k-NEAREST-NEIGBORS - BREAST CANCER PREDICTION

Case Study - Supervised lazy learner classifiers

Marcus Birkenkrahe

March 7, 2023

README

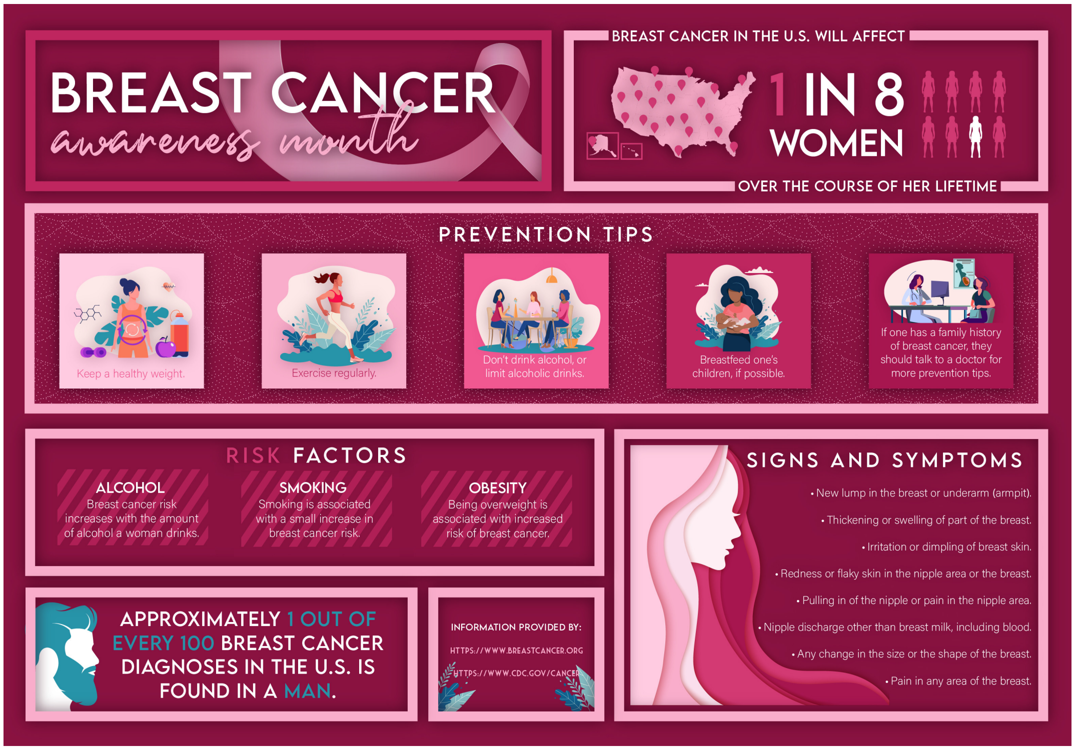

BREAST CANCER
awareness month
BREAST CANCER IN THE U.S. WILL AFFECT
1 IN 8 WOMEN
OVER THE COURSE OF HER LIFETIME
PREVENTION TIPS
Keep a healthy weight.
Exercise regularly.
Don't drink alcohol, or limit alcoholic drinks.
Breastfeed one's children, if possible.
If one has a family history of breast cancer, they should talk to a doctor for more prevention tips.
RISK FACTORS
ALCOHOL
Breast cancer risk increases with the amount of alcohol a woman drinks.
SMOKING
Smoking is associated with a small increase in breast cancer risk.
OBESITY
Being overweight is associated with increased risk of breast cancer.
SIGNS AND SYMPTOMS
• New lump in the breast or underarm (armpit).
• Thickening or swelling of part of the breast.
• Irritation or dimpling of breast skin.
• Redness or flaky skin in the nipple area or the breast.
• Pulling in of the nipple or pain in the nipple area.
• Nipple discharge other than breast milk, including blood.
• Any change in the size or the shape of the breast.
• Pain in any area of the breast.
APPROXIMATELY 1 OUT OF EVERY 100 BREAST CANCER DIAGNOSES IN THE U.S. IS FOUND IN A MAN.
INFORMATION PROVIDED BY:
HTTPS://WWW.BREASTCANCER.ORG
HTTPS://WWW.CDC.GOV/CANCER

Figure 1: Info graphic - breast cancer awareness month (October)

We will investigate the utility of ML for detecting cancer by applying the k-NN algorithm to measurements of biopsied cells from women with abnormal breast masses.

To code along with the lecture, download `5_knn_practice.org` from GitHub, complete the file and upload it to Canvas by the deadline.

Plan

- Routine breast cancer screening looks for abnormal lumps or masses
- Small cell samples are extracted via fine-needle aspiration biopsy
- Cells are examined to determine if the mass is benign or malignant
- Machine learning could automate cancerous cell identification
- Potential benefits: efficiency/time savings, detection accuracy

ML workflow

- Collecting the data (Data)
- Exploring and preparing the data (D)
- Normalizing (rescaling) numeric data (A)
- Creating training and test data sets (A)
- Training a model on the data (G)
- Evaluating model performance (E)
- Improving model performance (E)

Collecting the data

- Measurements from digitized images of fine-needle aspirate of a breast mass
- The data values represent characteristics of the cell nuclei present in the digital image
- Data were donated by researchers of the University of Wisconsin

