***Projekt 2***

**Ada Hryniewicka**

*1. Eksporacja*

a. Wczytanie danych

```
X_train <- read.csv('/Users/ads/Desktop/SAD/projekt2/1/X_train.csv', header = TRUE)
X_test <- read.csv('/Users/ads/Desktop/SAD/projekt2/1/X_test.csv', header = TRUE)
y_train <- read.csv('/Users/ads/Desktop/SAD/projekt2/1/y_train.csv', header = TRUE)
```

Sprawdzenie ilości obseracji i zmiennych w danych.

```
dim(X_train)
```

```
## [1] 3794 9000
```

```
dim(X_test)
```

```
## [1]  670 9000
```

```
dim(y_train)
```

```
## [1] 3794    1
```

Zbiór treningowy objaśniający objaśniany składa się z 3794 obseracji (w tym przypadku ilość komórek) oraz z 9000 wartości (genów). Natomiast zbiór treningowy objaśniany składa się z z 3794 obseracji i 1 wartości. Zbiór testowy objaśniający ma 670 oberwacji i 9000 wartości. Wymiary macierzy są sensowne, jednak należy poddać je sprawdzeniu w kierunku braków danych.

```
sum(is.na(X_train))
```

```
## [1] 0
```

```
sum(is.na(X_test))
```

```
## [1] 0
```

```
sum(is.na(y_train))
```

```
## [1] 0
```

```
summary(unique(colnames(X_train)))
```

```
##    Length     Class      Mode
##      9000 character character
```

```
summary(unique(colnames(X_test)))
```

```
##    Length     Class      Mode
##      9000 character character
```

```
summary(unique(colnames(y_train)))
```

```
##    Length     Class      Mode
##         1 character character
```

```
#str() daje pełną informację o danych w każdym wierszu, jednak ze względu na estety
kę raportu zostało zakomentowane
#str(X_test)
#str(y_train)
```

Dane są kompletne i w kolumnach zawierają się dane numeryczne.

b. Rozkład empiryczny zmiennej objaśnianej.

Podstawowe statystyki:

```
cat("Srednia:", mean(y_train$CD36))
```

```
## Srednia: 1.00913
```

```
cat("\nMediana:", median(y_train$CD36))
```

```
##
## Mediana: 0.2534892
```

```
cat("\nOdchylenie standardowe:", sd(y_train$CD36))
```

```
##
## Odchylenie standardowe: 1.355825
```

```
cat("\nKwantyle:\n")
```

```
##
## Kwantyle:
```

```
quantile(y_train$CD36, c(0.1, 0.25, 0.5, 0.75, 0.9))
```

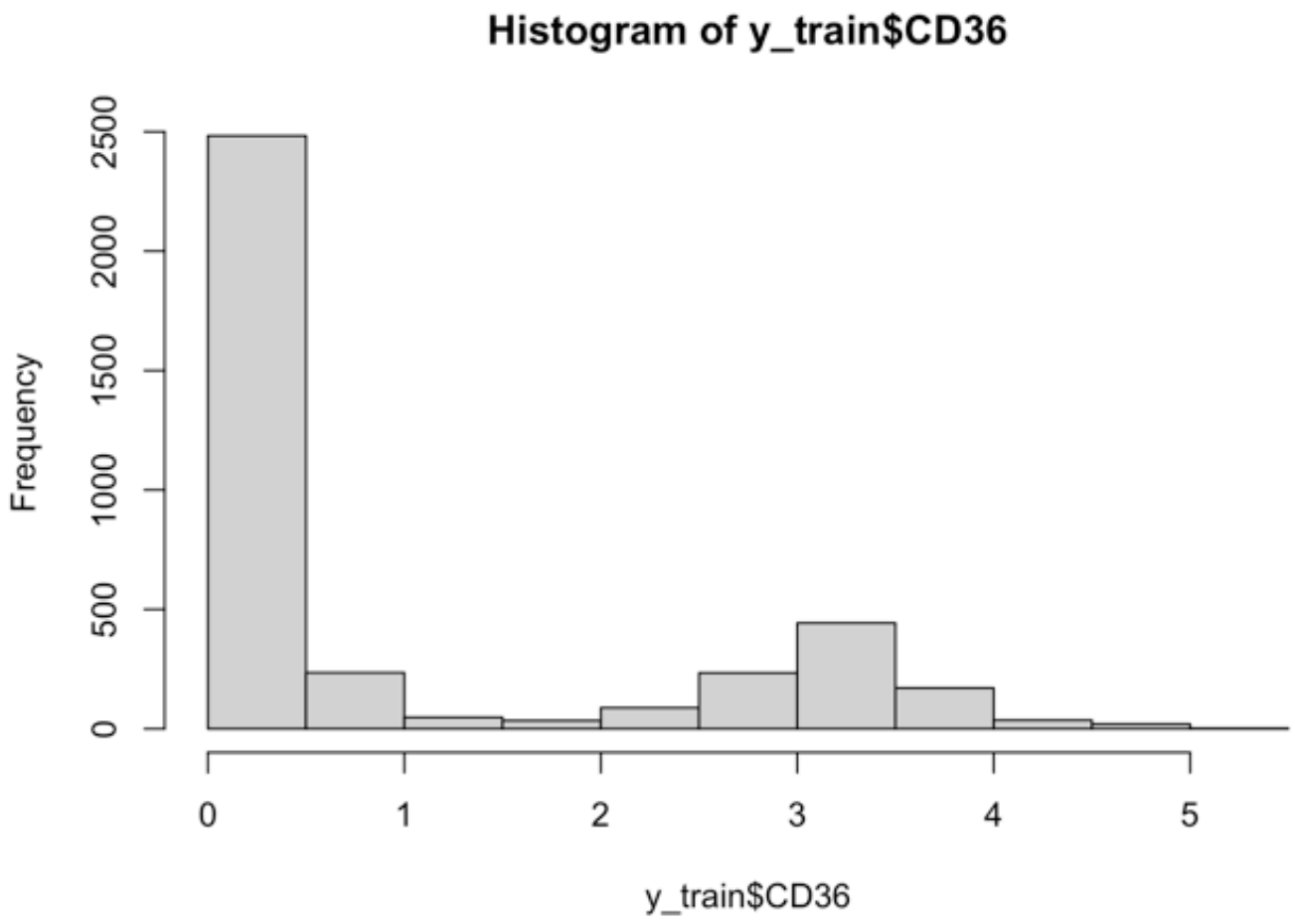
```
##        10%        25%        50%        75%        90%
## 0.00000000 0.09177912 0.25348917 2.30203277 3.32073979
```

