Poisson regression on real data in R

March 16, 2022

0.1 Poisson regression on real data in R

In this lesson, we will analyze real data using the Poisson regression model. A video will accompany this notebook.

The goal of the `bike` data is to keep count of cyclists entering and leaving Queens, Manhattan and Brooklyn via the East River Bridges. The Traffic Information Management System (TIMS) collected this count data for several months during 2017. Each record represents the total number of cyclists per 24 hours at Brooklyn Bridge, Manhattan Bridge, Williamsburg Bridge, and Queensboro Bridge. Also included in the dataset are date and temperature imformation.

Column Name and Column Description

1. `date`: Date the count was conducted
2. `day`: Day of the week the count was conducted
3. `temp_h`: The high temperature for that day in fahrenheit
4. `temp_l`: The low temperature for that night in fahrenheit
5. `precip`: The amount of precipitation for that day in inches
6. `bb`: Total number of cyclist counts at Brooklyn Bridge in a 24 hour period
7. `mb`: Total number of cyclist counts at Manhattan Bridge in a 24 hour period
8. `wb`: Total number of cyclist counts at Williamburg Bridge in a 24 hour period
9. `qb`: Total number of cyclist counts at Queensboro Bridge in a 24 hour period
10. `total`: The number of cyclist counts for all the East River Bridges combined in a 24 hour period

Our goal will be to try to use the weather data to explain the total number of cyclists on the Manhattan Bridge on any given day.

First, we'll load the data into R using the `RCurl` package's `getURL()` function:

[1]:
```
#similar analysis https://towardsdatascience.com/
↪an-illustrated-guide-to-the-poisson-regression-model-50cccba15958library(dplyr)
library(lubridate) #for the ymd() function
library(tidyverse)

#read in the data
```

```r
bike = read.csv("bike.csv", sep = ",", header = TRUE)

#check for NA
sum(is.na(bike$mb))

head(bike, 10)
```

```
Attaching package: 'lubridate'


The following objects are masked from 'package:base':

    date, intersect, setdiff, union


  Attaching packages                          tidyverse
1.3.0

  ggplot2 3.3.0      purrr   0.3.4
  tibble  3.0.1      dplyr   0.8.5
  tidyr   1.0.2      stringr 1.4.0
  readr   1.3.1      forcats 0.5.0

  Conflicts
tidyverse_conflicts()
  lubridate::as.difftime() masks
base::as.difftime()
  lubridate::date()        masks
base::date()
  dplyr::filter()          masks
stats::filter()
  lubridate::intersect()   masks
base::intersect()
  dplyr::lag()             masks
stats::lag()
  lubridate::setdiff()     masks
base::setdiff()
  lubridate::union()       masks
base::union()
```

0

A data.frame: 10 × 10

	date	day	temp_h	temp_l	precip	bb	mb	wb	qb
	<fct>	<fct>	<dbl>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl
1	4/1	Saturday	46.0	37.0	0.00	606	1446	1915	1430
2	4/2	Sunday	62.1	41.0	0.00	2021	3943	4207	2862
3	4/3	Monday	63.0	50.0	0.03	2470	4988	5178	3689
4	4/4	Tuesday	51.1	46.0	1.18	723	1913	2279	1666
5	4/5	Wednesday	63.0	46.0	0.00	2807	5276	5711	4197
6	4/6	Thursday	48.9	41.0	0.73	461	1324	1739	1372
7	4/7	Friday	48.0	43.0	T	1222	2955	3399	2765
8	4/8	Saturday	55.9	39.9	0.00	1674	3163	4082	2691
9	4/9	Sunday	66.0	45.0	0.00	2375	4377	4886	3261
10	4/10	Monday	73.9	55.0	0.00	3324	6359	6881	4731

Note: $T = $ trace precipitation. Let's think of trace precipitation as no precipitation for now...