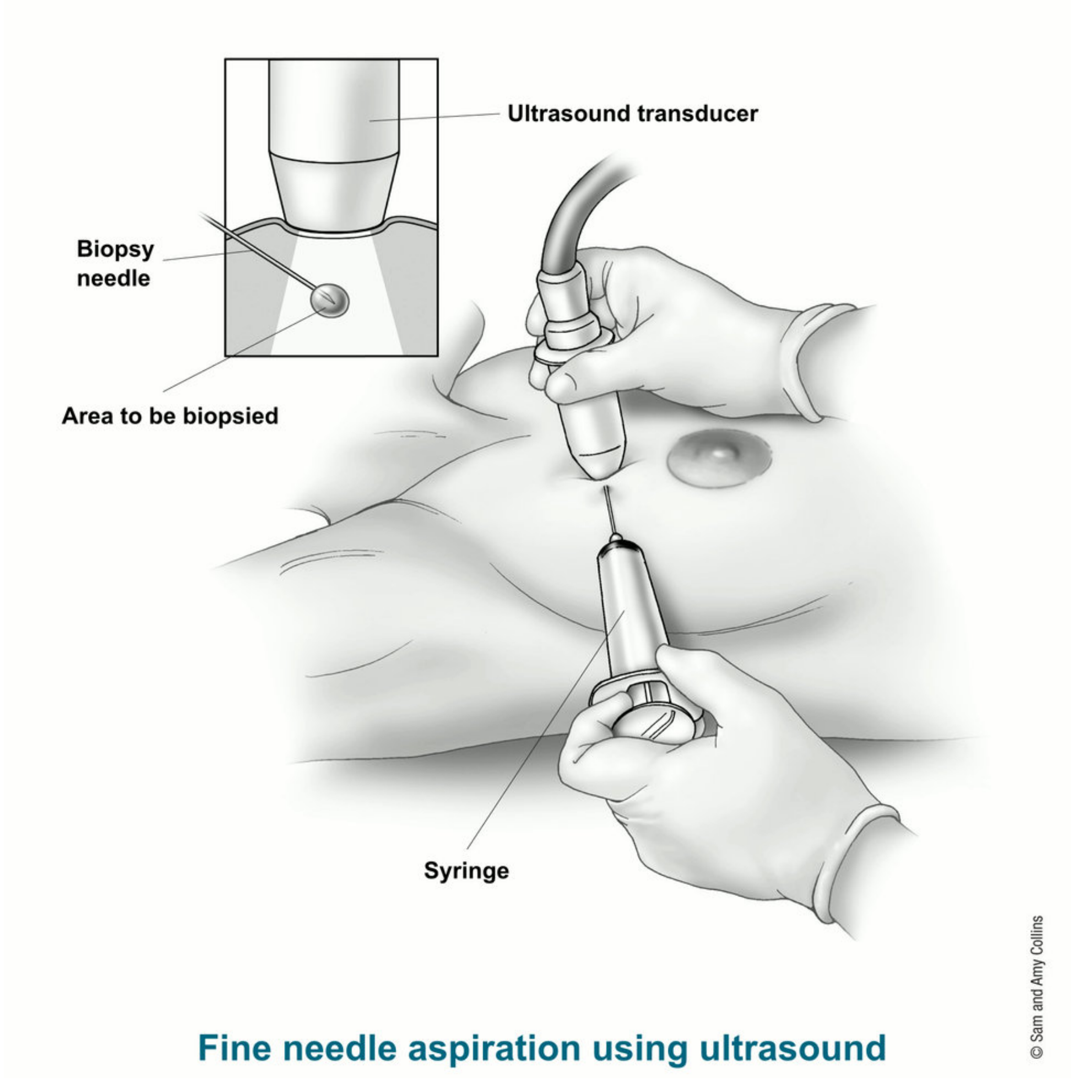


Figure 2: Fine needle aspiration using ultrasound (Source: Collins)

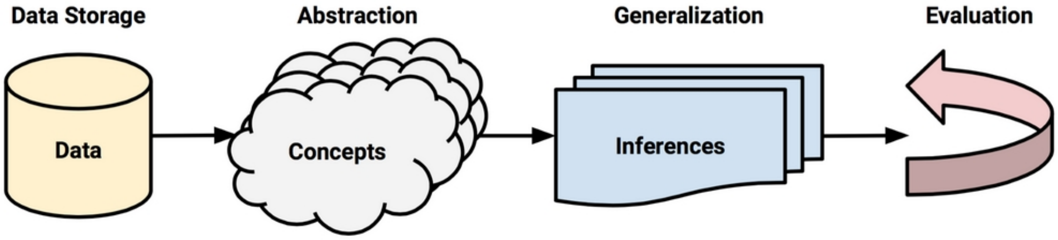


Figure 3: General machine learning process (Source: Lantz, 2019)

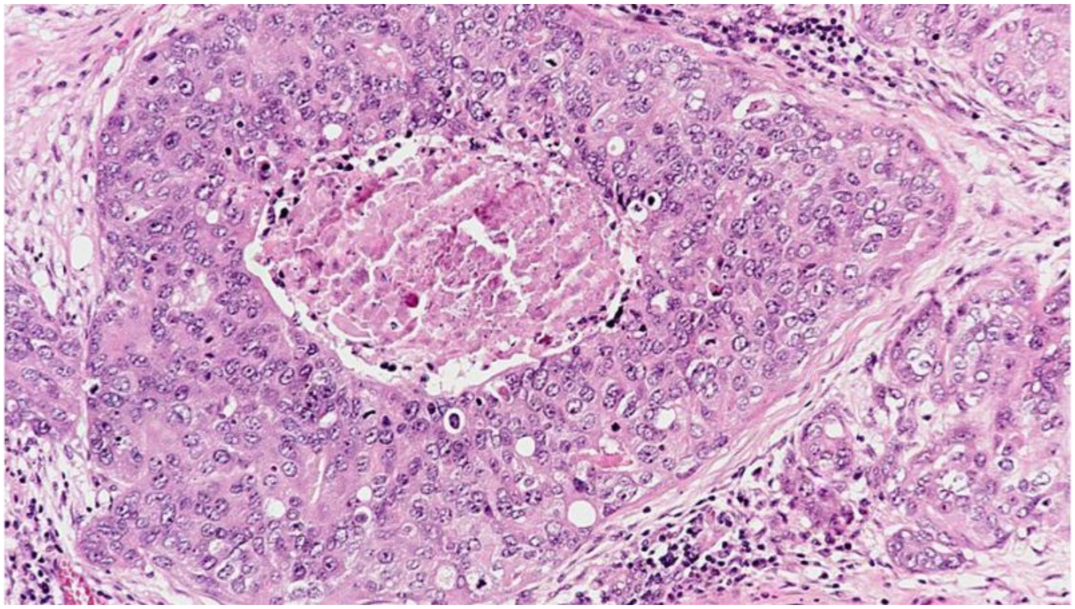
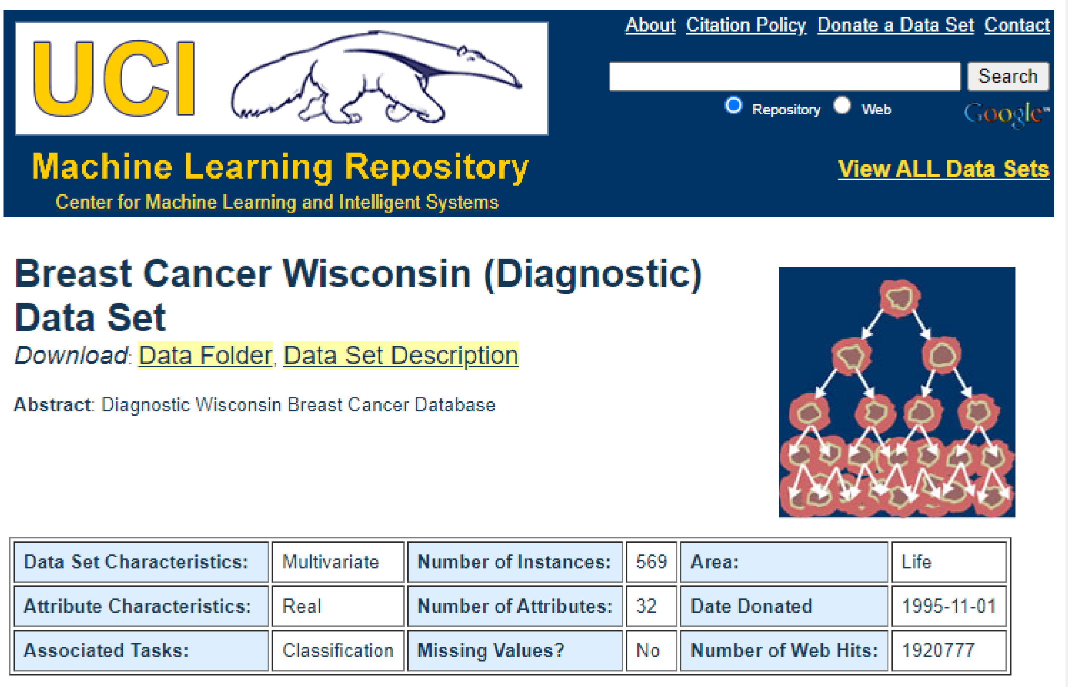


