Google Maps Route Planning Exploration

Conor Falvey, Troy Posadas

12/1/2019

We sought out to create a model that can achieve close times to the Google Maps API's best guess for its route timing. However, as we have no physical metric to compare what is "better" than Google Maps, we analyzed the performance of a polynomial spline regression against a linear regression. We analyzed the effects of departure time, departure day, and Google Maps' optimistic and pessimistic guesses of route times on the over all trip time on two seperate routes in hypothesising that this type of model will more closely represent the data. We wrote a custom API caller that queried the Google Maps trip timing API in 30 second intervals for approximately a week. Both of us have used API scrapers before in previous work, and attempting to collect usable data and perform an analysis of it brought our communal interests of data science and prediction together. We explain later the methodology behind choosing polynomial splines over trying to use our own linear model for this dataset.

API Caller

Here we have the python script used to query the API every 30 seconds for all of our routes and Google's optimistic, pessimistic, and best guess timings which we then piped into a local Postgres database and dumped into a CSV file for analysis.

```
import requests
from apscheduler.schedulers.background import BackgroundScheduler
import time
import configs
import datetime
import pg8000

models = ["optimistic", "pessimistic", "best_guess"]
routes = {"41.8839,-87.6319": "34.0537,-118.2427",
          "24.5551,-81.7800": "47.6062,-122.3321"}
api_key = configs.API_KEY
seconds = 30

def db_insert(curr_time, route, model, distance, duration_text, duration,
              duration_traffic_text, duration_traffic):
    cursor.execute("INSERT INTO maps (time, route, model, distance, durationText, "
                   "duration, durationTrafficText, "
                   "durationTraffic) VALUES (%s, %s, %s, %s, %s, %s, %s, %s);",
                   (curr_time, route, model, distance, duration_text, duration,
                    duration_traffic_text, duration_traffic))
    print("Values added to database at {}".format(curr_time))

def call(dest, origin):
    if dest == list(routes.keys())[0]:
        curr_route = 0
    else:
        curr_route = 1
    curr = int(time.time()) + 10
    for model in models:
```

```
        response = requests
        .get("https://maps.googleapis.com/maps/api/distancematrix/json?destinations={}"
                                "&origins={}&departure_time={}&traffic_model={}&key={}"
                                .format(dest, origin, curr, model, api_key)).json()
        if response is not None:
            db_insert(curr, curr_route, model, response.get("rows")[0]
                      .get("elements")[0].get("distance").get("value"),
                      response.get("rows")[0].get("elements")[0]
                              .get("duration").get("text"),
                      response.get("rows")[0].get("elements")[0]
                              .get("duration").get("value"),
                      response.get("rows")[0].get("elements")[0]
                              .get("duration_in_traffic").get("text"),
                      response.get("rows")[0].get("elements")[0]
                              .get("duration_in_traffic").get("value"))

def caller():
    for key, value in routes.items():
        call(key, value)

def clock():
    scheduler = BackgroundScheduler()
    scheduler.add_job(caller, 'interval', seconds=seconds)
    scheduler.start()

    try:
        while True:
            time.sleep(2)
    except (KeyboardInterrupt, SystemExit):
        scheduler.shutdown()

print("Starting Google Maps API Caller...")
print("Running {} routes every {} seconds".format(len(routes), seconds))
print("Opening connection to datbase...")
conn = pg8000.connect(user=configs.DB_USER, password=configs.DB_PASSWORD)
cursor = conn.cursor()
print("Successfully connected to database")
start = int(time.time())
clock()
end = int(time.time())
print("Total elapsed time: {}".format(str(datetime.timedelta(seconds=(end - start)))))
conn.commit()
```

After we dumped the database as a CSV, we can see a few metrics. As a caveat, the time is stored in Epoch time, and further conversion must be done. The "Index" number is the Epoch time equivalent for Wednesday, November 6th, 2019 at 9:10:49am, which is when the API recieved it's first data points.

```
index <- 1573103449
trafficData = read.csv("./maps.csv", header = FALSE)
header = c("time", "route", "model", "distance", "durationText", "duration"
           , "durationTrafficText", "durationTraffic")
names(trafficData) = header
trafficData$elapsed =  trafficData$time - index
trafficData$diff = trafficData$duration - trafficData$durationTraffic
```

