Untitled

Tata iles

2023-06-01

Libraries

DATA

```
library(reticulate)
use_python("C:/Users/tatai/AppData/Local/Programs/Python/Python311/python.exe")
```

```
import pandas as pd
```

```
data = pd.read_csv("D:/Downloads/base examen écrit.csv")
```

```
fraud <- py$data
```

Preprocessing

Adding age , hour , day columns because they tend to have influence on the fraud

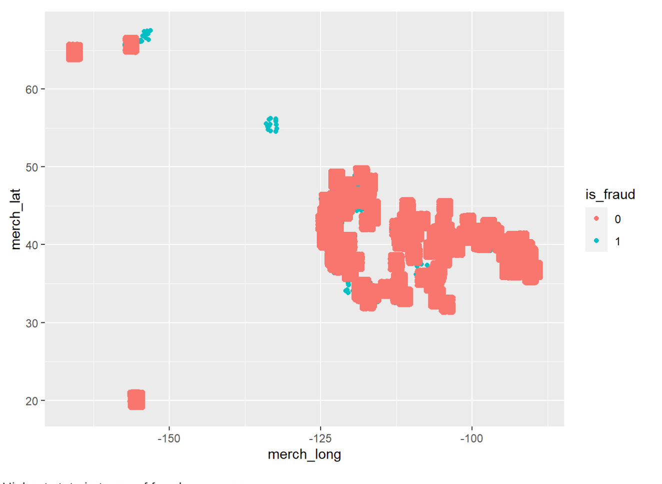
```
fraud <- fraud %>%
  mutate_if(is.character,as.factor)%>%
  mutate(
         year_b = str_split(dob,"-",simplify = TRUE)[,1],
         year_p = str_split(trans_date_trans_time," ",simplify = TRUE)[,1],
         year_p = str_split(year_p,"-",simplify = TRUE)[,1],
         age = as.integer(year_p) - as.integer(year_b),
         hour = hour(trans_date_trans_time),
         day = wday(trans_date_trans_time)) %>%
  select(-c(1,dob,year_b,year_p,trans_num))
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `hour = hour(trans_date_trans_time)`.
## Caused by warning:
## ! tz(): Don't know how to compute timezone for object of class factor; returning "UTC".
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
fraud$is_fraud <- as.factor(fraud$is_fraud)
```

Some coordinates tend to have more fraud occurrence

```
fraud %>% ggplot(aes(merch_long,merch_lat,color = is_fraud))+geom_point()
```

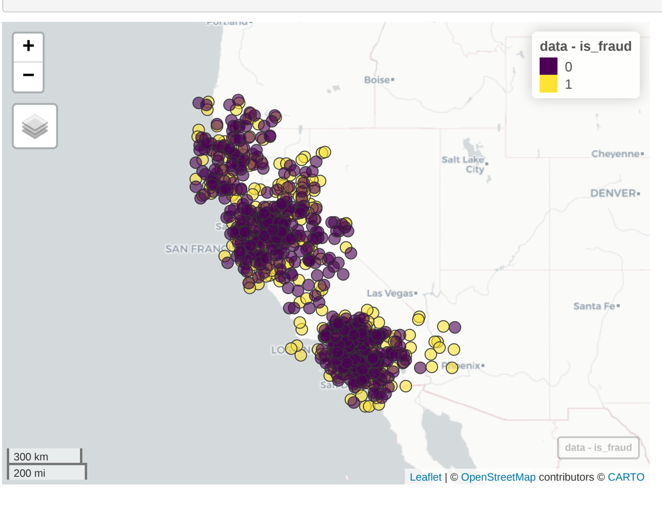


Highest state in terms of fraud occurrence

```
fraud %>%
  group_by(state) %>%
  count(is_fraud) %>%
  filter(is_fraud == 1) %>%
  arrange(-n) %>% head(3)
```

