EDX Capstone Stroke Predictions

Eduardo Almada

18/03/2021

Contents

1 Summary

This is a project report for Edx HarvardX: PH125.9 - Data Science: Capstone, in which every student must look for a dataset to analyse and use machine learning to predict an outcome, for this report I chose the **Stroke Prediction Dataset** in which the dataset can be used to predict whether a patient is likely to get a stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relevant information about the patient (id).

2 Introduction

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. so being able to predict a stroke based on the various inputs can have a greater impact on helping avoiding or containing them.

3 Pre-processing the data

In order to start the algorithm we should first prepare the data, so lets review it first:

```r
# Load data from a csv within a zip file
data <- read_csv(unz('archive.zip','healthcare-dataset-stroke-data.csv'))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   id = col_double(),
##   gender = col_character(),
##   age = col_double(),
##   hypertension = col_double(),
##   heart_disease = col_double(),
##   ever_married = col_character(),
##   work_type = col_character(),
##   Residence_type = col_character(),
##   avg_glucose_level = col_double(),
##   bmi = col_character(),
##   smoking_status = col_character(),
##   stroke = col_double()
## )
```

```r
# data preview
summary(data)# data summary
```

```
##        id           gender               age         hypertension
##  Min.   :   67   Length:5110        Min.   : 0.08   Min.   :0.00000
##  1st Qu.:17741   Class :character   1st Qu.:25.00   1st Qu.:0.00000
##  Median :36932   Mode  :character   Median :45.00   Median :0.00000
##  Mean   :36518                      Mean   :43.23   Mean   :0.09746
##  3rd Qu.:54682                      3rd Qu.:61.00   3rd Qu.:0.00000
##  Max.   :72940                      Max.   :82.00   Max.   :1.00000
##  heart_disease     ever_married        work_type         Residence_type
##  Min.   :0.00000   Length:5110        Length:5110        Length:5110
##  1st Qu.:0.00000   Class :character   Class :character   Class :character
##  Median :0.00000   Mode  :character   Mode  :character   Mode  :character
##  Mean   :0.05401
##  3rd Qu.:0.00000
##  Max.   :1.00000
##  avg_glucose_level     bmi           smoking_status         stroke
##  Min.   : 55.12    Length:5110        Length:5110        Min.   :0.00000
##  1st Qu.: 77.25    Class :character   Class :character   1st Qu.:0.00000
##  Median : 91.89    Mode  :character   Mode  :character   Median :0.00000
```

```
##  Mean   :106.15                                              Mean   :0.04873
##  3rd Qu.:114.09                                              3rd Qu.:0.00000
##  Max.   :271.74                                              Max.   :1.00000
```

Now that we know how the data is arranged, we will proceed by counting if any all the missing values.

Variable	n
id	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
work_type	0
Residence_type	0
avg_glucose_level	0
bmi	201
smoking_status	1544
stroke	0

