STATS 769 - Lab 05 - bole001

Bernard O'Leary

02 September 2019

Data format and API call details

Data is in JSON format as it comes down from the API. It is a JSON array of "trip" objects. The API allows query-sting based parameters to be passed as part of an HTTP GET request, which means that we don't need to send an HTTP body. The paramters that we use are *limit* to tell the API to give us back a specific number of JSON rows. Without this parameter we get back 1000 by default. We also send a *year* and a *vehicle_type* paramter to further limit our result set. Finally an *api_token* parameter is sent to authenticate our request. I am using a combination of the the *httr* and *jsonlite* libraries to make a simple GET request and then transform the result into a dataframe.

The following code imports the data from the API and lists dataframe dimensions and first 6 rows of the data for validation purposes.

```
library(httr)
library(jsonlite)
```

```
## Loading required package: methods
```

```
# Call the resource
json_result <- GET("https://data.austintexas.gov/resource/7d8e-dm7r.json?$limit=10000&year=2018&vehicle_

# Show data frame
trips <- fromJSON(content(json_result, as="text"))
head(trips)
```

```
##                                trip_id
## 1 c998b4ad-a6b0-4bda-bcf1-439f8dec79e0
## 2 51d3b1d9-90f0-4c10-a4f7-83123a697fe4
## 3 1b666bdb-8031-45db-9094-a0b4dcb19395
## 4 a858eeaf-062e-40e2-b275-cd31df2575a7
## 5 fb51603e-4695-4d1d-8646-2607b27444d4
## 6 0edca4b5-2aa2-4e83-a744-2017b76faa15
##                              device_id vehicle_type trip_duration
## 1 b14555c3-cb30-495b-92e1-18800f64d0df      scooter           486
## 2 830a59d0-78aa-4297-80ed-c40773c0a87d      scooter           197
## 3 49d245ec-f8f9-4529-a7df-276ff1d147d4      scooter             6
## 4 830a59d0-78aa-4297-80ed-c40773c0a87d      scooter           108
## 5 87804ff3-8249-4d63-a298-38fa0ceae797      scooter           219
## 6 6191306a-b12f-47b4-b270-b81a16b91e40      scooter          1116
##   trip_distance             start_time               end_time
## 1             0 2018-11-18T00:45:00.000 2018-11-18T01:00:00.000
## 2             0 2018-09-30T00:00:00.000 2018-09-30T00:00:00.000
## 3             0 2018-12-02T00:00:00.000 2018-12-02T00:00:00.000
## 4             0 2018-09-30T00:15:00.000 2018-09-30T00:15:00.000
## 5             0 2018-11-04T00:30:00.000 2018-11-04T00:30:00.000
```

```
## 6             0 2018-10-07T00:30:00.000 2018-10-07T01:00:00.000
##             modified_date month hour day_of_week council_district_start
## 1 2019-04-17T01:53:51.000    11    0           0                      3
## 2 2019-04-17T04:24:37.000     9    0           0                      9
## 3 2019-04-17T04:34:17.000    12    0           0                      9
## 4 2019-04-17T02:15:42.000     9    0           0                      9
## 5 2019-04-17T06:23:21.000    11    0           0                      3
## 6 2019-04-17T05:06:39.000    10    0           0                      9
##   council_district_end year census_geoid_start census_geoid_end
## 1                    9 2018        48453002304      48453001100
## 2                    9 2018        48453000604      48453000601
## 3                    9 2018        48453000601      48453000601
## 4                    9 2018        48453000603      48453000603
## 5                    3 2018        48453001308      48453001308
## 6                    9 2018        48453001100      48453000700
```

```
dim(trips)
```

```
## [1] 10000    16
```

Cleanse dataset

Subset only trips with non-negative distances and durations, create a new *long_trip* variable (where "long" means that the trip distance was greater than 1000m). We exclude non-positive durations and distances and make another new variable that is a log of the duration variable called *logged_trip_duration*. Show the first six records of the dataframe so that we can see these new variables and validate.

```
model_trips <- subset(trips, as.integer(trip_duration) > 0 & as.integer(trip_distance) > 0)
model_trips$logged_trip_duration <- log(as.integer(model_trips$trip_duration))
model_trips$long_trip <- as.integer(model_trips$trip_distance) > 1000
head(model_trips)
```

