STATS 769 - Lab 03 - bole001

Bernard O'Leary

18 August 2019

Read in data and Linux commands

Data format is CSV (Comma Separated Values), reading into R using the read.csv function. Becuase there are a large number of files an alternative approach has been taken, to list all files in the Lab02 directory accroding to the file structure we are looking for ("trips*.csv") and then pick up each file and add to a dataframe (called"trips.df").

The machine that was used to create the report is a Microsoft Windows 10 machine that has an Ubuntu Linux Bash shell built into it as part of the WSL (Windows Subsystem for Linux) feature that is an optional feature of the Windows 10 OS (https://docs.microsoft.com/en-us/windows/wsl/install-win10). Once access was gained to the STATS 769 Linux machines, the files were copied to the Windows 10 machine used to create the report using the following scp command:

```
bernardo@PKS10198:/mnt/d/Study/UoA-STATS-769$ scp bole001@sc-cer00014-04.its.auckland.ac.nz:/course/Labs
```

Whereas on the Windows 10 machine used to create the report, it was set to the local filesystem where the csv files for the lab were copied to. Code used to create the initial dataframe for the scooter Trips data follows.

```r
# Files are already in our local directory
directory = '.'
scooter_files.ls <- intersect(list.files(path=directory, pattern="scooter"), list.files(path=directory,
# First apply read.csv, then rbind
scooter_trips.df = do.call(rbind, lapply(scooter_files.ls, function(x) read.csv(x, stringsAsFactors = FA
```

Get rid of non-positive and non-finite values in duration and distance

Remove from the entire dataset.

```r
scooter_trips.df <- subset(scooter_trips.df, scooter_trips.df$Trip.Duration > 0)
scooter_trips.df <- subset(scooter_trips.df, scooter_trips.df$Trip.Distance > 0)
scooter_trips.df <- subset(scooter_trips.df, is.finite(scooter_trips.df$Trip.Duration))
scooter_trips.df <- subset(scooter_trips.df, is.finite(scooter_trips.df$Trip.Distance))
```

Add logged duration and distance variables

Name the variables Trip.Duration.Logged and Trip.Distance.Logged and add to the end of the dataframe. Note that because the data are not cleansed by this point NaNs are produced here.

```r
scooter_trips.df$Trip.Duration.Logged <- log(scooter_trips.df$Trip.Duration)
scooter_trips.df$Trip.Distance.Logged <- log(scooter_trips.df$Trip.Distance)
```

Perform 10-fold cross validation

```r
# Define RMSE function
RMSE <- function(m, o) {
    sqrt(mean((m - o)^2))
}

# Randomly shuffle the data
scooter_trips.df <- scooter_trips.df[sample(nrow(scooter_trips.df)),]

# Create 10 equally size folds
folds <- cut(seq(1,nrow(scooter_trips.df)),breaks=10,labels=FALSE)

# Perform 10 fold cross validation
for(i in 1:10){

  # Segement your data by fold using the which() function
  test_indexes <- which(folds==i,arr.ind=TRUE)
  test_set <- scooter_trips.df[test_indexes, ]
  training_set <- scooter_trips.df[-test_indexes, ]

  # Make the model
  y_train = training_set$Trip.Duration
  x_train = training_set$Trip.Distance
  fit_mean = mean(y_train)
  fit_lm <- lm(y ~ x, data.frame(y=y_train, x=x_train))

  # Test results
  y_test <- test_set$Trip.Duration
  x_test <- test_set$Trip.Distance

  # Make a vector that is populated with the fit_mean
  pred_mean <- rep(fit_mean, length(y_test))

  # Get a vector of predictions based on test data
  pred_lm <- predict(fit_lm, data.frame(x=x_test))

  # get the RMSE
  print(i)
  print(RMSE(pred_mean, y_test))
  print(RMSE(pred_lm, y_test))
}
```

```
## [1] 1
## [1] 1044.484
## [1] 1044.157
## [1] 2
## [1] 912.4832
## [1] 912.0982
## [1] 3
## [1] 1148.879
## [1] 1148.571
## [1] 4
## [1] 1124.961
## [1] 1124.626
## [1] 5
## [1] 1399.572
## [1] 1399.306
## [1] 6
## [1] 972.5498
## [1] 972.1615
## [1] 7
## [1] 1403.794
## [1] 1403.481
## [1] 8
## [1] 1052.661
## [1] 6821.529
## [1] 9
## [1] 945.9922
## [1] 945.599
## [1] 10
## [1] 917.1392
## [1] 916.758
```