Predicting the volume of clicks on FreeCodeCamp Facebook Page posts With Machine Learning.

Mohammed Chihab

5/15/2020

Executive Summary

As a brief summary, the main goal of this project is to create a clicks volume prediction system on Facebook posts based on “day”, “type” and “reach” variables. The data is retrieved on 5/14/2020 from “https://github.com/freeCodeCamp/open-data/tree/master/facebook-fCC-data”. As per the description of the data from the source as well as the data itself, the “day” variable refers to the day of post. The “type” variable refers to the categories “Link”, “Photo”, “Public”, “Status” or “Video”. The variable “reach” refers to the people who viewed the post even if not clicking the link in the post. The raw data features the “clicks” variable as a continuous variable which refers to the number of users who clicked the link. However, our interest is to split this variable into intervals representing the volumes of clicks i.e. “low”, “medium” “high”. Our objective is to predict these volumes based on the variables “day”, “type” and “reach”. The steps to reach our goals are explained in the Methods/Analysis Section.

Methods/Analysis

Introduction

In this project, many measures and steps will be taken and followed: prepare the data, explore its main features, scale the predictors data matrix, re-explore the data by performing a principal component analysis of the scaled matrix, split the data into training and testing sets, train models to predict the volume of clicks, then, finally apply the chosen model to predict the volumes of clicks on a simulated foreign validation dataset split from the original scaled dataset. To measure accuracy during the training of our models, the mean of both the predicted and the true data will be used.

Section 01: Libraries Requirements

The following libraries are required for our code to run. This will Install and/or load them to the R platform.

```
if(!require(tidyverse)) install.packages("tidyverse")
library(tidyverse)
if(!require(caret)) install.packages("caret")
library(caret)
if(!require(data.table)) install.packages("data.table")
library(data.table)
if(!require(lubridate)) install.packages("lubridate")
library(lubridate)
if(!require(matrixStats)) install.packages("matrixStats")
library(matrixStats)
if(!require(gam)) install.packages("gam")
library(gam)
if(!require(readr)) install.packages("readr")
library(readr)
```

Section 02: Data Upload

In this section, we will download and save our data automatically in a temp file. The data shall be named "facebkFCC_main".

Note: this process could take a couple of minutes.

If you would like to search the data on the net, the freeCodeCamp-facebook-page-activity dataset can be found here:
- datset as of 5/14/2020: https://raw.githubusercontent.com/chihabmmed/facebook-fCC-data/master/freeCodeCamp-facebook-page-activity.csv
- datset as of 5/14/2020: https://files.gitter.im/FreeCodeCamp/DataScience/l4Qf/freeCodeCamp-facebook-page-activity.csv
- datset as of 5/14/2020: https://raw.githubusercontent.com/freeCodeCamp/open-data/master/facebook-fCC-data/data/freeCodeCamp-facebook-page-activity.csv

```
ds <- tempfile()
download.file("https://raw.githubusercontent.com/chihabmmed/facebook-fCC-data/master/freeCodeCamp-facebook-page-activity.csv", ds)
facebkFCC_main <- read_csv(ds)
```

```
## Parsed with column specification:
## cols(
##   date = col_character(),
##   time = col_time(format = ""),
##   title = col_character(),
##   type = col_character(),
##   reach = col_double(),
##   clicks = col_double(),
##   reactions = col_character()
## )
```

Section 03: Data Cleaning

~~####~~ Data Wrangling: Cleaning and setting up the data set.

First, Create a function that will Split the clicks data into three intervals, and convert them into categories. The three intervals will represent the volumes: "low", "medium" and "high". The new variable will be named "volume". The function that will carry out this task will be named: "lmh". The function will categorize the number of clicks

below **450** as a low volume, between **450** and **1.16e+03** as a medium volume, above **1.16e+03** as a high volume of clicks.

```
lmh <- function(x) {
  ifelse(x<450,"low",ifelse(x<=1.16e+03,"medium","high"))
}
```