Histogram:

```
colnames(y_train)
```

```
## [1] "CD36"
```

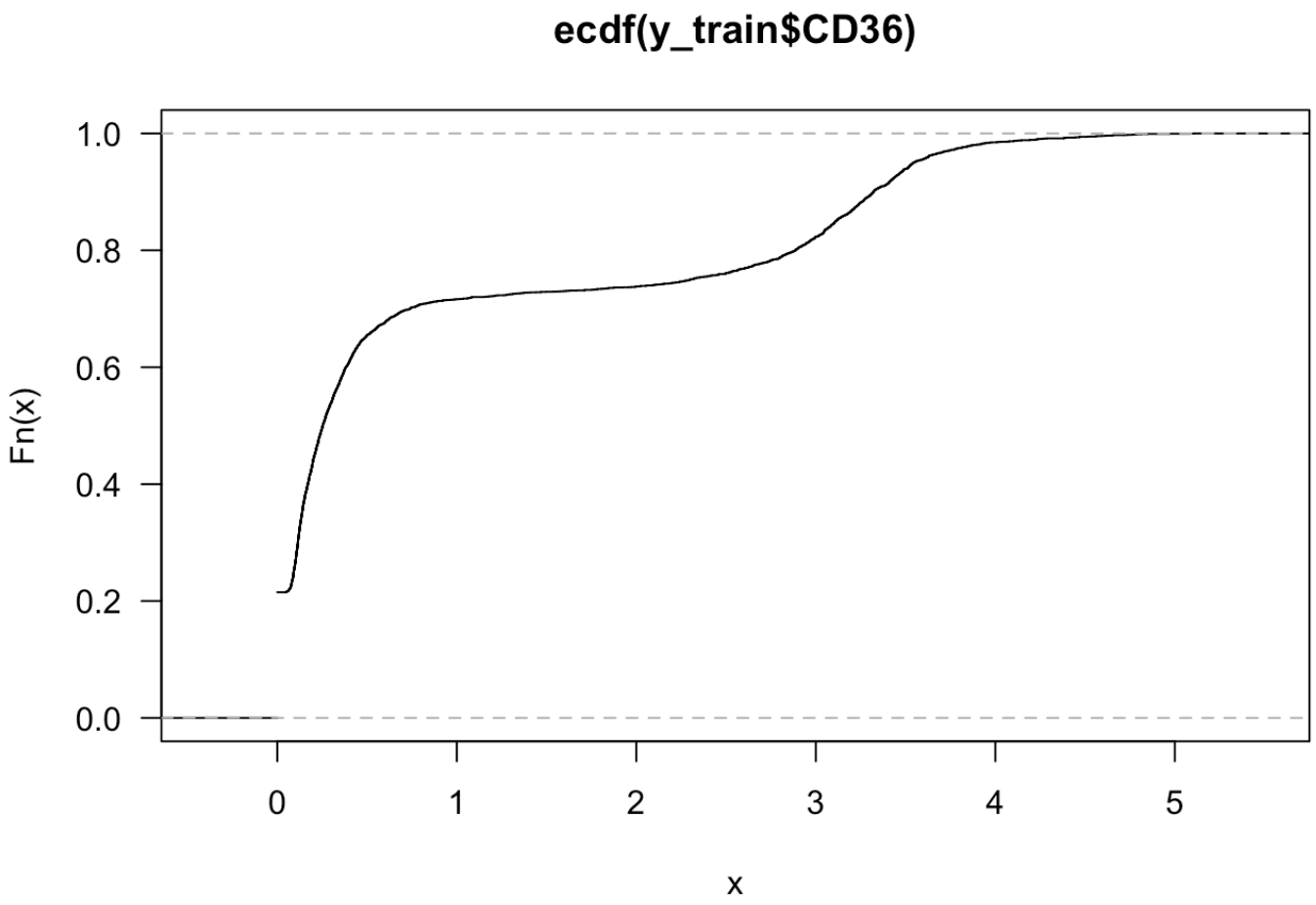
```
hist(y_train$CD36)
```



Po histogramie widać, że zdecydowanie nie mamy do czynienia z rozkładem normalnym, a większość danych skupiona jest wokół wartości 0.

Dystrybuanta empiryczna:

```
plot(ecdf(y_train$CD36), las=1)
```




Ze względów na czytelność wyników na wykresie pokazane zostało 100 najbardziej skorelowanych zmiennych. Liczbą tą można manipulować zmieniając wartość w wierszu 78.

```
library(data.table)
corMatrix <- cor(X_train, y_train$CD36)
ordered<-setDT(melt(corMatrix))[order(value)][c(0:100)] #w tym miejscu można zmienić wartość top genów branych do macierzy korelacji
```

```
## Warning in melt(corMatrix): The melt generic in data.table has been passed a
## matrix and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as well.
## To continue using melt methods from reshape2 while both libraries are attached,
## e.g. melt.list, you can prepend the namespace like reshape2::melt(corMatrix). In
## the next version, this warning will become an error.
```

```
top250<-ordered$Var1

cormat <- round(cor(X_train[c(top250)]),2)
library(reshape2)
```