3.1 Cleaning the data

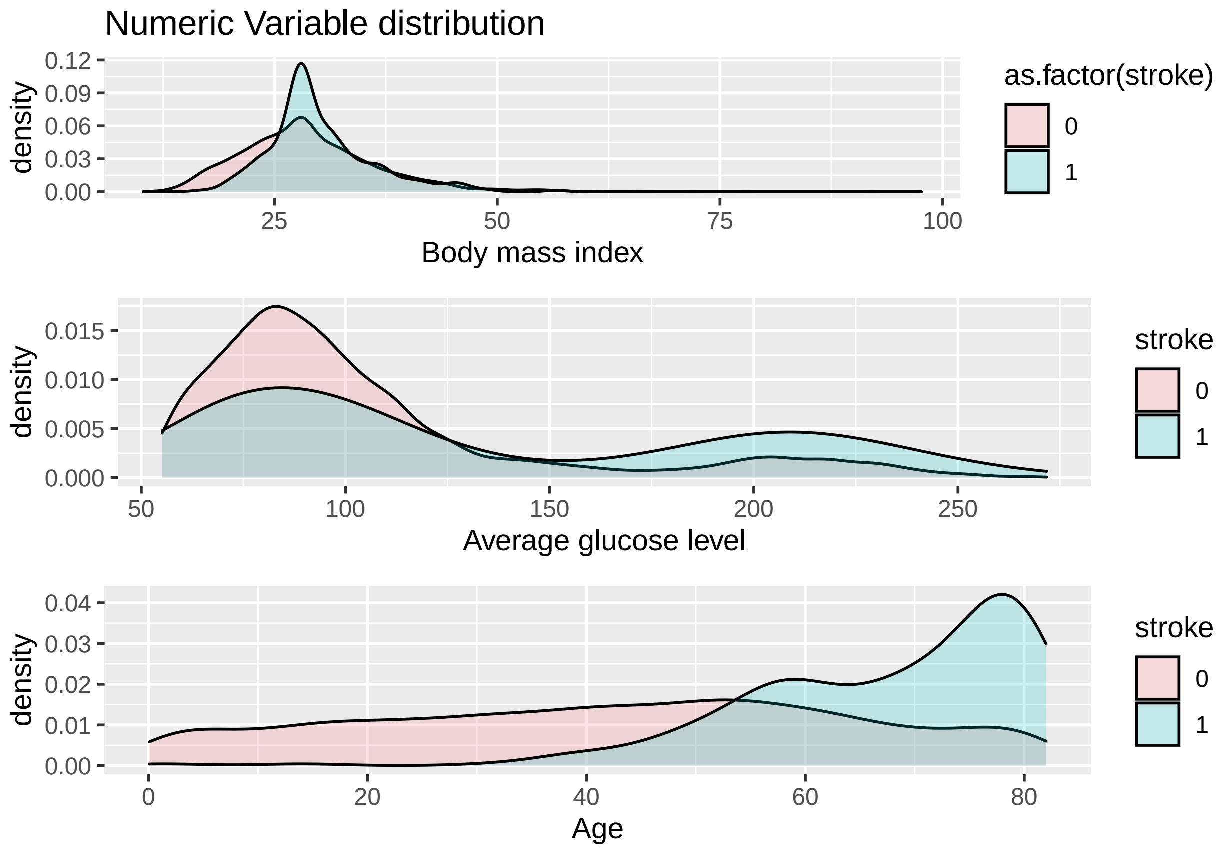
Two variables are missing data, so before we proceed with the analysis this values should be fix.

```
# replacing N/A and Unknown with NA
clean_data <- replace_with_na(data,replace = list(bmi = c('N/A'),
                              smoking_status = c("Unknown")))

# cleaning and formatting data
post_data <- clean_data %>% mutate(stroke = factor(stroke),
        bmi = ifelse(is.na(bmi),median(bmi,na.rm = T),bmi),
        hypertension = factor(hypertension),
        heart_disease = factor(heart_disease)) %>% filter(!gender == 'Other') %>%
        select(-id)
# filling smoking_status Na's with previous values
post_data <- post_data %>% fill(smoking_status)
```

3.2 Vizualising the data

Now lets create density plots for data exploration.



After analyzing the plots, it is concluded that the variable *age* is the one that drives the chance of a stroke, this will be useful when approaching the methods for ML.

4 Exploring the data

Let us see the proportions of people with stroke records:

stroke	n	prop
0	4860	0.95
1	249	0.05

It is evident that the data is imbalance, so if we were to always predict for people not to get a stroke we would have a 95% accuracy, but this is not our goal. Lets see if using maching learning we can improve the Specificity or capability of predicting negative negatives, assuming we set our positive as not having a stroke.

