Health No Show

darbaroch

31/5/2021

1.1. Executive Summary

This document corresponds to the “Choose your own” Project Submission for the EDX’s PH125.9x course.

The data set corresponds to medical appointment no-show. It comprises 110000+ observations of appointments in a Brazilian region in April, May and June 2016.

It is a classification problem of a dichotomous variable: show or no-show.

Area under the ROC curve is used as the key performance indicator.

Approaches with random forest and knn algorithms are used. Some work on the features is required for the algorithms to perform better.

Ultimate algorithm achieved ROC AUC = 0.77 on the test set.

1.2. Introduction

The data set was downloaded from Kaggle and made available through GitHub. It can be loaded with the following code.

```
library(tidyverse)
library(lubridate)
library(caret)

dl <- tempfile()
download.file("https://github.com/dbaroch/med_app_no_show/raw/main/KaggleV2-May-2016.csv", d
l)

df <- read_delim(file =  dl,
                    delim = ",")

rm(dl)

df
```

```
## # A tibble: 110,527 x 14
##    PatientId AppointmentID Gender ScheduledDay        AppointmentDay        Age
##        <dbl>         <dbl> <chr>  <dttm>              <dttm>              <dbl>
##  1   2.99e13       5642903 F      2016-04-29 18:38:08 2016-04-29 00:00:00    62
##  2   5.59e14       5642503 M      2016-04-29 16:08:27 2016-04-29 00:00:00    56
##  3   4.26e12       5642549 F      2016-04-29 16:19:04 2016-04-29 00:00:00    62
##  4   8.68e11       5642828 F      2016-04-29 17:29:31 2016-04-29 00:00:00     8
##  5   8.84e12       5642494 F      2016-04-29 16:07:23 2016-04-29 00:00:00    56
##  6   9.60e13       5626772 F      2016-04-27 08:36:51 2016-04-29 00:00:00    76
##  7   7.34e14       5630279 F      2016-04-27 15:05:12 2016-04-29 00:00:00    23
##  8   3.45e12       5630575 F      2016-04-27 15:39:58 2016-04-29 00:00:00    39
##  9   5.64e13       5638447 F      2016-04-29 08:02:16 2016-04-29 00:00:00    21
## 10   7.81e13       5629123 F      2016-04-27 12:48:25 2016-04-29 00:00:00    19
## # ... with 110,517 more rows, and 8 more variables: Neighbourhood <chr>,
## #   Scholarship <dbl>, Hipertension <dbl>, Diabetes <dbl>, Alcoholism <dbl>,
## #   Handcap <dbl>, SMS_received <dbl>, `No-show` <chr>
```

The data set has 110527 observations and 14 variables. When browsing the Kaggle's website it can be found that this is version number 5 of the data set and that there have been former versions of it with much more observations and a different no-show rate (about 30%). No versions other than number 5 are longer available for downloading.

```
str(df)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 110527 obs. of  14 variables:
##  $ PatientId     : num  2.99e+13 5.59e+14 4.26e+12 8.68e+11 8.84e+12 ...
##  $ AppointmentID : num  5642903 5642503 5642549 5642828 5642494 ...
##  $ Gender        : chr  "F" "M" "F" "F" ...
##  $ ScheduledDay  : POSIXct, format: "2016-04-29 18:38:08" "2016-04-29 16:08:27" ...
##  $ AppointmentDay: POSIXct, format: "2016-04-29" "2016-04-29" ...
##  $ Age           : num  62 56 62 8 56 76 23 39 21 19 ...
##  $ Neighbourhood : chr  "JARDIM DA PENHA" "JARDIM DA PENHA" "MATA DA PRAIA" "PONTAL DE CAM
BURI" ...
##  $ Scholarship   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hipertension  : num  1 0 0 0 1 1 0 0 0 0 ...
##  $ Diabetes      : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ Alcoholism    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Handcap       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SMS_received  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ No-show       : chr  "No" "No" "No" "No" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   PatientId = col_double(),
##   ..   AppointmentID = col_double(),
##   ..   Gender = col_character(),
##   ..   ScheduledDay = col_datetime(format = ""),
##   ..   AppointmentDay = col_datetime(format = ""),
##   ..   Age = col_double(),
##   ..   Neighbourhood = col_character(),
##   ..   Scholarship = col_double(),
##   ..   Hipertension = col_double(),
##   ..   Diabetes = col_double(),
##   ..   Alcoholism = col_double(),
##   ..   Handcap = col_double(),
##   ..   SMS_received = col_double(),
##   ..   `No-show` = col_character()
##   .. )
```

Scholarship, Hipertension, Diabetes, Alcoholism, SMS_received and No-Show are actually dichotomous variables that would better represented as logical. Gender is also dichotomous and could be assigned the TRUE value for women.

Handcap accounts for the number of handicap each patient has and integer representation would be ok (there have been some discussion in the Kaggle's forum regarding this variable and the SMS_received but the data set creator explained this is how it must be considered).

Age would be better represented as integer.

Patient's and appointments's IDs are represented with double number, but the number itself is meaningless. It only matters that every appointment and any patient has a unique number.

Neighbourhood is represented by character but would better be factor. As stated by the data set creator, the neighbourhood is the place where the health facility related to the appointment is and not where the patient lives.

2.1. Check for missing values and adapt variable classes

It can be seen that there are no missing values in the data set:

```
df %>% complete.cases() %>% all()
```

```
## [1] TRUE
```

Variables' classes are converted and "No-show" variable is re-written as "no_show".

```
df <- df %>%
  mutate(Gender = if_else(Gender == "F", TRUE, FALSE),
         `No-show` = if_else(`No-show` == "Yes", TRUE, FALSE),
         Neighbourhood = as.factor(Neighbourhood),
         Age = as.integer(Age),
         Handcap = as.integer(Handcap),
         Scholarship = as.logical(Scholarship),
         Hipertension = as.logical(Hipertension),
         Diabetes = as.logical(Diabetes),
         Alcoholism = as.logical(Alcoholism),
         SMS_received = as.logical(SMS_received))

names(df)[names(df) == "No-show"] <- "no_show"
```

2.2. Data split into training and final test sets

The data set is split into training and final test sets: 10% of the observations are separated for later use to evaluate the ultimate performance of the algorithm to be developed.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = df$AppointmentID, times = 1, p = 0.1, list = FALSE)
train <- df[-test_index[,1],]
final_test <- df[test_index[,1],]
rm(df,test_index)
```

2.3. Initial data exploration

2.3.1. Appointment Id

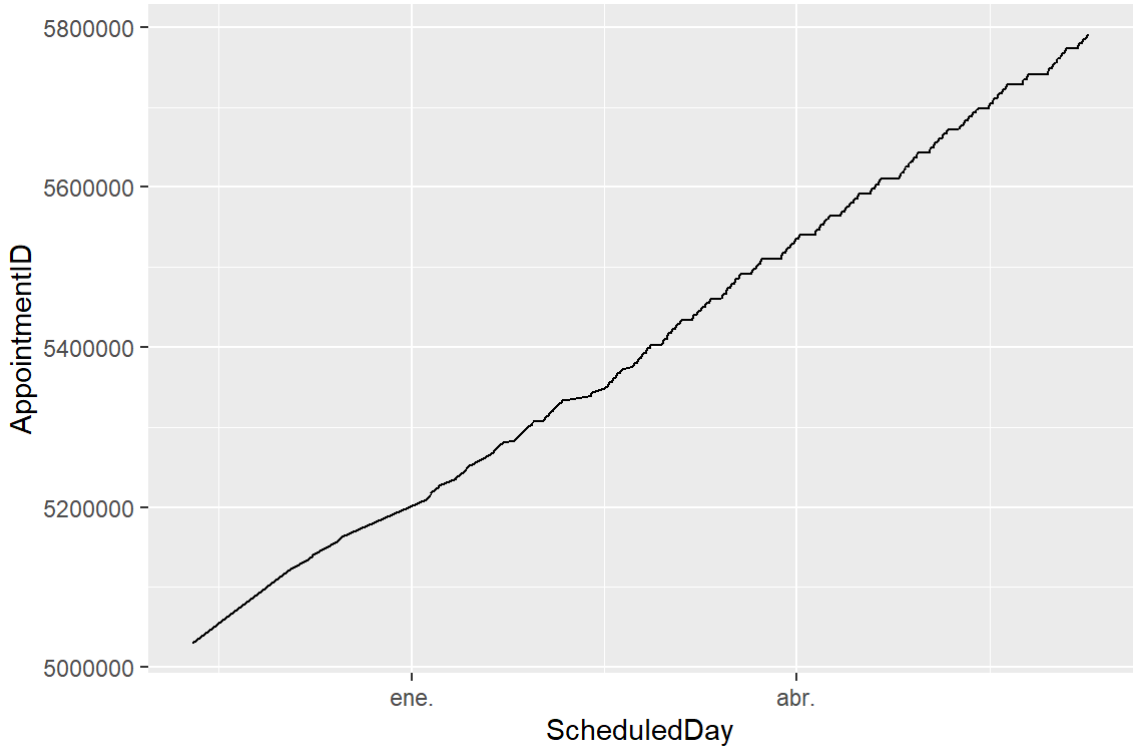
AppointmentId is just the reference for the observations. There are as many different AppointmentIds as rows in the data set.

```
train %>% pull(AppointmentID) %>% unique() %>% length()
```

```
## [1] 99471
```

Therefore, it may seem that appointmentId is not meaningful as a number. If plotted versus scheduledDay it can be seen that appointmentId reflects the moment when the appointment was made:

```
train %>%
  select(ScheduledDay, AppointmentID) %>%
  ggplot(aes(x=ScheduledDay, y = AppointmentID)) + geom_line()
```



It can be seen that correlation between them is almost 1.

```
cor(as.numeric(train$ScheduledDay), train$AppointmentID)
```

```
## [1] 0.9983071
```

2.3.2. Time span

The time span of the appointments is about 1.3 months, from 2016-04-29 to 2016-06-08.

```
train %>% pull(AppointmentDay) %>% unique() %>% min()
```

```
## [1] "2016-04-29 UTC"
```

```
train %>% pull(AppointmentDay) %>% unique() %>% max()
```

```
## [1] "2016-06-08 UTC"
```

2.3.3. No-show rate

The no-show rate in the training set is 20%.

