lecture_04_R_example

Jonathan Holt

1/22/2019

Introduction to R Markdown

This is an R Markdown Notebook. R Markdown Notebooks are similar to Jupyter notebooks, and they are very popular among data scientists using the R programming language. R Markdown Notebooks can be created using RStudio, a free software.

RStudio can be accessed from Anaconda Navigator, or downloaded independently.

The single greatest thing about R (in my opinion) is RStudio. RStudio is a well-maintained IDE (integrated development environment) that makes coding easy for non-programmers.

Notice that we do not need to use chuncks for text in R Markdown. We can simply type regular text, *italicized text*, **bold text**, and links directly into the notebook.

Mathematical expressions in R Markdown are the same as they are in Jupyter notbooks. For example,

$$y = bx + \epsilon$$

When we're ready to add some code, we need to insert a chunk. R Markdown supports code chunks in R, Python, and several other languages. First, let's insert a chunk of Python code.

```python
# I love Python
import numpy as np
np.random.seed(42)
print(np.random.rand())
```

```
## 0.374540118847
```

Now, let's try a chunk of R code.

```r
# I love R
set.seed(42)
print(runif(1,0,1)) # choose 1 random number between 0 and 1.
```

```
## [1] 0.914806
```

Did you notice how R required 1 less line of code compared to Python? One advantage of R is that many foundational packages are *pre-loaded*, so less code is required for simple things. This is also the reason that Python is more flexible.

R or Python?

Both R and Python are open-source languages developed in the 1990's. For a long time, R was preferred by data scientists because it had better analysis toolboxes and it generally looked better. R is is still very popular in statistical and academic settings. All of the statistics courses that I have taken at Duke have been taught in R.

Over the years, however, Python has matched R in terms of data science capability. Furthermore, Python is more robust and scalable. Many apps and websites can be coded in Python (but not R), so Python is more transferable. The Deep learning community prefers Python to R.

Python and R have similar syntax, and nearly identical functionality with respect to data science. They both have lots of community support, which means many, many packages for both.

Here is a link that explains some of the key differences between the two languages.

Reasons to do your homework in Python:

- if you have no previous coding experience
- if you think you will pursue deep learning or app development
- if you already know R, and want to learn a second language

Reasons to do your homework in R:

- if you already use R, and don't think you will pursue deep learning or app development
- if you already know Python, and want to learn a second language

Remember, you can always code R in Jupyter Notebooks, just like you can code Python in R Markdown. # Code Examples in R

Let's follow along with the test score example from the Jupyter notebook.