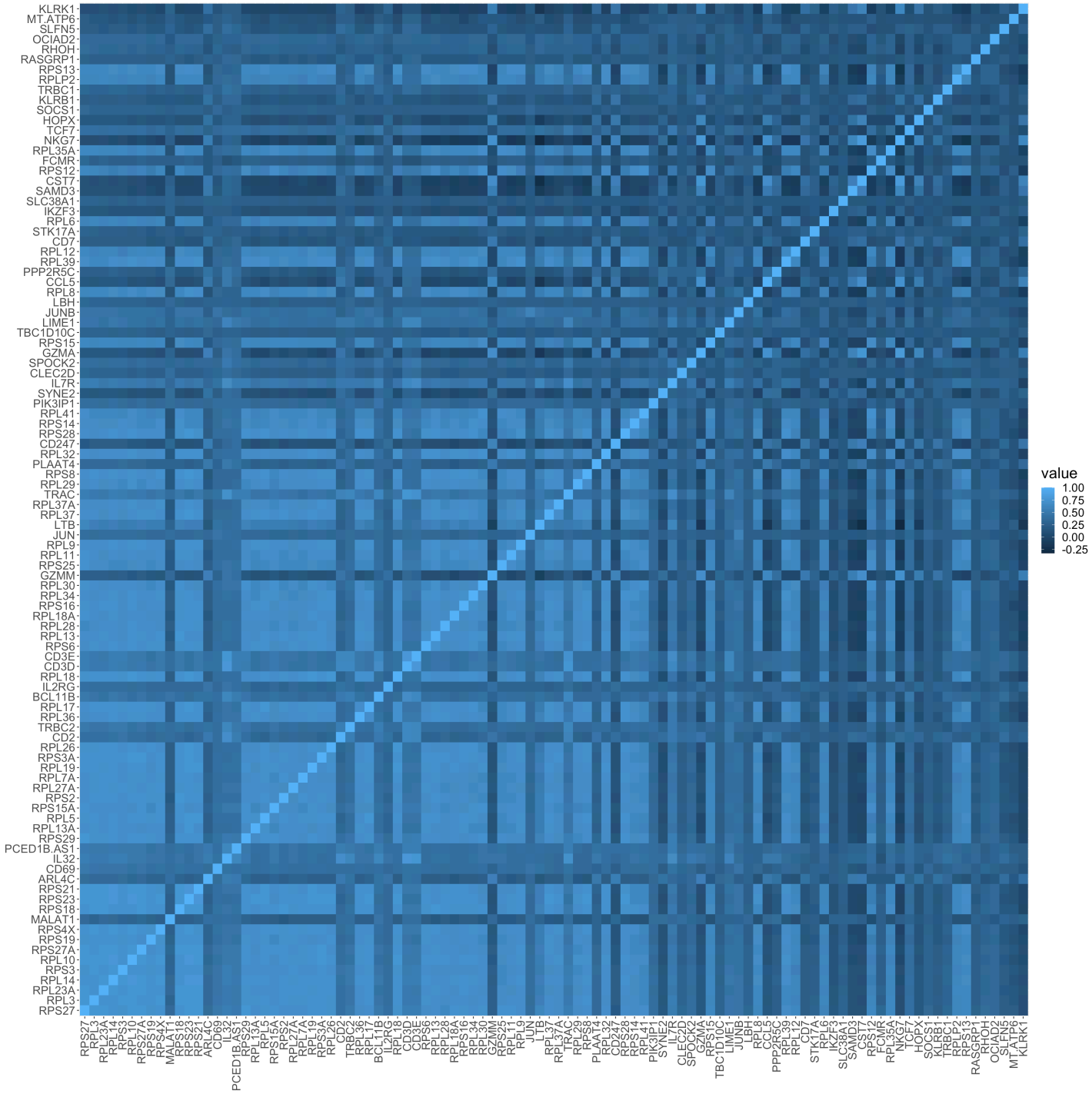
```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```
melted_cormat <- melt(cormat)
head(melted_cormat)
```

```
##     Var1  Var2 value
## 1  RPS27 RPS27  1.00
## 2   RPL3 RPS27  0.76
## 3 RPL23A RPS27  0.77
## 4  RPL14 RPS27  0.74
## 5   RPS3 RPS27  0.71
## 6  RPL10 RPS27  0.75
```

```
library(ggplot2)
#opcja umożliwiająca zapis wykresu
#png(filename="cor_heatmap.png", width=600, height=600)
ggplot(data=melted_cormat,aes(x = Var1, y = Var2, fill = value )) + geom_tile() + t
heme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1,size=15), axis.tex
t = element_text(size=15), axis.title=element_text(size=0,face="bold"),legend.text
= element_text(size=15),legend.title = element_text(size=20))
```




*2. ElasticNet*

a. Regresja grzbietowa i lasso służa do zapobiegania przetrenowania danych. Różnią się one sposobem dodawania biasu- tworzenia nowej lekko zaburzonej funkcji regresji, aby zmniejszyć wariancję. Regresja lasso może ściągnąć prosta do 0, nastomiast regresja grzbietowa może to zrobić tylko asympotycznie do 0, nie osiągając tej wartości. Dlatego lasso jest lepsze dla modeli w których jest więcej wartości, które są nieużyteczne dla modelu, bo funkcja