```
no_show_rate <- mean(train$no_show == TRUE)
no_show_rate
```

```
## [1] 0.2022499
```

2.3.4. Patients and number of appointments per patient

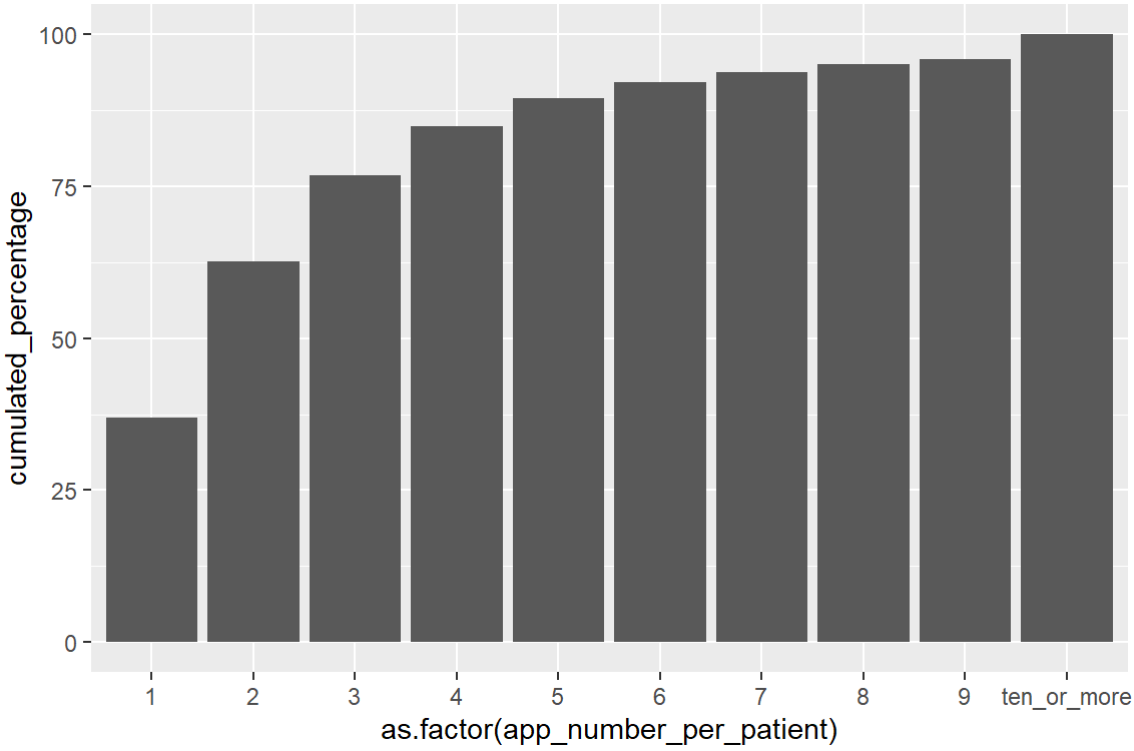
There are 58344 different patients in the training set.

```
train %>% pull(PatientId) %>% unique() %>% length()
```

```
## [1] 58344
```

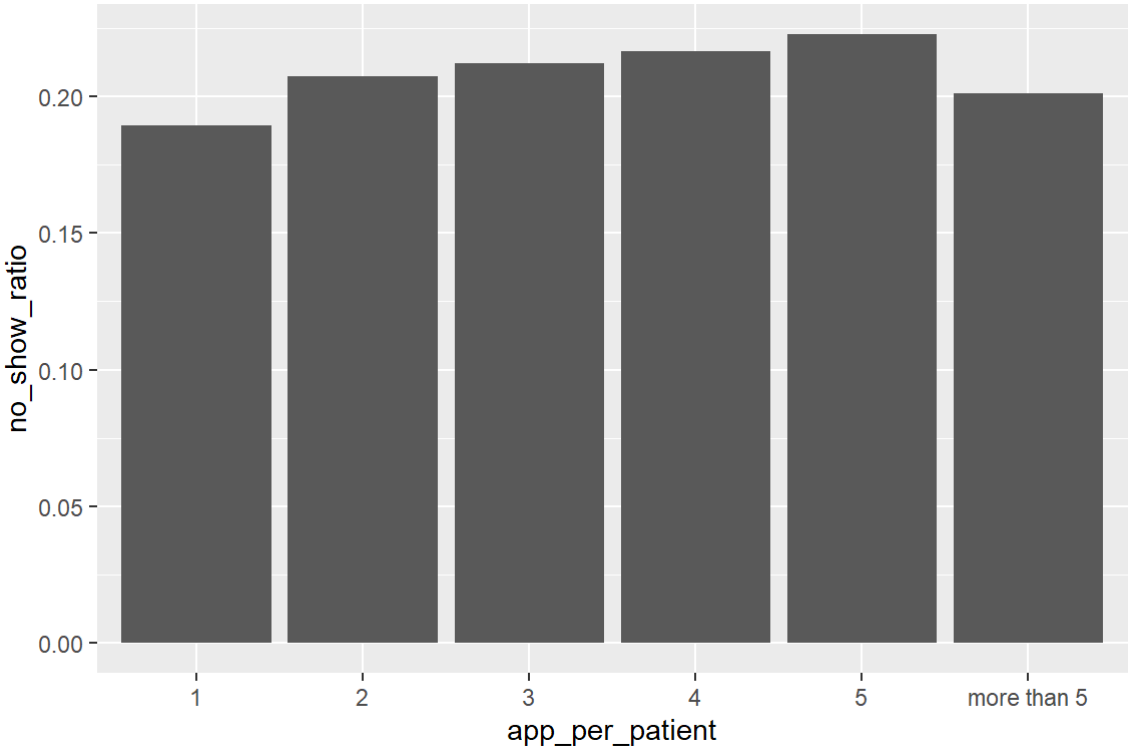
There are some patients for which several observations are available. The vast majority of the observations correspond to patients that have only 1 or just a few appointments. But there are a couple of patients with a significant amount of appointments made.

```
train %>%
  group_by(PatientId) %>%
  summarise(app_number_per_patient = n()) %>%
  group_by(app_number_per_patient) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  mutate(app_number = count * app_number_per_patient,
         app_number_per_patient = if_else(app_number_per_patient > 9, "ten_or_more", as.chara
cter(app_number_per_patient))) %>%
  group_by(app_number_per_patient) %>%
  summarise(app_number = sum(app_number)) %>%
  mutate(cumulated_percentage = 100*cumsum(app_number)/sum(app_number)) %>%
  ggplot(aes(x = as.factor(app_number_per_patient), y = cumulated_percentage)) +
  geom_bar(stat = "identity")
```



As much as 89% of the appointments correspond to patients that made just 5 or less. 75% of the data correspond to patients that made 3 appointments or less. Therefore, it is reasonable to think that characterization of patients on their tendency to absence will not be useful, unless it is considered only for the subset of patients for which data on several appointments is available, for example: patients for which there are 5 or more observations.

```
train %>%
  group_by(PatientId) %>%
  mutate(app_per_patient = n()) %>%
  mutate(app_per_patient = ifelse(app_per_patient > 5, "more than 5", as.character(app_per_pa
tient))) %>%
  ungroup() %>%
  group_by(app_per_patient) %>%
  summarise(no_show_ratio = mean(no_show == TRUE)) %>%
  ggplot(aes(x=app_per_patient, y = no_show_ratio)) + geom_bar(stat = "identity")
```



There seems to be a slightly lower ratio when compared to the global mean for patients with only 1 appointment and a slightly higher ratio for patients with up to 5 appointments.

2.3.5. Variables that characterize patients

Gender, Age, Scholarship, Hypertension, Diabetes, Alcoholism and Handcap are variables that characterize the patient rather than the observation.

Therefore, each PatientId should have only one combination of those variables. The following chunk shows it does not.

```
train %>%
  select(PatientId, Gender, Age, Scholarship, Hipertension, Diabetes, Alcoholism, Handcap) %
>%
  unique() %>%
  nrow()
```

```
## [1] 59389
```

There are 59389 different combinations for only 58344 patients. That is being caused by Age. Some patients got their birthday during the period under analysis. This can be shown with the following chunk that considers all patients with more than one age value and calculates the difference between each age registered and their mean. Its a result is a unique value: 0.5, half a year.

```
train %>%
  select(PatientId, Age) %>%
  unique() %>%
  group_by(PatientId) %>%
  mutate(count = n(),
         age_avg = mean(Age),
         age_diff = abs(Age - age_avg)) %>%
  filter(count > 1) %>%
  ungroup() %>%
  pull(age_diff) %>%
  unique()
```

```
## [1] 0.5
```

So, it is ok to have 2 different age values for some patients if their birthday occurred during the analyzed period.

Once age is not taken into account, there are 58344 combinations for 58344 patients meaning that patients' data is consistent.

```
train %>%
  select(PatientId, Gender, Neighbourhood, Scholarship, Hipertension, Diabetes, Alcoholism, H
andcap) %>%
  unique() %>%
  nrow()
```

```
## [1] 58344
```

2.3.6. Appointments and gender

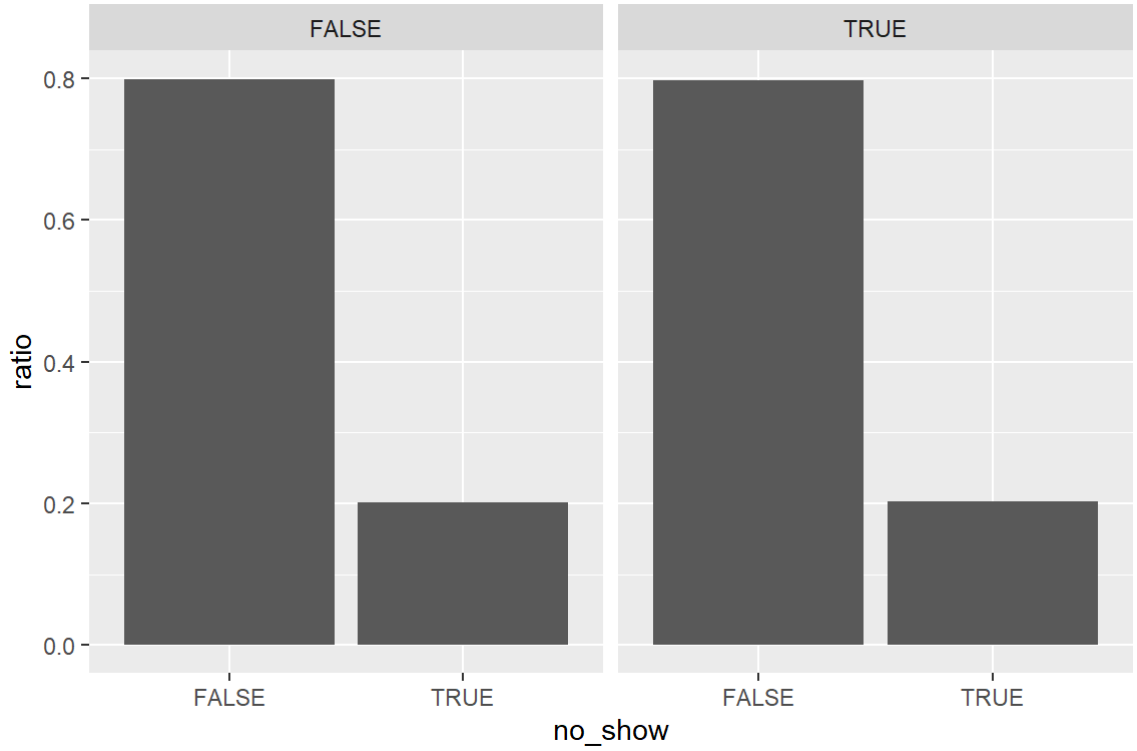
There is almost twice appointments made by women when compared to men.