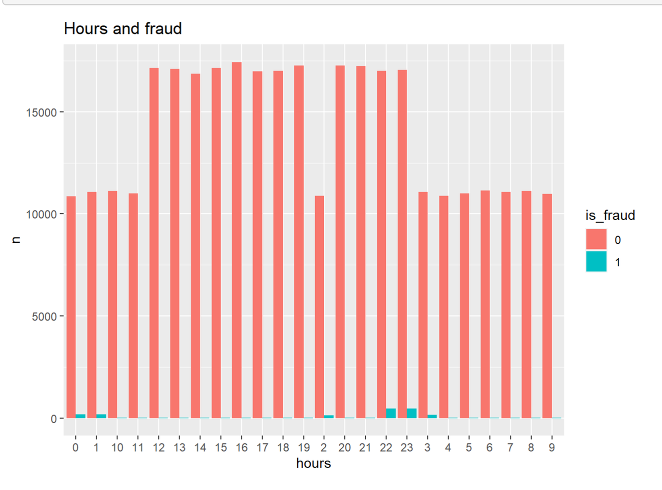
```
## # A tibble: 3 × 3
## # Groups:   state [3]
##   state is_fraud     n
##   <fct> <fct>    <int>
## 1 CA    1          402
## 2 MO    1          262
## 3 NE    1          216
```

```
CA <- rbind(fraud %>% filter(state == "CA"&is_fraud == 1),fraud %>% filter(state == "CA"&is_fraud == 0) %>% slice
(1:500))
mapview(CA, xcol = "merch_long", ycol = "merch_lat",
        crs = 4269, grid = FALSE,
        zcol = "is_fraud")
```



Some hours have more cases of frauds more than others

```
fraud %>% group_by(hour) %>% count(is_fraud) %>%
  ggplot(aes(as.character(hour),n,fill=is_fraud)) + geom_col(position=position_dodge()) + xlab("hours") + ggtitle
("Hours and fraud")
```



some categories tend to have more cases for fraud

```
fraud %>% group_by(category) %>% count(is_fraud) %>%
  ggplot(aes(n,category,fill=is_fraud)) + geom_col(position=position_dodge()) + ggtitle("Fraud and categories") +
scale_fill_manual(values=c('#999999','red'))
```

some ages tend to have more cases for fraud

```
#fraud %>% group_by(age) %>%
#  count(is_fraud) %>%
#  filter(is_fraud == 1) %>%
#  mutate(age = as.character(cut(age,
#                       seq(10, 100, 10)))) %>%
#  group_by(age) %>%
#  summarize(sums = sum(n)) %>%
#  as.data.frame() %>%
#  ggplot(aes(age,sums)) +
#  ggtitle("Ages and fraud") + geom_col(color="red", fill="white") + ylab("Number of frauds")
```

```
custom_colors <- viridis::mako(n = 9)

fraud %>% group_by(age) %>%
  count(is_fraud) %>%
  filter(is_fraud == 1) %>%
  mutate(age = as.character(cut(age,
                    seq(10, 100, 10)))) %>%
  group_by(age) %>%
  summarize(fraud = sum(n)) %>%
  as.data.frame() %>%
  hchart('column', hcaes(x = age, y = fraud, color = custom_colors)) %>%
  hc_add_theme(hc_theme_google()) %>%
  hc_tooltip(pointFormat = '<b>Number of sales: </b> {point.y} <br>') %>%
  hc_title(text = 'Fraud and Ages',
           style = list(fontSize = '25px', fontWeight = 'bold')) %>%
  hc_subtitle(text = 'Age bins by fraud occurence',
              style = list(fontSize = '16px'))
```

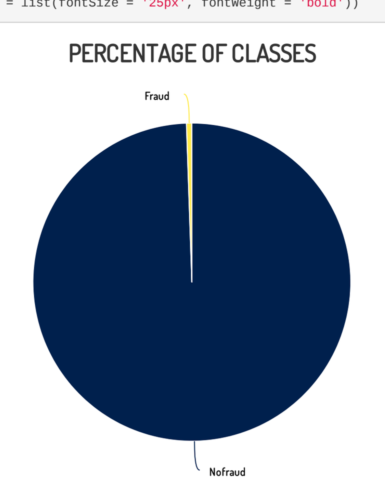
Unbalanced data

The data is highly unbalanced 99% non fraud

```
fraud %>%
  count(is_fraud) %>%
  mutate(n=paste(round(n*100/nrow(fraud),2),"%"))
```

```
##   is_fraud       n
## 1        0 99.48 %
## 2        1  0.52 %
```

```
custom_colors <- viridis::cividis(n = 2)
fraud %>% count(is_fraud)  %>% mutate(is_fraud = c("No fraud","Fraud")) %>%
  hchart('pie', hcaes(x = is_fraud, y = n,color = custom_colors)) %>%   hc_add_theme(hc_theme_gridlight()) %>%
  hc_tooltip(pointFormat = '<b>Number of sales: </b> {point.y} <br>') %>%
  hc_title(text = 'Percentage of classes',
           style = list(fontSize = '25px', fontWeight = 'bold'))
```



Converting the GPS coordinates to distance (new column)

```
df <- data.frame( fraud %>% select(long,lat,merch_long,merch_lat))


distances_km <- numeric(nrow(df))


for (i in 1:nrow(df)) {
  lon1 <- df$long[i]
  lat1 <- df$lat[i]
  lon2 <- df$merch_long[i]
  lat2 <- df$merch_lat[i]

  distances_km[i] <- distGeo(c(lon1, lat1), c(lon2, lat2)) / 1000
}


fraud$distance_km <- distances_km
rm(distances_km,lon1,lat1,lon2,lat2,i,df)
```

Categorical columns

. name of merchant has so many unique values and we can't one hot code it (693 new columns), label encoding also isn't a choice .

so i preffered dropping it and it also don't tend to have that much importance .