[2]:
```
#replace T for 0...
bike = bike %>%
    mutate(precip = fct_recode(precip, "0" = "T"))
    #mutate(precip=replace(precip, precip== "T", NA))

bike$precip
#check data types
sapply(bike, class)
```

1. 0.00 2. 0.00 3. 0.03 4. 1.18 5. 0.00 6. 0.73 7. 0 8. 0.00 9. 0.00 10. 0.00 11. 0.00 12. 0.02 13. 0.00 14. 0.00 15. 0.00 16. 0 17. 0 18. 0.00 19. 0 20. 0.17 21. 0.29 22. 0.11 23. 0.00 24. 0 25. 0.91 26. 0.34 27. 0.00 28. 0.00 29. 0.06 30. 0.00 31. 0.00 32. 0.00 33. 0.00 34. 0.00 35. 3.02 36. 0.18 37. 0.01 38. 0.00 39. 0.00 40. 0.00 41. 0.00 42. 0.00 43. 1.31 44. 0.02 45. 0.00 46. 0.00 47. 0.00 48. 0.00 49. 0.00 50. 0.01 51. 0.00 52. 0.59 53. 0.00 54. 0.04 55. 0.58 56. 0.10 57. 0.00 58. 0.00 59. 0.13 60. 0.06 61. 0.03 62. 0.00 63. 0.01 64. 0.01 65. 0.09 66. 0.02 67. 0.06 68. 0.00 69. 0.00 70. 0.00 71. 0.00 72. 0.00 73. 0.00 74. 0 75. 0.29 76. 0.00 77. 0.00 78. 1.39 79. 0 80. 1.35 81. 0.03 82. 0.00 83. 0.00 84. 0.04 85. 1.29 86. 0.00 87. 0.00 88. 0.18 89. 0.00 90. 0.00 91. 0 92. 0.23 93. 0.00 94. 0.45 95. 0.00 96. 0.00 97. 0 98. 1.78 99. 0.00 100. 0.00 101. 0.00 102. 0.00 103. 0.00 104. 0.00 105. 0.35 106. 0.00 107. 0.00 108. 0.00 109. 0.00 110. 0.00 111. 0.01 112. 0.00 113. 0.57 114. 0.06 115. 0.74 116. 0.00 117. 0.00 118. 0 119. 0.00 120. 0.00 121. 0.00 122. 0.00 123. 0.00 124. 0.09 125. 0.00 126. 0.15 127. 0.30 128. 0.00 129. 0.76 130. 0.00 131. 0.00 132. 0.00 133. 0 134. 0.11 135. 0.00 136. 0.00 137. 0.45 138. 0.00 139. 0.00 140. 0.88 141. 0.00 142. 0.00 143. 0.00 144. 0.30 145. 0 146. 0.00 147. 0.00 148. 0.00 149. 0.00 150. 0.00 151. 0.10 152. 0.01 153. 0.00 154. 0.00 155. 0.53 156. 0.74 157. 0.00 158. 0 159. 0.42 160. 0.01 161. 0.00 162. 0.00 163. 0.00 164. 0.00 165. 0.00 166. 0.06 167. 0.02 168. 0.00 169. 0.00 170. 0.00 171. 0.00 172. 0.22 173. 0.00 174. 0.00 175. 0.00 176. 0.00 177. 0.00 178. 0.00 179. 0.00 180. 0.00 181. 0.00 182. 0.00 183. 0.00 184. 0.00 185. 0.00 186. 0.00 187. 0.00 188. 0.00 189. 0.00 190. 0.00 191. 0.22 192. 0.26 193. 0.00 194. 0.06 195. 0.07 196. 0.00 197. 0.08 198. 0 199. 0.01 200. 0.00 201. 0.00 202. 0.00 203. 0.00 204. 0.00 205. 0.00 206. 0.00 207. 0.20 208. 0.00 209. 0.00 210. 0.00 211. 0.00 212. 3.03 213. 0.25 214. 0.00