Figure 4: Ductal carcinoma in situ (Source: pathology.jhu.edu)

Getting the data



Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

Abstract: Diagnostic Wisconsin Breast Cancer Database

Data Set Characteristics:	Multivariate	Number of Instances:	569	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	32	Date Donated	1995-11-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	1920777

Figure 5: UCI Machine Learning Repository - Breast Cancer WI data set

- Origin: Univ of CA at Irvine (UCI) ML Repository
- Breast cancer data included 569 **examples** (aka instances, rows) of cancer biopsies (our data have header and randomized records)
- For each example 32 **features** (aka attributes, columns) were recorded:
  1. **Identity** number
  2. Cancer **diagnosis** (`M` for "malignant" or `B` for "benign")
  3. 30 Numeric laboratory measurements: **mean**, **standard error**, and **largest** value for 10 different cell nuclei characteristics: Radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension.

- Unless you're an oncologist, you won't know how each feature relates to benign or malignant masses. What does this mean for the data?[1]

Importing the data

- Import the CSV data file to a dataframe `wbcd` from url=bit.ly/3khqmkp
    1. assume that the data have a header
    2. do not automatically convert strings (`chr`) into factors
    3. check the `args` of the importing function if you're not sure

```
args(read.csv)
```

```
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
NULL
```

```
wbcd <- read.csv(
  file="http://bit.ly/3khqmkp",
  stringsAsFactors=FALSE)
```

- Check the structure of the data frame:

```
str(wbcd)
```

```
'data.frame': 569 obs. of  32 variables:
 $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87
 $ diagnosis        : chr  "B" "B" "B" "B" ...
 $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
 $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
 $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
 $ area_mean        : num  464 346 373 385 712 ...
 $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
 $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
 $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
```

[1] The data contain expertise bias from the oncologists who labelled them, i.e. who made the measurements, and potential mislabelling of the diagnosis label. The extent of these can only be estimated from reading the research papers that accompany the data and contain information about the methodology of data collection and coding.

```
 $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
 $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
 $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
 $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
 $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
 $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
 $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
 $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
 $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
 $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
 $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
 $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
 $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
 $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
 $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
 $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
 $ area_worst       : num  549 425 471 434 819 ...
 $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
 $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
 $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
 $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
 $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
 $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

- The variable id is a unique identifier for each patient in the data.
- Regardless of ML method, ID variables **should always be excluded**: a model that includes an ID column will suffer from overfitting and generalize poor data - can you think why?[2].
- Overwrite the data frame with itself after removing the first column, then check the first four examples and features only:

  ```
  wbcd <- read.csv(
    file="http://bit.ly/3khqmkp",
    stringsAsFactors=FALSE)
  wbcd <- wbcd[-1]
  wbcd[1:4,1:4]
  ```

[2] The identity column is a perfect predictor of the output variable. The model will learn to associate specific IDs with certain outcomes, instead of learning general patterns that apply to all data: this is overfitting.

```
  diagnosis radius_mean texture_mean perimeter_mean
1         B       12.32        12.39          78.85
2         B       10.60        18.95          69.28
3         B       11.04        16.83          70.92
4         B       11.28        13.39          73.00
```

```
  diagnosis radius_mean texture_mean perimeter_mean
1         B       12.32        12.39          78.85
2         B       10.60        18.95          69.28
3         B       11.04        16.83          70.92
4         B       11.28        13.39          73.00
```

Exploring the `diagnosis` target data

- The `wbcd[,2] = diagnosis`, is the outcome we want to predict: this feature indicates if the example is from a benign or malignant mass.
- How many examples are benign or malignant, respectively?

  ```
  table(wbcd$diagnosis)
  ```

  ```

    B   M 
  357 212 
  ```

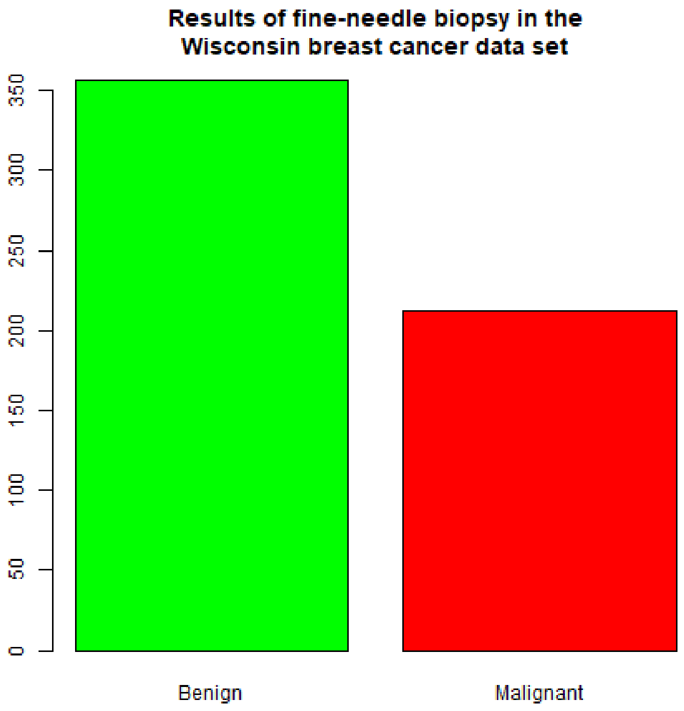
- kNN like many other ML classifiers require the target feature (aka class) to be coded as `factor` with `levels`.
- We recode `diagnosis` as a `factor` and add the `labels` "Benign" and "Malignant" - if you cannot remember `factor`, run `args` on it!

