Public copy

Rudy Nartker

November 25, 2019

Historically we've seen plenty of issues concerning search costs associated with transportation. In other words large portions of mark-up and inflation are capable because consumers aren't aware of all their options and competition is being stifled due to market obfuscation. The relevane in looking into some ways that one might use data science to bring consumer and producer closer together in terms of agreement on product price than ever before. Going into this project I would like to focus on the idea that many things play an important role in determining price or which service people choose. There are more variables than I have had time to go over in this paper but the ones I have chosen hopefully will be adequate in explaining those assumptions. With this project. I had set out to find proftible patterns in the data associated with New York city ride sharing data. Which is at first a rather broad endeavor. One way in which I had intend to show my completion of this goal is by using QDA in order to define the types of cab one might call upon.

```
cab = na.omit(cab)
cab = as.data.frame(cab)

weather = as.data.frame(weather)

cab[,3] = as.POSIXct(cab[,3]/1000, origin="1970-01-01")
weather[,6] = as.POSIXct(weather[,6], origin="1970-01-01")
```

However before we can engage in a QDA there is a tremendous amount of cleaning that must be done. First we remove NA values, this causes a loss of about 8% of raw data, not huge but worth mentioning. Next we must make sure they are data frames and in a format that is usable by our functions, makes them into readable formats for us. Finally our time stamps are unreadable, though they are in the same format the one for the cab data frame is in mili-seconds and the one for the weather data frame is in seconds.

This does however create a new set of problems entirely. While date stamps are a useful tool when creating models they can also be a hinderance when it comes to merging the two data frames. If you wish to merge by date and you can't match because the seconds don't line up, then something must be done. So we split up the date by date, and by time of day.

```
weather[,9] = format(as.POSIXct(strptime(weather$time_stamp, "%Y-%m-%d %H:%M:%S", tz="")) , format = "%H
weather[,10] = format(as.POSIXct(strptime(weather$time_stamp, "%Y-%m-%d %H:%M:%S", tz="")) , format = "%
weather = weather[, -6]
names(weather) = c("temp", "source", "clouds", "pressure", "rain", "humidity", "wind", "hour_stamp", "da

cab[,11] = format(as.POSIXct(strptime(cab$time_stamp, "%Y-%m-%d %H:%M:%S", tz="")) , format = "%H ")
cab[,12] = format(as.POSIXct(strptime(cab$time_stamp, "%Y-%m-%d %H:%M:%S", tz="")) , format = "%Y-%m-%d
cab = cab[, -3]
names(cab) = c("distance", "cab_type", "destination", "source", "price", "surge_multiplier","id", "produ

cab = cab[, -8]
cab = cab[, -3]
bigstupiddf = merge(cab, weather, by = c("date_stamp", "source", "hour_stamp"))
cleanBoi = distinct(bigstupiddf, bigstupiddf$id, .keep_all = T)
cleanBoi = cleanBoi[, -8]

head(cleanBoi)
```

```
##   date_stamp   source hour_stamp distance cab_type price surge_multiplier
```

```
## 1 2018-11-25 Back Bay        21      2.31     Lyft   3.0               1
## 2 2018-11-25 Back Bay        21      2.31     Lyft  16.5               1
## 3 2018-11-25 Back Bay        21      2.99     Lyft  19.5               1
## 4 2018-11-25 Back Bay        21      2.31     Lyft  32.5               1
## 5 2018-11-25 Back Bay        21      2.31     Lyft  16.5               1
## 6 2018-11-25 Back Bay        21      2.31     Lyft  10.5               1
##           name  temp clouds pressure rain humidity wind
## 1       Shared 41.04   0.87  1014.39   NA     0.92 1.46
## 2      Lyft XL 41.04   0.87  1014.39   NA     0.92 1.46
## 3      Lyft XL 41.04   0.87  1014.39   NA     0.92 1.46
## 4 Lux Black XL 41.04   0.87  1014.39   NA     0.92 1.46
## 5          Lux 41.04   0.87  1014.39   NA     0.92 1.46
## 6         Lyft 41.04   0.87  1014.39   NA     0.92 1.46
##                          bigstupiddf$id
## 1 52264c69-04ba-4536-b063-8bc658307e1f
## 2 dd905431-a139-4690-bfee-5f83b610d72d
## 3 861c3854-8614-4e65-bb1d-4a175a4742b7
## 4 ea7f9a8c-25f8-4e23-b2ec-65e0492fd417
## 5 76c23a0a-eb71-48c1-9eb4-6ea1a4fde87f
## 6 824fecb1-1503-41e8-a15a-5a5681d29259
```

