Imbalanced Fraud Detection Undersampling, SMOTE-NC, R-Python Hybrid

YOUNES AISSAOUI

2023

Make sure you have all the packages and modules both in .R and Python

```
library(tidymodels)
library(tidyverse)
library(reticulate)
library(ROSE)
library(themis)
library(bonsai)
use_python("C:/Users/YOUNES/AppData/Local/Programs/Python/Python310/python.exe")
```

```
import pandas as pd
import numpy as np
import imblearn
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTENC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

DATA

```
)"fraud <- read_csv("C:/Users/YOUNES/Desktop/FRAUD.csv
```

Imbalanced data

```
fraud %>%  group_by(is_fraud) %>% summarise(count=n())
```

```
## # A tibble: 2 x 2
##   is_fraud  count
##      <dbl>  <int>
## 1        0 337825
## 2        1   1782
```

1-Pre-treatments

-Deleting variables that need more work to be useful.

-Under sampling

-Oversampling using SMOTE-NC works with numeric and categorical variables

```
fraud <- fraud %>%
  mutate(category = as.factor(category),
         job = as.factor(job),is_fraud = as.factor(is_fraud))
fraud <- fraud[,!names(fraud) %in% c("trans_date_trans_time", "merchant", "city","job","dob","trans_num
set.seed(222)
data_split <-
  initial_split(fraud, prop = 0.75, strata = is_fraud)
train_data <- training(data_split)
test_data  <- testing(data_split)
data_balanced_under <- ovun.sample(is_fraud ~ ., data = train_data, method = "under", N = 70000, seed =
py$test_data <-test_data
py$train_data <-data_balanced_under
```

```
X_train = train_data.iloc[:,:-1]
y_train = train_data.iloc[:,-1]
X_test = test_data.iloc[:,:-1]
y_test = test_data.iloc[:,-1]
categorical_features = [0,2]  # Example indices of categorical columns
numerical_features = [1,3,4,5]
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_features),
        ('num', StandardScaler(), numerical_features)
    ])
X_train_encoded = preprocessor.fit_transform(X_train)
X_test_encoded = preprocessor.fit_transform(X_test)
oversampler = SMOTENC(sampling_strategy=0.5, categorical_features=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
,k_neighbors=5,random_state=252)
X_oversampled, y_oversampled = oversampler.fit_resample(X_train_encoded, y_train)
X_resampled  = X_oversampled
y_resampled = y_oversampled
cat_encoder = preprocessor.named_transformers_['cat']
cat_feature_names = cat_encoder.get_feature_names_out()
feature_names = list(cat_feature_names) +  ['amt',"lat" ,"long" ,"city_pop"]
X_resampled_dense = X_resampled.toarray()
df_resampled = pd.DataFrame(X_resampled_dense, columns=feature_names)
df_resampled['is_fraud'] = y_resampled
X_test_encoded = X_test_encoded.toarray()
df_test_encoded = pd.DataFrame(X_test_encoded, columns=feature_names)
X_train=df_resampled.iloc[:,:-1]
y_train=df_resampled.iloc[:,-1]
X_test = df_test_encoded
series_df = pd.DataFrame(y_test, columns=['is_fraud'])
series_df = series_df.reset_index(drop=True)
```

```
balanced_data_test = pd.concat([X_test, series_df], axis=1)
balanced_data_train = df_resampled
```

-Splitting the new balanced Data *NO Leakage*

```
balanced_data_test = as.data.frame(py$balanced_data_test)
balanced_data_train = as.data.frame(py$balanced_data_train)
dfb <- rbind(balanced_data_train,balanced_data_test)
ind <- list(analysis = seq(nrow(balanced_data_train)), assessment = nrow(balanced_data_train) + seq(nrow
balanced_data_split <- make_splits(ind, dfb)
balanced_data_split
```