  ```
  wbcd <- read.csv(
    file="http://bit.ly/3khqmkp",
    stringsAsFactors=FALSE)
  wbcd <- wbcd[-1]
  wbcd[1:4,1:4]
  wbcd$diagnosis <- factor(wbcd$diagnosis,
                           levels=c("B","M"),
                           labels=c("Benign","Malignant"))
  str(wbcd$diagnosis)
  ```

```
  diagnosis radius_mean texture_mean perimeter_mean
1         B       12.32        12.39          78.85
2         B       10.60        18.95          69.28
3         B       11.04        16.83          70.92
4         B       11.28        13.39          73.00
 Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 1 1 1 2 1 1 ...
```

- We visualize the frequencies of the two diagnoses in a barplot, coloring the benign results green, and the malignant results red:

```
barplot(table(wbcd$diagnosis),
        col=c("green","red"),
        main=
          "Results of fine-needle biopsy in the\nWisconsin breast cancer data set"
```



- To obtain the relative percentage of the diagnosis results, we look at the proportions table:

```
cat("Relative percentages of breast cancer\n")
cat("masses in the Wisconsin data set:\n")
round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)

Relative percentages of breast cancer
masses in the Wisconsin data set:

   Benign Malignant
     62.7      37.3
```

Exploring the predictors

- The remaining 30 features are `numeric` and consist of different measurements of the 10 characteristics.
- List the first 3 rows of three of these predictors: `radius_mean`, `area_mean`, and `smoothness_mean`:

```
wbcd[1:3,c("radius_mean","area_mean","smoothness_mean")]

  radius_mean area_mean smoothness_mean
1       12.32     464.1         0.10280
2       10.60     346.4         0.09688
3       11.04     373.2         0.10770
```

- Compute a statistical `summary` of these three features:

```
summary(wbcd[c("radius_mean","area_mean","smoothness_mean")])

 radius_mean       area_mean      smoothness_mean
Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
Median :13.370   Median : 551.1   Median :0.09587
Mean   :14.127   Mean   : 654.9   Mean   :0.09636
3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

- What do you notice when looking at the values? Remember that distance calculation for k-NN depends on the measurement scale of the input.[3]

  ```
  range(wbcd["area_mean"])
  range(wbcd["smoothness_mean"])
  ```

  ```
  [1]  143.5 2501.0
  [1] 0.05263 0.16340
  ```

Intermission - Review from Thu 23-Feb-23

- Run the code from the last session so that you're caught up:

  ```
  ## get the Wisconsin breast cancer data as data frame:
  wbcd <- read.csv(file="http://bit.ly/3khqmkp")
  ## drop the first (ID) column:
  wbcd <- wbcd[-1]
  ## recode target class as labeled 2-level factor
  wbcd$diagnosis |> factor(c("B","M"),c("Benign","Malignant")) -> wbcd$diagnosis
  wbcd$diagnosis |>  str()
  ```

  ```
  Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 1 1 1 2 1 1 ...
  ```

Interlude: `function`

- We normalize the data using the min-max normalization formula, which we encapsulate in a `function`.

- User-defined functions work like other R functions: they take arguments and `return` the result of their computations.

- Example: defining a `hello` world `function` in R

  ```
  helloWorld <- function() {
    return ("hello world")
  }
  helloWorld()
  ```

[3] Area has a much larger range than smoothness - it will dominate the distance calculation and could confuse our classifier. We need to rescale, normalize or standardize the values.

```
[1] "hello world"
```

- Example: `hello` world `function` with an argument in R

  ```
  hello <- function(name) {
    paste("Hello,", name)  # without return, the last result is returned
  }
  hello("Marcus")

  [1] "Hello, Marcus"
  ```

Transforming - numeric data normalization

- To apply the min-max formula to the whole dataset, we define a function `normalize`:

  ```
  normalize <- function(x) {
    return ((x-min(x))/(max(x)-min(x)))
  }
  ```

- We test the function on some vectors:

  ```
  normalize(c(1,2,3,4,5))
  normalize(c(10,20,30,40,50))

  [1] 0.00 0.25 0.50 0.75 1.00
  [1] 0.00 0.25 0.50 0.75 1.00
  ```

- Looking good! The normalized scale values are identical.

Interlude: `lapply` and `tapply`

- One reason to define a function is that R offers implicit looping with the `apply` family of functions.

- The `lapply` function takes a list and applies an argument to each list element and returns a list. A data frame is a list:

  ```
  is.list(wbcd)
  args(lapply)
  ```

```
[1] TRUE
function (X, FUN, ...)
NULL
```

- Example: What are the mean values of the variables in the `airquality` data frame?

```
str(airquality)
lapply(X=airquality[1:4],FUN=mean, na.rm=TRUE)

'data.frame': 153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
$Ozone
[1] 42.12931

$Solar.R
[1] 185.9315

$Wind
[1] 9.957516

$Temp
[1] 77.88235
```

- Another useful function is `tapply`: it allows running a function on any feature of a dataframe grouped by `factor` levels.
- Example: what is the average (`mean`) of the largest cell radius measurements (`radius_worst`) for `Benign` and `Malignant` labels?

```
tapply(X = wbcd$radius_worst,   # subset = largest cell radius
       INDEX = wbcd$diagnosis,  # group by = diagnosis label
       FUN = mean)              # function = average values

  Benign Malignant
13.37980  21.13481
```

Applying `normalize` to the data frame

- We apply the `normalize` function to all elements of `wbcd` and convert the resulting `list` to a data frame `wcbd_n` using `as.data.frame`:

  ```
  wbcd_n <- as.data.frame(lapply(wbcd[2:31],FUN=normalize))
  ## show the first 3 x 4 results
  wbcd_n[1:3,2:4]
  ```

  ```
    texture_mean perimeter_mean  area_mean
  1    0.0906324      0.2422777 0.13599152
  2    0.3124789      0.1761454 0.08606575
  3    0.2407846      0.1874784 0.09743372
  ```

- To confirm that the transformation worked, let's look at the summary stats for `area_mean` and `smoothness_mean` again:

  ```
  summary(wbcd_n$area_mean)
  summary(wbcd_n$smoothness_mean)
  ```

  ```
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.0000  0.1174  0.1729  0.2169  0.2711  1.0000
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.0000  0.3046  0.3904  0.3948  0.4755  1.0000
  ```

Simulating new patient scenario

- All our 569 biopsies are already labelled so we know which are benign or malignant.

- Using all data for training leaves us not knowing if the data has been overfitted or how well the generalization to new cases works.

- We want to know how our learner performs on **unseen** data: unless you have access to new patients, you need to simulate this scenario.