Before we train the data let us change the characters vectors into factors and the body mass index to the categories the Center of Disease control and Prevention standardizes according to the bmi of a person. (underweight, normal weight, overweight and obese)

```
# preparing data for ML algorithms
post_data <- post_data %>%
  mutate(across(c(hypertension, heart_disease), factor), # changing hypertension & heart_disease class
         across(where(is.character), as.factor), # changing character class to factors
```

```r
         across(where(is.factor), as.numeric), # changing factors to numeric
         stroke = factor(ifelse(stroke == 1, '0', '1')), # setting stroke factors to 1 for positive and
        # changing bmi vector from numeric to categorical according to the CDC categories
        # https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html
         bmi = case_when(bmi < 18.5 ~ 'underweight',
        bmi >= 18.5 & bmi < 25 ~ 'normal_weight',
        bmi >= 25 & bmi < 30 ~ 'overweight',
        bmi >= 30 ~ 'obese'),
        bmi = factor(bmi,
        levels = c("underweight", "normal_weight",
        "overweight", "obese"), order = TRUE)
        ) %>% as_tibble()
head(post_data)
```

```
## # A tibble: 6 x 11
##   gender   age hypertension heart_disease ever_married work_type Residence_type
##    <dbl> <dbl>        <dbl>         <dbl>        <dbl>     <dbl>          <dbl>
## 1      2    67            1             2            2         4              2
## 2      1    61            1             1            2         5              1
## 3      2    80            1             2            2         4              1
## 4      1    49            1             1            2         4              2
## 5      1    79            2             1            2         5              1
## 6      2    81            1             1            2         4              2
## # ... with 4 more variables: avg_glucose_level <dbl>, bmi <ord>,
## #   smoking_status <dbl>, stroke <fct>
```

4.1 Training data

```r
########### Set training and validation set #########
# setting root for repeatability purposes
set.seed(2021)

# creating train and test set
index <-  createDataPartition(post_data$stroke, times = 1, p = 0.3, list = F)
o_train_set <- post_data[-index,]
o_test_set <- post_data[index,]
```

```
## Dimensions of the train set:  3576 11

## Dimensions of the test set:  1533 11
```

5 Methods

In this section, two methods were used to develop the predictions of having a stroke:

5.1 GLM Model

The *generalized linear model* is a generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution like Gaussian distribution.

```r
##################### glm model ####################
set.seed(2021) # setting root for repeatability purposes
glm_model <- train(stroke ~ ., method = 'glm', data = o_train_set)
confusionMatrix(predict(glm_model, o_test_set), o_test_set$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1458   75
##          1    0    0
##
##                Accuracy : 0.9511
##                  95% CI : (0.9391, 0.9613)
##     No Information Rate : 0.9511
##     P-Value [Acc > NIR] : 0.5307
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9511
##          Neg Pred Value :    NaN
##              Prevalence : 0.9511
##          Detection Rate : 0.9511
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

As we predicted before we do get an accuracy of 95%, and a Specificity of zero, this means that this model is overperforming and should be rejected.

5.2 RF Model

The *Random forest* model consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction

```r
##################### random forest model ####################
set.seed(2021) # setting root for repeatability purposes

# setting the tune grid
rfGrid <- data.frame(.mtry = c(2,3,5,6),.splitrule = "gini",
  .min.node.size = 5)

# setting the control parameters
rfControl <- trainControl(
  method = "oob", number = 5,
```

```
  verboseIter = TRUE)

rf_model <- train(stroke ~ ., data = o_train_set,
  method = "ranger", tuneLength = 3,
  tuneGrid = rfGrid,trControl = rfControl)
```

```
## + : mtry=2, splitrule=gini, min.node.size=5
## - : mtry=2, splitrule=gini, min.node.size=5
## + : mtry=3, splitrule=gini, min.node.size=5
## - : mtry=3, splitrule=gini, min.node.size=5
## + : mtry=5, splitrule=gini, min.node.size=5
## - : mtry=5, splitrule=gini, min.node.size=5
## + : mtry=6, splitrule=gini, min.node.size=5
## - : mtry=6, splitrule=gini, min.node.size=5
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2, splitrule = gini, min.node.size = 5 on full training set
```

```
confusionMatrix(predict(rf_model,o_test_set),o_test_set$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1458   75
##          1    0    0
##
##                Accuracy : 0.9511
##                  95% CI : (0.9391, 0.9613)
##     No Information Rate : 0.9511
##     P-Value [Acc > NIR] : 0.5307
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9511
##          Neg Pred Value :    NaN
##              Prevalence : 0.9511
##          Detection Rate : 0.9511
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

The method used within the random forest is **Ranger** a fast implementation of random forests or recursive partitioning, particularly suited for high dimensional data.