After we do that we merge them together using Date, Time and Location. Then we filter out any extranious copies that the process may have made using the ID column which is unique to every ride. There we have our clean data and we can begin to work on our LDA and QDA.

```
lyftsample = sample_n(filter(cleanBoi, cleanBoi$cab_type == 'Lyft'), (0.1*nrow(filter(cleanBoi, cleanBo
ubersample = sample_n(filter(cleanBoi, cleanBoi$cab_type == 'Uber'), (0.1*nrow(filter(cleanBoi, cleanBo
cabsample = rbind(lyftsample, ubersample)
```

now we have the sample we're going to work with, this is a random sample of about half uber and half lyft. This was important to me for analysis sake we want to be sure that we don't accidentally end up with a skewed sample heavily favoring one company or another.

```
numsample = select_if(cabsample, is.numeric)
numsample[is.na(numsample)] = 0
numsample[, 10] = cabsample$cab_type
names(numsample) = c("distance", "price", "surge_multiplier", "temp", "clouds", "pressure", "rain", "hu
```

Here after cleaning re move everything we can't use for analysis. Such as ID number, Cab_type when analyzing name and vice versa.

```
Train = numsample$cab_type %>% createDataPartition(p = 0.8, list = F)
training = numsample[Train, ]
testing = numsample[-Train, ]

preproc.param = training %>% preProcess(method = c("center", "scale"))
training.trans = preproc.param %>% predict(training)
testing.trans = preproc.param %>% predict(testing)

model1 = lda(cab_type~., data=training.trans)
predictions = model1 %>% predict(testing.trans)
predictions$class = factor(predictions$class, levels=c("Uber", "Lyft"), ordered = T)
mean(predictions$class == testing$cab_type)
```

```
## [1] 0.5422702
```

Unfortunately the LDA is only slightly better than random accuracy... 53~54%. This is no better than

random chance. Some of us may do better at predicting via total guess. However we do have the tools to attempt to analyze something with more categories. We take a new sample and begin preparing a new data frame for QDA.

```
lyftsample = sample_n(filter(cleanBoi, cleanBoi$cab_type == 'Lyft'), (0.2*nrow(filter(cleanBoi, cleanBo
ubersample = sample_n(filter(cleanBoi, cleanBoi$cab_type == 'Uber'), (0.2*nrow(filter(cleanBoi, cleanBo
numsample = select_if(rbind(lyftsample, ubersample), is.numeric)


numsample[is.na(numsample)] = 0
numsample[, 10] = cabsample$name
names(numsample) = c("distance", "price", "surge_multiplier", "temp", "clouds", "pressure", "rain", "hu

Train2 = numsample$name %>% createDataPartition(p = 0.8, list = F)
training2 = numsample[Train2, ]
testing2 = numsample[-Train2, ]

preproc.param = training2 %>% preProcess(method = c("center", "scale"))
training.trans = preproc.param %>% predict(training2)
testing.trans = preproc.param %>% predict(testing2)


model2 = qda(name~., data=training.trans)
predictions = model2 %>%  predict(testing.trans)
predictions$class = factor(predictions$class, levels = c("Lyft XL", "Lyft", "Lux Black XL", "Lux", "Lux
mean(predictions$class == testing2$name)
```

```
## [1] 0.08293969
```

