MQIM R BOOTCAMP

Advanced topics

Monday, Dec 7, 2020

1. Bootcamp Week Schedule
2. R Markdown, R Projects and Github
3. Basic Machine Learning, R for KNN
4. Introduction to deep learning (keras library)
5. Matlab
6. Python

Section 1

Bootcamp Week Schedule

Table of Content

- Introduction to R
- The R Language, Data Types, Functions, Loops, Import/Export, Plot
- Basic Exploratory Data Analysis
- Basic Statistics
- Getting Financial Data in R
- R Class, Object and functions
- Data Preparation, Transformation and Visualization (tidyverse package)
- Model Building
- Advanced topics: Rmarkdown, Shiny, Github
- Basic Machine Learning and Deep Learning using R
- Introduction to Python/Matlab
- Introduction to BQL/BQUANT

Section 2

R Markdown, R Projects and Github

R Markdown

Why R Markdown?

- R Markdown provides various great formats for communicating, presenting as well as publishing rearch results.
- It enables the final report to include code chunks and output (table, plot etc.) while executing the codes.
- In addition, it makes typing complicated formulas and equations much less painful.
- It also ensures reproducibility and consistency. You can keep your code, notes, graphs and relevant links all in one place.
- Of course, a great way to submit your assignment.

R Markdown

How it works?

- create *.Rmd* file, select *File > New File > R Markdown.* in the menubar. RStudio will launch a wizard that you can use to pre-populate your file with useful content that reminds you how the key features of R Markdown work.
- **knit** the document, R Markdown sends the .Rmd file to knitr, http://yihui.name/knitr/. R Markdown executes all of the code chunks and creates a new markdown (.md) document which includes the code and its output.
- The markdown file generated by *knitr* is then processed by **pandoc**, http://pandoc.org/. R Markdown is responsible for creating the final file.

R Markdown

R Markdown formats:

- Documents:
  - html
  - pdf
  - word
- Presentations:
  - ioslides - HTML presentation with ioslides
  - slidy - HTML presentation with W3C Slidy
  - beamer - PDF presentation with LaTeX Beamer.
  - ppt

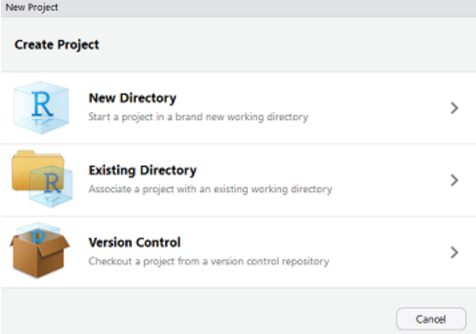
Rstudio Projects

It is a good practice to create a projects for an ongoing research process with kept developing codes. The beauty of using projects are:

- Holds all the files relevant to that particular piece of work in a folder in your computer in a desired project directory. - Set unite working directory to Project directory, save the trouble for typing *setwd("C:/Users/path)* and *rm(list = ls())* for each r script.
- Dedicated R process. File browser pointed at project directory.
- The way to create RStuodio Project is quite flexible, you can create it in a new folder, in a existing folder, or link it to a version control repository (we will talk about GitHub here).

Rstudio Projects

To create a new Rstudio Projects, go to *RStudio -> New Project*, you will see options:



Click **New Directory** if you'd like to make a new folder as project, or **Existing Directory** to make existing folder into an RStudio Project.

GitHub

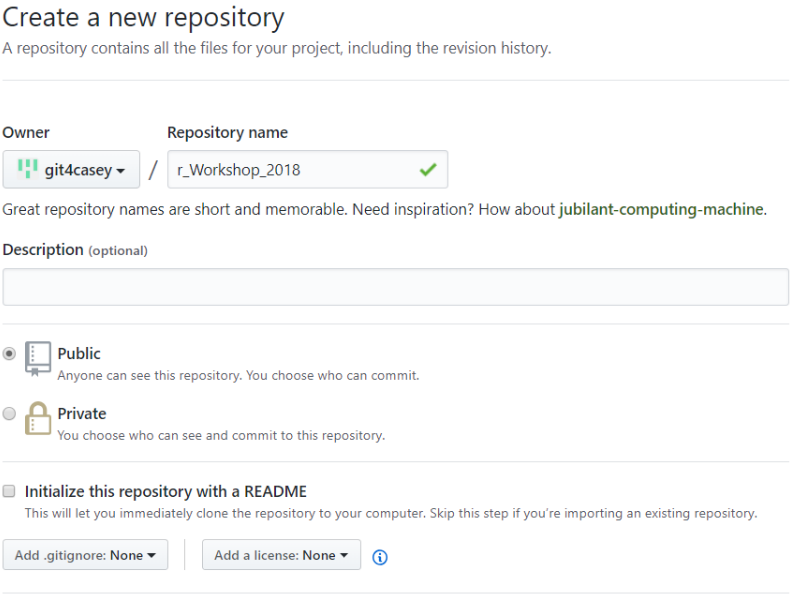
What is it?