Levels: 1. '0.00' 2. '0.01' 3. '0.02' 4. '0.03' 5. '0.04' 6. '0.06' 7. '0.07' 8. '0.08' 9. '0.09' 10. '0.10' 11. '0.11' 12. '0.13' 13. '0.15' 14. '0.17' 15. '0.18' 16. '0.20' 17. '0.22' 18. '0.23' 19. '0.25' 20. '0.26' 21. '0.29' 22. '0.30' 23. '0.34' 24. '0.35' 25. '0.42' 26. '0.45' 27. '0.53' 28. '0.57' 29. '0.58' 30. '0.59' 31. '0.73' 32. '0.74' 33. '0.76' 34. '0.88' 35. '0.91' 36. '1.18' 37. '1.29' 38. '1.31' 39. '1.35' 40. '1.39'

41. '1.78' 42. '3.02' 43. '3.03' 44. '0'

date 'factor' **day** 'factor' **temp_h** 'numeric' **temp_l** 'numeric' **precip** 'factor' **bb** 'numeric' **mb** 'numeric' **wb** 'numeric' **qb** 'numeric' **total** 'numeric'

Notice that we'll need to do some additional data cleaning. In particular, many of the variables are being recognized as factors even though they shouldn't be (e.g., the bridge counts, and precip). Also, we can store the date variable as a date.

[3]:

```
#wrangle the data
bike = bike %>%
    mutate(temp_h = as.numeric(temp_h),temp_l = as.numeric(temp_l)) %>%
    mutate(mb = as.numeric(as.character(mb)),bb = as.numeric(as.character(bb)),
           wb = as.numeric(as.character(wb)), qb = as.numeric(as.character(qb)),
           total = as.numeric(as.character(total))) %>%
    mutate(date = as.Date(as.character(date),format='%m/%d')) %>%
    mutate(precip = as.numeric(as.character(precip)))

#fix the year of the date variable
bike$date = ymd(as.character(bike$date)) - years(3)

#summarize and confirm data types
summary(bike)
sapply(bike, class)
head(bike)
```

```
      date                   day         temp_h          temp_l     
 Min.   :2019-04-01   Friday   :30   Min.   :46.0   Min.   :37.00  
 1st Qu.:2019-05-24   Monday   :31   1st Qu.:66.9   1st Qu.:55.23  
 Median :2019-07-16   Saturday :31   Median :75.9   Median :64.00  
 Mean   :2019-07-16   Sunday   :31   Mean   :74.2   Mean   :62.03  
 3rd Qu.:2019-09-07   Thursday :30   3rd Qu.:82.0   3rd Qu.:70.00  
 Max.   :2019-10-31   Tuesday  :31   Max.   :93.9   Max.   :78.10  
                      Wednesday:30                                 
     precip             bb             mb             wb             qb      
 Min.   :0.0000   Min.   : 151   Min.   : 484   Min.   : 874   Min.   : 865  
 1st Qu.:0.0000   1st Qu.:2298   1st Qu.:4308   1st Qu.:5115   1st Qu.:3746  
 Median :0.0000   Median :2857   Median :5608   Median :6287   Median :4681  
 Mean   :0.1318   Mean   :2680   Mean   :5345   Mean   :6052   Mean   :4550  
 3rd Qu.:0.0375   3rd Qu.:3285   3rd Qu.:6760   3rd Qu.:7512   3rd Qu.:5692  
 Max.   :3.0300   Max.   :4960   Max.   :8239   Max.   :8873   Max.   :6582  

     total      
 Min.   : 2374  
 1st Qu.:15705  
 Median :19367  
 Mean   :18628  
 3rd Qu.:23152  
 Max.   :26969  
```

date 'Date' **day** 'factor' **temp_h** 'numeric' **temp_l** 'numeric' **precip** 'numeric' **bb** 'numeric' **mb** 'numeric' **wb** 'numeric' **qb** 'numeric' **total** 'numeric'