In this part of the section, we will clean and set up the data. all variable will be converted into numericals.
- The day variable which represents: “Sun” “Mon” “Tue” “Wed” “Thu” “Fri” “Sat” , will be converted into “1” “2” “3” “4” “5” “6” “7” respectively.
- The type variable which represents: “Link” “Photo” “Public” “Status” “Video”, will be converted into “1” “2” “3” “4” “5” respectively.
- The reach variable will be converted into numerical.

```
facebkFCC <- facebkFCC_main %>%
  # use lubridate to wrangle the date variable, keep only the days, convert it into numerical
  mutate(day = as.numeric(wday(mdy(date)))) %>%
  #Factor the type variable, unclass it and convert it into numerical
  mutate(type = as.numeric(unclass(factor(type))))%>%
  #Convert the reach variable into numerical
  mutate(reach = as.numeric(as.character(reach))) %>%
  #Apply the function lmh created above to split it into three levels, name it volume
  mutate(volume = factor(lmh(clicks))) %>%
  #Select only day, type, reach, volume
  select(day, type, reach, volume)
```

Section 04: Data Exploration - First Part

In this parsection, we will explore the main features of the data in our hands.

The dimentions and properties

```
dim(facebkFCC)
```

```
## [1] 420   4
```

The facebkFCC data set has 420 observations and 4 variables.

The structure of the data

```
str(facebkFCC)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 420 obs. of  4 variables:
##  $ day   : num  6 6 5 5 5 4 4 2 2 1 ...
##  $ type  : num  1 1 5 1 1 1 2 2 1 1 ...
##  $ reach : num  1768 6941 17399 3751 18248 ...
##  $ volume: Factor w/ 3 levels "high","low","medium": 2 3 1 2 1 2 3 1 1 3 ...
```

The structure shows us that all variables in the data are numericals except the volume variable which is categorical.

The first three rows in our data

```
head(facebkFCC, 3)
```

```
## # A tibble: 3 x 4
##     day  type reach volume
##   <dbl> <dbl> <dbl> <fct>
## 1     6     1  1768 low
## 2     6     1  6941 medium
## 3     5     5 17399 high
```

####———————— The summary function provide us with a brief glimps on our data

```
summary(facebkFCC)
```

```
##       day             type           reach           volume
##  Min.   :1.000   Min.   :1.000   Min.   :   688   high  :141
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:  6477   low   :140
##  Median :4.000   Median :1.000   Median :  9956   medium:139
##  Mean   :3.757   Mean   :1.455   Mean   : 13524
##  3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.: 15124
##  Max.   :7.000   Max.   :5.000   Max.   :408191
```

####———————— The number of predictors in the dataset

```
dim(facebkFCC[,-4])
```

```
## [1] 420   3
```

There are 3 predictors in the data set, with 420 observations.

####———————— Levels in the test set

```
levels(facebkFCC$volume)
```

```
## [1] "high"   "low"    "medium"
```

####———————— Proportions of the volume samples in the test set

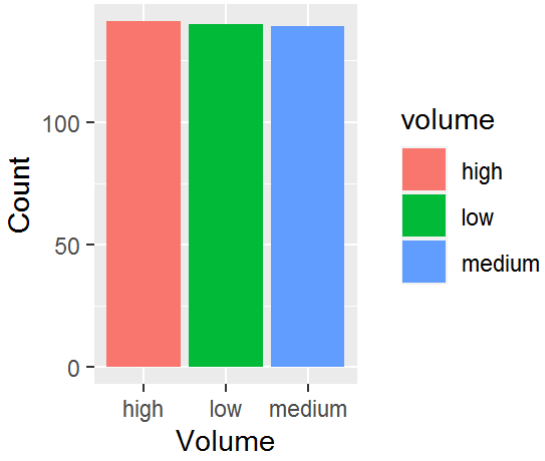
```
prop_volume <- data.frame(
  #Proportion of the "high" volume samples in the test set
  prop_high =  mean(facebkFCC$volume == "high"),
  #Proportion of the "medium" volume samples in the test set
  prop_medium = mean(facebkFCC$volume == "medium"),
  #Proportion of the "low" volume samples in the test set
  prop_low = mean(facebkFCC$volume == "low"))
prop_volume
```

```
##   prop_high prop_medium  prop_low
## 1 0.3357143   0.3309524 0.3333333
```

The porportions are almost identical this can also be manifested as below in the bar figure.