```
train %>%
  group_by(Gender) %>%
  summarise(count = n())
```

```
## # A tibble: 2 x 2
##   Gender count
##   <lgl>  <int>
## 1 FALSE  34753
## 2 TRUE   64718
```

Respect to no show ratio, it seems that there is no difference regarding gender, at least globally. The following plot compares no-show rate. The left-hand is Gender = FALSE and therefore correspond to men, while the right-hand correspond to women.

```
train %>%
  group_by(Gender, no_show) %>%
  summarise(count = n()) %>%
  ungroup() %>%
  group_by(Gender) %>%
  mutate(ratio = count/sum(count)) %>%
  ggplot(aes(x = no_show, y = ratio)) +
  geom_bar(stat = "identity") +
  facet_grid(. ~ Gender)
```



2.3.7. Appointments and age

The maximum and minimum age are -1 and 115.

```
train %>% pull(Age) %>% max()
```

```
## [1] 115
```

```
train %>% pull(Age) %>% min()
```

```
## [1] -1
```

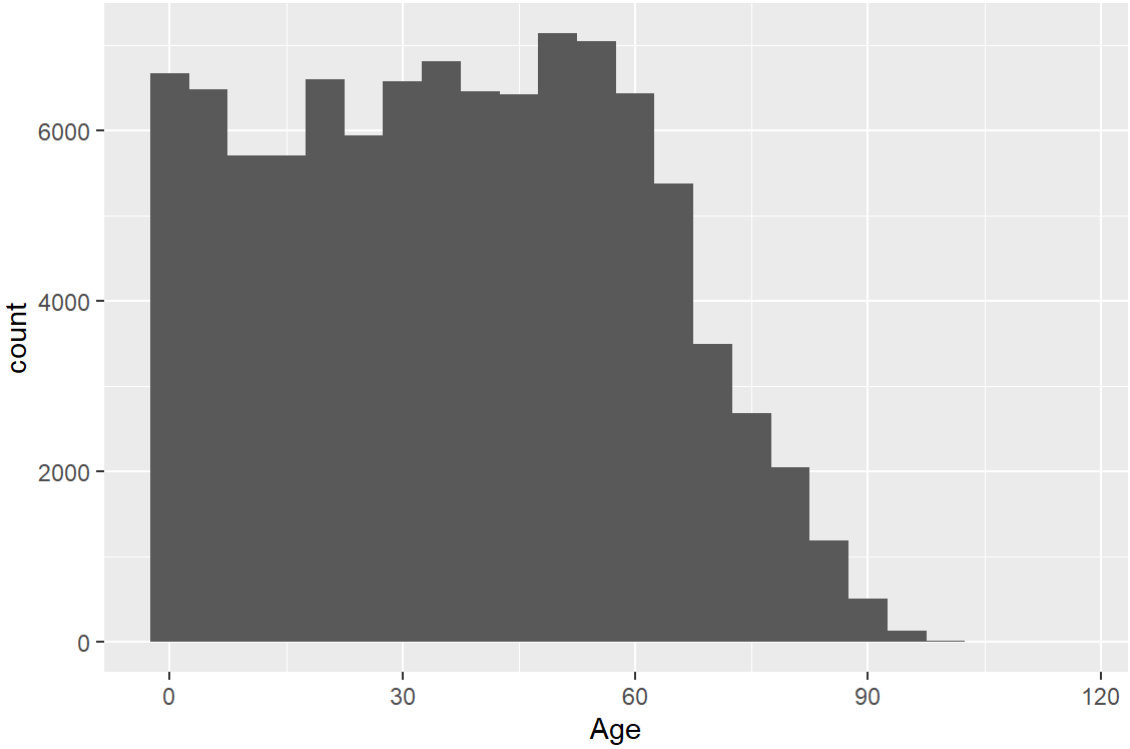
There seems to be a mistake with the minimum (-1) and it will not be taken into account. Although the maximum is quite big (115), it is considered as possible for the purposes of this task.

```
train <- train %>% filter(Age >= 0)
```

It is important to note that if there is a similar error in the test set, the algorithm is not likely to perform well on it as it will not have trained with similar data.

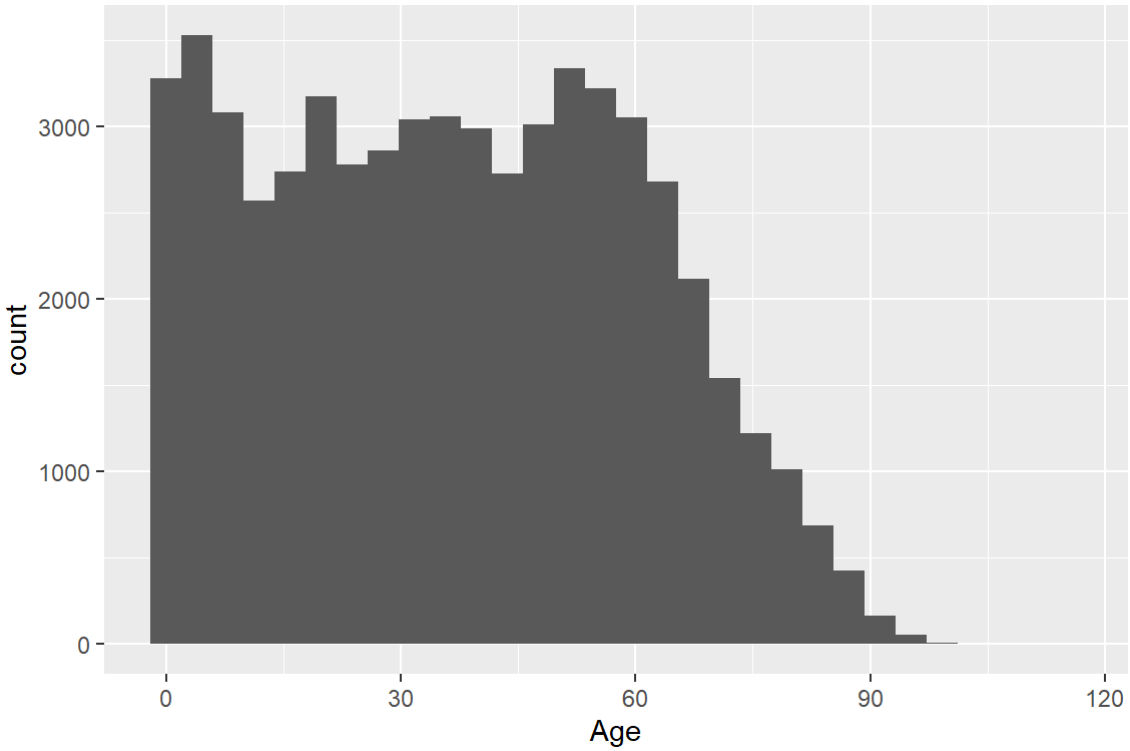
The following plot shows a histogram of the appointments made respect to the patient's age.

```
train %>%
  select(Age) %>%
  ggplot(aes(x=Age)) + geom_histogram(binwidth = 5)
```



It seems that until the age of 60 the number of appointments is about the same, and then is less, probably due to the fact that there are less patients with ages higher than 60.

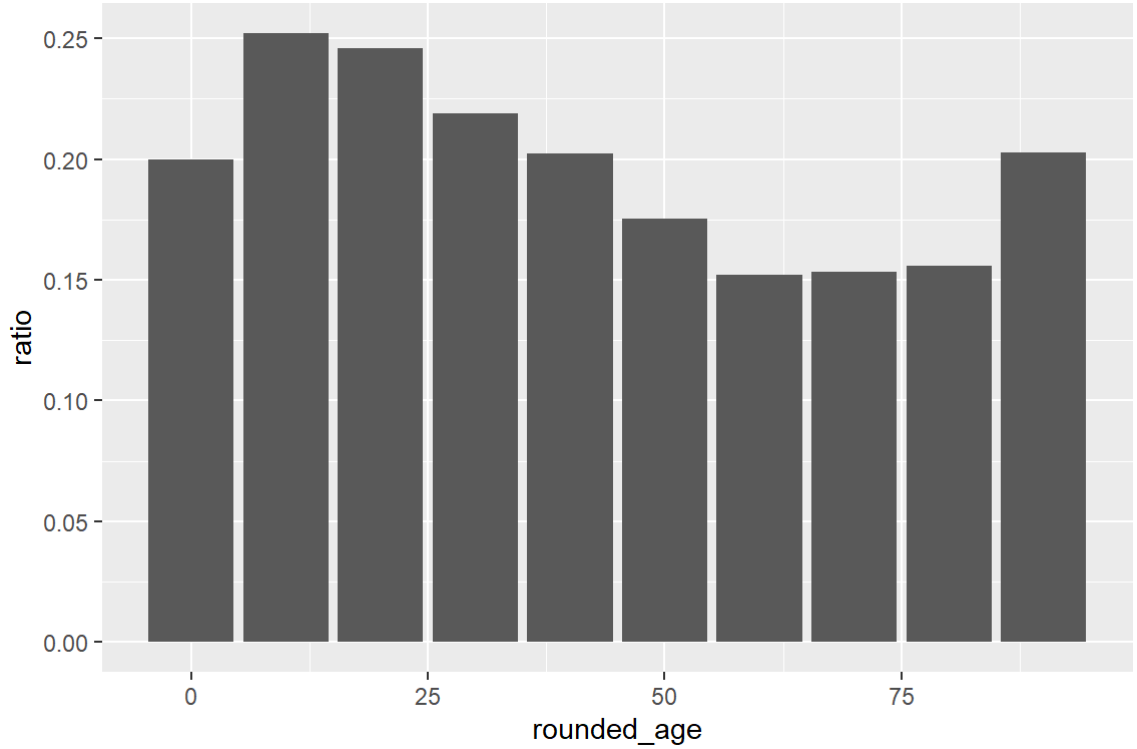
```
train %>%
  select(PatientId, Age) %>%
  group_by(PatientId) %>%
  arrange(-Age) %>%
  slice(1) %>%
  ggplot(aes(x=Age)) + geom_histogram()
```



The patient's age histogram shows that in fact there are less patients aged 60 or more when compared to other ages.

Regarding no show ratio, a 10-year stratification is used to analyze the data. The previous histograms showed that there is a relatively low amount of observations for the higher ages. Therefore, for the purpose of this stratification, the highest category is cap to 90 or more.