```
## <Analysis/Assess/Total>
## <103014/84902/187916>
```

2-Recipe, Setting engines, Selecting Metrics, workflow set, Tuning

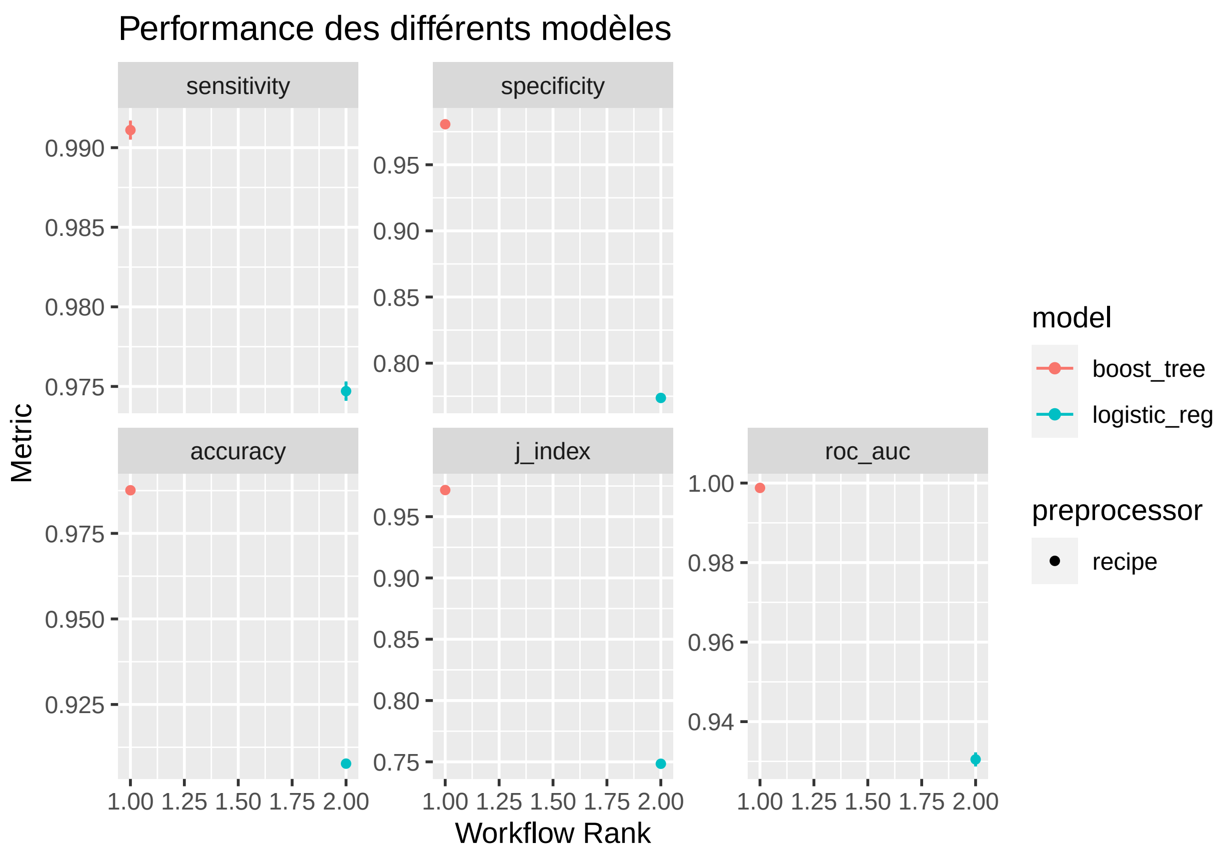
```
recipe_plain <-
  recipe(is_fraud ~ ., data = balanced_data_train)
logreg_spec <-
  logistic_reg() %>%
  set_engine("glm")

rf_spec <-
  rand_forest(trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("classification")

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")

fraud_metrics <-
  metric_set(roc_auc, accuracy, sensitivity, specificity, j_index)
wf_set_tune <-
  workflow_set(
    list(plain = recipe_plain),
    list(
      lightgbm = lightgbm_spec,
      logreg = logreg_spec
     )
```