- Simulation means splitting the data randomly in two sets:

  1. a **training data** set used to build the k-NN model
  2. a **test data** set used to estimate its predictive accuracy

- We'll use 469 records (82%) for the training dataset and the remaining 100 records (18%) to simulate new patients.
- For the simulation to work, it is important that each dataset is a **representative subset** of the full set of data.
- The data would not be representative if it was ordered chronologically or grouped by similar values.

Creating training and test data sets

- Split the normalized data frame, `wbcd_n` into two sets `wbcd_train` and `wbcd_test` using the first 469 and the next 100 values, respectively, and display the length of the results:

```
wbcd_train <- wbcd_n[1:469,]   # all normalized columns
wbcd_test <- wbcd_n[470:569,]  # all normalized columns
nrow(wbcd_train)
nrow(wbcd_test)
```

```
[1] 469
[1] 100
```

- To normalize the data, we excluded the target variable `diagnosis`. For training and testing, it needs to be stored.
- The `diagnosis` is the **class** that we want the learner to predict. Class variables are stored in `factor` vectors or labels, split between both data sets.
- Create `wbcd_train_labels` and `wbcd_test_labels` from `wcbd[,1]` by splitting the records in 469 training and 100 test records, then display the structure of the resulting vectors.

```
wbcd_train_labels <- wbcd[1:469,1]  # from the original dataset
wbcd_test_labels <- wbcd[470:569,1]  # from the original dataset
str(wbcd_train_labels[1:3])
str(wbcd_test_labels[1:3])
```

```
Factor w/ 2 levels "Benign","Malignant": 1 1 1
Factor w/ 2 levels "Benign","Malignant": 1 1 1
```

Getting the k-NN algorithm

- For the k-NN algorithm, the training phase involves no model building: training a "lazy learner" means storing the input data in a structured format.
- To classify the test instances, we use the `knn` function from the `class` package. Install and load it, then list all loaded packages:

  ```
  install.packages("class")
  library(class)
  search()
  ```

  ```
  Warning: package 'class' is in use and will not be installed
   [1] ".GlobalEnv"        "package:gmodels"   "package:class"
   [4] "ESSR"              "package:stats"     "package:graphics"
   [7] "package:grDevices" "package:utils"     "package:datasets"
  [10] "package:stringr"   "package:httr"      "package:methods"
  [13] "Autoloads"         "package:base"
  ```

- Look at the arguments of `knn`:

  ```
  args(knn)
  ```

  ```
  function (train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
  NULL
  ```

- Look at the `help` for `knn`:

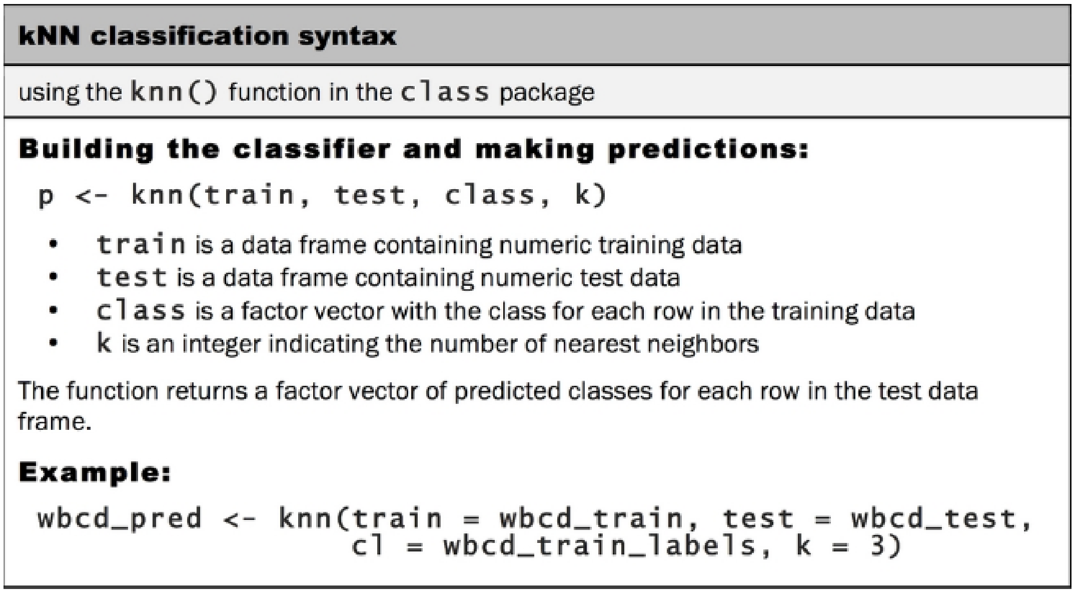
  ```
  help(knn)
  ```

- You can check in the R console if there are any other `knn` like functions available to you already, with the fuzzy search `??`. You can also search for kNN in the CRAN package repository.
- You can run the examples for `knn` (listed at the end of the `help`) file, with `example(knn)`:

  ```
  example(knn)
  ```

Classification with `class::knn`

- For each instance/row/record in the test data, `knn` will identify the `k` nearest neighbors using Euclidean distance, where `k` is a user-specified number.
- The test instance is classified by taking a "vote" among the `k` nearest neighbors - this involves assigning the class of the majority of the neighbors. A tie vote is broken at random.
- Training and classification is performed in a single command - we only use four of the available 7 parameters:



kNN classification syntax

using the `knn()` function in the `class` package

Building the classifier and making predictions:

```
p <- knn(train, test, class, k)
```

- `train` is a data frame containing numeric training data
- `test` is a data frame containing numeric test data
- `class` is a factor vector with the class for each row in the training data
- `k` is an integer indicating the number of nearest neighbors

The function returns a factor vector of predicted classes for each row in the test data frame.

Example:

```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

Figure 6: kNN classification syntax (Source: Lantz p. 83)

- The only parameter not discussed or set is `k`, the number of neighbors to include in the vote - a standard initial choice is to take the square root of the training data set size:

```
as.integer(sqrt(469))
```

```
[1] 21
```

- With a 2-category (benign or malignant) outcome, using an odd number eliminates the chance of ending with a tie vote.

- Use knn to classify the test data:

```
wbcd_test_pred <- knn(train = wbcd_train, # training data
                      test = wbcd_test,  # test data
                      cl = wbcd_train_labels, # class factor
                      k = 21)  # nearest neighbors
```

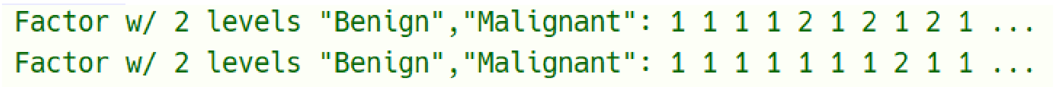
- What data structure do you expect as a result, and what will be its size?[4] How can you check?

