomes part of the training set
4. So we estimate the test set accuracy with the training set

To do this, we use **Cross-validation**

8.2 Cross-validation

Approach:

1. **Use** the *training set*
2. **Split** further into *training set* and *test set*
3. **Build a model** on the *training set*
4. **Evaluate** on the *test set*
5. **Repeat** over and over and **average** the *estimated error*

The original *test set* is never used in this process. The most common way to do this, is to do **random subsampling** of the training set, into *training* and *testing* sets.

Used for:

1. **Picking variables** to include in a model
2. **Picking the type of prediction function** to use OR
3. **Picking the parameters** in the prediction function and estimate them
4. **Comparing** different predictors

8.2.1 Approaches to cross-validation

There are different approaches to cross-validation. In general, it means to split the training dataset into training and testing samples, several times, and then average the errors, to get an estimate of the error rate we would get in an out of sample estimation procedure.

```
knitr::include_graphics("Cross-validation.png")
```

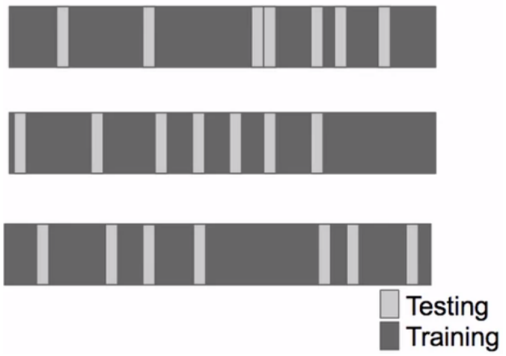

Figure 9: Random subsampling. From the original **training set**, consecutive random subsamples are selected for training and testing. In this example, it would mean iterations of building the model on the dark-grey training samples, and testing it on the light-grey testing samples (i.e. applying it to evaluate the out of sample accuracy rate).

8.2.1.1 Random subsampling

8.2.1.2 K-fold cross validation

Another approach is ***K-fold** cross validation*

```
knitr::include_graphics("K-fold.png")
```

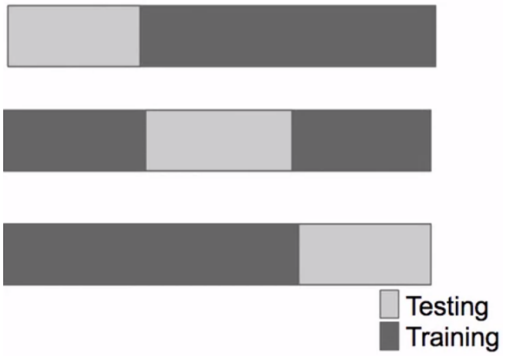

Figure 10: K-fold cross validation. The idea is to split the original **training set** into K equal-sized datasets. In this example, the training dataset is split into 3 (i.e. a 3-fold cross validation). As before, in these K iterations, the model is build on the dark-grey training samples, and tested on the light-grey testing samples, to evaluate the out of sample accuracy rate.

8.2.1.3 Leave one out Another very common approach is called ***Leave one out*** *cross validation*. In this case, we just leave out exactly one sample, and we build the predictive function on all the remaining samples. And then we predict the value on the one sample that we left out. Then we leave out the next sample, and build on all the remaining values. And then predict the sample that we left out, and so forth until we've done that for every single sample in our data set.

This is another way to estimate the out of sample accuracy rate.

```
knitr::include_graphics("Leave-one-out.png")
```

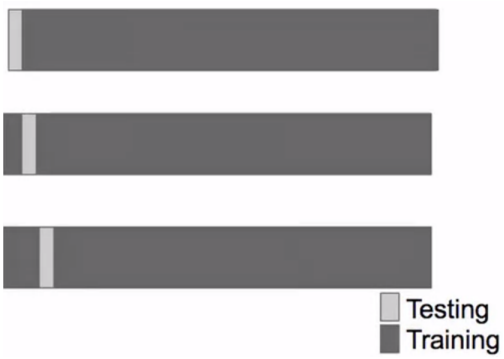

Figure 11: Leave one out cross validation. The idea is to exclude exactly one sample from the original **training set** to build the model on the dark-grey remaining samples samples, and predict the value on the light-grey one that was left out. Then repeat this process until each individual sample has been excluded from the training, to evaluate the out of sample accuracy rate.