A data.frame: 6 × 10

	date	day	temp_h	temp_l	precip	bb	mb	wb	qb
	<date>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<d
1	2019-04-01	Saturday	46.0	37	0.00	606	1446	1915	143
2	2019-04-02	Sunday	62.1	41	0.00	2021	3943	4207	286
3	2019-04-03	Monday	63.0	50	0.03	2470	4988	5178	368
4	2019-04-04	Tuesday	51.1	46	1.18	723	1913	2279	166
5	2019-04-05	Wednesday	63.0	46	0.00	2807	5276	5711	419
6	2019-04-06	Thursday	48.9	41	0.73	461	1324	1739	137

Now, we see that each variable is stored correctly.

For predictive performance purposes, let's split the data into a training set - on which we'll fit the model - and a testing set - on which we'll make "out of sample" predictions". Let's train the model on 80% of the data, and save about 20% for validation.

[4]:
```
set.seed(8585)
bound = floor(nrow(bike)*0.8) #define % of training and test set

df = bike[sample(nrow(bike)), ]        #sample rows
df_train = df[1:bound, ]               #get training set
df_test = df[(bound+1):nrow(bike), ]   #get test set
```

Let's fit a Poisson regression model on the data!

[5]:
```
glm_bike = glm(mb ~ precip + temp_h + temp_l + day, data = df_train, family =␣
 ↪poisson)
summary(glm_bike)
cat("If precipitation is increased by one inch, we would expect the mean number␣
 ↪of bikes across the manhattan bridge to be multiplied by",
    exp(coef(glm_bike)[2]))
```

```
Call:
glm(formula = mb ~ precip + temp_h + temp_l + day, family = poisson,
    data = df_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-41.577   -7.122    2.309    8.523   38.973

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  7.7605007  0.0088089  880.98   <2e-16 ***
precip      -0.6733753  0.0049519 -135.98   <2e-16 ***
temp_h       0.0232326  0.0002333   99.59   <2e-16 ***
```

```
temp_l       -0.0139666  0.0002524  -55.34   <2e-16 ***
dayMonday     0.0758621  0.0038756   19.57   <2e-16 ***
daySaturday  -0.1993917  0.0042215  -47.23   <2e-16 ***
daySunday    -0.2618583  0.0043404  -60.33   <2e-16 ***
dayThursday   0.0999425  0.0038607   25.89   <2e-16 ***
dayTuesday    0.1419263  0.0038452   36.91   <2e-16 ***
dayWednesday  0.1580572  0.0038413   41.15   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 110729  on 170  degrees of freedom
Residual deviance:  34454  on 161  degrees of freedom
AIC: 36242

Number of Fisher Scoring iterations: 4
```

```
If precipitation is increased by one inch, we would expect the mean number of
bikes across the manhattan bridge to be multiplied by 0.5099843
```

This is helpful, but we might want to be more fine-grained in our interpretation. A one inch increase in precipitation can be a lot. Perhaps we'd want to know what a one standard deviation increase would look like. To do so, let's scale the data, and note the standard deviation of the precipitation variable (in the training set):

[6]:
```
cat("The standard deviation of precipitation is", sd(df_train$precip),".")
df_train_scale = df_train %>% mutate_at(c("precip", "temp_h", "temp_l"), scale)
```

```
The standard deviation of precipitation is 0.3755631 .
```

[7]:
```
glm_bike_scale = glm(mb ~ precip + temp_h + temp_l + day, data =␣
 ↪df_train_scale, family = poisson)
coef(glm_bike_scale)
cat(paste0("Assuming the model is correct, if precipitation is increased by",
           "one standard deviation (0.38 inches), we would expect the mean ",
           "number of bikes across the manhattan bridge to be multiplied by"),
    exp(coef(glm_bike_scale)[2]),"adjusting for tempteratures, and day of week.
 ↪")
```