```
str(wbcd_test_pred)
str(wbcd_train_labels)
length(wbcd_test_pred)
```

```
 Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 2 1 2 1 2 1 ...
 Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 1 1 1 2 1 1 ...
[1] 100
```

Evaluating model performance

- A performing model will have identified the labels in the test data set with high accuracy. Low accuracy means mis-identified labels:



```
Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 2 1 2 1 2 1 ...
Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 1 1 1 2 1 1 ...
```

- The tool to show accuracy is the **confusion matrix**, which shows the number of true and false positive and negative classification results.

- To build this table, we use the CrossTable function of the gmodels package. After installing the package, we can load it, look at the loaded packages.

[4] A factor vector, of course: one entry for each of the 100 values of the test data set, classified according to one of the levels/labels.

```
install.packages("gmodels")
library(gmodels)
search()

Warning: package ’gmodels’ is in use and will not be installed
 [1] ".GlobalEnv"        "package:gmodels"   "package:class"
 [4] "ESSR"              "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:stringr"   "package:httr"      "package:methods"
[13] "Autoloads"         "package:base"
```

- Look at the arguments of the function `CrossTable`:

```
args(CrossTable)

function (x, y, digits = 3, max.width = 5, expected = FALSE,
    prop.r = TRUE, prop.c = TRUE, prop.t = TRUE, prop.chisq = TRUE,
    chisq = FALSE, fisher = FALSE, mcnemar = FALSE, resid = FALSE,
    sresid = FALSE, asresid = FALSE, missing.include = FALSE,
    format = c("SAS", "SPSS"), dnn = NULL, ...)
NULL
```

- Fortunately, we only need two arguments (x,y). We also exclude the chi-square values from the output to make it more readable:

  1. x is the set of test data set labels used for classification
  2. y is the data set of predicted labels by `knn`

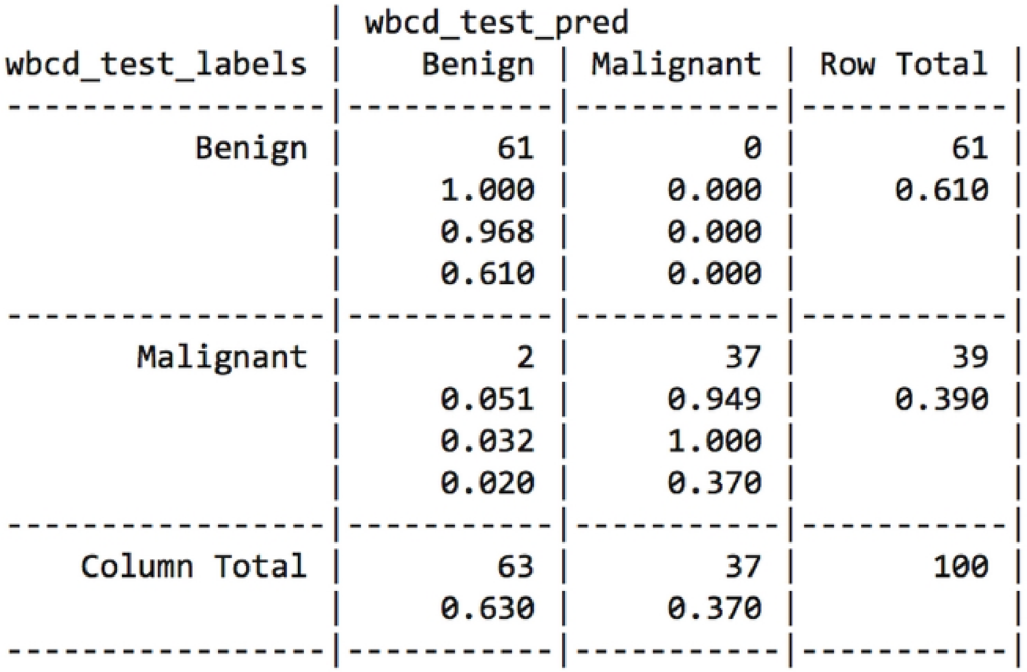
```
library(gmodels)
CrossTable(x = wbcd_test_labels,
           y = wbcd_test_pred,
           prop.chisq = FALSE)



   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|          N / Col Total |
|        N / Table Total |
|-------------------------|


Total Observations in Table:  100


 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
  Benign |        61 |         0 |        61 |
 |     1.000 |     0.000 |     0.610 |
 |     0.968 |     0.000 |           |
 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        37 |        39 |
 |     0.051 |     0.949 |     0.390 |
 |     0.032 |     1.000 |           |
 |     0.020 |     0.370 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        63 |        37 |       100 |
 |     0.630 |     0.370 |           |
-----------------|-----------|-----------|-----------|
```

- You can also just replace `gmodels::CrossTable` by `base::table`

```
table(x = wbcd_test_labels,
      y = wbcd_test_pred)
```

```
           y
x           Benign Malignant
  Benign        61         0
  Malignant      2        37
```

Analyze the confusion table



```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.968 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        37 |        39 |
                 |     0.051 |     0.949 |     0.390 |
                 |     0.032 |     1.000 |           |
                 |     0.020 |     0.370 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        63 |        37 |       100 |
                 |     0.630 |     0.370 |           |
-----------------|-----------|-----------|-----------|
```

1. Top-left: TRUE NEGATIVE results - 61/100
2. Bottom-right: TRUE POSITIVE results - 37/100
3. Bottom-left: FALSE NEGATIVE results - 2/100
4. Top-right: FALSE POSITIVES results - 0/100

What do these results mean?

1. "True negative" means that the patient had no tumor and the model recognized this.
2. "True positive" means that the patient had a tumor and the model recognized this.
3. "False negative" means that the patient had a tumor but the model did not recognize it.

4. "False positive" means that the patient had no tumor but the model found one.

Computing accuracy as an average

- The arithmetic average between the predicted and the original labels for the test data set corresponds to the percentage of cells correctly identified:

```
mean(wbcd_test_pred==wbcd_test_labels, na.rm=TRUE)
```

```
[1] 0.98
```

- This works because the `TRUE` and `FALSE` values of the `logical` argument are interpreted as `1` and `0` by the `mean` function:

```
wbcd_test_pred==wbcd_test_labels
```

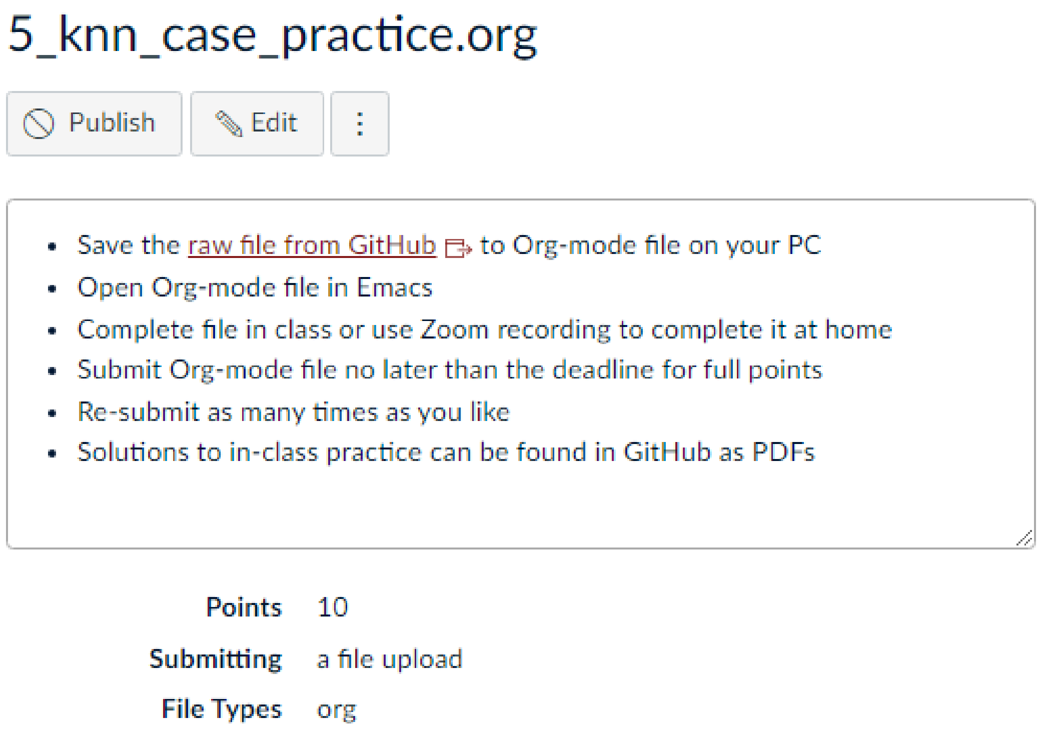
```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[49]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[85]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[97]  TRUE  TRUE  TRUE  TRUE
```

Improving model performance

- Perform an alternative rescaling of numeric features (z-score)

- Run the model for different `k` values to find the optium value

Upload the completed practice file to Canvas



5_knn_case_practice.org

- Save the raw file from GitHub to Org-mode file on your PC
- Open Org-mode file in Emacs
- Complete file in class or use Zoom recording to complete it at home
- Submit Org-mode file no later than the deadline for full points
- Re-submit as many times as you like
- Solutions to in-class practice can be found in GitHub as PDFs

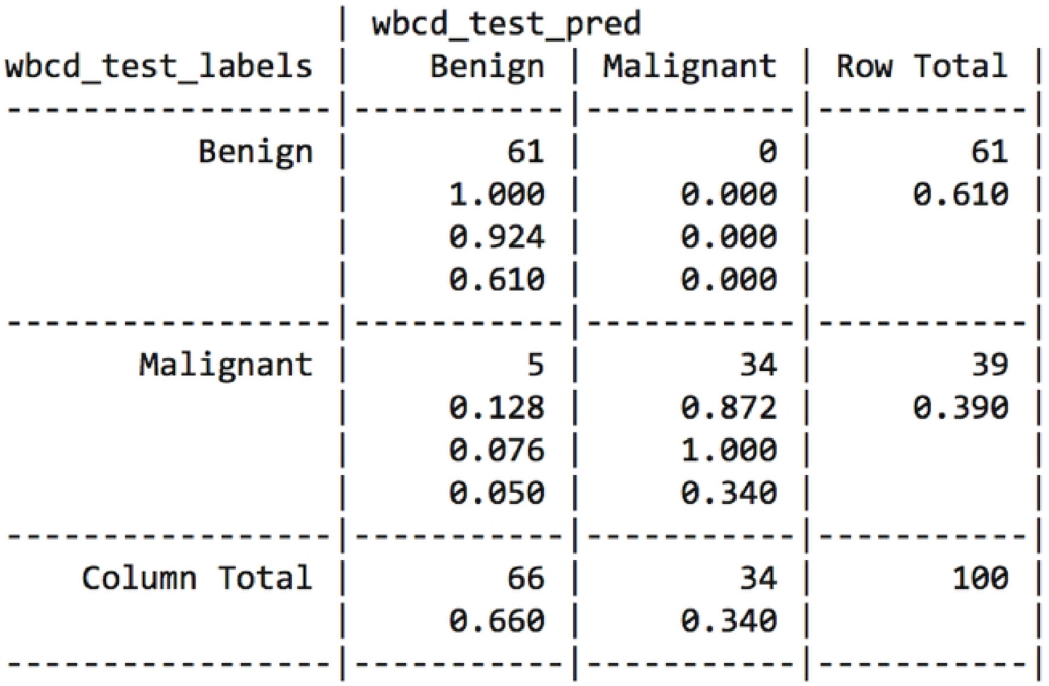
Points	10
Submitting	a file upload
File Types	org

Bonus exercises - improve the model performance

You find these exercises in GitHub as `5_knn_exercise_1.org` and `5_knn_exercise_2.org`.

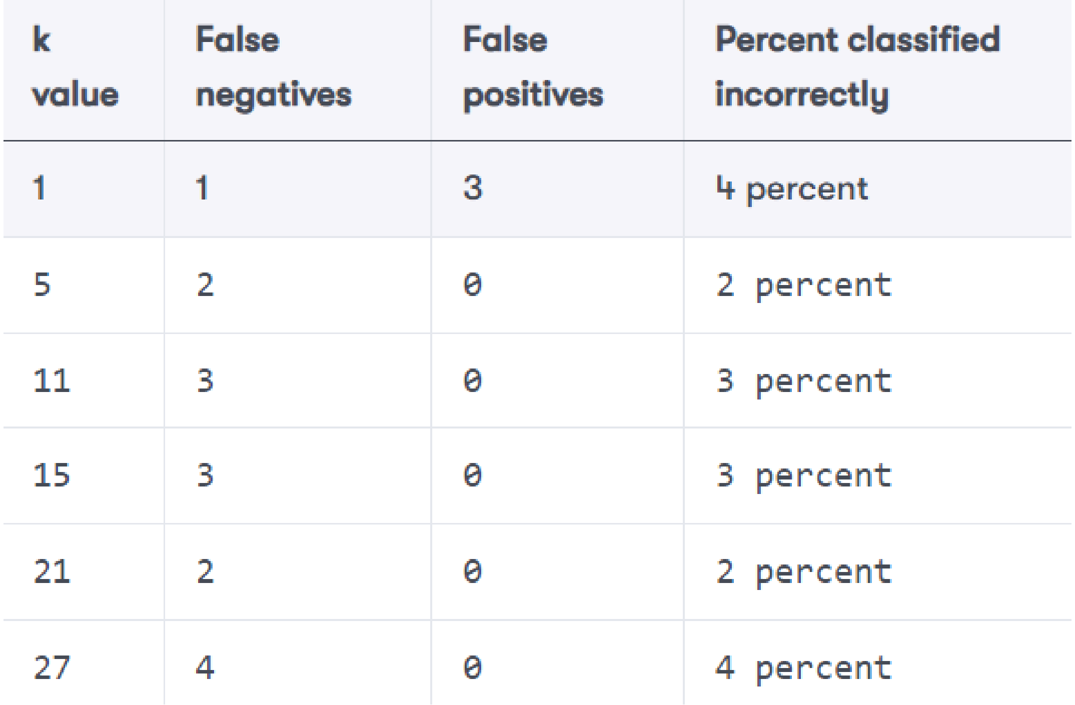
1. Rescaling: Use the z-score standardization to transform the data, check and interpret the predictions.

2. Measuring: Use different values of `k`, check and interpret the predictions: k = 1, 5, 11, 15, 21, 27.