```
train %>%
  mutate(rounded_age = 10*floor(Age/10),
         rounded_age = if_else(rounded_age >= 90, 90, rounded_age)) %>%
  filter(rounded_age >= 0) %>%
  group_by(rounded_age) %>%
  summarise(ratio = mean(no_show ==  TRUE)) %>%
  ggplot(aes(x=rounded_age, y = ratio)) + geom_bar(stat = "identity")
```



There seems to be biases related to age. It seems that ages from 10 - 30 tend to have a higher no show rate than the average and that ages from 50 to 80 tend to have a lower no show rate.

2.3.8. Appointments and neighbourhood

There are 81 different neighbourhoods.

```
train %>%
  pull(Neighbourhood) %>%
  unique() %>%
  length()
```

```
## [1] 81
```

There are neighbourhoods with quite a lot of appointments:

```
train %>%
  group_by(Neighbourhood) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  slice(1:5)
```

```
## # A tibble: 5 x 2
##   Neighbourhood   count
##   <fct>           <int>
## 1 JARDIM CAMBURI   6937
## 2 MARIA ORTIZ      5228
## 3 RESISTÊNCIA      3948
## 4 JARDIM DA PENHA  3527
## 5 ITARARÉ          3133
```

And neighbourhoods with just a few appointments:

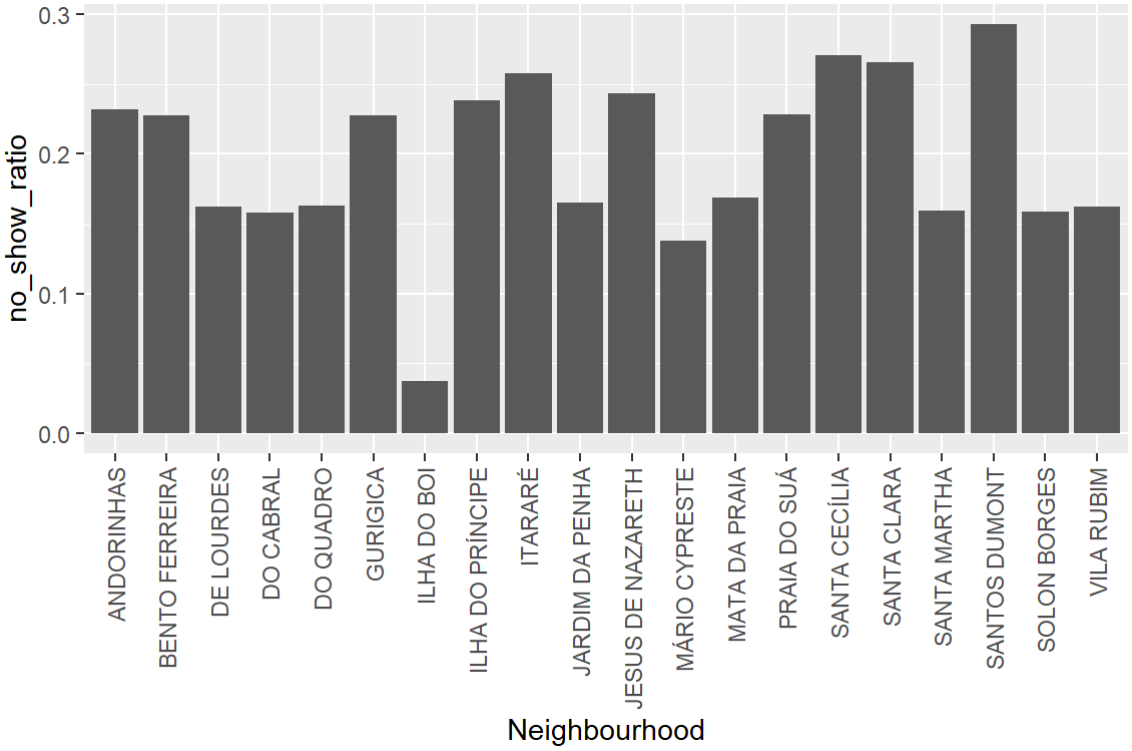
```
train %>%
  group_by(Neighbourhood) %>%
  summarise(count = n()) %>%
  arrange(count) %>%
  slice(1:5)
```

```
## # A tibble: 5 x 2
##   Neighbourhood               count
##   <fct>                       <int>
## 1 PARQUE INDUSTRIAL               1
## 2 ILHAS OCEÂNICAS DE TRINDADE     2
## 3 AEROPORTO                       7
## 4 ILHA DO FRADE                  10
## 5 ILHA DO BOI                    27
```

“Parque industrial” has only 1 appointment. It seems reasonable to doubt whether the neighbourhood is actually related to the place where the health facility is and not where the patient lives. Nevertheless, the neighbourhood informtation will not be used with any of those 2 interpretations.

Neighbourhoods can differ quite enough in their no-show ratio.

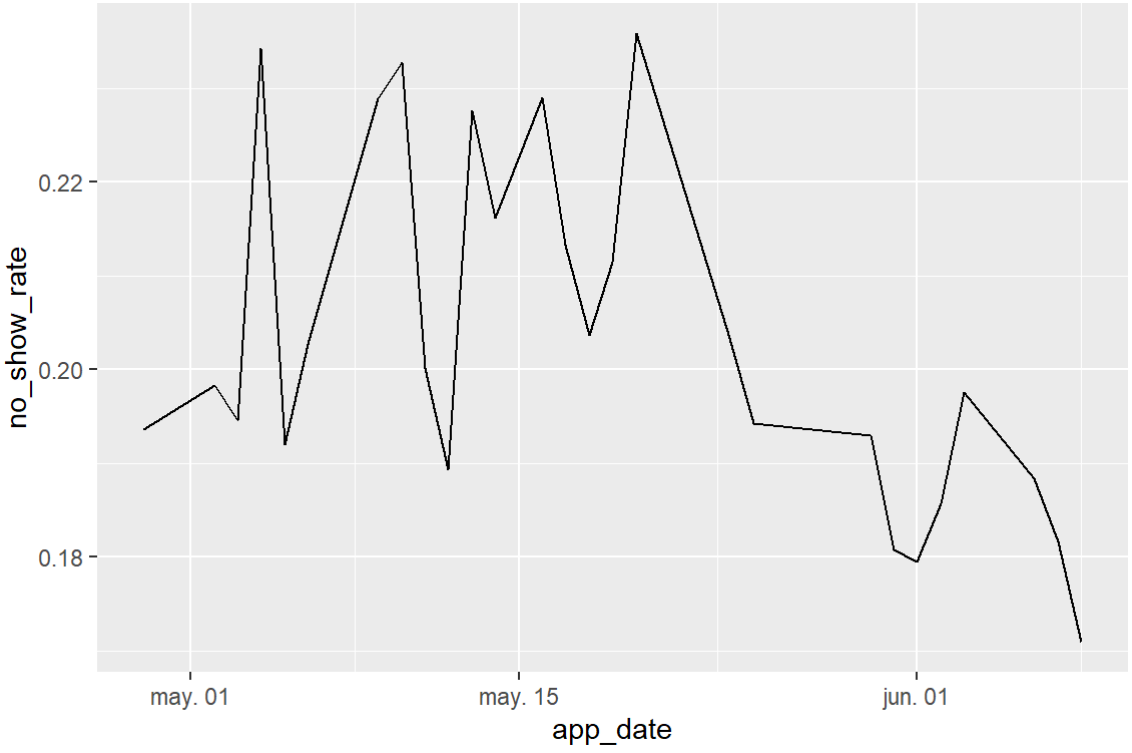
```
train %>%
  group_by(Neighbourhood) %>%
  summarise(no_show_ratio = mean(no_show == TRUE),
            count = n()) %>%
  filter(count >= 20) %>%
  arrange(-no_show_ratio) %>%
  slice(-(11:(nrow(.)-10))) %>%
  ggplot(aes(x=Neighbourhood, y = no_show_ratio)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



2.3.9. No-show rate along time span

No-show rate seems to variate along time, having lower values by the end of May and June:

```
train %>%
  group_by(app_date = date(AppointmentDay)) %>%
  summarise(no_show_rate = mean(no_show == TRUE)) %>%
  ggplot(aes(x=app_date, y =  no_show_rate)) + geom_line()
```



2.3.10. Time between schedulling and the appointment

The following code presents the no-show rate for the different values of days between scheduling and the appointment.

```
train %>%
  mutate(days_between = date(AppointmentDay) - date(ScheduledDay)) %>%
  group_by(days_between) %>%
  summarise(ratio = mean(no_show == TRUE))
```

```
## # A tibble: 130 x 2
##    days_between  ratio
##    <drtn>        <dbl>
##  1 -6 days      1
##  2 -1 days      1
##  3  0 days      0.0467
##  4  1 days      0.215
##  5  2 days      0.240
##  6  3 days      0.239
##  7  4 days      0.230
##  8  5 days      0.266
##  9  6 days      0.246
## 10  7 days      0.267
## # ... with 120 more rows
```

There are some observations with days_between negative values, -6 and -1. They correspond to 100% no-show. AppointmentDay or ScheduledDay must be wrong for those observations.