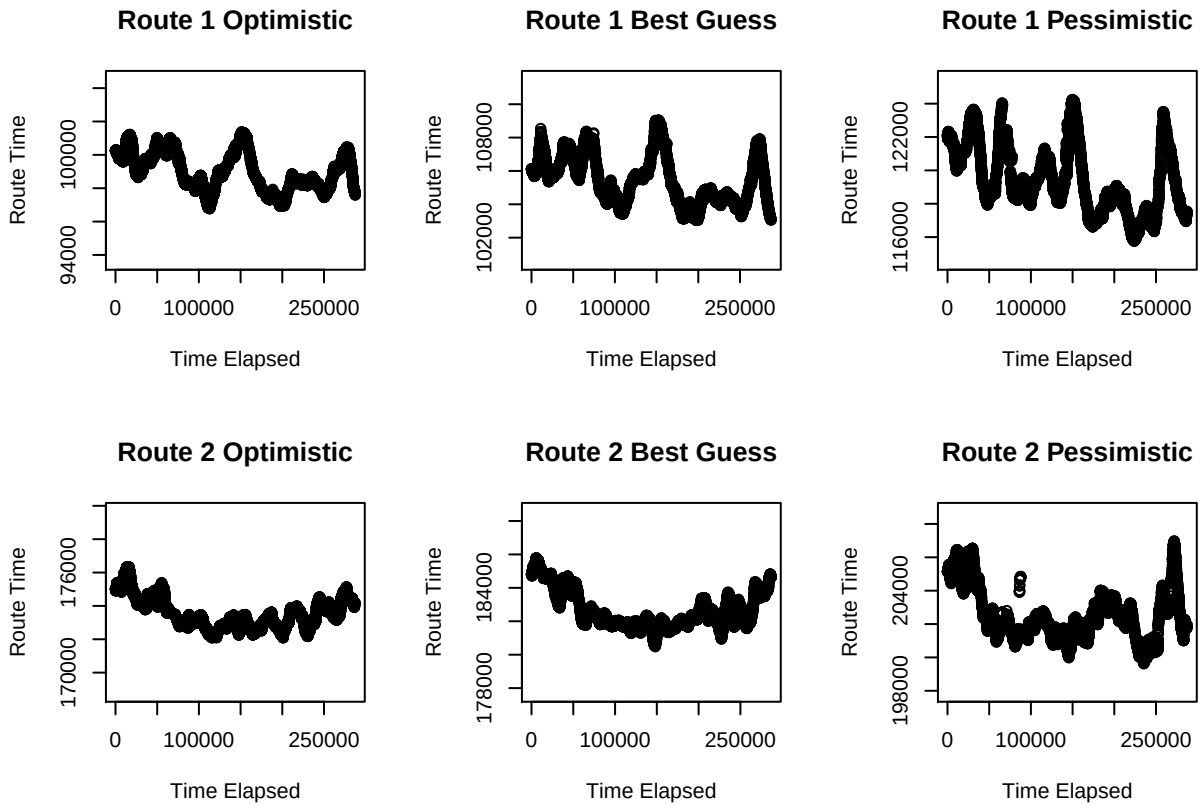
```r
trafficData$Percent = trafficData$durationTraffic / max(trafficData$durationTraffic)

route1 = trafficData[trafficData$route == 0,]
route2 = trafficData[trafficData$route == 1,]

cat(paste("Starting Time: ", as.POSIXct(1573103449, tz = "America/Los_Angeles",
                                        origin="1970-01-01"), sep = ""))
```

```
## Starting Time: 2019-11-06 21:10:49
```

Beginning Visualization

An initial visualization provides a few insights into the data we collected. Here we notice a few big things that can draw away from using linear regression here: It's not linear in the slightest. With this in mind, we could have used data transformation such as Box-Cox or KS tests. Instead, we opted to go further and test our foundations in linear regression by branching out into polynomial splines.

```r
par(mfrow = c(2, 3))
plot(route1[route1$model == "optimistic", ]$elapsed,
     route1[route1$model == "optimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Optimistic", asp = 20)

plot(route1[route1$model == "best_guess", ]$elapsed,
     route1[route1$model == "best_guess", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Best Guess", asp = 20)

plot(route1[route1$model == "pessimistic", ]$elapsed,
     route1[route1$model == "pessimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Pessimistic", asp = 20)

plot(route2[route2$model == "optimistic", ]$elapsed,
     route2[route2$model == "optimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Optimistic", asp = 20)

plot(route2[route2$model == "best_guess", ]$elapsed,
     route2[route2$model == "best_guess", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Best Guess", asp = 20)

plot(route2[route2$model == "pessimistic", ]$elapsed,
     route2[route2$model == "pessimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Pessimistic", asp = 20)
```



