# Projekt Statistical Learning w Praktyce

Grzegorz Maciąg, Kacper Kowalczyk

Luty 2023

## 1 Wprowadzenie problemu

Dane, które wybraliśmy do projektu zostały udostępnione przez meksykański rząd i dotyczą osób chorych na COVID–19.
Źródło: https://www.kaggle.com/datasets/meirnizri/covid19-dataset.

Celem analizy będzie przewidzenie czy pacjent zginie czy przeżyje infekcje. Z racji natury problemu postaramy się zminimalizować błąd, w którym algorytm źle przewidzi, że pacjent przeżyje.

## 2 Przegląd danych

Ładowanie danych do R i podstawowe statystyki:

```
dane <- read.csv("RRR/Covid_Data.csv", stringsAsFactors = T)
summary(dane)
     DATE_DIED
9999-99-99: 971633
06/07/2020:   1000
07/07/2020:    996
13/07/2020:    990
16/06/2020:    979
16/07/2020:    938
(Other)    : 72039
```

Zainteresowani jesteśmy kolumną nazwaną "DATE_DIED". Ze strony, z której pochodzą dane wiadomo, że wartość "9999–99–99" oznacza, że pacjent przeżył, a jakakolwiek inna wartość datę śmierci. Prosta linijka kodu zamienia te wartości na zmienna factor o dwóch stopniach:

```
y <- factor(ifelse(DATE_DIED != "9999-99-99", 'Yes', 'No'))
summary(y)
     No      Yes
971633   76942
```

Po wywołaniu funkcji "summary" na zmiennej objaśnianej możemy zaobserwować, że nasze klasy są bardzo niezbalansowane tj. pacjentów, którzy przeżyli jest około 93%, a 7% pacjentów, którzy zgineli. Jest to duży problem, ponieważ nawet prosty klasyfikator, który niezależnie od danych zwracałby klasę "No" miałby 93% dokładności. W związku z tym nie powinniśmy sugerować się tylko miarą dokładności algorytmu, bo może ona "oszukiwać".

Druga rzecz, która może rzucać sie w oczy to ilość danych do naszej dyspozycji. Pozycji mamy 1048575 czyli około 1 milion. Powodować to będzie duży czas oczekiwania na zakończenie jakichkolwiek obliczeń, w tym budowy modelu.

Na stronie skąd pochodzą dane możemy przeczytać także, że istnieją w danych braki, które oznaczane są przez wartości 99 i 97. Nie usuwamy tych braków w danych, bo mogą one wyznaczać jakąś zależność. Przykładowo zmienna "OTHER_DISEASE" oznacza czy pacjent był chory na jakąś inna chorobę w trakcie badań. Brak wartości w tym przypadku może oznaczać, że nie było możliwości na zapytanie o to pacjenta przed śmiercią.

Przed budową właściwego modelu trzeba podzielić dane na dwie części aby sprawdzać jak dobrze działa nasz model. Postanowiliśmy robić to losowo zostawiając 10% danych na zbiór testowy.

```
X <- dane[ ,-5]

train = sample(1:nrow(X), 0.9*nrow(X))

x_train = X[train, ]
x_test = X[-train, ]

y_train = y[train]
y_test = y[-train]

dane_train <- data.frame(x_train,y_train)
summary(dane_train)
y_train
No :874461
Yes: 69256
```

Zbiór uczący ma około 90% wszystkich instancji klasy "Yes", więc jest dobrym wyborem na zbiór uczący.

# 3 Budowa i ewaluacja modelu

Jako modeli predykcyjnego użyjemy lasu losowego i XGBoost. W pierwszej kolejności zbudowalimy las z wszystkimi domyślnymi parametrami.

```
forest <- randomForest(x_train, y_train, x_test, y_test)
```

```
forest
Call:
randomForest(x = x_train, y = y_train, xtest = x_test, ytest = y_test)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 4.84%
Confusion matrix:
       No   Yes class.error
No  862197 12264  0.01402464
Yes  33376 35880  0.48192214
                Test set error rate: 4.85%
Confusion matrix:
      No  Yes class.error
No  95740 1432  0.01473676
Yes  3655 4031  0.47553994
```

Jak widać z ostatniej macierzy krzyżowej algorytm słabo radzi sobie z przewidywaniem klasy "Yes" (pacjent zginie) tj. przewidział, że 3655 osób przeżyje chociaż zginie.