```r
  )
set.seed(345)
fraud_folds <- vfold_cv(balanced_data_train, v = 3, strata = is_fraud)
tune_results <-
  workflow_map(
    wf_set_tune,
    "tune_grid",
    resamples = fraud_folds,
    grid = 10,
    metrics = fraud_metrics,
    verbose = TRUE
    )
rank_results(tune_results, rank_metric = "j_index") %>%
  select(-`.config`, -n,-preprocessor) %>%
  filter(.metric == "j_index")
```

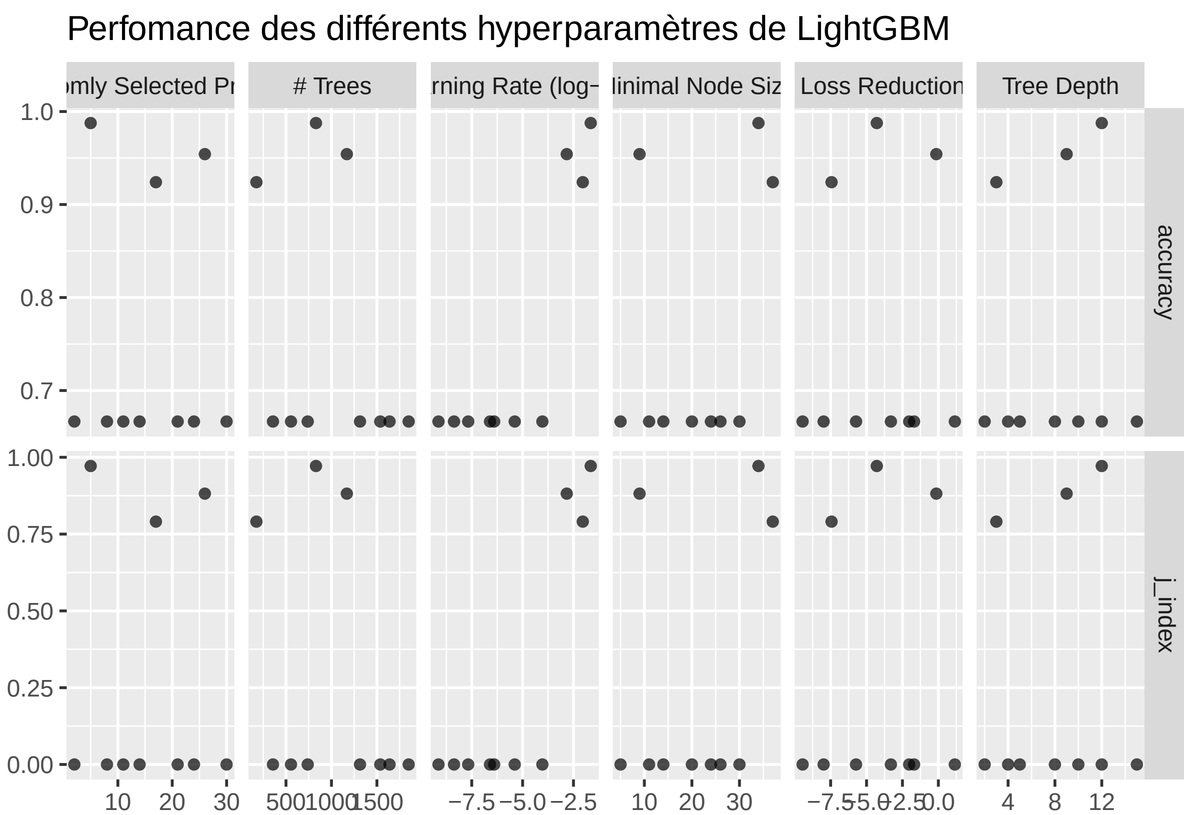
```
## # A tibble: 11 x 6
##    wflow_id       .metric  mean  std_err model         rank
##    <chr>          <chr>   <dbl>    <dbl> <chr>        <int>
##  1 plain_lightgbm j_index 0.972 0.00121  boost_tree       1
##  2 plain_lightgbm j_index 0.882 0.00419  boost_tree       2
##  3 plain_lightgbm j_index 0.791 0.0289   boost_tree       3
##  4 plain_logreg   j_index 0.748 0.000648 logistic_reg     4
##  5 plain_lightgbm j_index 0     0        boost_tree       5
##  6 plain_lightgbm j_index 0     0        boost_tree       6
##  7 plain_lightgbm j_index 0     0        boost_tree       7
##  8 plain_lightgbm j_index 0     0        boost_tree       8
##  9 plain_lightgbm j_index 0     0        boost_tree       9
## 10 plain_lightgbm j_index 0     0        boost_tree      10
## 11 plain_lightgbm j_index 0     0        boost_tree      11
```

3- visualizations

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