Splitting the Data

From here, we split the data into each route and the three sub classes of route information gathered.

```
OptDatar1 = route1[route1$model == "optimistic",]
PessDatar1 = route1[route1$model == "pessimistic",]
BestDatar1 = route1[route1$model == "best_guess",]

OptDatar2 = route2[route2$model == "optimistic",]
PessDatar2 = route2[route2$model == "pessimistic",]
BestDatar2 = route2[route2$model == "best_guess",]
```

Define Metrics

In order to get some validation of the data and to score the models with, we performed a set validation with an 80/20 split between training and testing data respectively. This can be achieved by sampling the initial data and subsetting the the exclusion of the initial data to form our testing set.

```
#df = data.frame(elapsed = 0)
r1OptTrain <- OptDatar1[sample(nrow(OptDatar1), as.integer(nrow(OptDatar1) * 0.8)), ]
r1PessTrain <- PessDatar1[sample(nrow(PessDatar1), as.integer(nrow(PessDatar1) * 0.8)), ]
r1BestTrain <- BestDatar1[sample(nrow(BestDatar1), as.integer(nrow(BestDatar1) * 0.8)), ]
r2OptTrain <- OptDatar2[sample(nrow(OptDatar2), as.integer(nrow(OptDatar2)) * 0.8), ]
r2PessTrain <- PessDatar2[sample(nrow(PessDatar2), as.integer(nrow(PessDatar2) * 0.8)), ]
```

```
r2BestTrain <- BestDatar2[sample(nrow(BestDatar2), as.integer(nrow(BestDatar2) * 0.8)), ]

r1OptTest <- anti_join(OptDatar1, r1OptTrain)
r1PessTest <- anti_join(PessDatar1, r1PessTrain)
r1BestTest <- anti_join(BestDatar1, r1BestTrain)
r2OptTest <- anti_join(OptDatar2, r2OptTrain)
r2PessTest <- anti_join(PessDatar2, r2PessTrain)
r2BestTest <- anti_join(BestDatar2, r2BestTrain)

#Dataset Lengths
cat(paste("Training Size: ", nrow(r1OptTrain)
          , "\nTesting Size: ", nrow(r1OptTest), sep = ""))
```

```
## Training Size: 7491
## Testing Size: 1873
```

Linear Models for Comparison

To have some performance metric to compare our MSEs against, we run linear models of all the routes and store their MSEs. This will serve as a foundation on which to base the success or failure of our polynomial spline models.

```
r1OptLinear <- lm(Percent ~ elapsed, data = r1OptTrain)
r1PessLinear <- lm(Percent ~ elapsed, data = r1PessTrain)
r1BestLinear <- lm(Percent ~ elapsed, data = r1BestTrain)
r2OptLinear <- lm(Percent ~ elapsed, data = r2OptTrain)
r2PessLinear <- lm(Percent ~ elapsed, data = r2PessTrain)
r2BestLinear <- lm(Percent ~ elapsed, data = r2BestTrain)

MSEs <- data.frame(R1Opt = mean(r1OptLinear$residuals^2),
                   R1Pess = mean(r1PessLinear$residuals^2),
                   R1Best = mean(r1BestLinear$residuals^2),
                   R2Opt = mean(r2OptLinear$residuals^2),
                   R2Pess = mean(r2PessLinear$residuals^2),
                   R2Best = mean(r2BestLinear$residuals^2))
```

Run Models

Here, we run the polynomial spline models. One important aspect to note of here is that the knots we selected are not the optimal knots. In looking at the graphs and with some manual tweaking, we were able to get the models to perform at an acceptable level, but with more fine tuning, perhaps by using a penalized spline regression function instead, or grid searching for the optimal knot values. Nonetheless, the chosen knots function as desired.