(Intercept) 8.5214006831787 **precip** -0.252894897252218 **temp_h** 0.237264729556711 **temp_l** -0.128276909705292 **dayMonday** 0.0758620891615072 **daySaturday** -0.199391723214535 **daySunday** -0.261858265426246 **dayThursday** 0.0999425319296252 **dayTuesday** 0.141926290043043 **dayWednesday** 0.158057155721602

```
Assuming the model is correct, if precipitation is increased byone standard
deviation (0.38 inches), we would expect the mean number of bikes across the
```

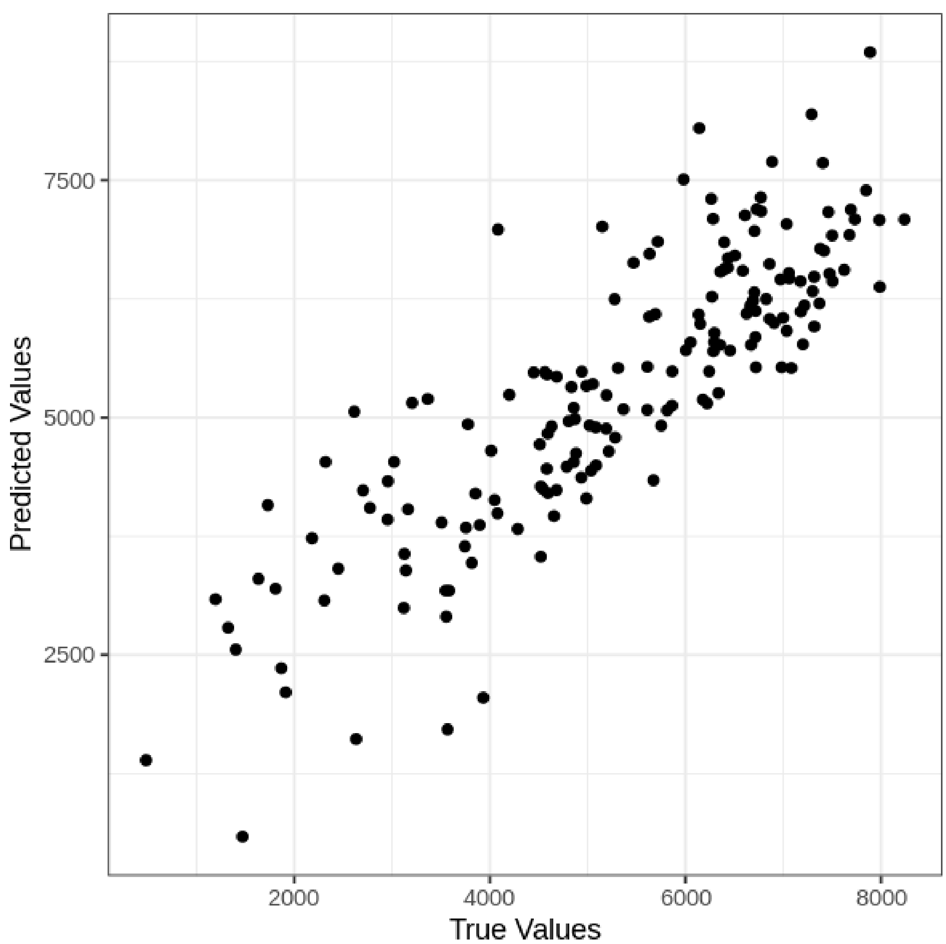
```
manhattan bridge to be multiplied by 0.7765495 adjusting for tempteratures, and
day of week.
```

Of course, we have not yet studied whether this model is correct. One simple assessment that we can look at is a plot of the predicted values ($\widehat{\mu}$) vs the true values for either the training set, or the the out-of-sample (test) set. In the training set, we see points aligning along $y = x$, but with some variability, suggesting that the fit could be better.