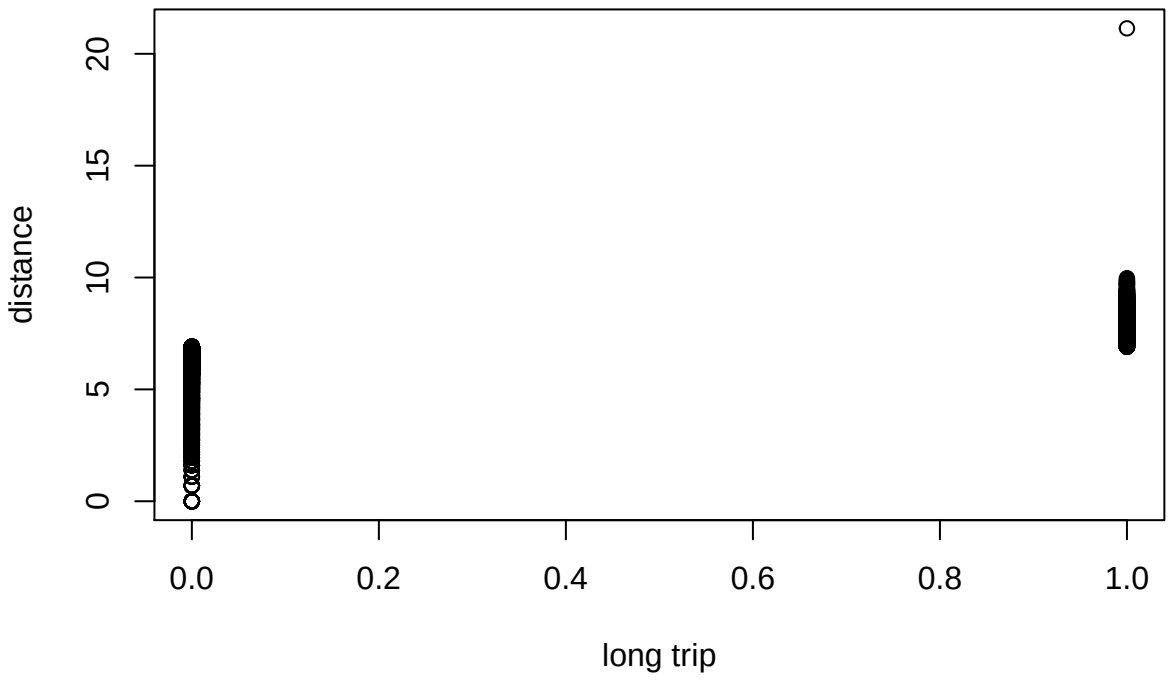
```
##                                trip_id
## 22 8df8fc9d-1272-4cc4-8646-789263296977
## 23 ee90620e-e662-4a45-b34d-97c147b2be45
## 24 e3332e1c-432b-4204-a9d6-04637e8dab7e
## 25 e63ac1b4-e46f-40de-899b-afcd9e6a7f75
## 26 ee0c177d-4d6b-4ef3-9d49-f65565989c45
## 27 97f38c39-18ed-4620-9c4b-c6e55f6b7bd4
##                                device_id vehicle_type trip_duration
## 22 8aaf96e5-6e22-4e4c-a2f8-6db00c900aca      scooter           221
## 23 c537d2ef-f112-44fa-a0ea-55a4152e53f3      scooter          1107
## 24 771b40ad-8d45-47ff-b94d-58d4f5bd2371      scooter           451
## 25 16740fa3-07fa-482b-940f-1902b3c4be36      scooter           437
## 26 e3b7a489-92b6-43a6-8ceb-c837a38621d5      scooter           901
## 27 430bd20e-666e-41d6-81fb-cdec7db29cc4      scooter           150
##    trip_distance              start_time                end_time
## 22           960 2018-06-21T17:45:00.000 2018-06-21T17:45:00.000
## 23          4555 2018-08-07T18:00:00.000 2018-08-07T18:15:00.000
## 24          1155 2018-06-21T17:45:00.000 2018-06-21T18:00:00.000
## 25          1322 2018-06-21T17:45:00.000 2018-06-21T18:00:00.000
## 26          1867 2018-06-21T17:45:00.000 2018-06-21T18:00:00.000
```

```
## 27           161 2018-06-21T17:45:00.000 2018-06-21T17:45:00.000
##              modified_date month hour day_of_week council_district_start
## 22 2019-04-17T01:39:54.000     6   17           4                      1
## 23 2019-04-17T01:43:38.000     8   18           2                      9
## 24 2019-04-17T01:39:54.000     6   17           4                      3
## 25 2019-04-17T01:39:54.000     6   17           4                      9
## 26 2019-04-17T01:39:54.000     6   17           4                      9
## 27 2019-04-17T01:39:54.000     6   17           4                      9
##    council_district_end year census_geoid_start census_geoid_end
## 22                    9 2018        48453000401      48453000601
## 23                    9 2018        48453001100      48453001305
## 24                    1 2018        48453000902      48453000901
## 25                    9 2018        48453001100      48453001100
## 26                    9 2018        48453001100      48453000604
## 27                    9 2018        48453000401      48453000401
##    logged_trip_duration long_trip
## 22             5.398163     FALSE
## 23             7.009409      TRUE
## 24             6.111467      TRUE
## 25             6.079933      TRUE
## 26             6.803505      TRUE
## 27             5.010635     FALSE
```