####———————— Plotting the counts per volume (each volume represents an interval of clicks)

```
ggplot(aes(x = volume), data = facebkFCC, echo = FALSE) +
  geom_bar(aes(fill = volume), stat = 'count') +
  xlab('Volume') + ylab('Count') + labs(fill = 'volume')
```



Section 05: Data Scaling

####———————— Converting the data into a list of a matrix and a vector

```
facebkFCC_list <- list(x = data.matrix(facebkFCC[,-4]), y = facebkFCC$volume)
```

####———————— Scaling the matrix of the predictors

```
x_centered <- sweep(facebkFCC_list$x, 2, colMeans(facebkFCC_list$x))
x_scaled <- sweep(x_centered, 2, colSds(facebkFCC_list$x), FUN = "/")
```

Section 06: Data Exploration - Second Part

Performing principal component analysis of the scaled matrix.

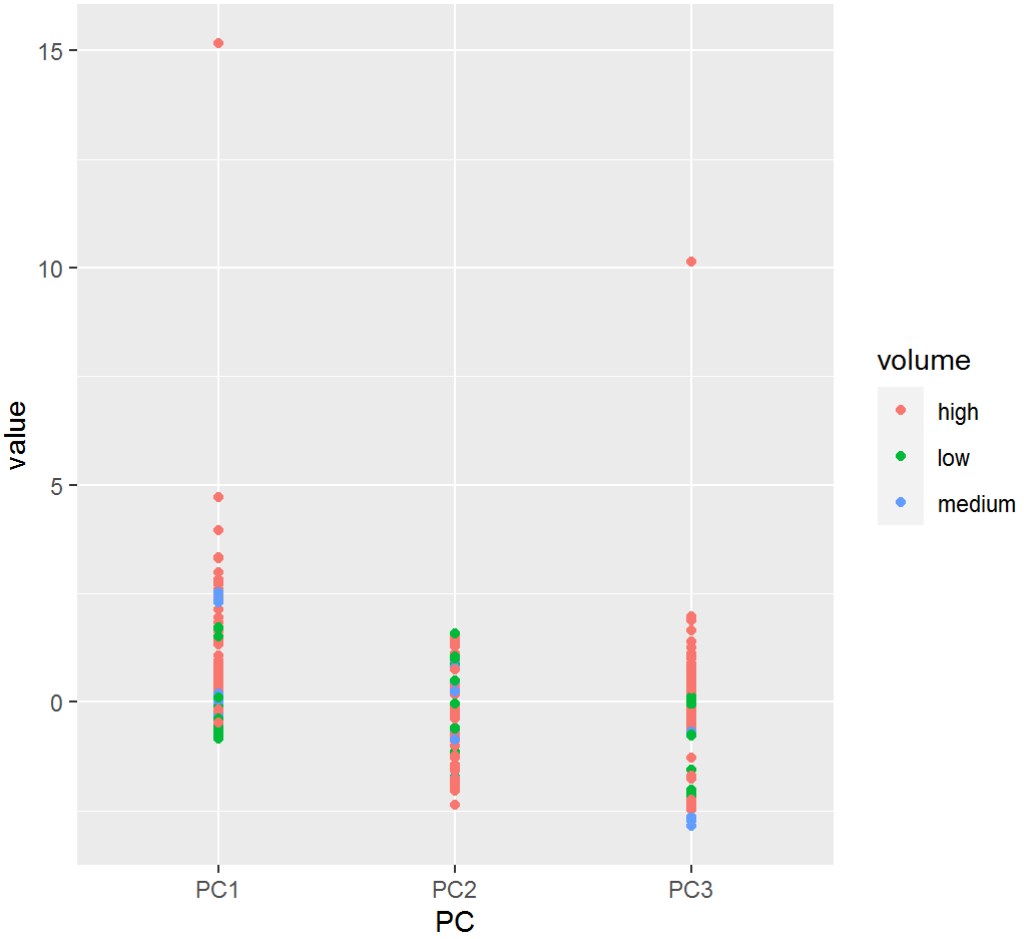
####———————— The proportion of variance explained by each principal component.

```
pca <- prcomp(x_scaled)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     1.1168 0.9985 0.8693
## Proportion of Variance 0.4158 0.3323 0.2519
## Cumulative Proportion  0.4158 0.7481 1.0000
```