[8]:
```
n_train = length(df_train$mb)
n_test = length(df_test$mb)

mu_train = predict(glm_bike, df_train, type="response")
y_train = df_train$mb

options(repr.plot.width = 5, repr.plot.height = 5)
df_test_predict = data.frame(y_train, mu_train)
p = ggplot(df_test_predict) + geom_point(aes(y_train, mu_train))
p = p + theme_bw()
p = p + xlab("True Values") + ylab("Predicted Values")
p
```



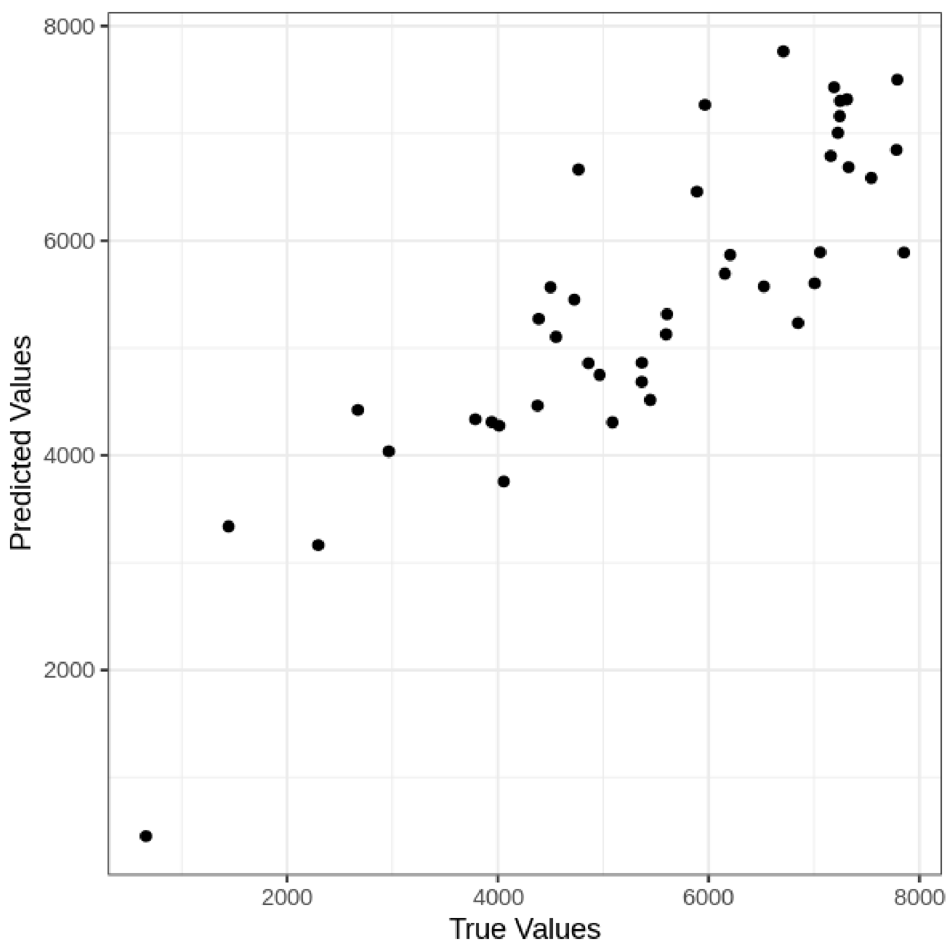
The same plot on the test set shows a bit more variability, further suggesting that the model fit is less than optimal.

[9]:

```
n_train = length(df_train$mb)
n_test = length(df_test$mb)

mu_test = predict(glm_bike, df_test, type="response")
y_test = df_test$mb

options(repr.plot.width = 5, repr.plot.height = 5)
df_test_predict = data.frame(y_test, mu_test)
p = ggplot(df_test_predict) + geom_point(aes(y_test, mu_test))
p = p + theme_bw()
p = p + xlab("True Values") + ylab("Predicted Values")
p
```



[ ]:

```

```