- Git is an open-source version control system that was started by Linus Trovalds, whom created Linux.
- When developers create something (an app, for example), they make constant changes to the code, releasing new versions after the first official release.
- Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.
- People who have nothing to do with the development of a project can still download the files and use them.
- "Hub" part in GitHub is https://github.com/

GitHub

- Repository: a repository("repo") is a location where all the files for a particular project are stored. Each project has its own repo, and you can access it with a unique URL.
- Forking a Repo
  - create a new project based off of another project that already exists.
  - fork the repo that you'd like to contribute to, make the changes you like and release the revised project as a new repo.
- You can fork the **R_Workshop** repo from my github to yours to access some R files and data.

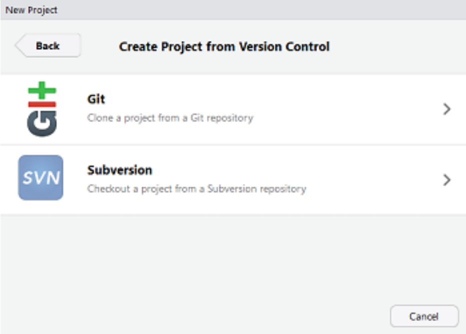
GitHub


Create a new repository
A repository contains all the files for your project, including the revision history.
Owner
Repository name
git4casey
/
r_Workshop_2018
Great repository names are short and memorable. Need inspiration? How about jubilant-computing-machine.
Description (optional)
Public
Anyone can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
Add .gitignore: None
Add a license: None

Link Rstudio Project to Github

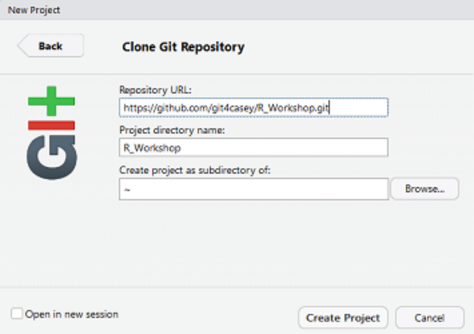
How to use it along with Rstudio Project?

- When creating the Rstudio Project, go to the third option **Version Control**

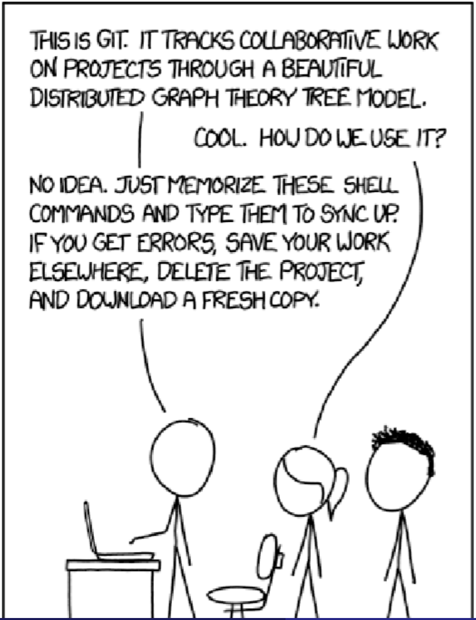


Link Rstudio Project to Github

- Select **Git**, and paste the repo URL you'd like to work on. For example, it can be the forked **R_Workshop** repo on your github.
- Select the preferred the directory in your computer by *Browse...*, then click *Create Project*. Now you should be able to see all the files in the repo from Rstudio, and edit and push to update your github repo.