. same for the city .

. category could be one hot encoded and has influence on the fraud as represented above.

. job might have some importance on the fraud so i tried to frequency-encode it (replace the values by the frequency) .

```
uniques <- fraud %>% summarise(merchant = length(unique(merchant)),
                    category = length(unique(category)),
                    job = length(unique(job)),
                    city = length(unique(city))) %>% t() %>% as.data.frame()
uniques
```

```
##           V1
## merchant 693
## category  14
## job      163
## city     176
```

city and state columns dropped because they have the same info and they are hard to encode so i chose to use city pop instead

```
fraud <- fraud %>% select(-c(long,merch_long,merch_lat,lat,merchant,city,state))
```

Splitting

```
set.seed(123)
data_split <-
  initial_split(fraud,strata = is_fraud)

train <- training(data_split)
test  <- testing(data_split)

#cross validation object
fraud_folds <- vfold_cv(train, v = 3, strata = is_fraud)
```

Recipe

i tried 3 different recipes to tune the model and choose both the best recipe and best hyperparameters

```
#basic recipe
recipe_plain <-
  recipe(is_fraud ~ ., data = train) %>%
  step_normalize(all_numeric_predictors())  %>%
  step_mutate(job, count = n()) %>%
  step_integer(job)%>%
  step_rm(count)%>%
  step_dummy(all_nominal_predictors())

#rebalancing using smote
smote <- recipe_plain %>%
  step_smote(is_fraud,over_ratio = 0.85) %>%
  step_sample(size = nrow(train))

#rebalancing using random undersampling
rus <- recipe_plain %>% step_downsample(is_fraud)
```

Metric set

```
metric <- metric_set(sens, precision,yardstick::spec, j_index, f_meas)
```

Model

Spec

i chose the lightgbm model because it was the best model

```
set.seed(123)
lightgbm_spec <-
  boost_tree(
    mtry = tune(),
    trees = tune(),
    tree_depth = tune(),
    learn_rate = tune(),
    min_n = tune(),
    loss_reduction = tune()
  ) %>%
  set_engine(engine = "lightgbm") %>%
  set_mode(mode = "classification")
```

Workflow

```
wf_set_tune <-
  workflow_set(
    list(plain = recipe_plain,
         smote = smote,
         rus = rus),
    list(lightgmb = lightgbm_spec)
  )
```

Tune

tune the hyperparameters of the model and evaluate the model accross different recipes (simple, smote, rus)

```
set.seed(123)
tune_results <-
  workflow_map(
    wf_set_tune,
    "tune_grid",
    resamples = fraud_folds,
    grid = 6,
    metrics = metric,
    verbose = TRUE
    )
```

```
## i 1 of 3 tuning:     plain_lightgmb
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
## ✔ 1 of 3 tuning:     plain_lightgmb (2m 28.9s)
```

```
## i 2 of 3 tuning:     smote_lightgmb
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
## ✔ 2 of 3 tuning:     smote_lightgmb (3m 44.7s)
```

```
## i 3 of 3 tuning:     rus_lightgmb
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
## ✔ 3 of 3 tuning:     rus_lightgmb (1m 28.8s)
```