2.3.11. Number of appointments that day

There are some patients with several appointments on a single date.

```
train %>%
  group_by(PatientId, AppointmentDay) %>%
  mutate(nbr_app_that_day = n()) %>%
  ungroup() %>%
  group_by(nbr_app_that_day) %>%
  summarise(bias = mean(no_show == TRUE) - no_show_rate,
            count = n()) %>%
  mutate(count = count/nbr_app_that_day)
```

```
## # A tibble: 8 x 3
##   nbr_app_that_day     bias count
##              <int>    <dbl> <dbl>
## 1                1 -0.00282 86140
## 2                2  0.0210   5468
## 3                3  0.0219    583
## 4                4 -0.0333    111
## 5                5 -0.0393     27
## 6                6 -0.140       8
## 7                8  0.548       1
## 8               10 -0.202       1
```

Biases for 5 or less appointments in a day are very little, no useful information seems to be found there. Relevant information is only available for more than 5 appointments in a single date which are quite rare cases.

2.3.12. Weekday influence

The no-show rate can be calculated for each week day separately:

```
train %>%
  mutate(weekday = wday(AppointmentDay)) %>%
  group_by(weekday) %>%
  summarise(no_show_mean = mean(no_show == TRUE))
```

```
## # A tibble: 6 x 2
##   weekday no_show_mean
##     <dbl>        <dbl>
## 1       2        0.208
## 2       3        0.201
## 3       4        0.197
## 4       5        0.195
## 5       6        0.212
## 6       7        0.216
```

There seems to be no big difference, at least globally.

2.3.13. SMS and days between

No patient with less than 3 days between scheduling and appointment was sent an SMS. There seems to be a rule: SMS are only sent if the appointment occurs 3 days or more from scheduling.

If only appointments with 3 or more days between scheduling and appointment are considered it seems that sending SMS renders the no-show rate a bit lower:

```
train %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay)),
         SMS_received = factor(if_else(days_between < 3, "does_not_apply",
                                       if_else(SMS_received == TRUE, "yes", "no")))) %>%
  select(days_between, SMS_received, no_show) %>%
  group_by(SMS_received) %>%
  summarise(count = n(),
            no_show_rate = mean(no_show ==  TRUE))
```

```
## # A tibble: 3 x 3
##   SMS_received   count no_show_rate
##   <fct>          <int>        <dbl>
## 1 does_not_apply 45467       0.0902
## 2 no             22103       0.327
## 3 yes            31900       0.276
```

2.4. Questions and approaches

1. Having a 20% no-show rate accuracy does not seem to be a good performance indicator. Which indicator should be considered?
2. There seems that relevant information (medical specialty, weather, other) is missing. How good can an algorithm lacking this information perform? Spoiler alert: no performance target is available.
3. Classification algorithms shall be used. Random forest and KNN can be considered.

4. How should imbalance be dealt with?
5. Are the features suitable for being used just the way they are or do they need to be transformed?

2.5. Metrics and target discussion.

Accuracy is not a good metric to evaluate a classification system for this problem. A naïve approach predicting “show” everytime would achieve about 80% accuracy because of prevalence.

Sensitivity, specificity, F1-score would depend on a predefined cutoff used to weight how important it is to predict accurately on one of the classes. This is an arbitrary choice that should be evaluated regarding the use for which the system’s output is meant. This information is unavailable.

Therefore the area under the ROC curve (“ROC AUC” from now on) is chosen as the metric to evaluate performance. It does not depend on a predefined cutoff and it is robust to class imbalance.

An interesting discussion regarding the use of ROC Vs. Precision-Recall can be found in this thread: https://stats.stackexchange.com/questions/7207/roc-vs-precision-and-recall-curves (https://stats.stackexchange.com/questions/7207/roc-vs-precision-and-recall-curves)

For the sake of this task, and not having strong background on the health system, the ROC AUC is chosen.

A ROC AUC target is also unavailable, no reference is available in the data set publication. For some systems human performance can be regarded as a reference but it does not apply here. 80% accuracy could be thought of a reference to build upon, but as discussed before it is not a good one, because a system with less accuracy (for a predefined cutoff) but better ROC AUC might be preferred. Therefore, just for illustration purposes a logistic regression will be modeled to determine a baseline.

2.6. Test set for development purposes

For developing and testing models, the training set is split in “train” and “dev” set, as the final_test shall only be used to evaluate the ultimate model’s performance.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = train$AppointmentID, times = 1, p = 0.1, list = FALSE)
dev_set <- train[test_index[,1],]
train <- train[-test_index[,1],]

rm(test_index)
```

2.7. Model

2.7.1. Logistic regression

AppointmentId and Neighbouuhood variables are removed because data exploration showed that AppointmentId is almost the same as ScheduledDay and the model is unable to handle the neighbourhood variable because it has 81 classes.

```
library(PRROC)
library(AUC)

train_for_model <- train %>% select(-AppointmentID, -Neighbourhood)
test_for_model <- dev_set %>% select(-AppointmentID, -Neighbourhood)

train_glm <- train(as.factor(no_show) ~ ., method = "glm", data = train_for_model)

test_preds <- predict(train_glm, newdata = test_for_model, type= "prob")

roc_test_glm <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test_glm)
```

```
## [1] 0.6556534
```

0.66 is achieved for the ROC AUC in the test set.

2.7.2. Random forest with almost no feature manipulation

Just as for logistic regression, a first random forest model is tried removing AppointmentID and Neighbourhood.

```
library(randomForest)

train_for_model <- train %>% select(-Neighbourhood, -AppointmentID)
test_for_model <- dev_set %>% select(-Neighbourhood, -AppointmentID)

rf_model_1 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_1$votes[,2],factor(1 * (rf_model_1$y == TRUE)))
auc(roc)
```

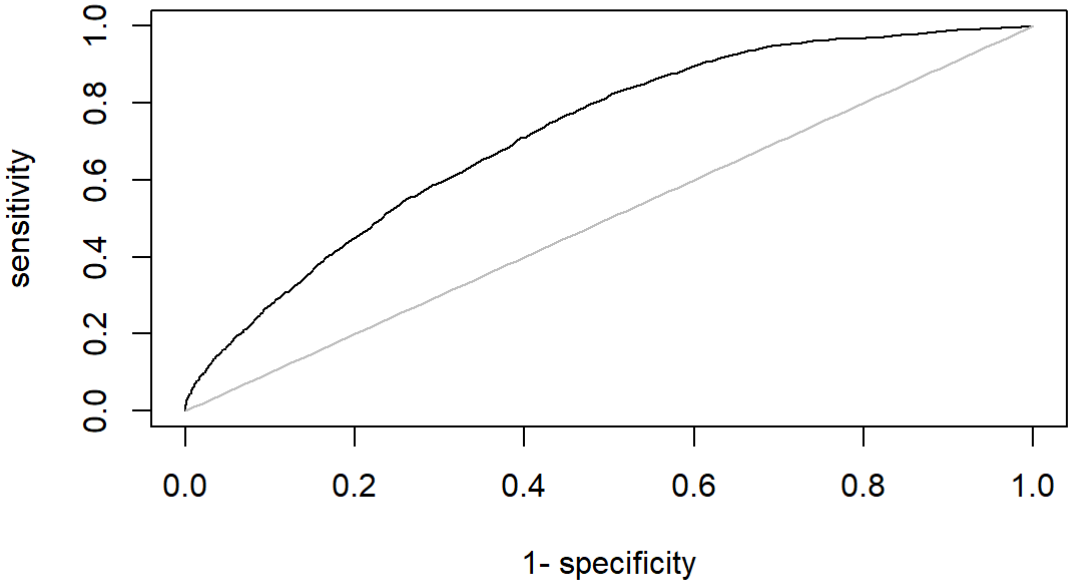
```
## [1] 0.7177507
```

```
test_preds <- predict(rf_model_1, newdata = test_for_model, type= "prob")

roc_test_1 <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test_1)
```

```
## [1] 0.7193481
```

Similar values were achieved for training and dev sets. It may be considered that the algorithm lacks complexity in order to fit the data better.

```
plot(roc_test_1)
```



The ROC plot shows that sensitivity is very hard to achieve: it comes with lots of false positives. Specificity is also very hard.

```
rf_model_1$importance
```

```
##                MeanDecreaseGini
## PatientId              3501.1982
## Gender                  303.2020
## ScheduledDay           5097.1117
## AppointmentDay         1896.1669
## Age                    2597.7508
## Scholarship             163.3545
## Hipertension            160.9153
## Diabetes                141.7629
## Alcoholism              128.7064
## Handcap                 124.0412
## SMS_received            444.4168
```

2.7.3. Random forest without PatientId variable.

PatientId is meaningless as a number, as long as different numbers are assigned to different patients but it is the second most important variable in the first model. Another approach is considered with the same approach but removing PatientId.

```
train_for_model <- train %>% select(-Neighbourhood, -AppointmentID, -PatientId)
test_for_model <- dev_set %>% select(-Neighbourhood, -AppointmentID, -PatientId)

rf_model_2 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_2$votes[,2],factor(1 * (rf_model_2$y == TRUE)))
auc(roc)
```

```
## [1] 0.7089233
```

```
test_preds <- predict(rf_model_2, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7113429
```

Performance is worse, it seems that relevant information is being lost.

2.7.4. Random forest with bias per patient

A different approach is considered by converting the PatientID variable. Instead of using it just the way it is, a no-show rate bias per patient is calculated for patients that have at list a couple of appointments (to avoid calculating biases for patients for which only 1 observation is available).

5 appointments per patient is used here to calculate a bias. This is a parameter to be tuned.