8.3 Considerations

- For **time series** data, data must be used in "chunks"
- For **k-fold** cross validation
  - Larger k (e.g. 10 or 20) = less bias, more variance
  - Smaller k = more bias, less variance
- **Random subsampling** must be done *without* replacement
- Random subsampling with replacement is the **bootstrap**
  - Underestimates the error
  - Can be corrected, but it is complicated. This is called the 0.632+ bootstrap, which weights the bootstrapped resubstitution error.
- If you cross-validate to pick predictors you must estimate errors on independent data, in order to get a true out of sample error. Since you always picked the best model, it will not necessarily be a good representation of what the real out of sample error rate is[3]

---

9 Lecture 9: What data should you use?

After defining the question of interest, the next most important thing is to identify the data set that will allow you to create the best possible predictions, which your machine learning algorithm.

9.1 The FiveThirtyEight example

A great example is FiveThirtyEight. Nate Silver, the statistician who created this blog, used polling information from a wide variety of polls and averaged them together to try to get a best prediction of who is

[3]The best way to do this, it to apply your prediction function, just one time, to an independent test set.

going to win which votes and which states in order to get the best vote (i.e. a prediction of who would vote for who when it came time for the election). It was very successful in predicting both the 2008 and 2012 elections.

He had several important ideas:

- **Using like data to predict like:** he took polling data from organizations like the Gallup polling agencies in the United States, and specifically all that data that was asking people directly, *who are you likely to vote for in the upcoming election?*
  - In other words, he used data where people were asked the same question they were going to be asked when they went into the polls the ballot and decide who they were going to vote for
- **Recognising the quirks of the data:** He realised that some polls were actually biased in one way or the other. A particular pollster might ask questions that led people to say that they would vote for one candidate more than the other, even when they might not necessarily vote that way when it came time to vote for real
  - **Weighting the polls:** he weighted them by how close they were to being unbiased or accurate polls

This is an example where finding the right dataset and combining it with simple understanding of what's really going on scientifically can have maximum possible benefit in terms of making strong predictions.

9.2 Key idea

If you want to predict something about X, use data as closely related to X as you possibly can.

9.2.1 Examples

An example is *Moneyball* (also a movie with Brad Pitt), that used a baseball player's performance to predict his performance.

Another examples are the *Netflix prize*, where people were trying to predict movie preferences, and they did that based on the past movie preferences of those people, or the *Heritage Health prize*, where the goal was to try to predict which people would be hospitalised, and they used data about previous hospitalisations.

9.2.2 Counterexamples

People used Google searches to try and predict flu outbreaks, but it has recently been pointed out that it has have major flaws, in the sense that, if people's searches changed, or if the properties of the ways those searches were connected to the flu changed, their predictions would actually be quite far off (as has been demonstrated).

Another example is the reported positive association between chocolate consumption per capita, and Nobel Laureates per 10 million population ($r = 0.791, p < 0.0001$).

```
knitr::include_graphics("choco.jpeg")
```

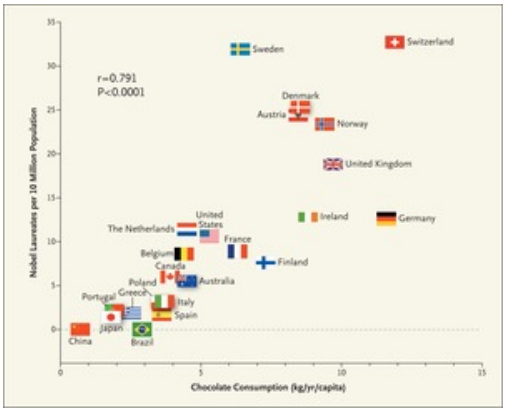

Figure 12: association between chocolate consumption per capita, and Nobel Laureates per 10 million population. From Messerli, F. H.. (2012). Chocolate Consumption, Cognitive Function, and Nobel Laureates. *New England Journal of Medicine, 367*(16), 1562–1564. https://www.nejm.org/doi/full/10.1056/nejmon1211064.

There are many variables that might relate these two things to each other. This has nothing to do with the ability of chocolate consumption *per se* to predict whether you will get a Nobel prize or not.

When you're using data that is not related, be very careful about interpreting why your prediction algorithm works or does not work.