GitHub

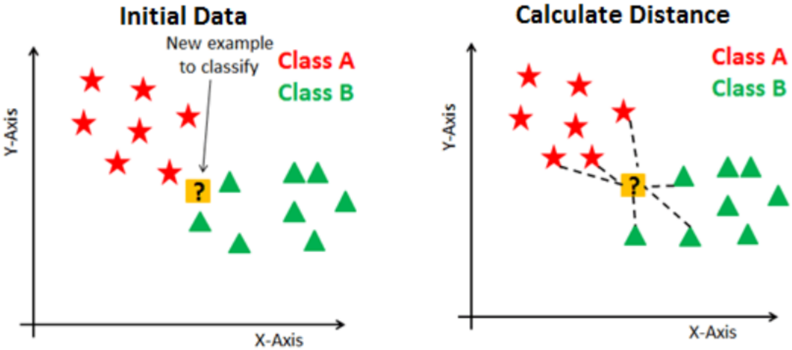

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Section 3

Basic Machine Learning, R for KNN

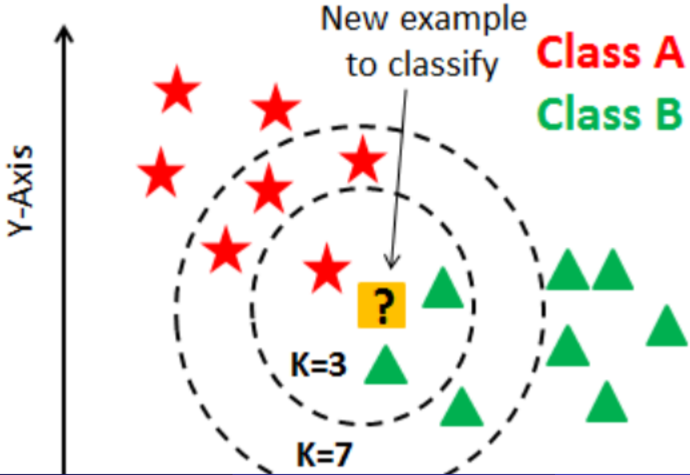
KNN concepts

- The KNN or k-nearest neighbors algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances.
- More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure.
- This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance.



KNN

The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. Look at the case below: The question is how to choose the optimal number of *k* neighbors?



KNN

No unified answer that suits all kind of data sets. In general, requires test on different k and validate the performance. Research has also shown that

- a small number of neighbors are most flexible fit which will have low bias but high variance
- a large number of neighbors will have a smoother decision boundary, which implies lower variance but higher bias.

KNN Iris Example in R

Let's use famous iris dataset to understand how knn works in R. To inspect the data, some simple summary:

```
## -- Attaching packages --------------------------------------- tic

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
```

KNN Iris Example in R

To visualize if there is any correlation between variables.