####———————— Plotting PCs against their value wrapped by volume

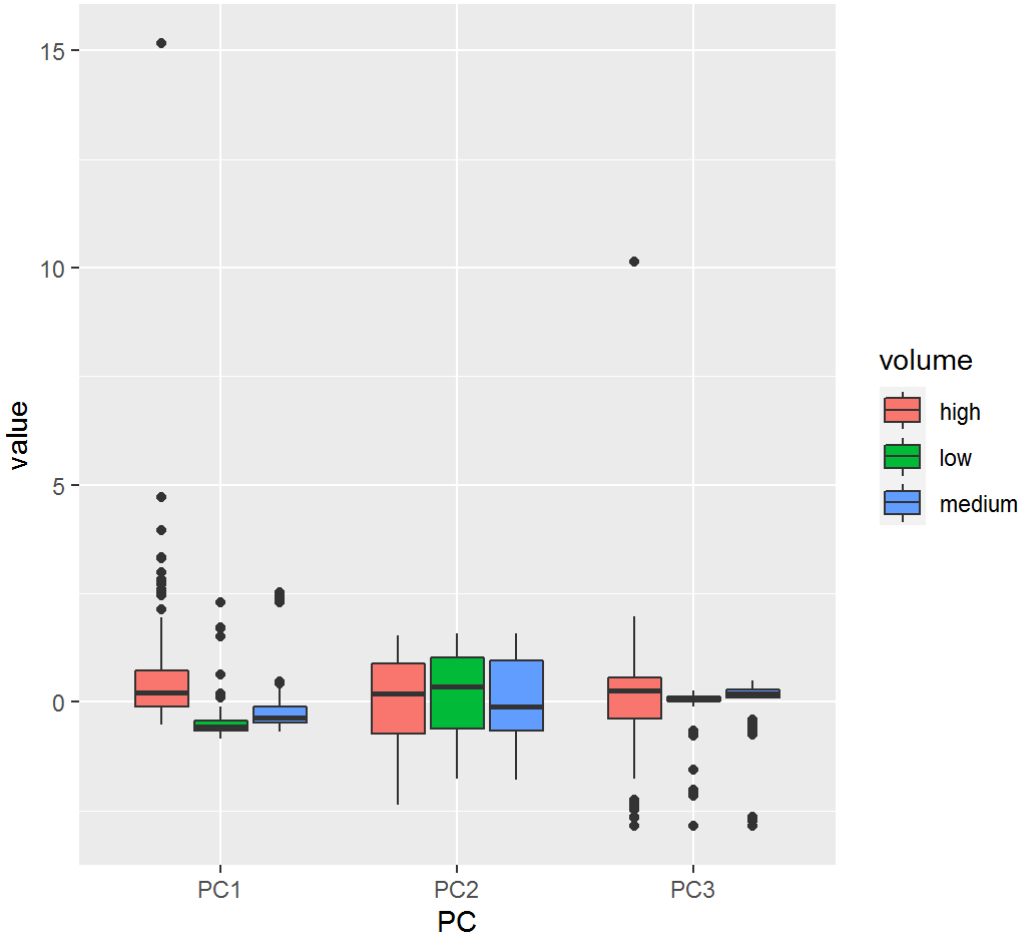
```
data.frame(volume = facebkFCC_list$y, pca$x[,1:3]) %>%
  gather(key = "PC", value = "value", -volume) %>%
  ggplot(aes(PC, value, fill = volume, color = volume)) +
  #facet_wrap(~volume) # this will wrap based on volume
  geom_point()
```



High volume tends to have high values of pc1 and pc3. Low volume tends to have high values of pc2.

PC boxplot

```
data.frame(volume = facebkFCC_list$y, pca$x[,1:3]) %>%
  gather(key = "PC", value = "value", -volume) %>%
  ggplot(aes(PC, value, fill = volume)) +
  geom_boxplot()
```



This box plot shows that PC1 is significantly different enough by volume that there is little overlap in the interquantile ranges for low, medium and high volumes.

Section 07: Training and testing data sets creation

####———————— Creating development sets as well as validation sets

```
set.seed(1, sample.kind = "Rounding")
validation_index <- createDataPartition(facebkFCC_list$y, times = 1, p = 0.2, list = FALSE)
validation_x <- x_scaled[validation_index,]
validation_y <- facebkFCC_list$y[validation_index]
dev_x <- x_scaled[-validation_index,]
dev_y <- facebkFCC_list$y[-validation_index]
```

####———————— Splitting the development set into a train set and a test set

```
#putting data into a list
dev_data <- list(x = dev_x, y = dev_y)

#Splitting data
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(dev_data$y, times = 1, p = 0.2, list = FALSE)
test_x <- dev_data$x[test_index,]
test_y <- dev_data$y[test_index]
train_x <- dev_data$x[-test_index,]
train_y <- dev_data$y[-test_index]
```