The same results from the past method is obtained, this means that due to the imbalance in the strokes column our algorithm will always miss the recall. The next step shoulb be balancing the data.

6 Balancing the data

For this we will use the oversample function from the *imbalance* package, which generates data for binary class datasets, so that ML models can perform better in predicting for both positives and negatives.

Data before balancing it:

stroke	n	prop
0	4860	0.95
1	249	0.05

Data after balancing it:

stroke	n	prop
0	4860	0.5
1	4860	0.5

Now that the proportions are even, let us try the ML methods again.

6.1 Training data pt. 2

```
# setting root for repeatability purposes
set.seed(2021)

# creating train and test set
index <-  createDataPartition(post_dataa$stroke, times = 1, p = 0.3, list = F)
train_set <- post_dataa[-index,]
test_set <- post_dataa[index,]
```

```
## Dimensions of the train set:  6804 11
```

```
## Dimensions of the test set:  2916 11
```

7 Methods with balanced data

7.1 GLM Model

```
##################### glm model ####################
set.seed(2021) # setting root for repeatability purposes
glm_model <- train(stroke ~ ., method = 'glm', data = train_set)
confusionMatrix(predict(glm_model, test_set), test_set$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    0    1
##          0 1458   64
##          1    0 1394
##
##                Accuracy : 0.9781
##                  95% CI : (0.9721, 0.9831)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9561
##
##  Mcnemar's Test P-Value : 3.407e-15
##
##             Sensitivity : 1.0000
##             Specificity : 0.9561
##          Pos Pred Value : 0.9580
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5219
##       Balanced Accuracy : 0.9781
##
##        'Positive' Class : 0
##
```

The accuracy went up, and the Specificity is no longer zero, now our model can predict stokes with a 95% accuracy.

7.2 RF Model

```
##################### random forest model ####################
set.seed(2021) # setting root for repeatability purposes

# setting the tune grid
rfGrid <- data.frame(.mtry = c(2,3,5,6),.splitrule = "gini",
                     .min.node.size = 5)

# setting the control parameters
rfControl <- trainControl(
  method = "oob", number = 5,
  verboseIter = TRUE)

rf_model <- train(stroke ~ .,train_set,
                  method = "ranger",tuneLength = 3,
                  tuneGrid = rfGrid,trControl = rfControl)
```

```
## + : mtry=2, splitrule=gini, min.node.size=5
## - : mtry=2, splitrule=gini, min.node.size=5
## + : mtry=3, splitrule=gini, min.node.size=5
## - : mtry=3, splitrule=gini, min.node.size=5
## + : mtry=5, splitrule=gini, min.node.size=5
## - : mtry=5, splitrule=gini, min.node.size=5
```

```
## + : mtry=6, splitrule=gini, min.node.size=5
## - : mtry=6, splitrule=gini, min.node.size=5
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2, splitrule = gini, min.node.size = 5 on full training set
```

```
# testing with balanced test set
confusionMatrix(predict(rf_model,test_set),test_set$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1458   64
##          1    0 1394
##
##                Accuracy : 0.9781
##                  95% CI : (0.9721, 0.9831)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9561
##
##  Mcnemar's Test P-Value : 3.407e-15
##
##             Sensitivity : 1.0000
##             Specificity : 0.9561
##          Pos Pred Value : 0.9580
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5219
##       Balanced Accuracy : 0.9781
##
##        'Positive' Class : 0
##
```

Same outcome for this model, but can it predict when using the whole dataset?