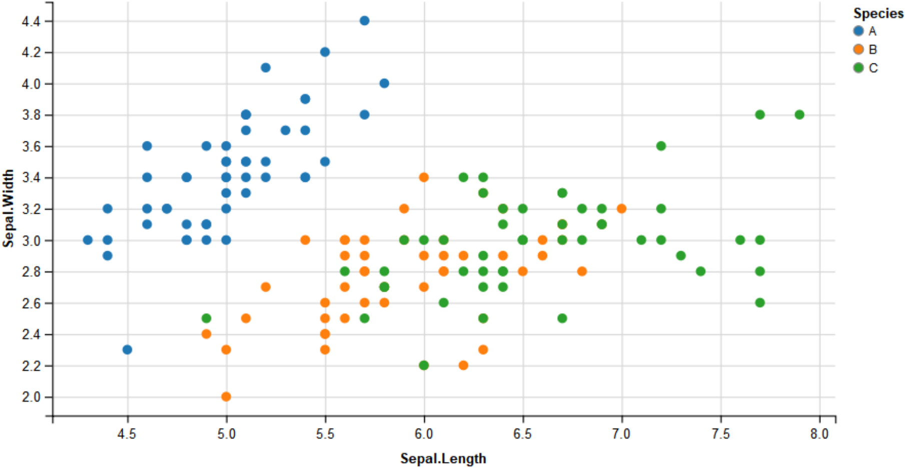


Figure 2: flowchart

KNN Iris Example in R

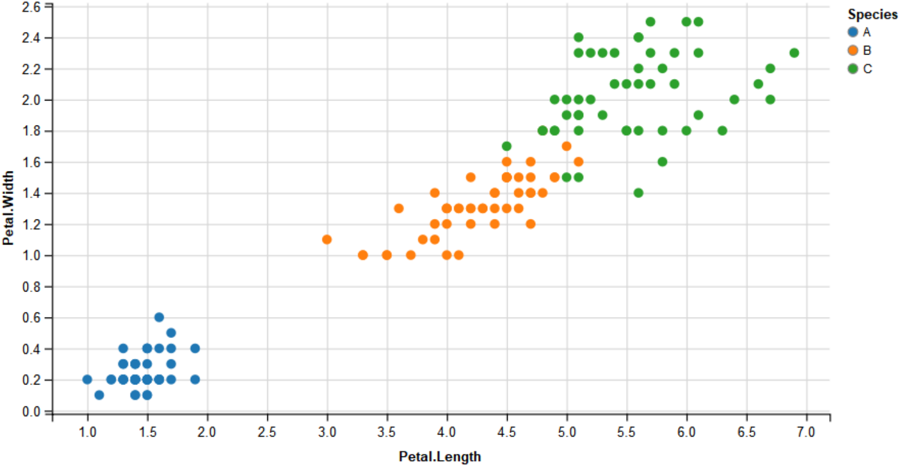


Figure 3: flowchart

KNN Iris Example in R

- Divide the data set into two parts: a training set and a test set. (2/3, 1/3 split)

```
> ind <- sample(2, nrow(iris_new), replace=TRUE, prob=c(0.67, 0.33))
> iris_training <- iris_new[ind==1, 1:4]
> iris_test <- iris_new[ind==2, 1:4]
> iris_trainLabels <- iris_new[ind==1,5]
> iris_testLabels <- iris_new[ind==2, 5]
```

- Using the knn() function, which uses the Euclidian distance measure in order to find the k-nearest neighbors to your new, unknown instance.

```
> library(class)
> iris_pred <- knn(train = iris_training, test = iris_test,
+                  cl = iris_trainLabels, k=3)
> iris_pred
```

```
##  [1] A A A A A A A A A A A A A A A A A A A B B B B B B B B B B B
## [39] C C C B C C B C C C C C B C B
## Levels: A B C
```

KNN Iris Example in R

- Compare the predicted class to the test labels.

```
##   test_lable pred_lable diff
## 1          A          A    1
## 2          A          A    1
## 3          A          A    1
## 4          A          A    1
## 5          A          A    1
## 6          A          A    1
```

```
## [1] "Accuracy: 92.4528301886792%"
```

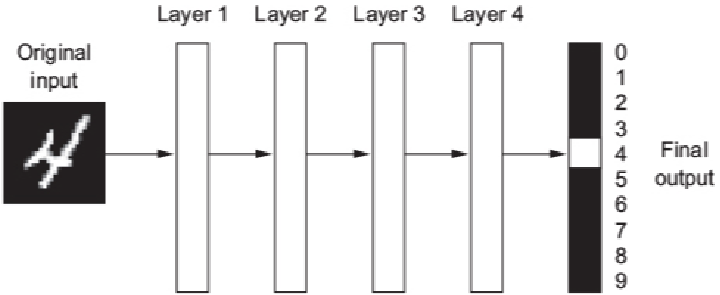
Section 4

Introduction to deep learning (keras library)

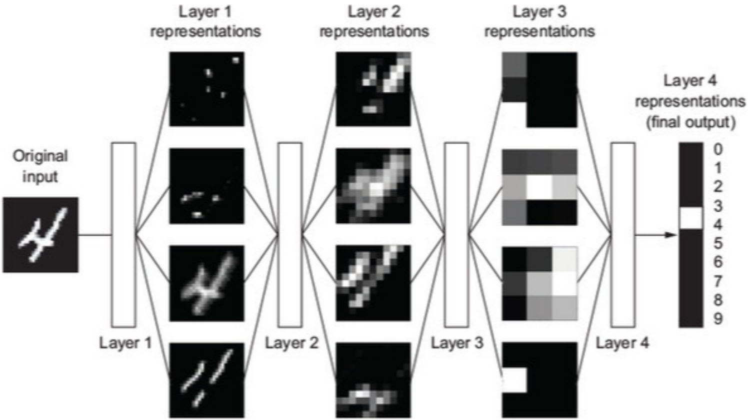
What's the "deep"