```
no_show_rate <- mean(train$no_show == TRUE)

bias_per_patient <- train %>%
  group_by(PatientId) %>%
  mutate(count = n()) %>%
  filter(count > 4) %>%
  summarise(bias_per_patient = mean(no_show == TRUE) - no_show_rate)
```

```
train_for_model <- train %>%
  left_join(bias_per_patient) %>%
  mutate(bias_per_patient = if_else(is.na(bias_per_patient), 0, bias_per_patient)) %>%
  select(-Neighbourhood, -AppointmentID, -PatientId)

test_for_model <- dev_set %>%
  left_join(bias_per_patient) %>%
  mutate(bias_per_patient = if_else(is.na(bias_per_patient), 0, bias_per_patient)) %>%
  select(-Neighbourhood, -AppointmentID, -PatientId)

rf_model_3 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_3$votes[,2],factor(1 * (rf_model_3$y == TRUE)))
auc(roc)
```

```
## [1] 0.7335459
```

```
test_preds <- predict(rf_model_3, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7152872
```

It performs better when compared to the case where PatientId is just removed but it is worse than the first model. The information lost because of PatientId removal was not fully retrieved by the new bias per patient feature. Now there also seems to be some overtraining, performance is lower in the test set. Not removing the PatientId could also be tried as Random Forest itself can withstand feature correlation.

2.7.5. Random forest with both bias per patient and PatientId

```
train_for_model <- train %>%
  left_join(bias_per_patient) %>%
  mutate(bias_per_patient = if_else(is.na(bias_per_patient), 0, bias_per_patient)) %>%
  select(-Neighbourhood, -AppointmentID)

test_for_model <- dev_set %>%
  left_join(bias_per_patient) %>%
  mutate(bias_per_patient = if_else(is.na(bias_per_patient), 0, bias_per_patient)) %>%
  select(-Neighbourhood, -AppointmentID)

rf_model_4 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_4$votes[,2],factor(1 * (rf_model_4$y == TRUE)))
auc(roc)
```

```
## [1] 0.7406225
```

```
test_preds <- predict(rf_model_4, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7191954
```

Performance increased but it is still worse than the first approach. Therefore, PatientId is kept will the bias per patient is discarded. The exploratory analysis had shown that although patient's no-show previous record might be important, the vast majority of observations correspond to patients with only 1 or very few appointments, so previous records is very difficult if not impossible to be considered for this data set.

2.7.6. Random forest with bias per neighbourhood

A bias approach may be tried for the neighbourhood: instead of using a categorical variables with 81 classes, it can be turned into a numerical bias per neighbourhood variable.

15 appointments per neighbourhood is a threshold being used to decide whether a bias is going to be calculated for that neighbourhood. This is a parameter to be tuned.

```
no_show_rate <- mean(train$no_show == TRUE)

bias_per_nbh <- train %>%
  group_by(Neighbourhood) %>%
  mutate(count = n()) %>%
  filter(count > 15) %>%
  summarise(bias_per_nbh = mean(no_show == TRUE) - no_show_rate)
```

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID)

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID)

rf_model_5 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_5$votes[,2],factor(1 * (rf_model_5$y == TRUE)))
auc(roc)
```

```
## [1] 0.733763
```

```
test_preds <- predict(rf_model_5, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7290421
```

Performance improved.

```
rf_model_5$importance
```

```
##                MeanDecreaseGini
## PatientId              3975.3181
## Gender                  416.7603
## ScheduledDay           5614.5873
## AppointmentDay         2360.7642
## Age                    3143.8368
## Scholarship             222.1505
## Hipertension            216.9707
## Diabetes                169.8150
## Alcoholism              143.8400
## Handcap                 133.8628
## SMS_received            467.9749
## bias_per_nbh           2721.0081
```

2.7.7. Random forest with days between scheduling and appointment

The exploratory analysis showed that the days between scheduling and the appointment seemed to be important. Although correlated to ScheduledDay and AppointmentDay, adding a days between variable may help the algorithm to do better.

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
))

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
))

rf_model_6 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_6$votes[,2],factor(1 * (rf_model_6$y == TRUE)))
auc(roc)
```

```
## [1] 0.7487357
```

```
test_preds <- predict(rf_model_6, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7476196
```

Performance improved.

```
rf_model_6$importance
```

```
##                MeanDecreaseGini
## PatientId              3889.6780
## Gender                  447.3966
## ScheduledDay           4465.7386
## AppointmentDay         1881.9904
## Age                    3193.6434
## Scholarship             224.0588
## Hipertension            219.2916
## Diabetes                165.8777
## Alcoholism              139.6537
## Handcap                 128.6447
## SMS_received            350.1536
## bias_per_nbh           2740.7049
## days_between           3743.9534
```

2.7.8. Random forest with a modified SMS_received variable

The exploratory analysis also showed that SMS were only sent to patients with 3 or more days between scheduling and the appointment. So there are a lot of patients with SMS_received = “No” that were actually not eligible. Reconverting the SMS_received variable into a 3 category (“Yes” / “No” / “does not apply”) may help

the algorithm to do better.

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
),
         SMS_received = factor(if_else(days_between < 3, "does_not_apply",
                                       if_else(SMS_received == TRUE, "yes", "no"))))

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
),
         SMS_received = factor(if_else(days_between < 3, "does_not_apply",
                                       if_else(SMS_received == TRUE, "yes", "no"))))

rf_model_7 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_7$votes[,2],factor(1 * (rf_model_7$y == TRUE)))
auc(roc)
```

```
## [1] 0.7483197
```

```
test_preds <- predict(rf_model_7, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.748047
```

```
rf_model_7$importance
```

```
##                MeanDecreaseGini
## PatientId              3946.2154
## Gender                  458.3282
## ScheduledDay           4315.8968
## AppointmentDay         1868.0327
## Age                    3243.2861
## Scholarship             226.6438
## Hipertension            216.7881
## Diabetes                170.8132
## Alcoholism              144.6533
## Handcap                 128.8187
## SMS_received           1077.5823
## bias_per_nbh           2776.1633
## days_between           3230.6152
```

Performance increased very little. It can also be seen that the SMS_received variable increased its relative importance mostly at the expense of days_between: it seems that the change in the SMS_received variable is not adding relevant information that was already considered in the days between variable. Therefore, changing the SMS_received variable is discarded.

2.7.9. Random forest with weekdays and hour

In order to add some more complexity the weekday for the AppoinmentDay and ScheduledDay and the hour for the ScheduledDay (not available for AppointmentDay) can be included as features.

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay))

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay))

rf_model_8 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_8$votes[,2],factor(1 * (rf_model_8$y == TRUE)))
auc(roc)
```

```
## [1] 0.7598046
```

```
test_preds <- predict(rf_model_8, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7546434
```

There is some improvement.

2.7.10. Random forest incorporating number of appointment that day and per patient

For this model the number for appointment each patient has that day and the total number of appointments the patiend had in the period are calculated:

```
nbr_app_that_day <- train %>%
          group_by(PatientId, AppointmentDay) %>%
          summarise(nbr_app_that_day = n())

nbr_app_per_patient <- train %>%
          group_by(PatientId) %>%
          summarise(nbr_app_per_patient = n())
```

This information is then incorporated in the model.

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay)) %>%
  left_join(nbr_app_per_patient) %>%
  left_join(nbr_app_that_day)

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay))  %>%
  left_join(nbr_app_per_patient) %>%
  left_join(nbr_app_that_day) %>%
  mutate(nbr_app_that_day = if_else(is.na(nbr_app_that_day), 0, as.double(nbr_app_that_day)),
         nbr_app_per_patient = if_else(is.na(nbr_app_per_patient), 0, as.double(nbr_app_per_p
atient)))

rf_model_9 <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500)
```

```
roc <- roc(rf_model_9$votes[,2],factor(1 * (rf_model_9$y == TRUE)))
auc(roc)
```

```
## [1] 0.7661951
```

```
test_preds <- predict(rf_model_9, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7625664
```

Some improvement is achieved, but it is not quite big. A large proportion of the data set correspond to patients with few appointments and only 1 appointment per day. So the new features may be relevant for only a tiny fraction of the data.

```
rf_model_9$importance
```

```
##                     MeanDecreaseGini
## PatientId                   3867.7432
## Gender                       545.1972
## ScheduledDay                4324.7555
## AppointmentDay              1679.7530
## Age                         3368.2022
## Scholarship                  282.5019
## Hipertension                 253.3792
## Diabetes                     183.8547
## Alcoholism                   147.7542
## Handcap                      116.6400
## SMS_received                 376.3551
## bias_per_nbh                2896.1143
## days_between                3639.0029
## wday_sched                  1069.2975
## wday_app                     996.5181
## hour_sched                  1958.2080
## nbr_app_per_patient         1550.9817
## nbr_app_that_day             372.2198
```

2.7.11. Downsampling approach for the FALSE class

In this problem class imbalance is found, but it is not quite big: 80/20. Therefore, a downsampling approach is a priori not very helpful, but it was tried anyway.

The following code shows an example. For the downsampling on the FALSE class to be conducted the “strata” and “sampsize” arguments are used. In order to compensate for the smaller size of the samples, more trees are grown using the “ntree” argument.

The following thread is quite useful to understand the approach:
https://stats.stackexchange.com/questions/157714/r-package-for-weighted-random-forest-classwt-option (https://stats.stackexchange.com/questions/157714/r-package-for-weighted-random-forest-classwt-option)

```
train_for_model <- train %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay)) %>%
  left_join(nbr_app_per_patient) %>%
  left_join(nbr_app_that_day)

test_for_model <- dev_set %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay))  %>%
  left_join(nbr_app_per_patient) %>%
  left_join(nbr_app_that_day) %>%
  mutate(nbr_app_that_day = if_else(is.na(nbr_app_that_day), 0, as.double(nbr_app_that_day)),
         nbr_app_per_patient = if_else(is.na(nbr_app_per_patient), 0, as.double(nbr_app_per_p
atient)))

rf_model_10 <- randomForest(as.factor(no_show) ~ ., data = train_for_model,
                            strata = train_for_model$no_show,
                            sampsize = c(500,500),
                            ntree = 5000)
```

```
roc <- roc(rf_model_10$votes[,2],factor(1 * (rf_model_10$y == TRUE)))
auc(roc)
```

```
## [1] 0.7408099
```

```
test_preds <- predict(rf_model_10, newdata = test_for_model, type= "prob")

roc_test <- roc(test_preds[,2], factor(1 * (test_for_model$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7309237
```

Several sample sizes were tried but no performance improvement was found.

2.7.12. Random forest. Final parameter tuning.

There are 2 parameters to be tuned: the number of appointments per neighbourhood used to compute or not a neighbourhood bias, and the algorithm's parameter "mtry".