Model XGBoost z przykładowymi parametrami:

```
library(xgboost)
x_train <- as.matrix(x_train)
x_test <- as.matrix(x_test)
y_train <- as.matrix(as.numeric(y_train) - 1) # Yes/No -> 1/0
y_test <- as.matrix(as.numeric(y_test) - 1)

xgb.model <- xgboost(data = x_train, label = y_train,
                     max.depth = 2, eta = 1, nthread = 2, nrounds = 100,
                     objective = "binary:logistic")

pred <- predict(xgb.model, x_test)
y_pred <- as.factor(ifelse(pred>0.5, 1, 0))
ConfusionMatrix(y_pred, y_test)
        y_pred
y_true     0     1
     0 95388  1826
     1  3693  3951
```

Podobny problem pojawia się w modelu XGBoost. Chcielibyśmy wyeliminować ten błąd, w najgorszym przypadku, kosztem drugiego.

W tym celu wykorzystamy tak zwany "Under Sampling". Ideą metody jest wybór do zbioru uczącego takich obserwacji, aby klasy były zbalansowane. Najczęściej wybór obserwacji z wiekszej klasy jest losowy. Osiągnąć można to za pomocą funkcji RandUnderClassif z biblioteki UBL.

```
library(UBL)
DANE_UNDER <- RandUnderClassif(y_train~. , dat = dane_train)

summary(DANE_UNDER)
y_train
No :69256
Yes:69256
```

Jak widzimy klasy są teraz idealnie zbalansowane. Uzyskaliśmy to kosztem niewykorzystania dużej ilości danych. Na szczęście obserwacji z klasy "Yes" też mamy pod dostatkiem. Zbudujmy zatem modele oparte na tych danych i sprawdźmy jak dobrze opisuje dane z wcześniejszego zbioru testowego.

```
forest_under <- randomForest(x_train_under, y_train_under)
y_pred_under <- predict(forest_under, x_test)
library(MLmetrics)
ConfusionMatrix(y_pred_under, y_test)
```

```
           y_pred
y_true      No    Yes
     No   85899  11273
     Yes    399   7287
```

```
x_train_under <- as.matrix(x_train_under)
y_train_under <- as.matrix(as.numeric(y_train_under) - 1)
xgb.under <- xgboost(data = x_train_under, label = y_test_under,
                     max.depth = 2, eta = 1, nthread = 2, nrounds = 100,
                     objective = "binary:logistic")

pred <- predict(xgb.under, x_test)
y_pred <- as.factor(ifelse(pred>0.5, 1, 0))
ConfusionMatrix(y_pred, y_test)
        y_pred
y_true      0      1
     0  86101  11113
     1    425   7219
```

Z macierzy krzyżowych zauważyć można, że zmniejszyliśmy błąd, w którym algorytm przypisuje klasę "przeżyje" przy prawdziwości "nie przeżyje". Wszystko to udało się kosztem drugiego błędu. Według nas drugi błąd jest mniej istotny gdyż dla osób przewidzianych jako śmiertelnie chorych można np. przenieść na oddział intensywnej terapii.

Z dokumentacji funkcji RandUnderClassif można wyczytać, że istnieje parametr C.perc, który kontroluje ile procent danej klasy znajduje się w wylosowanym zbiorze. Przykładowo:

```
dane_under02 <- RandUnderClassif(y_train~., dat = dane_train,
                                 C.perc = list( No = 0.15))
summary(dane_under02)
     y_train
No :131169
Yes: 69256
```

W następej sekcji przy pomocy walidacji krzyżowej wybierzemy odpowiedni parametr C.perc.

