Data Frame: A case study.

Song Liu (song.liu@bristol.ac.uk)

GA 18, Fry Building,

Microsoft Teams (search "song liu").

In today's lecture, we will introduce a widely used feature in R, data frame.

Creating a Scoring Table

Imagine we would like to create a "table" which stores data of different types:

```
student number, name, male, score
1               song,     T,  70
2               jack,     T,  80
3               emma,     F,  70
...
```

We cannot use `matrix` as it only allows single type data.

Using `list`

- Since we know that `list` can be used to store data of different types, can we use it to create the above table?

Using `list`

```
student number, name, male, score
1               song,     T,   70
2               jack,     T,   80
3               emma,     F,   70
...
```

Can be stored as

```
# creating vectors storing each column of the table.
stu_no <- c(1,2,3)
name <- c("song", "jack", "emma")
male <- c(T,T,F)
score <- c(70, 80, 70)
# combine these lists together
score_table <- list(no = stu_no,
          name=name, male = male, score = score)
```

Using `list`

This solution does work:

```
> score_table$name
[1] "song" "jack" "emma"
> score_table[[2]]
[1] "song" "jack" "emma"

> score_table[[2]][1]
[1] "song"
```

Using `list`

However, you are stuck with list style index which can be cumbersome when you are dealing with a table:

```
# indexing a single element, works
> score_table$name[1]
[1] "song"
```

What if I want to find names of students whose score > 72?

```
# works, but a bit too wordy...
score_table$name[score_table$score > 72]
[1] "jack"
```

Data Frame

- Data frame is a list whose elements are equal-length vectors/lists.

```r
stu_no <- c(1,2,3)
name <- c("song", "jack", "emma")
male <- c(T,T,F)
score <- c(70, 80, 70)
# no need to specify the names of each column,
# like what we did in the list case.
score_table <- data.frame(stu_no, name, male, score)
```

Displaying Data Frame

You can see `data.frame` automatically selected names for the columns.

```
> score_frame
  stu_no name  male score
1      1 song  TRUE    70
2      2 jack  TRUE    80
3      3 emma FALSE    70
```

Summarizing Data Frame

You can also get a summary of your data frame:

```
> summary(score_frame)
     stu_no          name              male
 Min.   :1.0   Length:3          Mode :logical
 1st Qu.:1.5   Class :character  FALSE:1
 Median :2.0   Mode  :character  TRUE :2
 Mean   :2.0
 3rd Qu.:2.5
 Max.   :3.0
     score
 Min.   :70.00
 1st Qu.:70.00
 Median :70.00
 Mean   :73.33
 3rd Qu.:75.00
 Max.   :80.00
```

Indexing Data Frame like a Matrix

You can index the data frame as if it was a matrix:

```
> score_frame[1,2]
[1] "song"
> score_frame[1:2,]
  stu_no name male score
1      1 song TRUE    70
2      2 jack TRUE    80
> score_frame[,1:2]
  stu_no name
1      1 song
2      2 jack
3      3 emma
```

Indexing Data Frame like a List

Since data frames are lists. You can use list-style indexing.

```
> score_frame$name[1]
[1] "song"
> score_frame$name[1:2]
[1] "song" "jack"
```

You can also mix matrix and list style indexing:

```
# displaying rows with scores > 75.
> score_frame[score_frame$score > 75,]
  stu_no name male score
2      2 jack TRUE    80
```

Adding/Deleting Data from Data Frame

Since data frames are lists, you can add a new column to the data frame as if you were adding a new element to the list.

```
score_frame$age <- c(23, 23, 21)
```

Similarly, you can remove a column by setting the relevant list element to `NULL`.

```
score_frame$age <- NULL
```

Since data frame can be indexed as it if was a matrix, you can also add/delete rows/columns using matrix style deletion by reassignment.

```
score_frame <- score_frame[-1, ] #delete the first row
```

Case Study: Predicting MNIST Digits.

- Let us revisit the TB1 project and how data frame can be useful in this specific application.