Mtry is the number of features randomly selected to be considerered for a new partition when each tree is grown. By default its number is equal to sqrt(number of features) for a classification problem. As 19 features are being considered in the ultimate random forest model mtry is 4. So 2, 4 and 6 will be tried.

A variable “lambda_nbr” will be used for the number of appointments per neighbourhood. Values 50, 100 and 300 will be tried.

10 fold k-fold cross validation will be used.

Feature manipulation is performed within a function “covert_df”. The function includes the “if_else” commands that apply for the test set that convert “NA” to zero.

For each fold, the no-show variable of the test set (“dev set”) is not used anywhere until ROC AUC is calculated. The bias per neighbourhood is calculated using the training set ONLY.

```
convert_df <- function(df, bias_per_nbh, nbr_app_per_patient, nbr_app_that_day){
  df %>%
  left_join(bias_per_nbh) %>%
  mutate(bias_per_nbh = if_else(is.na(bias_per_nbh), 0, bias_per_nbh)) %>%
  select(-Neighbourhood, -AppointmentID) %>%
  mutate(days_between = as.integer(date(AppointmentDay) - date(ScheduledDay), units = "days"
)) %>%
  mutate(wday_sched = wday(ScheduledDay),
         wday_app = wday(AppointmentDay),
         hour_sched = hour(ScheduledDay))  %>%
  left_join(nbr_app_per_patient) %>%
  left_join(nbr_app_that_day) %>%
  mutate(nbr_app_that_day = if_else(is.na(nbr_app_that_day), 0, as.double(nbr_app_that_day)),
         nbr_app_per_patient = if_else(is.na(nbr_app_per_patient), 0, as.double(nbr_app_per_p
atient)))
}
```

```
entire_train <- train %>% bind_rows(dev_set)

set.seed(1, sample.kind="Rounding")
indexes <- sample(length(train$no_show))

folds_nbr <- 2 #change to 10 to run the 10-fold cross validation

params <- expand.grid(tune_mtry = c(2,4,6), # 2,3,4,5,6)
                lambda_nbh = c(50, 100, 300)) #50, 100, 300, 500, 1000, 1500, 5000 ,10000))

rf_1_fold <- function(df, indexes, k, folds, params){
  valid_index <- indexes[(1+length(indexes)/folds*(k-1)):(length(indexes)/folds*k)]
  train_set <- df[-valid_index,]
  valid_set <- df[valid_index,]

  nbr_app_per_patient <- train_set %>%
          group_by(PatientId) %>%
          summarise(nbr_app_per_patient = n())

  nbr_app_that_day <- train_set %>%
          group_by(PatientId, AppointmentDay) %>%
          summarise(nbr_app_that_day = n())


  lapply(1:length(params$tune_mtry), function(i){

    params <- params[i,]

    no_show_rate <- mean(train_set$no_show == TRUE)

    bias_per_nbh <- train_set %>%
      group_by(Neighbourhood) %>%
      mutate(count = n()) %>%
      filter(count > params$lambda_nbh) %>%
      summarise(bias_per_nbh = mean(no_show == TRUE) - no_show_rate)


    train_para_rf <- convert_df(train_set, bias_per_nbh, nbr_app_per_patient, nbr_app_that_da
y)

    rf_model <- randomForest(as.factor(no_show) ~ ., data = train_para_rf, ntree = 500, mtry
 = params$tune_mtry)

    test_para_rf <- convert_df(valid_set, bias_per_nbh, nbr_app_per_patient, nbr_app_that_da
y)

    test_preds <- predict(rf_model, newdata = test_para_rf, type= "prob")

    params %>%
      mutate(k = k,
             auc_roc = auc(roc(test_preds[,2], factor(1 * (test_para_rf$no_show==TRUE)))))

  }) %>%
    bind_rows()

}
```

```
results <- lapply(1:folds_nbr, rf_1_fold, df = entire_train, indexes = indexes, folds = 10, p
arams = params)

results %>%
  bind_rows() %>%
  group_by(tune_mtry, lambda_nbh) %>%
  summarise(auc_roc = mean(auc_roc))
```

```
## # A tibble: 9 x 3
## # Groups:   tune_mtry [3]
##   tune_mtry lambda_nbh auc_roc
##       <dbl>      <dbl>   <dbl>
## 1         2         50   0.759
## 2         2        100   0.760
## 3         2        300   0.759
## 4         4         50   0.770
## 5         4        100   0.771
## 6         4        300   0.770
## 7         6         50   0.770
## 8         6        100   0.769
## 9         6        300   0.769
```

Top performance is achieved for:

```
results %>%
  bind_rows() %>%
  group_by(tune_mtry, lambda_nbh) %>%
  summarise(auc_roc = mean(auc_roc)) %>%
  ungroup() %>%
  arrange(-auc_roc) %>%
  slice(1)
```

```
## # A tibble: 1 x 3
##   tune_mtry lambda_nbh auc_roc
##       <dbl>      <dbl>   <dbl>
## 1         4        100   0.771
```

2.7.13. KNN model

Experience gained during RF9 development can be used for KNN but feature selection should also be considered as feature correlation (for which random forest is better prepared to withstand) may negatively impact performance.

If KNN is trained using the same feature manipulation as per RF9 performance is very low, about 0.53:

```
train_for_knn <- train_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No"))

test_for_knn <- test_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No"))

fitControl <- trainControl(method = "cv",
                           number = 10,
                           # repeats = 10, # uncomment for repeatedcv
                           ## Estimate class probabilities
                           classProbs = TRUE,
                           ## Evaluate performance using
                           ## the following function
                           summaryFunction = twoClassSummary)

train_knn <- train(no_show ~ ., method = "knn",
                   data = train_for_knn,
                   tuneGrid = data.frame(k = seq(35, 35, 10)), #5, 65, 10)),
                   trControl = fitControl,
                   metric = "ROC")

test_preds_knn <- predict(train_knn, newdata = test_for_knn, type= "prob")
auc(roc(test_preds_knn[,2], factor(1 * (test_for_knn$no_show == "Yes"))))
```

```
## [1] 0.5292445
```

Several features were removed and it was found that removing PatientId (which a priori had been considered meaningless as a number) makes a significant difference.

```
train_for_knn <- train_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-PatientId)

test_for_knn <- test_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
    select(-PatientId)

train_knn <- train(no_show ~ ., method = "knn",
                   data = train_for_knn,
                   tuneGrid = data.frame(k = seq(35, 35, 10)), #5, 65, 10)),
                   trControl = fitControl,
                   metric = "ROC")

test_preds_knn <- predict(train_knn, newdata = test_for_knn, type= "prob")
auc(roc(test_preds_knn[,2], factor(1 * (test_for_knn$no_show == "Yes"))))
```

```
## [1] 0.7090923
```

If only bias per neighbourhood is added performance gets a little higher.

```
train_for_knn <- train_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-PatientId, -nbr_app_per_patient, -nbr_app_that_day, -days_between, -wday_sched, -wd
ay_app, -hour_sched)

test_for_knn <- test_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-PatientId, -nbr_app_per_patient, -nbr_app_that_day, -days_between, -wday_sched, -wd
ay_app, -hour_sched)

train_knn <- train(no_show ~ ., method = "knn",
                   data = train_for_knn,
                   tuneGrid = data.frame(k = seq(35, 35, 10)), #5, 65, 10)),
                   trControl = fitControl,
                   metric = "ROC")

test_preds_knn <- predict(train_knn, newdata = test_for_knn, type= "prob")
auc(roc(test_preds_knn[,2], factor(1 * (test_for_knn$no_show == "Yes"))))
```

```
## [1] 0.7091183
```

If additional features are removed (those identified as less important in the random forest approach) performance gets slightly higher.

```
train_for_knn <- train_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-PatientId, -nbr_app_per_patient, -nbr_app_that_day, -days_between, -wday_sched, -wd
ay_app, -hour_sched, -Diabetes, -Alcoholism, -Hipertension, -Handcap)

test_for_knn <- test_for_model %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-PatientId, -nbr_app_per_patient, -nbr_app_that_day, -days_between, -wday_sched, -wd
ay_app, -hour_sched, -Diabetes, -Alcoholism, -Hipertension, -Handcap)

train_knn <- train(no_show ~ ., method = "knn",
                   data = train_for_knn,
                   tuneGrid = data.frame(k = seq(25, 45, 10)),
                   trControl = fitControl,
                   metric = "ROC")

test_preds_knn <- predict(train_knn, newdata = test_for_knn, type= "prob")
auc(roc(test_preds_knn[,2], factor(1 * (test_for_knn$no_show == "Yes"))))
```

```
## [1] 0.7087102
```

If instead data set features are used just the way they are in the data set with the exception of AppointmentID (that is very much the same as ScheduledDay), Neighbourhood (because the algorithm can not handle 81 classes) and PatientId (same reason as before) ROC AUC is also about 0.71.

```
train_for_knn <- train %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-Neighbourhood, -AppointmentID, -PatientId)

test_for_knn <- dev_set %>%
  mutate(no_show = if_else(no_show == TRUE, "Yes", "No")) %>%
  select(-Neighbourhood, -AppointmentID, -PatientId)

train_knn <- train(no_show ~ ., method = "knn",
                   data = train_for_knn,
                   tuneGrid = data.frame(k = seq(25, 45, 10)), #5, 65, 10)),
                   trControl = fitControl,
                   metric = "ROC")

test_preds_knn <- predict(train_knn, newdata = test_for_knn, type= "prob")
auc(roc(test_preds_knn[,2], factor(1 * (test_for_knn$no_show == "Yes"))))
```

```
## [1] 0.7091183
```

Removing additional features did not increase performance.

2.8. Models conclussion

The best model was the RF9 with a ROC AUC = 0.77 when lambda_nbh = 50 and tune_mtry = 4.

KNN did not perform as good as RF9 for any of the variants tried. It was not even close and therefore it was discarded. Should KNN have got a similar performance an ensemble between both algorithms would have been tried.

In the following section the whole training set (including the “dev set”) is used to train the ultimate model. Then predictions and evaluation on the final test set are made.