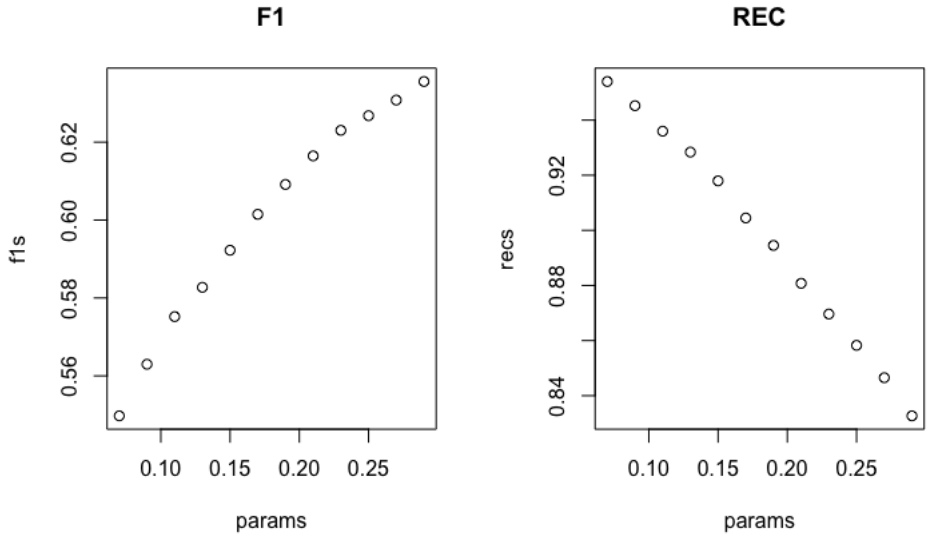
# 4 Walidacja krzyżowa

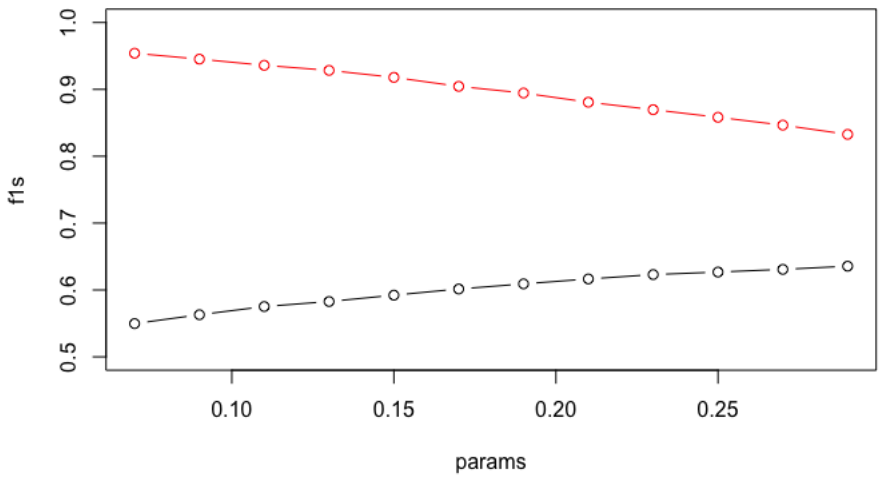
## 4.1 Walidacja krzyżowa dla parametru C.perc

W celu wyboru najlepszego parametru C.perc użyliśmy walidacji krzyżowej, z podziałem zbioru testowego na pięć podzbiorów. Do predykcji użyliśmy lasu losowego. Poniżej znajdują się wykresy F1 oraz czułości dla wybranych parametrów.

```
k <- 5
n <- nrow(dane)
params <- seq(0.07, 0.3, 0.02)
cv.matrix.f1 <- matrix(NA, k, length(params),
                       dimnames = list(NULL, paste(params)))
cv.matrix.rec <- matrix(NA, k, length(params),
                        dimnames = list(NULL, paste(params)))
folds <- sample(rep(1:k, length = n))

for (param in params){
  for (f in 1:k){
    dane_train_cv <- dane_train[f != folds, ]
    dane_test_cv <- dane_train[f == folds, ]
    DANE_UNDER_CV <- RandUnderClassif(y_train~., dat = dane_train_cv,
                                      C.perc = list( No = param))

    forest_cv_under <- randomForest(DANE_UNDER_CV[, -21],
                                    DANE_UNDER_CV[, 21],
                                    dane_test_cv[,-21],
                                    dane_test_cv[, 21])

    y_pred_cv_under <- forest_cv_under$test$predicted

    f1 <- F1_Score(dane_test_cv[, 21], y_pred_cv_under,
                   positive="Yes")
    cv.matrix.f1[f, paste(param)] = f1
    rec <- Recall(dane_test_cv[, 21], y_pred_cv_under,
```

```
                    positive="Yes")
      cv.matrix.rec[f, paste(param)] = rec
    }
  }

cv.matrix.f1
f1s <- apply(cv.matrix.f1, 2, mean)
recs <- apply(cv.matrix.rec, 2, mean)
par(mfrow=c(1,2))
plot(params, f1s, main="F1")
plot(params, recs, main="REC")

par(mfrow = c(1,1))
plot(params, f1s,ylim = c(0.5,1), type = "b")
lines(params, recs,type = "b", col = "red")
```