Ranking the tuning results by j-index on validation sets

the balanced data ranks better

. both rebalancing methods increased the j_index

smote 0.82

rus 0.91

. the under sampling did a better job

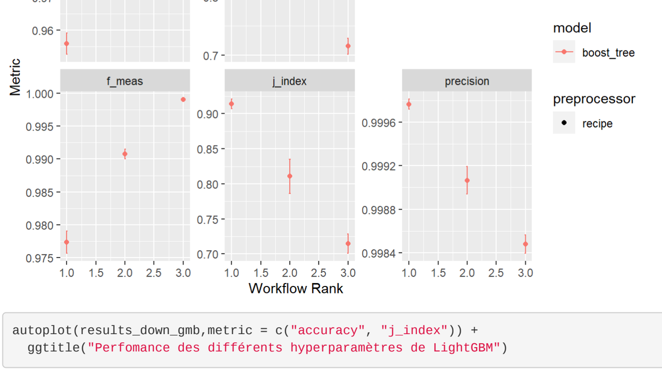
```
rank_results(tune_results, rank_metric = "j_index")
```

```
## # A tibble: 90 × 9
##    wflow_id     .config     .metric  mean std_err     n preprocessor model  rank
##    <chr>        <chr>       <chr>   <dbl>   <dbl> <int> <chr>        <chr> <int>
##  1 rus_lightgmb Preprocess… f_meas  0.977 1.03e-3     3 recipe       boos…     1
##  2 rus_lightgmb Preprocess… j_index 0.914 4.22e-3     3 recipe       boos…     1
##  3 rus_lightgmb Preprocess… precis… 1.00  2.83e-5     3 recipe       boos…     1
##  4 rus_lightgmb Preprocess… sens    0.956 1.97e-3     3 recipe       boos…     1
##  5 rus_lightgmb Preprocess… spec    0.958 5.26e-3     3 recipe       boos…     1
##  6 rus_lightgmb Preprocess… f_meas  0.966 1.05e-3     3 recipe       boos…     2
##  7 rus_lightgmb Preprocess… j_index 0.874 1.25e-2     3 recipe       boos…     2
##  8 rus_lightgmb Preprocess… precis… 1.00  5.61e-5     3 recipe       boos…     2
##  9 rus_lightgmb Preprocess… sens    0.935 1.92e-3     3 recipe       boos…     2
## 10 rus_lightgmb Preprocess… spec    0.939 1.06e-2     3 recipe       boos…     2
## # ℹ 80 more rows
```

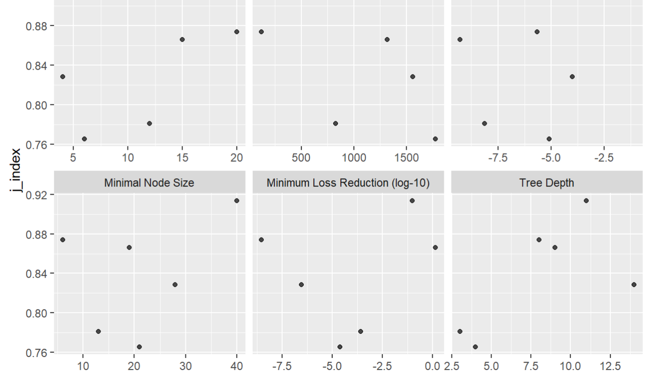
Selecting best model

```
results_down_gmb <- tune_results %>%
  extract_workflow_set_result("rus_lightgmb")
```

```
autoplot(tune_results, rank_metric = "j_index", select_best = TRUE) +
  ggtitle("Performance des différents modèles")
```




```
autoplot(results_down_gmb,metric = c("accuracy", "j_index")) +
  ggtitle("Perfomance des différents hyperparamètres de LightGBM")
```



Finalizing workflow

```
best_hyperparameters <- tune_results %>%
  extract_workflow_set_result("rus_lightgmb") %>%
  select_best(metric = "j_index")

validation_results <- tune_results %>%
  extract_workflow("rus_lightgmb") %>%
  finalize_workflow(best_hyperparameters) %>%
  last_fit(data_split, metrics = metric)
```

Performance on test data

as we can see we get we got stable metrics and stability between test and validation :

high accuracy 0.96 it was 0.998 which could lead to over fit

higher j-index for both test(0.919) and validation (0.913)

```
rbind(validation_results %>% collect_metrics() %>% select(-.config),
      validation_results %>%
        collect_predictions() %>%
        accuracy(truth = is_fraud,estimate = .pred_class))
```

```
## # A tibble: 6 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 sens      binary         0.962
## 2 precision binary         1.00
## 3 spec      binary         0.958
## 4 j_index   binary         0.920
## 5 f_meas    binary         0.981
## 6 accuracy  binary         0.962
```

Confusion matrix

```
validation_results %>% collect_predictions() %>% conf_mat(truth = is_fraud, estimate = .pred_class)
```

```
##           Truth
## Prediction     0     1
##          0 81261    18
##          1  3214   409
```

The matrix indicates that the models focuses more on catching fraudulent transactions rather than getting non fraud as non fraud and that might be useful as one fraud predicted as fraud is more important than predicting non frauds as fraud … one single fraudulent transaction could cause much more lost than loosing a non fraudulent customer