3.1. Training the ultimate model, predicting and testing on the test set

The whole training set is used to train the ultimate model.

```
lambda_nbh <- 50
tune_mtry <- 4

train <- train %>% bind_rows(dev_set) #put together all the training set back again

no_show_rate <- mean(train$no_show == TRUE)

bias_per_nbh <- train %>%
  group_by(Neighbourhood) %>%
  mutate(count = n()) %>%
  filter(count > lambda_nbh) %>%
  summarise(bias_per_nbh = mean(no_show == TRUE) - no_show_rate)

nbr_app_that_day <- train %>%
  group_by(PatientId, AppointmentDay) %>%
  summarise(nbr_app_that_day = n())

nbr_app_per_patient <- train %>%
  group_by(PatientId) %>%
  summarise(nbr_app_per_patient = n())


train_for_model <- convert_df(train, bias_per_nbh, nbr_app_per_patient, nbr_app_that_day)

rf_ultimate_model <- randomForest(as.factor(no_show) ~ ., data = train_for_model, ntree = 500
, mtry = tune_mtry)
```

Predictions are made on the final test set (this is the only time it is used for anything).

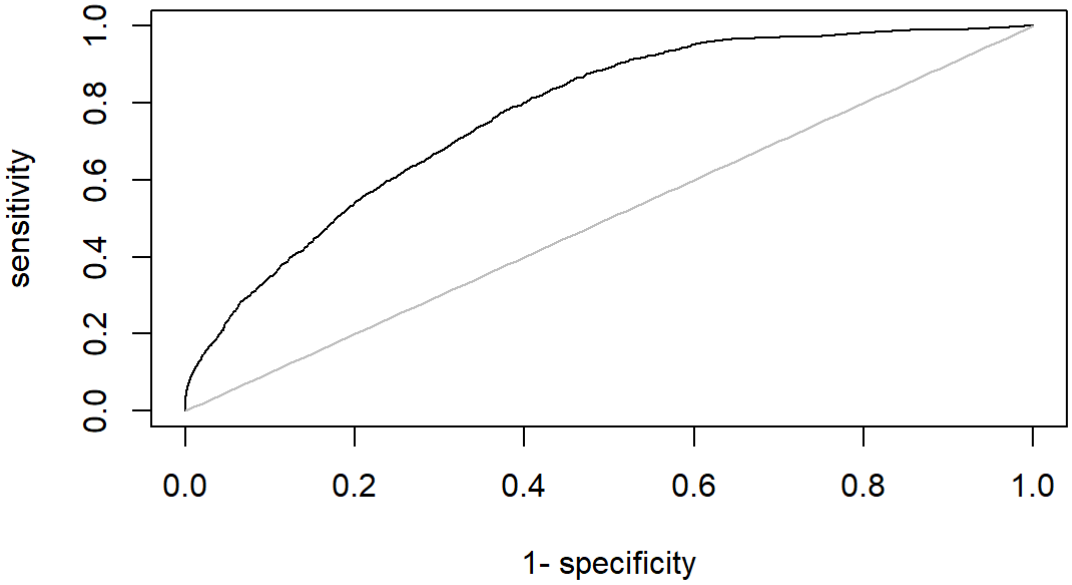
```
test_for_eval <- convert_df(final_test, bias_per_nbh, nbr_app_per_patient, nbr_app_that_day)

test_preds <- predict(rf_ultimate_model, newdata = test_for_eval, type= "prob")
```

And finally ROC AUC is evaluated for the final test set.

```
roc_test <- roc(test_preds[,2], factor(1 * (test_for_eval$no_show==TRUE)))
auc(roc_test)
```

```
## [1] 0.7721474
```

```
plot(roc_test)
```



In order to further evaluate and discuss results some particular cutoff values are used to evaluate sensitivity and true positive rate:

```
confusionMatrix(predict(rf_ultimate_model, newdata = test_for_eval, cutoff = c(0.68, 0.32)),
 as.factor(test_for_eval$no_show), positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  7310 1115
##      TRUE   1545 1086
##
##                Accuracy : 0.7594
##                  95% CI : (0.7513, 0.7674)
##     No Information Rate : 0.8009
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2971
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.49341
##             Specificity : 0.82552
##          Pos Pred Value : 0.41277
##          Neg Pred Value : 0.86766
##              Prevalence : 0.19908
##          Detection Rate : 0.09823
##    Detection Prevalence : 0.23797
##       Balanced Accuracy : 0.65947
##
##        'Positive' Class : TRUE
##
```

With the selected cutoff sensitivity = 0.5 and True Positive Rate = 0.41.

Another cutoff allows to achieve True Positive Rate = 0.75 with Sensitivity = 0.1.

```
confusionMatrix(predict(rf_ultimate_model, newdata = test_for_eval, cutoff = c(0.45, 0.55)),
 as.factor(test_for_eval$no_show), positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  8777 1985
##      TRUE     78  216
##
##                Accuracy : 0.8134
##                  95% CI : (0.806, 0.8206)
##     No Information Rate : 0.8009
##     P-Value [Acc > NIR] : 0.0004854
##
##                   Kappa : 0.1324
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.09814
##             Specificity : 0.99119
##          Pos Pred Value : 0.73469
##          Neg Pred Value : 0.81555
##              Prevalence : 0.19908
##          Detection Rate : 0.01954
##    Detection Prevalence : 0.02659
##       Balanced Accuracy : 0.54466
##
##        'Positive' Class : TRUE
##
```

3.2. Results discussion

ROC AUC = 0.77 may not seem bad but, despite not having a health background to evaluate the usefulness of these results, I posed this question ir order to try to do that: 20% appointments are being missed. If detecting half of them was intended: how good would the true positive rate be?

Several cutoffs were tried to evaluate that question and it was found that for sensitivity to be 0.5, the true positive rate is just 0.4. The majority of positive predictions the algorithm delivers are false positives.

If a reliable true prediction is intended, for example True Positive Rate = 0.75, then only 10% of the real positives are being detected.

Personally, I do not consider that to be useful.

As stated in the metrics and target section, no target is available. It could be the case that the best achievable performance is not much higher than 0.77 with the available data. It is unknown. Significant variability could be explained for features that are unavailable such as medical specialty. No-show rate might be much higher for the dermatologist’s appointments than for the cardiologist’s appointments. Rainy or stormy days might have a higher no-show rate than sunny days. The distance from patient’s home to hospital in not available. 33% of the observations correspond to patients for which that is the only available observation. 75% correspond to patients which have 3 or less appointments in the data set. The amount of appointments corresponding to

patients for which there are several appointments and a previous no-show record can be considered is so little that prevents the use of such a variable. Appointment reservation type (phone / in person / web) is also unavailable. Medical appointment no-show papers claim that those are relevant variables. There could be several factors that explain no-show behaviour that are not represented in the available variables.

After achieving ROC AUC = 0.77 the Kaggle website related to this data set was searched for other results. By June 2021 there are 328 notebooks addressing this data set.

The vast majority of them just depict an exploratory analysis. There are no significant differences with the exploratory analysis conducted here with the sole exception of a couple of notebooks that include heatmaps with a few variables.

Many other notebooks try algorithms (mainly random forest and XGBoost) but without any feature manipulation and only targeting accuracy, that is not a good performance indicator for this task. Accuracy achieved in most of them is about 80%, that is: about the same a naïve approach predicting “show” every time would do.

There is only one notebook that achieved outstanding results, almost perfect (ROC AUC = 0.999): https://www.kaggle.com/molihn/xgboost-to-predict-no-show-part-2-surprice (https://www.kaggle.com/molihn/xgboost-to-predict-no-show-part-2-surprice)

But a user found a leak that explains those results: https://www.kaggle.com/molihn/xgboost-to-predict-no-show-part-2-surprice#1011980 (https://www.kaggle.com/molihn/xgboost-to-predict-no-show-part-2-surprice#1011980)

This user’s explanation claims the algorithm is using a variable that is a linear transformation of the variable to be predicted, that is: when predicting no-show in the test set, the values of the no-show variable of the test set (notice it is TEST set both times) are being used. Therefore, although outstanding, that explanation prevents these results from being considered.

There is another notebook that achieved good results in metrics other than accuracy. It deals with imbalanced data with a downsampling approach (although imbalance is not quite high):

https://www.kaggle.com/belagoesr/predicting-no-show-downsampling-approach-with-rf (https://www.kaggle.com/belagoesr/predicting-no-show-downsampling-approach-with-rf)

It can be seen that Out[43] shows Precision = 83% and Recall = 97%. It is awesome. But a careful look shows in Out[44] that those results correspond to a downsampled test set where classes have been balanced, that is: a significant amount of “show” observations have been removed in the test set that is being used for performance evaluation. But then the test set distribution has been modified and no longer represents the nature of the problem. Should the classes be balanced instead of 80/20, then higher recall values can be achieved without paying a huge price in precision. It would be interesting to see the metrics for a complete test set constructed by randomly sampling the whole data set.

4. Conclusions

Identifying features that explain no-show behaviour proved to be hard in the data exploration conducted. Age, SMS received, number of days elapsed between scheduling and appointment, and other features explain some variability but it is just too little. For example: it was identified that SMS were only sent to patients with 3 or more days elapsed between scheduling and the appointment, and then a positive effect of the SMS was observed. But the effect was not very high: 0.327 for no SMS Vs. 0.276 for SMS. It was not possible to identify a single group within the data set for which there was a dominant tendency to no-show (say $p > 0.5$), with the exception of the variable PatientId. But then patients with several appointments for which it is possible to build a previous record of no-show represent a very tiny fraction of the data set. Therefore, although it is possible to use the previous record for them, it does not make a significant difference when considering the entire data set.

A classification algorithm was developed with a random forest approach. Its performance was evaluated with ROC AUC on the test set and achieved 0.77. Increasing performance (from 0.72 to 0.77) required successive feature manipulation and trying several approaches.

A downsampling approach was also tried in order to improve the algorithm's performance on this imbalanced data set, but the imbalance was not very high (just 80/20) and thus the downsampling did no good to performance.

KNN models were also developed but performance did not match that corresponding to random forest's for this task.

The best result achieved (ROC AUC = 0.77) was not considered very useful for the decision making process within a hospital setting because of the low recall or low precision observed.

The best ROC AUC achievable is unknown and it was questioned whether some important features would be missing if higher performance is to be reached.

It would be very interesting to try an approach like this for a data set comprising other variables, such as patients' no-show previous records, medical specialty, reservation system, the time of the appointment and a larger time span.