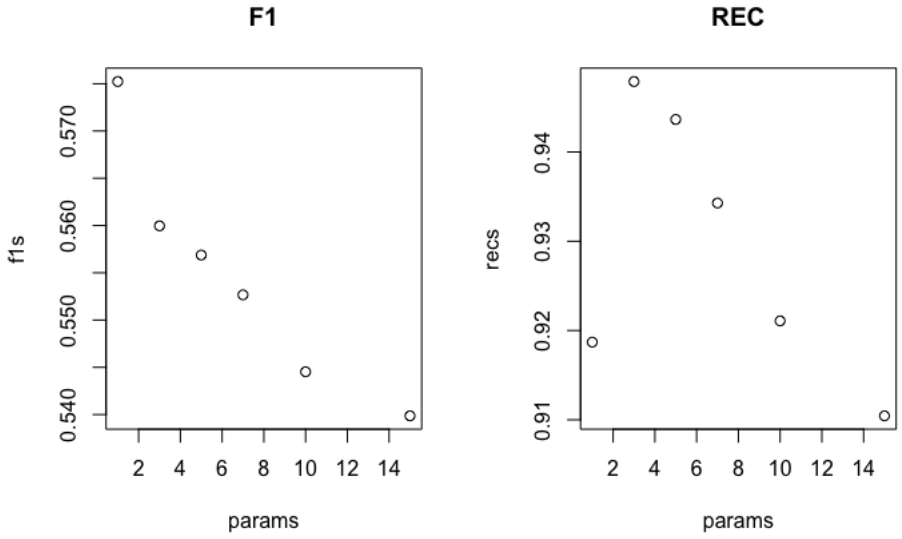

Na wykresach widać, że najlepszą czułość osiągniemy biorąc najmniejszy rozważany parametr odpowiadający podziałowi wykorzystującemu tyle samo elementów z klasy negatywnej co pozytywnej.

## 4.2 Walidacja krzyżowa dla XGBoost

Następnie użyliżny walidacji krzyżowej w celu wybrania najlepszego parametru max.depth dla modelu XGBoost. Podobnie jak poprzednio podzieliliśmy zbiór testowy na pięć podzbiorów.

```
k <- 5
n <- nrow(x_train)
params <- c(1, 3, 5, 7, 10, 15)
cv.matrix.f1 <- matrix(NA, k, length(params),
                       dimnames = list(NULL, paste(params)))
cv.matrix.rec <- matrix(NA, k, length(params),
                        dimnames = list(NULL, paste(params)))
folds <- sample(rep(1:k, length = n))

for (param in params){
  for (f in 1:k){
    x_train_cv <- x_train[f != folds, ]
    y_train_cv <- y_train[f != folds, ]
    x_test_cv <- x_train[f == folds, ]
    y_test_cv <- y_train[f == folds, ]

    dane_train_cv <- data.frame(x_train_cv, y_train_cv)
```

```
    dane_train_u_cv <- RandUnderClassif(y_train_cv~., dat = dane_train_cv)

    x_train_cv <- as.matrix(dane_train_u_cv[, -21])
    y_train_cv <- as.matrix(dane_train_u_cv[, 21])

    xgb.model <- xgboost(data = x_train_cv, label = y_train_cv,
                         max.depth = param, eta = 1, nthread = 2,
                         nrounds = 100,
                         objective = "binary:logistic")

    pred <- predict(xgb.model, x_test_cv)
    y_pred_cv <- as.factor(ifelse(pred>0.5, 1, 0))
    f1 <- F1_Score(y_test_cv, y_pred_cv, positive="1")
    cv.matrix.f1[f, paste(param)] = f1
    rec <- Recall(y_test_cv, y_pred_cv, positive="1")
    cv.matrix.rec[f, paste(param)] = rec
  }
}

cv.matrix.f1
f1s <- apply(cv.matrix.f1, 2, mean)
recs <- apply(cv.matrix.rec, 2, mean)
par(mfrow=c(1,2))
plot(params, f1s, main="F1")
plot(params, recs, main="REC")
```