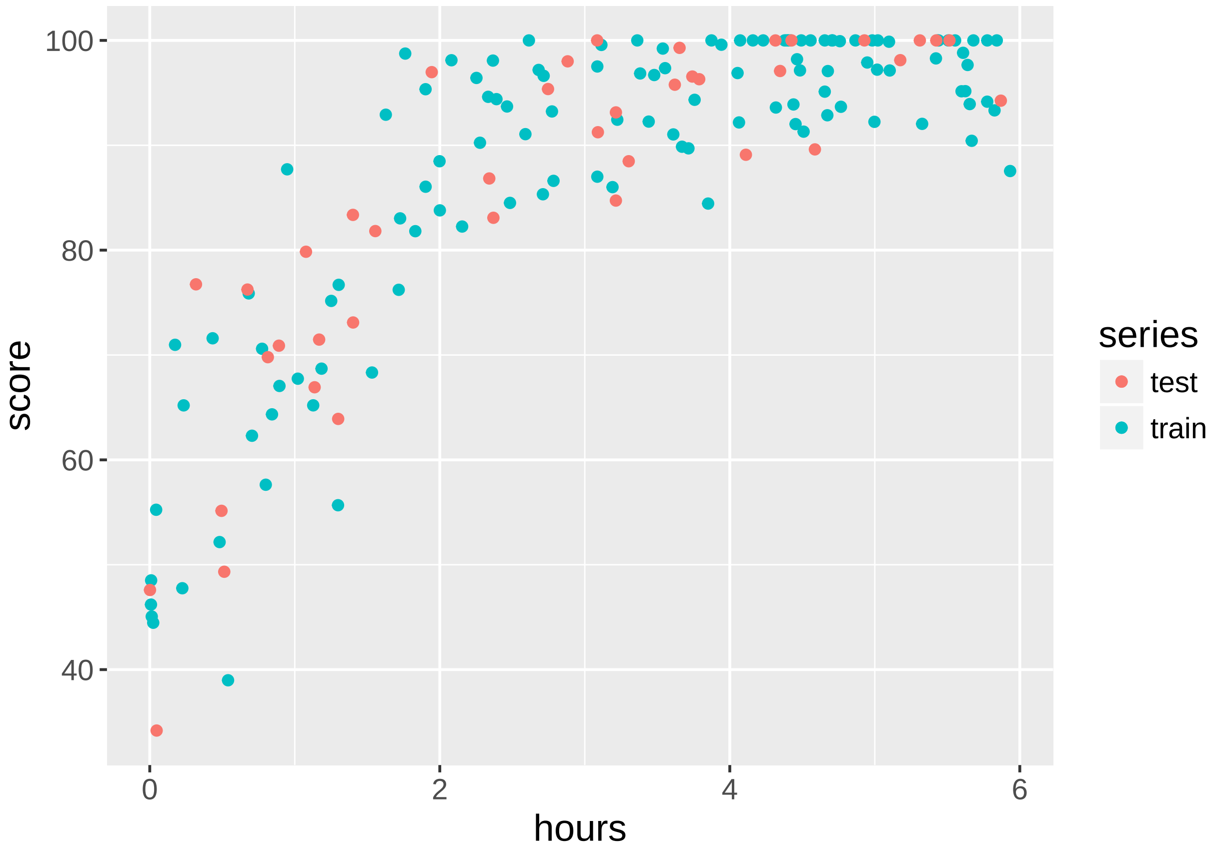
```
# sigmoid function
sigmoid <- function(x) {
  return(1/(1+exp(-x)))
}
# generate the student data
generate_students <- function(n=100) {
  x = 6*runif(n,0,1)
  x = sort(x)
  y = 100*sigmoid(x+0.4*rnorm(n)) + 5*rnorm(n)
  y[y>100] = 100
  return(data.frame(hours = x, score = y)) # put the data in a "data frame"
}
```

```
df <- generate_students(n=200) # this data frame contains both the hours and the score
```

```
n_train <- 120
n_valid <- 40
n_test <- 40

# first, split into train and test
train_index <- sample(seq_len(nrow(df)), size = n_train)
train <- df[train_index, ]
test <- df[-train_index, ] # "-" means "everything except"

# then, split the test set into test and validation
test_index <- sample(seq_len(nrow(test)), size = n_test)
valid <- test[-test_index, ]
test <- test[test_index, ]
```

Next, plot the data. R has plenty of built-in plotting functionality. However, ggplot2 is a very popular plotting library that makes things look much better than default.

```r
library(ggplot2)

ggplot(data = train, aes(x=hours, y=score)) +
  geom_point(aes(color = "train")) +
  geom_point(data = test, aes(color = "test")) +
  labs(color = "series") +
  theme(text = element_text(size = 14))
```



The primary Python library used in this class is scikit-learn. The R analogue is caret.

```r
# load the package
#install.packages("caret")
library(caret)
```