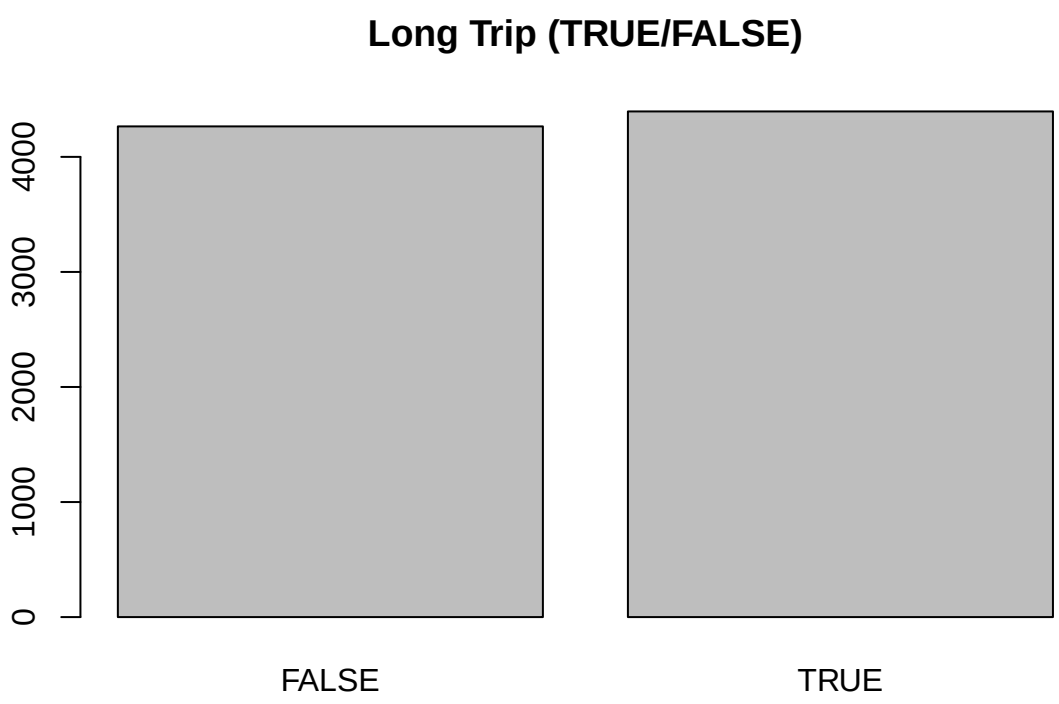
Boxplot the data (to see what we have)

Take a look at the data that we have come back from the API call. We have at least one significant outlier. Other than that looks like we have perhaps a

```
plot(log(as.integer(model_trips$trip_distance)) ~ model_trips$long_trip, ylab="distance", xlab="long tri
```




```
barplot(table(model_trips$long_trip), main="Long Trip (TRUE/FALSE)")
```