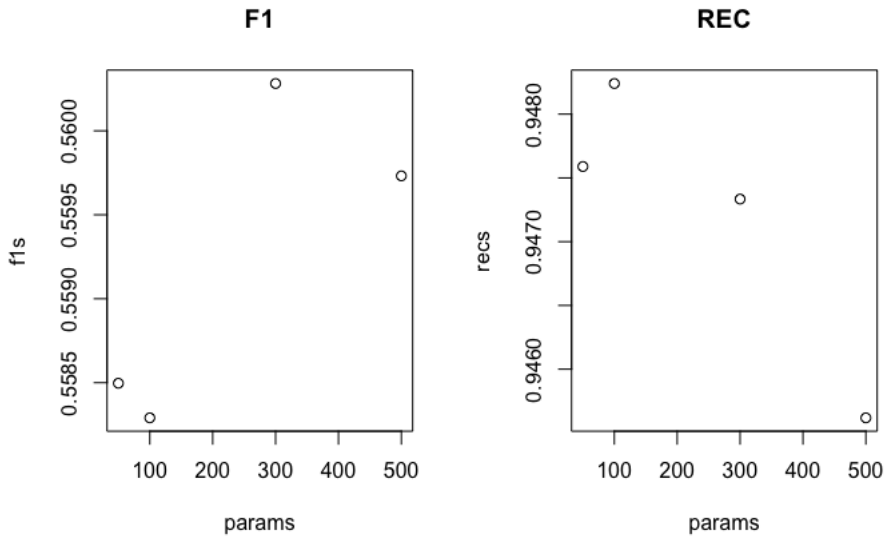



Analizując powyższe wykresy, zdecydowaliśmy się wybrać parametr max.depth

równy 3.

Następnie przeprowadziliśmy walidację krzyżową dla parametru nrounds.

```
    k <- 5
n <- nrow(x_train)
params <- c(50, 100, 300, 500)
cv.matrix.f1 <- matrix(NA, k, length(params),
                       dimnames = list(NULL, paste(params)))
cv.matrix.rec <- matrix(NA, k, length(params),
                        dimnames = list(NULL, paste(params)))
folds <- sample(rep(1:k, length = n))

for (param in params){
  for (f in 1:k){
    x_train_cv <- x_train[f != folds, ]
    y_train_cv <- y_train[f != folds, ]
    x_test_cv <- x_train[f == folds, ]
    y_test_cv <- y_train[f == folds, ]

    dane_train_cv <- data.frame(x_train_cv, y_train_cv)
    dane_train_u_cv <- RandUnderClassif(y_train_cv~., dat = dane_train_cv)

    x_train_cv <- as.matrix(dane_train_u_cv[, -21])
    y_train_cv <- as.matrix(dane_train_u_cv[, 21])

    xgb.model <- xgboost(data = x_train_cv, label = y_train_cv,
                         max.depth = 3, eta = 1, nthread = 2,
                         nrounds = param,
                         objective = "binary:logistic")

    pred <- predict(xgb.model, x_test_cv)
    y_pred_cv <- as.factor(ifelse(pred>0.5, 1, 0))
    f1 <- F1_Score(y_test_cv, y_pred_cv, positive="1")
    cv.matrix.f1[f, paste(param)] = f1
    rec <- Recall(y_test_cv, y_pred_cv, positive="1")
    cv.matrix.rec[f, paste(param)] = rec
  }
}

cv.matrix.f1
f1s <- apply(cv.matrix.f1, 2, mean)
recs <- apply(cv.matrix.rec, 2, mean)
par(mfrow=c(1,2))
plot(params, f1s, main="F1")
plot(params, recs, main="REC")
```




Zdecydowaliśmy się wybrać model zbudowany z 100 drzew, ponieważ posiada on największą czułość.

# 5 Wybór miedzy lasem i XGB

W celu wyboru najepszego modelu porównaliśmy las losowy z XGBoost, z wykorzystaniem parametrów wybranych w walidacjach krzyżowych.

Wyniki dla Lasu Losowego:

```
          No    Yes class.error
No   85932 11201   0.11531611
Yes    384  7341   0.04970874
F1_score: 0.5589523
Recall: 0.9502913
```

Wyniki dla modelu XGBoost:

```
        y_pred
y_true      0      1
     0 86073 11060
     1   370  7355
     F1_score: 0.5627391
     Recall: 0.9521036
```

# 6 Podsumowanie

Z powodu wysokiego niezbalansowania naszego zbioru danych, las losowy oraz XGBoost nie dały zadowalających wyników. W celu poprawy zdolności predykcyjnych modeli, zdecydowaliśmy się użyć undersamplingu polegającego na większym zbalansowaniu klas w zbiorze uczącym. W celu wybrania najlepszego podziału użyliśmy walidacji krzyżowej, która pokazała że najlepsze wyniki otrzymamy biorąc podział zawierający tyle samo klas negatywnych co pozytywnych. Następnie w celu dostrojenia modeli po raz kolejny użyliśmy walidacji krzyżowej dla różnych parametrów w modelach.

Końcowa analiza pokazała, że najlepsze wyniki otrzymujemy używając modelu XGBoost. Za jego pomocą maksymalnie wyeliminowaliśmy błąd, w którym algorytm źle przewidzi, że pacjent przeżyje oraz jego błąd jest najmniejszy.