```r
results_down_gmb <- tune_results %>%
  extract_workflow_set_result("plain_lightgbm")
autoplot(results_down_gmb,metric = c("accuracy", "j_index")) +
  ggtitle("Perfomance des différents hyperparamètres de LightGBM")
```



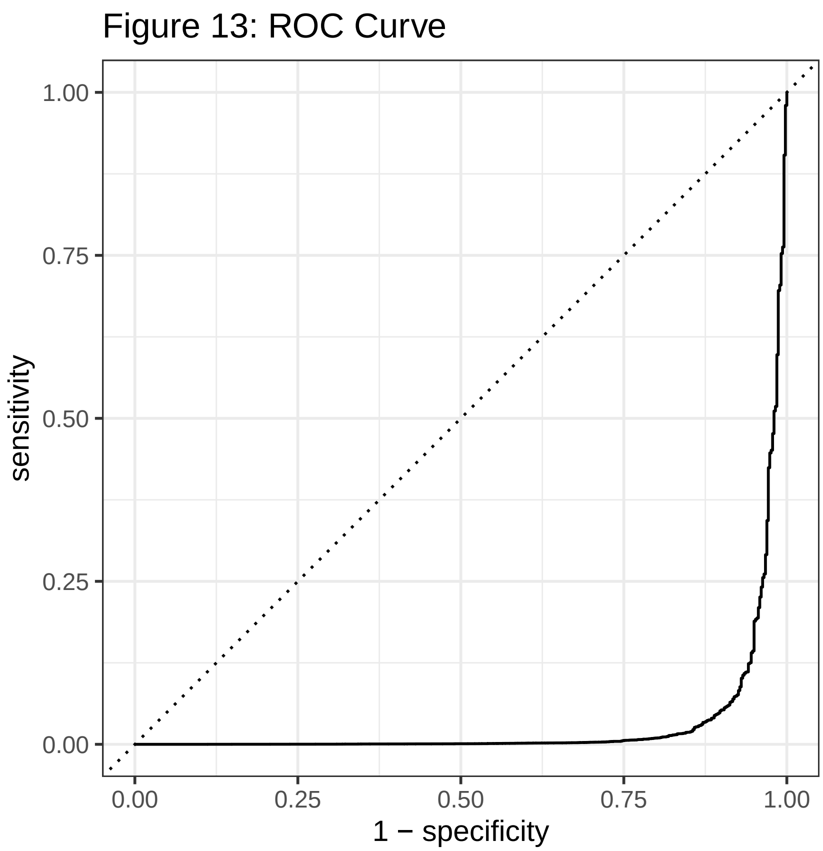
4- The best Model

```
best_hyperparameters <- tune_results %>%
  extract_workflow_set_result("plain_lightgbm") %>%
  select_best(metric = "j_index")
validation_results <- tune_results %>%
  extract_workflow("plain_lightgbm") %>%
  finalize_workflow(best_hyperparameters) %>%
  last_fit(split =  balanced_data_split, metrics = fraud_metrics)
collect_metrics(validation_results)
```

```
## # A tibble: 5 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.983 Preprocessor1_Model1
## 2 sensitivity binary         0.984 Preprocessor1_Model1
## 3 specificity binary         0.841 Preprocessor1_Model1
## 4 j_index     binary         0.824 Preprocessor1_Model1
## 5 roc_auc     binary         0.969 Preprocessor1_Model1
```

5-ROC Curve

```
validation_results %>%
  collect_predictions() %>%
  roc_curve(is_fraud, .pred_1) %>%
  autoplot() +
  ggtitle("Figure 13: ROC Curve")
```



6- Confusion matrix

```
val <- validation_results[[5]][[1]]
val %>% conf_mat(truth = is_fraud, estimate = .pred_class)
```

```
##           Truth
## Prediction     0     1
##          0 83058    73
##          1  1386   385
```