Section 08: Training models

####_____ Training models to predict the volume of clicks based on day, type and reach_____ ####

####————————— lda

Here we shall use the Linear Discriminant Analysis to predict the test set, Which is based on Bayes theorem. It first estimates the mean and variance for each class, then calculates the covariance to discriminate each class, and then it uses Bayes theorem to estimate the probability of each class.

```
train_lda <- train(train_x, train_y,
                   method = "lda")
lda_preds <- predict(train_lda, test_x)
acc_lda <- mean(lda_preds == test_y)
acc_lda
```

```
## [1] 0.6666667
```

####————————— qda

The Quadratic Discriminant Analysis algorithm is also based on Bayes theorem. Yet, its approach is different from that of linear regression. It is an extension of Linear Discriminant Analysis. However, It considers each class as having its own variance or covariance matrix rather than to have a common one.

```
train_qda <- train(train_x, train_y,
                   method = "qda")
qda_preds <- predict(train_qda, test_x)
acc_qda <- mean(qda_preds == test_y)
acc_qda
```

```
## [1] 0.7246377
```

####————————— knn

knn algorithm uses distance function to classify data, sort this distance, takes the k-th minimum distance, decide on the nearest neighbors, keeps the categories which are included in the k-th nearest neighbor. Then, it uses vote majority, i.e. simple majority based on the categories kept in the previous step to conclude the result.

```
set.seed(1, sample.kind = "Rounding")
tuning <- data.frame(k = c(3, 5, 7, 9))
train_knn <- train(train_x, train_y,
                   method = "knn",
                   tuneGrid = tuning)
train_knn$bestTune
```

```
##   k
## 2 5
```

```
knn_preds <- predict(train_knn, test_x)
acc_knn <- mean(knn_preds == test_y)
acc_knn
```

```
## [1] 0.7101449
```

rf

Random Forest is an algorithm that uses randomness to the model while growing multiple trees. Random forest algorithm average these trees to draw the final prediction. For more information and interpretability, we shall use **varImp** function from caret to shed light on variable importance.

```
set.seed(1, sample.kind = "Rounding")
tuning <- data.frame(mtry = c(3, 5, 7, 9))    # can expand to seq(3, 21, 2), same
train_rf <- train(train_x, train_y,
                  method = "rf",
                  tuneGrid = tuning,
                  importance = TRUE)
train_rf$bestTune
```

```
##   mtry
## 1    3
```

```
rf_preds <- predict(train_rf, test_x)
acc_rf <- mean(rf_preds == test_y)
acc_rf
```

```
## [1] 0.7681159
```

```
varImp(train_rf)
```

```
## rf variable importance
##
##   variables are sorted by maximum importance across the classes
##          high    low medium
## reach 100.000 99.931  47.86
## type   21.869 44.045  13.37
## day     6.523  8.608   0.00
```

nnet

Here, we train a nnet model with caret package which is, as the name suggests, based on the neural networks. According to caret documentation, amongst the most important parameters in this model are size and decay. While size is the number of units in the hidden layer, which Can be zero if there are skip-layer units, decay is the parameter for weight decay and has a default of 0. It is the regularization parameter that will be used to prevent over-fitting. However, in the training of our model we will keep the default values of these parameters.

```
set.seed(1, sample.kind = "Rounding")
trContol <- trainControl(method = "repeatedcv", number = 10, repeats=10)

train_nnet <- train(train_x, train_y,
                    method = "nnet",
                    trControl= trContol,
                    na.action = na.omit,
                    trace = FALSE)
nnet_preds <-predict(train_nnet, test_x)
acc_nnet <- mean(nnet_preds == test_y)
acc_nnet
```

```
## [1] 0.8115942
```

———————— **pcaNNet**

Here, we apply pcaNNet model on our scaled data to predict the volumes.
Neural Networks with a Principal Component Step or pcaNNet is an algorith that According to caret documentation process the data as follows:

- The train function will run PCA analysis on the data set, and then will use it in a neural network model.
- After running pca on the data, it draws the cumulative percentage of variance for each pc.
- Then the amount of variance in the predictors is captured.
- The next step is when the neural network model is applied.
- For new samples, the same transformation based on the pca analysis is applied to the new predictor values.

```
set.seed(1, sample.kind = "Rounding")
trContol <- trainControl(method = "repeatedcv", number = 10, repeats=10)

train_pcaNNet <- train(train_x, train_y,
                       method = "pcaNNet",
                       trControl= trContol,
                       na.action = na.omit,
                       trace = FALSE)
pcaNNet_preds <-predict(train_pcaNNet, test_x)
acc_pcaNNet <- mean(pcaNNet_preds == test_y)
acc_pcaNNet
```

```
## [1] 0.826087
```

Section 09: Results and Comparisson of Trained Models

In this section we shall look at the output accuracy of all previously trained models and compare them. based on this comparison we shall arrange them ascendingly from the lowest performing model to the highest performing model, and based on that we will choose one.

```
models <- c("LDA", "QDA", "K nearest neighbors", "Random forest", "nnet", "pcaNNet")
accuracy <- c(acc_lda,
              acc_qda,
              acc_knn,
              acc_rf,
              acc_nnet,
              acc_pcaNNet)
data.frame(Model = models, Accuracy = accuracy) %>% arrange(Accuracy)
```

```
##                 Model  Accuracy
## 1                 LDA 0.6666667
## 2 K nearest neighbors 0.7101449
## 3                 QDA 0.7246377
## 4       Random forest 0.7681159
## 5                nnet 0.8115942
## 6             pcaNNet 0.8260870
```

According to this order of the results above, the highest performing model is **pcaNNet** with **0.826087** accuracy. It also shows us that performing different models led us to find a better one from **0.6666667** accuracy achieved by **lda** to **0.8115942** achieved by **nnet** and **0.826087** achieved by **pcaNNet**. Since our model of choice will be the one that has the highest performance amongst all the trained models, **pcaNNet** will be our **model of choice**.

Results: using our model of choice on the validation dataset

Section 01: Applying pcNNet Model on the Validation dataset

In the previous sections we base our analysis and trained our models on a development data set that was itself split into a training set and a testing set. The development data set itself comes from a larger data set that was split into development set and a validation set. We considered the validation set as simulating the real life data that will not be used in our training of the models, nut rather will be used at the end to validate our model of choice in a final step. this data is, thus, considered foreign and real-life simulation.

```
set.seed(1, sample.kind = "Rounding")
trContol <- trainControl(method = "repeatedcv", number = 10, repeats=10)

val_train_pcaNNet <- train(dev_data$x, dev_data$y,
                           method = "pcaNNet",
                           trControl= trContol,
                           na.action = na.omit,
                           trace = FALSE)
val_pcaNNet_preds <-predict(val_train_pcaNNet, validation_x)
val_pcaNNet_acc <- mean(val_pcaNNet_preds == validation_y)
val_pcaNNet_acc
```

```
## [1] 0.8
```

Based on the results of the application of our model on the novel, foreign and real-life simulated validation set which have achieved an accuracy of **0.8**, that our model based on **pcaNNet** is performing as expected.

Conclusion

To conclude, our model performed as expected. However, we have achieved **0.8** accuracy on our validation set and **0.826087** accuracy on the test set of the development set. This gives us space for improvement and provides possiblities for more training of other tailored models. Other facilities, such working on large data might also provide us with more insights.

References

####————————— References as of 5/14/2020

- https://raw.githubusercontent.com/chihabmmed/facebook-fCC-data/master/freeCodeCamp-facebook-page-activity.csv
- https://files.gitter.im/FreeCodeCamp/DataScience/l4Qf/freeCodeCamp-facebook-page-activity.csv
- https://raw.githubusercontent.com/freeCodeCamp/open-data/master/facebook-fCC-data/data/freeCodeCamp-facebook-page-activity.csv
- https://quantdev.ssri.psu.edu/tutorials/supervised-machine-learning-caret-package
- http://topepo.github.io/caret/miscellaneous-model-functions.html#neural-networks-with-a-principal-component-step
- https://cran.r-project.org/web/packages/caret/caret.pdf
- https://cran.r-project.org/web/packages/nnet/nnet.pdf
- https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html
- https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/qda.html

- https://datascienceplus.com/how-to-perform-logistic-regression-lda-qda-in-r/
- https://www.rdocumentation.org/packages/MASS/versions/7.3-51.5/topics/qda
- https://www.datatechnotes.com/2018/10/lda-classification-in-r.html#more