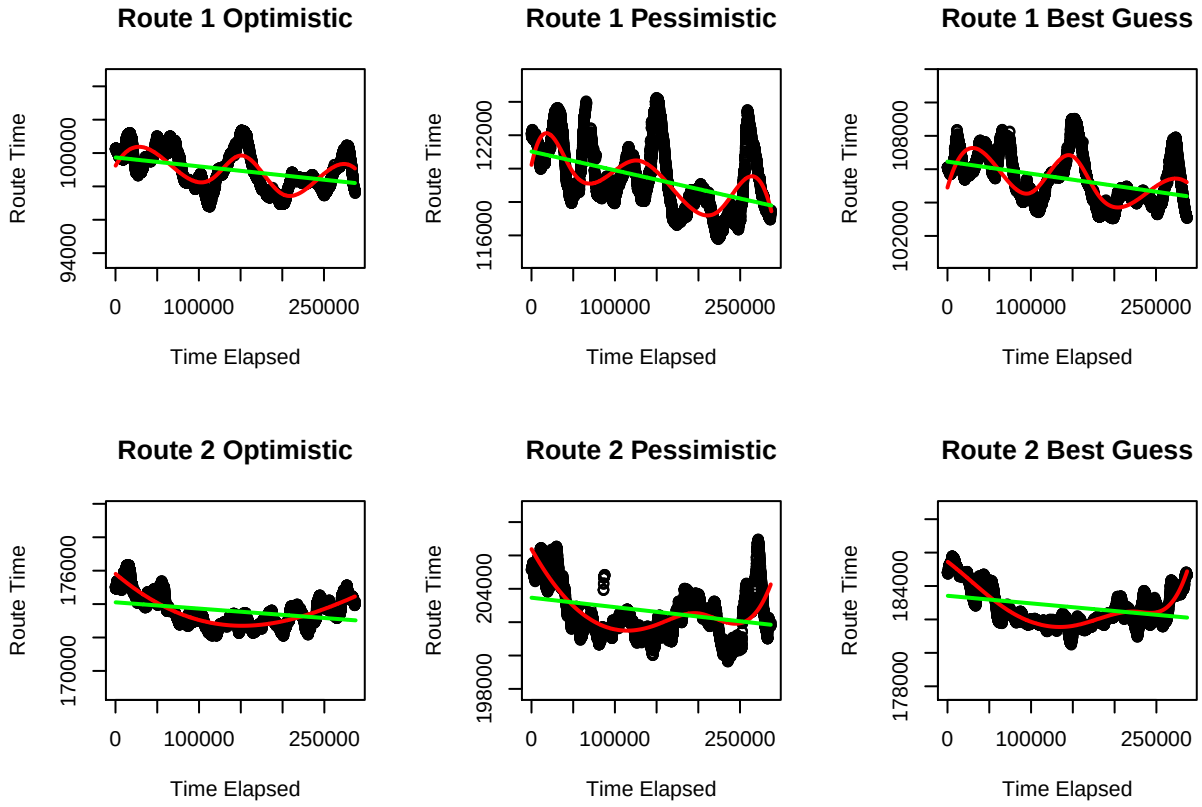
```
spline1 = lm(Percent ~ bs(elapsed, knots = c(115000, 150000, 200000)), data = r1OptTrain)
spline2 = lm(Percent ~ bs(elapsed, knots = c(50000, 125000, 225000)),data = r1PessTrain)
spline3 = lm(Percent ~  bs(elapsed, knots = c(100000, 150000, 180000)),data = r1BestTrain)
spline4 = lm(Percent ~ bs(elapsed, knots = c(0, 150000, 300000)),data = r2OptTrain)
spline5 = lm(Percent ~ bs(elapsed, knots = c(115000, 150000, 200000)),data = r2PessTrain)
spline6 = lm(Percent ~  bs(elapsed, knots = c(72500, 150000, 225000)),data = r2BestTrain)
```