Data Structure Analysis

- Recall, we have the following matrices
  - Labelled Images: $X \in \mathbb{R}^{1987 \times 784}$
  - Labels: $Y \in \mathbb{R}^{1987 \times 1}$
  - Testing Images: $T \in \mathbb{R}^{100 \times 784}$
- We can directly use these matrices and use them in our program (as we did in the TB1 project).
- However, by using data frame, we can organize them in a more meaningful way, making our program easily understandable.

Structuring Data using Data Frame

- Let us create two data frames. One for the labelled image dataset, the other for our test datasets.
  - Labelled image dataset will have two columns.
    ```
    image_data, label
    ```
  - Test image dataset will also have two columns.
    ```
    image_data, prediction
    ```

Structuring Data using Data Frame

- Let us construct the labelled data frame.

```r
# loading data from matrix files.
# R IO functions are similar to C,
# See definition of readmat function provided by
# the lecturer.
X <- readmat("X.matrix")
Y <- readmat("Y.matrix")
T <- readmat("T.matrix")
```

- Columns in data frame must be equal-length vectors/lists. In this case, we create lists where each element is a flattend image.

```r
Xlist <- list()
for(i in 1:dim(X)[1]){
    Xlist[[i]] <- X[i,]
}
```

Structuring Data using Data Frame

- Create the labelled data frame:

```
labelled_data <- data.frame(imagedata = I(Xlist), labels = Y)
```

- Here `labelled_data` contains two columns, `imagedata` and `labels` .
- The `I` function indicates we want the list elements stored in the data frame **as is**. Without using `I` function, R will treat each component of `Xlist` vectors as a column.

```
> dim(labelled_data) # labelled_data
[1] 1987    2
```

Visualizing Images

- With this data structure, we can easily visualize images.

```r
# [[]] imagedata is a list
img <- labelled_data$imagedata[[1]]
# convert the flattend vector to 28 * 28 matrix
img <- matrix(img, nrow = 28, ncol = 28)
# use as.cimg to convert img matrix to an image object
plot(as.cimg(img))
# add plot title.
title(labelled_data$label[1])
```

Visualizing Images

- For our conveniences, let us add the image objects for each digit as a new column to the data frame, so we can easily plot them.

```
cvrt <- function(imgdata){
    img <- imgdata
    img <- matrix(img, nrow = 28, ncol = 28)
    return(as.cimg(img))
}

labelled_data$pics <- lapply(labelled_data$imagedata, cvrt)
> dim(labelled_data) # now we have three columns.
[1] 1987    3
```

- To plot a digit

```
plot(labelled_data$pics[[1]])
```

Test Data Frame

- Now let us construct test data frame.
- First, add flattened image data and converted pictures

```
test_data <- data.frame(imagedata = I(Tlist))
test_data$pics <- lapply(test_data$imagedata, convert)
```

- Now, let us generate a prediction for each digit in the test data.

Find Neighbours

- First, let us write a function produces the 5 nearest neighbours (has the smallest distances) to a test image

```r
# takes a flattend image, return indices its 5 nearest neighbours
find_nei <- function(a){
    dist <- c()
    for(i in 1:length(labelled_data$imagedata)){
        dist[i] <- sqrt(sum((a-labelled_data$imagedata[[i]])^2))
    }

    return(order(dist)[1:5])
}
```

- Verify our function

```r
> nei <- find_nei(test_data$imagedata[[1]])
> labelled_data$labels[nei]
[1] 7 7 7 7 7
> plot(test_data$pics[[1]])
```

Prediction

- Let us make a prediction based on our neighbours' labels.

```r
predict <- function(a){
    nei <- find_nei(a)
    # creating a count vector, recording label counts.
    count <- matrix(0, nrow = 1, ncol = 10)
    # add the label by one, so they start from 1.
    labels <- labelled_data$labels[nei] + 1
    # counting labels
    for(i in 1:5){
        count[labels[i]] <- count[labels[i]] + 1
    }

    print(count)
    # find index of the biggest count, -1 ,
    # as our prediction
    return(which.max(count)-1)
}
```

Apply Prediction

Apply `predict` function to all test images.

```
test_data$pred <- sapply(test_data$imagedata, predict)
> dim(test_data)
[1] 100   3
```

Check the prediction of test images and see they match.

```
for(i in 1:100){
  plot(test_data$pics[[i]])
  title(test_data$pred[i])
  Sys.sleep(1)
}
```