Here we are our QDA, does NOT fair much better. However if you consider we "guess blindly" and are equally likely to be correct we are at nearly at double the expected accuracy. Next lets take a look at something a little more lean.

```
bigCab = select_if(cab, is.numeric)
bigCab[, 4] = cab$name
names(bigCab) = c('distance', 'price', 'surge_multiplier', 'name')
head(bigCab)
```

```
##   distance price surge_multiplier         name
## 1     0.44   5.0                1       Shared
## 2     0.44  11.0                1          Lux
## 3     0.44   7.0                1         Lyft
## 4     0.44  26.0                1 Lux Black XL
## 5     0.44   9.0                1      Lyft XL
## 6     0.44  16.5                1    Lux Black
```

```
bigCab = filter(bigCab, bigCab$name != "Black")
bigCab = filter(bigCab, bigCab$name != "Black SUV")
bigCab = filter(bigCab, bigCab$name != "Shared")
bigCab = filter(bigCab, bigCab$name != "UberPool")
bigCab = filter(bigCab, bigCab$name != "UberX")
bigCab = filter(bigCab, bigCab$name != "UberXL")
bigCab = filter(bigCab, bigCab$name != "WAV")


Train3 = bigCab$name %>% createDataPartition(p = 0.8, list = F)
```

```
training3 = bigCab[Train3, ]
testing3 = bigCab[-Train3, ]

preproc.param = training3 %>% preProcess(method = c("center", "scale"))
training.trans3 = preproc.param %>% predict(training3)
testing.trans3 = preproc.param %>% predict(testing3)

model3 = qda(name~., data = training.trans3)
predictions3 = model3 %>% predict(testing.trans3)
predictions3$class = factor(predictions$class, levels = c("Lyft XL", "Lyft", "Lux Black XL", "Lux", "Lu
predictions3$class = na.omit(predictions3$class)

mean(predictions3$class == testing3$name)
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
## Warning in `==.default`(predictions3$class, testing3$name): longer object
## length is not a multiple of shorter object length
```

```
## [1] 0.200527
```

Six classification of ride had to be discarded due to insuficient rank. Which seems strange considering the smaller sample did not have the same problem. I think the biggest problem whch we currently face is insufficient rank which happens to occur with nearly every single Uber service. The reason for this eludes me. additionally it may be worth noting that we an be constructing a model that is over fit. This could be very useful because one could be able to develope a third party app that uses lyft and uber API's. In order to set "Maximum price, distance, party size, etc." specifications in addition to a destination. This app could then querey any local ride share app and classify that which would have the lowest price based on customer preferences and needs. Both Uber and Lyft could also use this inorder to querey eachother's prices and compete. It could be also that one or both companies have already thought of this, and are using some sort of machine learning techniques in order to beat the other to the consumer with prices, deals, and deployment.

However now i think there may be room to try and build a regression model in order to predict price.

```
numsample = numsample[,-10]
linModel1 = lm(price~., numsample)
```

Now lets try and make it a little more lean. using StepAIC we can peel back some variables.

```
linModel2 = stepAIC(linModel1)
```

```
## Start:  AIC=543355.7
## price ~ distance + surge_multiplier + temp + clouds + pressure +
##     rain + humidity + wind
##
##                    Df Sum of Sq      RSS    AIC
## - clouds            1         0  9147157 543354
## - wind              1         6  9147163 543354
## - rain              1        56  9147213 543354
## - humidity          1       109  9147266 543355
## - pressure          1       118  9147275 543355
## <none>                          9147157 543356
## - temp              1       202  9147359 543357
## - surge_multiplier  1    606956  9754113 551516
## - distance          1   1292633 10439790 560147
##
```

```
## Step:  AIC=543353.7
## price ~ distance + surge_multiplier + temp + pressure + rain +
##     humidity + wind
##
##                    Df Sum of Sq      RSS    AIC
## - wind              1         7  9147164 543352
## - rain              1        56  9147213 543353
## - pressure          1       118  9147275 543353
## <none>                           9147157 543354
## - humidity          1       145  9147302 543354
## - temp              1       214  9147371 543355
## - surge_multiplier  1    606984  9754141 551514
## - distance          1   1292705 10439862 560146
##
## Step:  AIC=543351.8
## price ~ distance + surge_multiplier + temp + pressure + rain +
##     humidity
##
##                    Df Sum of Sq      RSS    AIC
## - rain              1        49  9147213 543351
## - humidity          1       142  9147306 543352
## <none>                           9147164 543352
## - temp              1       216  9147380 543353
## - pressure          1       246  9147411 543353
## - surge_multiplier  1    606993  9754158 551513
## - distance          1   1292747 10439912 560145
##
## Step:  AIC=543350.5
## price ~ distance + surge_multiplier + temp + pressure + humidity
##
##                    Df Sum of Sq      RSS    AIC
## - humidity          1       121  9147334 543350
## <none>                           9147213 543351
## - temp              1       224  9147437 543352
## - pressure          1       242  9147455 543352
## - surge_multiplier  1    606975  9754188 551511
## - distance          1   1292736 10439950 560143
##
## Step:  AIC=543350.2
## price ~ distance + surge_multiplier + temp + pressure
##
##                    Df Sum of Sq      RSS    AIC
## - temp              1       143  9147477 543350
## <none>                           9147334 543350
## - pressure          1       243  9147577 543352
## - surge_multiplier  1    606942  9754276 551510
## - distance          1   1292925 10440259 560145
##
## Step:  AIC=543350.2
## price ~ distance + surge_multiplier + pressure
##
##                    Df Sum of Sq      RSS    AIC
## - pressure          1       144  9147621 543350
## <none>                           9147477 543350
```

```
## - surge_multiplier  1    606832  9754309 551509
## - distance          1   1292800 10440277 560143
##
## Step:  AIC=543350.2
## price ~ distance + surge_multiplier
##
##                    Df Sum of Sq      RSS    AIC
## <none>                           9147621 543350
## - surge_multiplier  1    606750  9754370 551507
## - distance          1   1292988 10440609 560145
```

```
summary(linModel2)
```

```
##
## Call:
## lm(formula = price ~ distance + surge_multiplier, data = numsample)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.017  -6.761  -1.537   4.885  52.199
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -12.75309    0.25614  -49.79   <2e-16 ***
## distance           2.81504    0.02101  134.00   <2e-16 ***
## surge_multiplier  22.79156    0.24828   91.80   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.485 on 127045 degrees of freedom
## Multiple R-squared:  0.1755, Adjusted R-squared:  0.1755
## F-statistic: 1.352e+04 on 2 and 127045 DF,  p-value: < 2.2e-16
```