Comparison of logistic regression and k-nearest neighbours models

We fit a logistic model that predicts the proportion of long trips as a function of trip duration and output the confusion matrix. We do that same for KNN and then try to plot a chat of the result. Unfortunately I have not been able to produce a chart successfully.

```
test_index <- sample(nrow(model_trips), nrow(model_trips) * 0.1)
test_trips <- model_trips[test_index, ]
train_trips <- model_trips[-test_index, ]

overall_proportion <- mean(model_trips$long_trip)
glm_fit <- glm(y ~ x, data.frame(x=train_trips$logged_trip_duration, y=train_trips$long_trip), family="

library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

## Warning: replacing previous import by 'rlang:::=' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::.data' when loading 'dplyr'
```

```
## Warning: replacing previous import by 'rlang::as_label' when loading
## 'dplyr'

## Warning: replacing previous import by 'rlang::as_name' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::dots_n' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::enquo' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::enquos' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::expr' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::sym' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::syms' when loading 'dplyr'

## Warning: replacing previous import by 'rlang::!!' when loading 'recipes'

## Warning: replacing previous import by 'rlang::as_character' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::call2' when loading 'recipes'

## Warning: replacing previous import by 'rlang::exec' when loading 'recipes'

## Warning: replacing previous import by 'rlang::expr' when loading 'recipes'

## Warning: replacing previous import by 'rlang::f_lhs' when loading 'recipes'

## Warning: replacing previous import by 'rlang::f_rhs' when loading 'recipes'

## Warning: replacing previous import by 'rlang::is_empty' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::is_quosure' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::na_dbl' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::names2' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::quo' when loading 'recipes'

## Warning: replacing previous import by 'rlang::quo_get_expr' when loading
## 'recipes'
```

```
## Warning: replacing previous import by 'rlang::quo_squash' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::quo_text' when loading
## 'recipes'

## Warning: replacing previous import by 'rlang::quos' when loading 'recipes'

## Warning: replacing previous import by 'rlang::sym' when loading 'recipes'

## Warning: replacing previous import by 'rlang::syms' when loading 'recipes'

## Warning: replacing previous import by 'tibble::tibble' when loading
## 'recipes'

## Warning: replacing previous import by 'plyr::ddply' when loading 'caret'

## Warning: replacing previous import by 'recipes::all_outcomes' when loading
## 'caret'

## Warning: replacing previous import by 'recipes::all_predictors' when
## loading 'caret'

## Warning: replacing previous import by 'recipes::bake' when loading 'caret'

## Warning: replacing previous import by 'recipes::has_role' when loading
## 'caret'

## Warning: replacing previous import by 'recipes::juice' when loading 'caret'

## Warning: replacing previous import by 'recipes::prep' when loading 'caret'

##
## Attaching package: 'caret'

## The following object is masked from 'package:httr':
##
##     progress
```

```
glm_prob <- predict(glm_fit, data.frame(x=as.integer(test_trips$trip_distance)), type="response")
glm_pred <- ifelse(glm_prob > .5, 1, 0)
glm_diag <- confusionMatrix(factor(glm_pred, levels = 1:0),
                            factor(as.integer(test_trips$long_trip), levels = 1:0))
glm_diag$table
```

```
##           Reference
## Prediction   1   0
##          1 416 442
##          0   0   8
```

```
glm_diag$overall["Accuracy"]
```

```
##  Accuracy
## 0.4896074
```

```
train_x <- as.data.frame(train_trips$logged_trip_duration)
test_x <- as.data.frame(test_trips$logged_trip_duration)
train_y <- as.data.frame(as.integer(train_trips$long_trip))
cl = train_y[,1, drop = TRUE]

library(class)
knn_pred <- knn(train_x, test_x, cl, k=31, prob=TRUE)
knn_prob <- ifelse(knn_pred == 1, attr(knn_pred, "prob"), 1 - attr(knn_pred, "prob"))
knn_diag <- confusionMatrix(factor(knn_pred, levels = 1:0),
                            factor(as.integer(test_trips$long_trip), levels = 1:0))
knn_diag$table
```

```
##           Reference
## Prediction   1   0
##          1 369 109
##          0  47 341
```

```
knn_diag$overall["Accuracy"]
```

```
##  Accuracy
## 0.8198614
```

```
par(mar=c(3, 3, 2, 2))
breaks <- seq(0, 1, .1)
midbreaks <- breaks[-1] - diff(breaks)/2
class(test_x)
```

```
## [1] "data.frame"
```