- Emphasize on learning through successive *layers* of increasingly meaningful representations.
- these layered representations are (mostly) learned via models called *neural networks*, it structured in layers stacked on top of each other

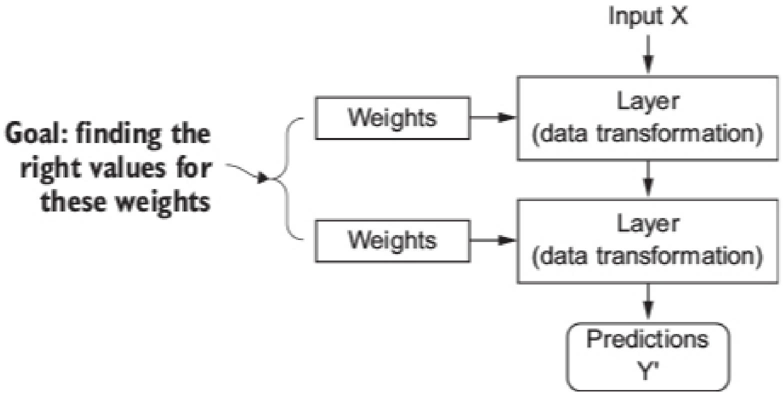
Layers representation



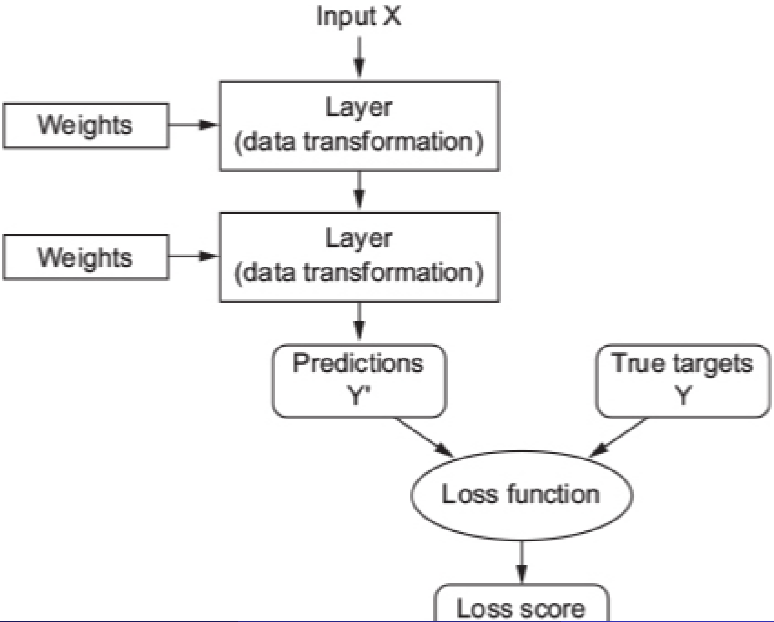
Layers representation



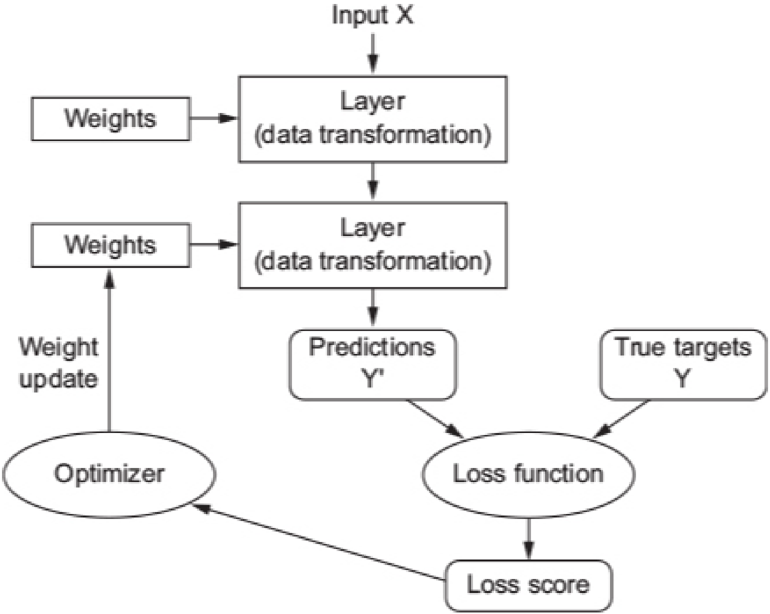
The workflow



The workflow



The workflow



Data structure

In general, all current machine-learning systems use multidimensional arrays as their basic data structure, it also called *tensors*.

Tensors are a generalization of vectors and matrices to an arbitrary number of dimensions.

Tensors

- 0D tensors(Scalars)

A tensor that contains only one number is called a *scalar* or *zero-dimensional tensor*. R doesn't have a *scalar* data type, but you can think R vector with length 1 is conceptually a scalar.

- 1D tensors(Vectors)
- 2D tensors(Matrices)
- 3D tensors and higher-dimensional tensors

Tensors

If you pack 2D tensors(matrices) in a new array, you will get a 3D tensor.

```
> x <- array(rep(0,2*3*2), dim = c(2,3,2))
> dim(x)
```

```
## [1] 2 3 2
```

By packing 3D tensors in an array, u can create a 4D tensor, and so on. In deep learning, you'll generally manipulate tensors up to 5D.

MNIST dataset

Let's look at a concrete example of a neural network that uses the Keras R package to learn to classify handwritten digits. The MNIST dataset comes preloaded in Keras, in the form of *train* and *test* lists, each of which includes a set of images (x) and associated labels (y).

```
> library(keras)
> library(tensorflow)
> #install_tensorflow()
> #mnist <- dataset_mnist()
>
> load("data/mnist.RData")
```

MNIST dataset

The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 X 28 pixels) into their 10 categories ($0 \sim 9$).

```
> train_images <- mnist$train$x
> train_labels <- mnist$train$y
> test_images <- mnist$test$x
> test_labels <- mnist$test$y
```

MNIST dataset

The network architecture