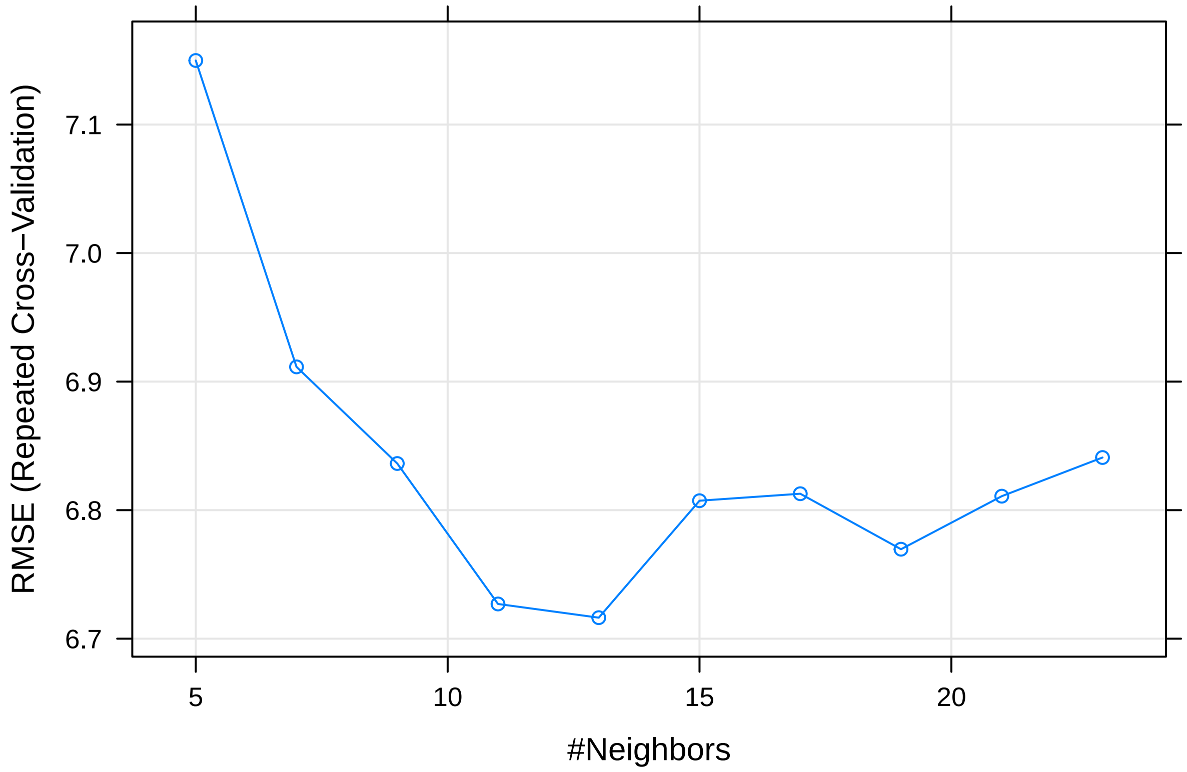
```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2018i.
## 1.0/zoneinfo/America/New_York'
```

```r
# set parameters for training
train_control <- trainControl(method = "repeatedcv", number = 15, repeats = 3)

# train the model
set.seed(42)
knn_fit <- train(score ~ hours, data = df, method = "knn",
 trControl=train_control,
 preProcess = c("center", "scale"),
 tuneLength = 10)

knn_fit
```

```
## k-Nearest Neighbors
##
## 200 samples
##   1 predictor
##
## Pre-processing: centered (1), scaled (1)
## Resampling: Cross-Validated (15 fold, repeated 3 times)
## Summary of sample sizes: 186, 188, 186, 188, 185, 187, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  7.149844  0.7991526  5.429807
##    7  6.911482  0.8120816  5.302158
##    9  6.836275  0.8181916  5.215313
##   11  6.726996  0.8251213  5.170951
##   13  6.716339  0.8255806  5.147268
##   15  6.807352  0.8216649  5.184001