Model Visualization

We can visualize the data by plotting the polynomial splines and linear model against the data for each set. Here, we use the test data as the drawn polynomial model as secondary validation that the models appear visually correct.

```
par(mfrow = c(2, 3))

plot(r1OptTrain[r1OptTrain$model == "optimistic", ]$elapsed,
     r1OptTrain[r1OptTrain$model == "optimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Optimistic", asp = 20)
points(r1OptTest$elapsed,
       predict(spline1, newdata = list(elapsed = r1OptTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r1OptTest$elapsed,
       predict(r1OptLinear, newdata = list(elapsed = r1OptTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")

plot(r1PessTrain[r1PessTrain$model == "pessimistic", ]$elapsed,
     r1PessTrain[r1PessTrain$model == "pessimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Pessimistic", asp = 20)
points(r1PessTest$elapsed,
       predict(spline2, newdata = list(elapsed = r1PessTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r1PessTest$elapsed,
       predict(r1PessLinear, newdata = list(elapsed = r1PessTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")

plot(r1BestTrain[r1BestTrain$model == "best_guess", ]$elapsed,
     r1BestTrain[r1BestTrain$model == "best_guess", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 1 Best Guess", asp = 20)
points(r1BestTest$elapsed,
       predict(spline3, newdata = list(elapsed = r1BestTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r1BestTest$elapsed,
       predict(r1BestLinear, newdata = list(elapsed = r1BestTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")

plot(r2OptTrain[r2OptTrain$model == "optimistic", ]$elapsed,
     r2OptTrain[r2OptTrain$model == "optimistic", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Optimistic", asp = 20)
points(r2OptTest$elapsed,
       predict(spline4, newdata = list(elapsed = r2OptTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r2OptTest$elapsed,
       predict(r2OptLinear, newdata = list(elapsed = r2OptTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")

plot(r2PessTrain[r2PessTrain$model == "pessimistic", ]$elapsed,
     r2PessTrain[r2PessTrain$model == "pessimistic", ]$durationTraffic,
```

```
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Pessimistic", asp = 20)
points(r2PessTest$elapsed,
       predict(spline5, newdata = list(elapsed = r2PessTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r2PessTest$elapsed,
       predict(r2PessLinear, newdata = list(elapsed = r2PessTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")

plot(r2BestTrain[r2BestTrain$model == "best_guess", ]$elapsed,
     r2BestTrain[r2BestTrain$model == "best_guess", ]$durationTraffic,
     xlab = "Time Elapsed", ylab = "Route Time",
     main = "Route 2 Best Guess", asp = 20)
points(r2BestTest$elapsed,
       predict(spline6, newdata = list(elapsed = r2BestTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "red")
points(r2BestTest$elapsed,
       predict(r2BestLinear, newdata = list(elapsed = r2BestTest$elapsed)) *
         max(trafficData$durationTraffic), lwd = 2, type = "l",col = "green")
```



MSE Comparison

We can take the MSEs from the polynomial models and compare them to the MSEs from the linear models to see the improvement over a linear model. Row three in the dataframe below is the percentage improvement attained in performance against a linear model.

```
MSEs <- rbind(MSEs, data.frame(R1Opt = mean(spline1$residuals^2),
                               R1Pess = mean(spline2$residuals^2),
                               R1Best = mean(spline3$residuals^2),
                               R2Opt = mean(spline4$residuals^2),
                               R2Pess = mean(spline5$residuals^2),
                               R2Best = mean(spline6$residuals^2)))

MSEs <- rbind(MSEs, format(MSEs[1, ] / MSEs[2, ] * 100, scientific = FALSE))

MSEs[, 1:3]
```

```
##                    R1Opt               R1Pess               R1Best
## 1   2.58302506182507e-05 7.55605930385304e-05 4.29469114667687e-05
## 2   1.37148715247583e-05  5.6660391509817e-05 2.50351698761199e-05
## 11             188.3375              133.357              171.5463
```

```
MSEs[, 4:6]
```

```
##                    R2Opt               R2Pess               R2Best
## 1   1.84405251961262e-05  5.4325047441068e-05 2.78610315232979e-05
## 2      5.54067228286e-06 3.04688632094934e-05 5.56202903640818e-06
## 11             332.8211             178.2969              500.9149
```

Conclusion

We achieved a large increase in the accuracy of a polynomial spline model against a standard linear model. This was to be expected, although knowing how much better it performed agaisnt a real life data usage goes to show that linear models are useful, however, they serve as a solid foundation for other methodologies of finding predictive models. Some of the challenges we faced were the incompleteness of our dataset over time, writing a custom API scheduler, and not having physical metrics to compare hypothetical data to. If we were to redo this project, we would create multiple Google Developer accounts in order to scrape more data and have complete weekly cycles, perform a k-fold cross validation on our input sets instead of a set validation, and fit a weighted spline model that could more precisely find the knot points of our polynomials.