```
> network <- keras_model_sequential() %>%
+   layer_dense(units = 512, activation = "relu",
+               input_shape = c(28 * 28)) %>%
+   layer_dense(units = 10, activation = "softmax")
```

MNIST dataset

The compilation step

```
> network %>% compile(
+   optimizer = "rmsprop",
+   loss = "categorical_crossentropy",
+   metrics = c("accuracy")
+ )
```

MNIST dataset

Preparing the image data

```
> train_images <- array_reshape(train_images, c(60000, 28 * 28))
> train_images <- train_images / 255
>
> test_images <- array_reshape(test_images, c(10000, 28 * 28))
> test_images <- test_images / 255
>
> train_labels <- to_categorical(train_labels)
> test_labels <- to_categorical(test_labels)
```

MNIST dataset

Now ready to train the network, which in Keras is done via a call to the networks's *fit* method - we *fit* the model to its training data:

```
> network %>% fit(train_images, train_labels, epochs = 3,
+                 batch_size = 125)
```

MNIST dataset

Generate predictions for the first 10 samples of the test set:

```
> network %>% predict_classes(test_images[1:10,])
```

```
##  [1] 7 2 1 0 4 1 4 9 5 9
```

MNIST dataset

Now let's evaluate the model performs on the test set

```
> metrics <- network %>% evaluate(test_images, test_labels)
```

Section 5

Matlab

Matlab

- MATLAB is a programming platform designed specifically for engineers and scientists
- Like R, MATLAB has many specialized toolboxes for making things easier for us
- Using MATLAB, you can:
  - Analyze data
  - Develop algorithms
  - Create models and applications

Matlab

- No need for types. i.e.,

~~int a;~~
~~double b;~~
~~float c;~~

- All variables are created with double precision unless specified and they are matrices.

```
Example:
>>x=5;
>>x1=2;
```

- After these statements, the variables are 1x1 matrices with double precision

Matlab

- a vector `x = [1 2 5 1]`

```
x =
      1    2    5    1
```

- a matrix `x = [1 2 3; 5 1 4; 3 2 -1]`

```
x =
      1      2      3
      5      1      4
      3      2     -1
```