##   17  6.812785  0.8224183  5.248376
##   19  6.769591  0.8256487  5.218818
##   21  6.810865  0.8225034  5.267032
##   23  6.840946  0.8239006  5.299582
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

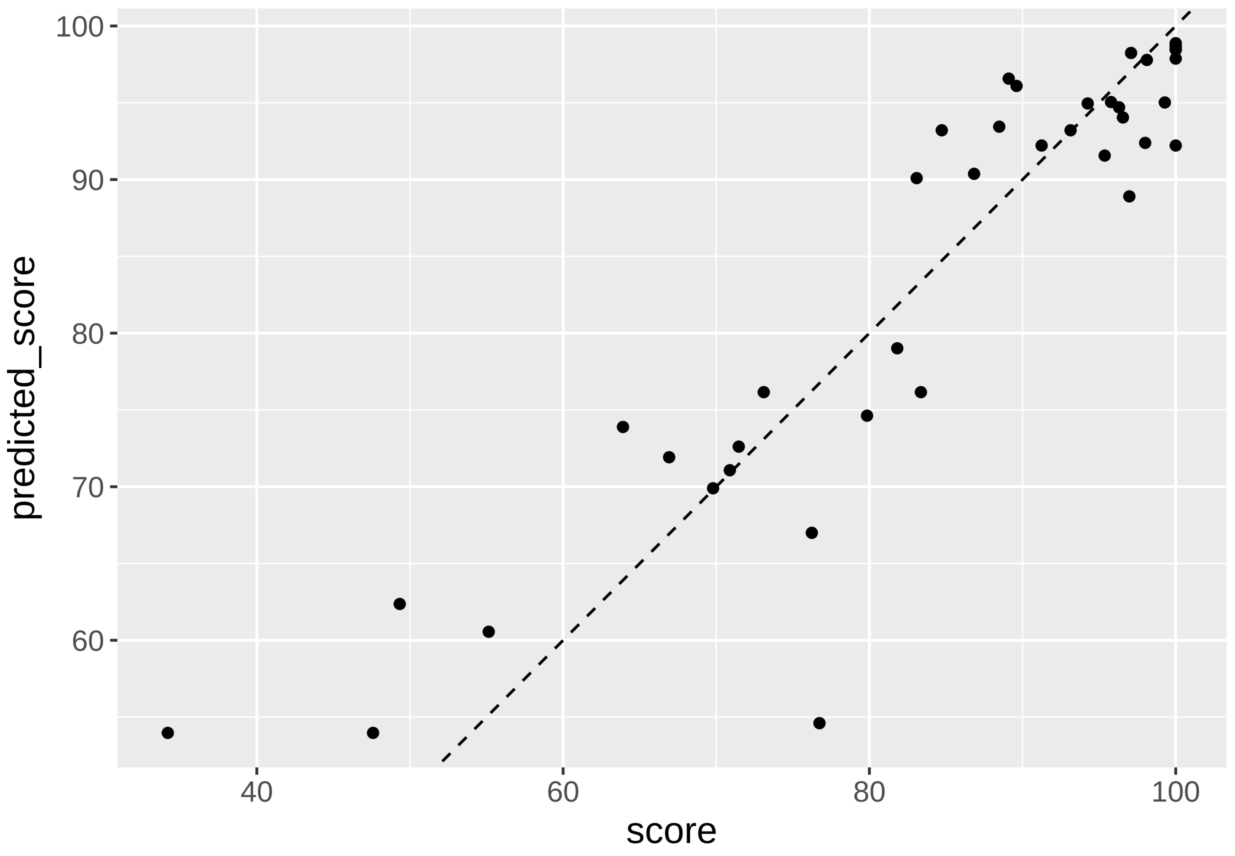
Caret has default plotting functionality.

```r
plot(knn_fit)
```



```r
test$predicted_score <- predict(knn_fit, newdata = test)
```

```r
ggplot(data = test, aes(x=score, y=predicted_score)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  theme(text = element_text(size = 14))
```



Recall that these are synthetic data, so don't worry if the results are different from the ones from the Jupyter notebook!

Exporting the notebook

Exporting the notebook as a PDF is a great way to share your work. The figures and text will be rendered in a clean, professional format.

Alternatively, the .rmd file can be shared directly. This allows the reader to run and/or edit your code live.