8 Results/Conclusions

The improvement obtained by balancing the data was notorious, but can it preform with the same accuracy when using the complete data?.

Let us check using the random forest model:

```
# testing with original data
confusionMatrix(predict(rf_model, post_dataa) ,post_dataa$stroke)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 4860  249
##          1    0 4611
##
##                Accuracy : 0.9744
##                  95% CI : (0.971, 0.9774)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9488
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.9488
##          Pos Pred Value : 0.9513
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5256
##       Balanced Accuracy : 0.9744
##
##        'Positive' Class : 0
##
```

```
# Accuracy for the random forest with original data
F1_Score(post_dataa$stroke,predict(rf_model,post_dataa))
```

```
## [1] 0.9750226
```

In practical terms we get the same accuracy of about 97%, and a high Sensitivity & Specificity ,

9 Session info

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_Canada.1252  LC_CTYPE=English_Canada.1252
## [3] LC_MONETARY=English_Canada.1252 LC_NUMERIC=C
## [5] LC_TIME=English_Canada.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
```

```
##
## other attached packages:
##  [1] ranger_0.12.1     MLmetrics_1.1.1   caret_6.0-86      lattice_0.20-41
##  [5] skimr_2.1.3       forcats_0.5.0     stringr_1.4.0     dplyr_1.0.4
##  [9] purrr_0.3.4       readr_1.4.0       tidyr_1.1.2       tibble_3.0.4
## [13] ggplot2_3.3.2     tidyverse_1.3.0   naniar_0.6.0      imbalance_1.0.2.1
## [17] gridExtra_2.3
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-149         fs_1.5.0             lubridate_1.7.9.2
##  [4] httr_1.4.2           repr_1.1.3           tools_4.0.3
##  [7] backports_1.2.0      utf8_1.1.4           R6_2.5.0
## [10] rpart_4.1-15         DBI_1.1.1            colorspace_2.0-0
## [13] nnet_7.3-14          withr_2.4.1          tidyselect_1.1.0
## [16] compiler_4.0.3       KernelKnn_1.1.0      cli_2.3.1
## [19] rvest_0.3.6          xml2_1.3.2           labeling_0.4.2
## [22] scales_1.1.1         digest_0.6.27        rmarkdown_2.7
## [25] base64enc_0.1-3      pkgconfig_2.0.3      htmltools_0.5.1.1
## [28] dbplyr_2.1.0         highr_0.8            rlang_0.4.10
## [31] readxl_1.3.1         rstudioapi_0.13      generics_0.1.0
## [34] farver_2.0.3         jsonlite_1.7.2       ModelMetrics_1.2.2.2
## [37] magrittr_2.0.1       Matrix_1.2-18        fansi_0.4.1
## [40] Rcpp_1.0.5           munsell_0.5.0        lifecycle_1.0.0
## [43] visdat_0.5.3         stringi_1.5.3        pROC_1.16.2
## [46] yaml_2.2.1           MASS_7.3-53          plyr_1.8.6
## [49] recipes_0.1.15       grid_4.0.3           crayon_1.4.1
## [52] haven_2.3.1          splines_4.0.3        hms_1.0.0
## [55] knitr_1.31           pillar_1.4.7         reshape2_1.4.4
## [58] codetools_0.2-16     stats4_4.0.3         reprex_1.0.0
## [61] glue_1.4.2           evaluate_0.14        data.table_1.13.4
## [64] modelr_0.1.8         vctrs_0.3.6          foreach_1.5.1
## [67] cellranger_1.1.0     gtable_0.3.0         assertthat_0.2.1
## [70] xfun_0.19            gower_0.2.2          prodlim_2019.11.13
## [73] broom_0.7.4          e1071_1.7-4          class_7.3-17
## [76] survival_3.2-7       timeDate_3043.102    smotefamily_1.3.1
## [79] iterators_1.0.13     lava_1.6.8.1         ellipsis_0.3.1
## [82] ipred_0.9-9
```