Holy Moly there are only Two Variables left. Distance and Surge Multiplier. Which make total sense that these do affect price. Lets take a closer look at these variables. This means we can reasonably assume weather has no statistically meaningful impact on price of ride sharing. Though the idea of Rain driving a price is interesting. Perhaps people will simply ride share less and plan on making fewer trips on days predicted to have bad weather?

```
anova(linModel2)
```

```
## Analysis of Variance Table
##
## Response: price
##                      Df  Sum Sq Mean Sq F value    Pr(>F)
## distance              1 1340638 1340638 18619.2 < 2.2e-16 ***
## surge_multiplier      1  606750  606750  8426.7 < 2.2e-16 ***
## Residuals        127045 9147621      72
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
thing = filter(cleanBoi, surge_multiplier > 1)
unique(thing$cab_type)
```

```
## [1] "Lyft"
```

This is something i noticed when reading over the Kernals and other research people did on this that there

are no surge multipliers in this data set atatched to Uber. This is frustrating from a modeling perspective because our model may dump ANY surge multiplier not set to 1 into a Lyft price point.

```
library(nnet)
numsample = select_if(cleanBoi, is.numeric)
numsample[is.na(numsample)] = 0

targets = class.ind( c(rep("lyft", length(which(cleanBoi$name == 'Lyft'))), rep("Shared", length(which(c
                       rep("Lux Black XL", length(which(cleanBoi$name == "Lux Black XL"))), rep("Lux",
                       rep("Black", length(which(cleanBoi$name == "Black"))), rep("UberX", length(whic
                       rep("Lux Black", length(which(cleanBoi$name == "Lux Black"))), rep("WAV", lengt
                       rep("UberXL", length(which(cleanBoi$name == "UberXL"))), rep("UberPool", length
                       rep("Black SUV", length(which(cleanBoi$name == "Black SUV")))))

samp = c( sample(1:90749, 63525), sample(90750:181498, 63525), sample(181499:272247, 63524),
          sample(272248:362995, 63525), sample(362996:453744, 63524), sample(453744:544493, 63524),
          sample(544494:635242, 63524))

irl = nnet(numsample[samp,], targets[samp,], size = 4, decay = 5e-4, maxit=200)
```

```
## # weights:  100
## initial  value 1346623.069838
## iter  10 value 444250.150006
## iter  20 value 440392.135450
## iter  30 value 438240.414743
## iter  40 value 432829.738522
## iter  50 value 429090.472352
## iter  60 value 428783.778341
## iter  70 value 425787.807763
## iter  80 value 422912.745732
## final  value 422813.390526
## converged
```

This was our first attempet at a neural net, it seemed to have worked out alright with much debuging. A word to the wise . Most of these classification techniques require you to remove or reconfigure all of your data into some numeric format. As above we re assign all of the various types of cabs (i.e Uber, UberX, Lyft, LyftXL) to the target data frame and it's a binary predictor 1 for yes and 0 for no in each collumn. The Neural Net Package will attempt to coerce any non-numeric variables you have into numeric ones, which will then make them N/A's and then return an Error.