```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.924 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         5 |        34 |        39 |
                 |     0.128 |     0.872 |     0.390 |
                 |     0.076 |     1.000 |           |
                 |     0.050 |     0.340 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        66 |        34 |       100 |
                 |     0.660 |     0.340 |           |
-----------------|-----------|-----------|-----------|
```

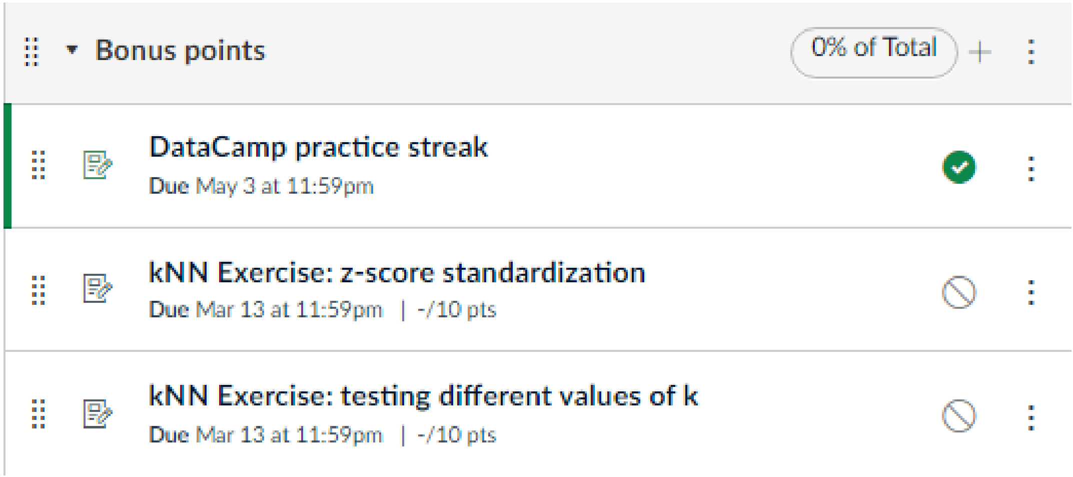
Figure 7: Sample results - Confusion matrix after z-score standardization



k value	False negatives	False positives	Percent classified incorrectly
1	1	3	4 percent
5	2	0	2 percent
11	3	0	3 percent
15	3	0	3 percent
21	2	0	2 percent
27	4	0	4 percent

Figure 8: Sample results - kNN On wbcd for different k

Upload the completed exercise files to Canvas



Bonus points — 0% of Total

DataCamp practice streak
Due May 3 at 11:59pm ✓

kNN Exercise: z-score standardization
Due Mar 13 at 11:59pm | -/10 pts ⃠

kNN Exercise: testing different values of k
Due Mar 13 at 11:59pm | -/10 pts ⃠

References

- Image: Ductal carcinoma in situ (URL: pathology.jhu.edu)
- Image: Fine-needle aspiration using ultrasound (URL: cancer.org)
- Data: Breast Cancer Diagnosis and Prognosis via Linear Programming, Mangasarian OL, Street WN, Wolberg WH, Operations Research, 1995, Vol. 43, pp. 570-577. URL: archive.ics.uci.edu/ml/
- Lantz (2019). Machine Learning with R (3e). Packt.

Glossary of Code

COMMAND	MEANING
`args`	function arguments
`read.csv`	read CSV file into data frame
`stringsAsFactors`	`read.csv` argument to turn char into factors
`str`	structure of R object
`df[-1]`	remove first column from data frame `df`
`table(x)`	frequency table for categorical vector `x`
`table(x,y)`	cross table for vectors `x` and `y`
`labels`	`factor` labels
`levels`	`factor` levels
`cat`	paste strings to screen
`prop.table`	proportions for frequency table
`round`	rounding function
`summary`	statistical summary
`range`	difference between min and max
`function`	create function
`return`	return function argument
`normalize`	user-defined function to normalize list
`unlist`	turn `list` into vector
`lapply`	apply `FUN` to all `list` argument members
`nrow`	number of rows
`search`	R environment (session) search path
`library`	load package
`as.integer`	turn argument into `integer`
`sqrt`	square root
`length`	length of vector
`gmodels::CrossTable`	cross tabulation
`class::knn`	k-Nearest Neighbor model building
`scale`	z-score standardization

Summary

- The case study used the Wisconsin Breast Cancer Dataset of cell data obtained from The UCI Machine Learning repository and randomized thereafter.

- The target class (2-category cancer diagnosis) was converted to a nominal, labelled factor vector, while the predictors were normalized using the min-max normalization method.

- Data were split in training and test data (80/20) and classified using the `knn` function from the `class` package.
- Model performance was evaluated using `base::table` and `CrossTable` from the `gmodels` package to create a confusion matrix.
- Model improvements were attempted with z-score standardization and by testing a variety of k values.

References

- Lantz (2019). Machine learning with R (3e). Packt. URL: packtpub.com.
- R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.
- Ripley/Venables (January 23, 2023). Package 'class': Various functions for classification, including k-nearest neighbour, Learning Vector Quantization, and Self-Organizing Maps. URL: cran.r-project.org.
- Warnes (October 13, 2022). Package 'gmodels': Various R Programming Tools for Model Fitting. URL: cran.r-project.org.