- transpose `y = x'`

```
y =
      1
      2
      5
      1
```

Matlab

- `t =1:10`

```
t =
      1    2    3    4   5   6    7   8    9    10
```

- `k =2:-0.5:-1`

```
k =
      2   1.5   1   0.5   0   -0.5   -1
```

- `B = [1:4; 5:8]`

```
x =
      1      2      3     4
      5      6      7     8
```

Matlab

- zeros(M,N) MxN matrix of zeros

```
x = zeros(1,3)
x =
   0     0     0
```

- ones(M,N) MxN matrix of ones

```
x = ones(1,3)
x =
   1     1     1
```

- rand(M,N) MxN matrix of uniformly distributed random numbers on (0,1)

```
x = rand(1,3)
x =
 0.9501  0.2311 0.6068
```

Matlab

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

Given:

```
A =

     3     5     3
     6     8     2
     2     7     3
```

```
>> A(6)

ans =

     7
```

```
>> A(3,2)

ans =

     7
```

```
>> A(2,:)

ans =

     6     8     2
```

```
>> A(1:2,2)

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)
Error: ??? Index exceeds matrix dimensions.

Matlab

- ```
  x = [1 2], y = [4 5], z=[ 0 0]

  A = [ x y]

      1   2   4   5

   B = [x ; y]

       1 2
       4 5
  ```

```
C = [x y ;z]
Error:
??? Error using ==> vertcat CAT arguments dimensions are not consistent.
```
```

Matlab

Operations:

+ addition

– Subtraction

* multiplication

/ division

^ power

‘ transpose

Matlab

Given A and B:

```
>> A = [1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```

```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition

```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction

```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product

```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose

```
>> T = A'

T =

     1     4     7
     2     5     8
     3     6     9
```

Section 6

Python

Python

It's free (open source)

- Downloading and installing Python is free and easy
- Source code is easily accessible
- Free doesn't mean unsupported! Online Python community is huge

It's portable

- Python runs virtually every major platform used today
- As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform It's powerful

Dynamic typing

- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, SciPy)
- Automatic memory management

Python - Operations on Numbers

Basic algebraic operations

- Four arithmetic operations: a+b, a-b, a*b, a/b
- Exponentiation: a**b
- Other elementary functions are not part of standard Python, but included in packages like NumPy and SciPy

Comparison operators

- Greater than, less than, etc.: a $<$ b, a $>$ b, a $<=$ b, a $>=$ b
- Identity tests: a $==$ b, a $!=$ b

Python - String

Strings are ordered blocks of text

- Strings are enclosed in single or double quotation marks
- Double quotation marks allow the user to extend strings over multiple lines without backslashes, which usually signal the continuation of an expression
- Examples: ‘abc’, “ABC”

Concatenation and repetition

- Strings are concatenated with the + sign:
- Enter: ‘abc’+‘def’
- You get: ‘abcdef’
- Strings are repeated with the * sign:
- Enter ‘abc’*3
- You get: ‘abcabcabc’

Python - Indexing and Slicing

Indexing and slicing

- Python starts indexing at 0. A string s will have indexes running from 0 to len(s)-1 (where len(s) is the length of s) in integer quantities.
- s[i] fetches the i+1 th element in s
- Assume s = ‘string’
- Enter: s[1]
- You get: ‘t’
- s[i:j] fetches elements i (inclusive) through j (not inclusive)

Python - Indexing and Slicing

- Enter: s[1:4]
- You get: ‘tri’
- s[:j] fetches all elements up to, but not including j
- Enter: s[:3]
- You get: ‘str’
- s[i:] fetches all elements from i onward (inclusive)
- Enter: s[2:]
- You get: ‘ring’

Python - List

Basic properties:

- Lists are contained in square brackets []
- Lists can contain numbers, strings, nested subsists, or nothing
- Examples: L1 = [0,1,2,3], L2 = [‘zero’, ‘one’], L3 = [0,1,[2,3],‘three’,[‘four,one’]], L4 = []
- List indexing works just like string indexing
- Lists are mutable: individual elements can be reassigned in place.
- Example: L1 = [0,1,2,3], L1[0] = 4
- Enter: L1
- You get: [4,1,2,3]