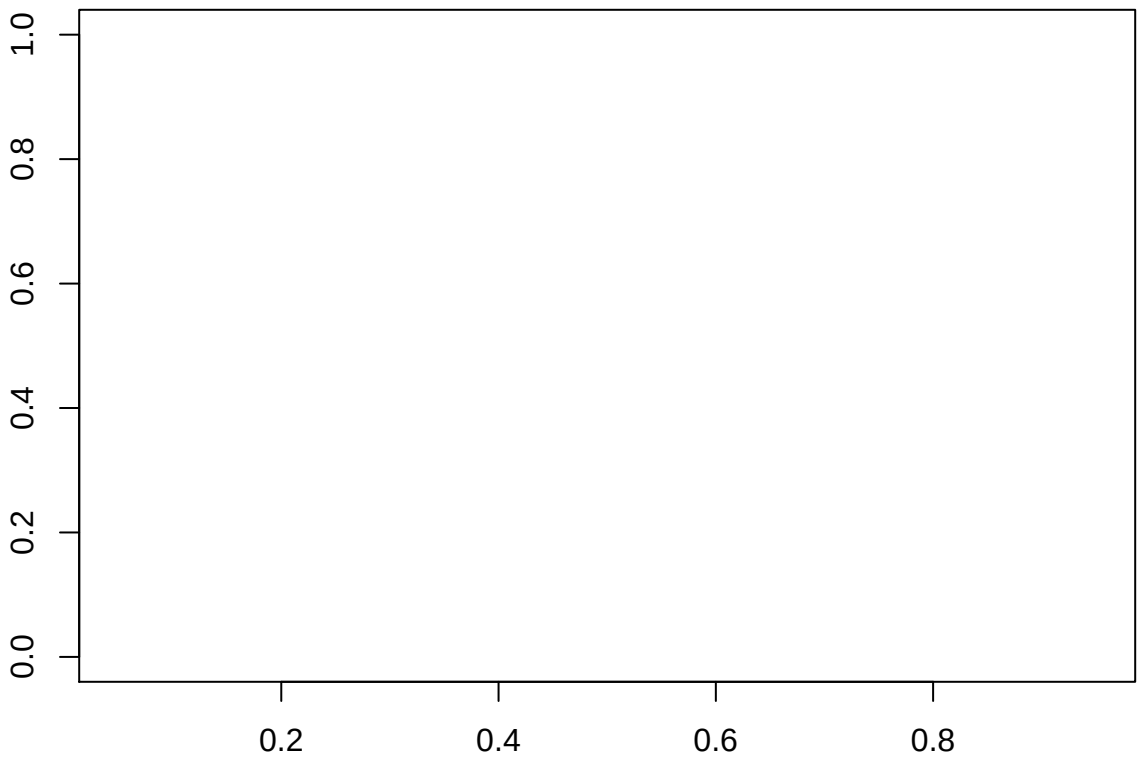
```
props <- tapply(as.integer(test_trips$long_trip), cut(test_trips$logged_trip_duration, breaks), mean)
props
```

```
##   (0,0.1] (0.1,0.2] (0.2,0.3] (0.3,0.4] (0.4,0.5] (0.5,0.6] (0.6,0.7]
##        NA        NA        NA        NA        NA        NA        NA
## (0.7,0.8] (0.8,0.9]   (0.9,1]
##        NA        NA        NA
```

```
midbreaks
```

```
##  [1] 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95
```

```
plot(midbreaks, props, pch=16, ylim = c(0,1))
lines(test_trips$logged_trip_duration, glm_prob, col="red")
lines(test_trips$logged_trip_duration, knn_prob, col="green")
```



Summary of analysis

The result of the KNN model is better than for the logistic regression, at least for the Accuracy metric. For logistic regression we have Accuracy of about 51% and for KNN we have accuracy of over 80%. For this dataset and scenario at least KNN is a better model than loogistic regression in terms of Accuracy.