je wyzeruje. Gdy mamy dużo parametrów i nie wiemy, czy będą przydatne, czy nie można zastosować model ElasticNet. ElasticNet łączy składniki kary z modelu regresji grzbietowej(L2) oraz z modelu lasso (L1). Kiedy alfa jest równa 0 mamy regresje grzbietową, a jeśli jest równa 1 mamy model lasso. Parametr lambda jest współczynnikiem ściągającym.

b. Przygotowanie danych pod metodę ElasticNet oraz wybranie siatki parametrów. Hiperparametrami w przypadu tej funkcji jest alfa i lambda. Dodatkowo dokonany został podział dancych na walidacyjne i treningowe.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
set.seed(42)


X <- as.matrix(X_train)
#skalowanie jednak w tym przypadku psuło wyniki dla przewidywań z kaggle, ale znacz
nie poprawiało szybkość trenowania lasów
#X <- scale(X)

n<-nrow(X_train)
train_rows <- sample(1:n, .9*n)
x.train <- X[train_rows, ]
x.test <- X[-train_rows, ]


Y <- as.matrix(y_train)
#Y <- scale(Y)

y.train <- Y[train_rows]
y.test <- Y[-train_rows]
```

#Elastic_Net_1

W metodzie ElasticNet zastosowane zostały dwa rodzaje podejśc 1, z lambda dobieraną przez model, gdzie zakres alf jest narzucony przez badacza. Natomiast drugie podejście zakładało narzucenie przez badacza obu wartości parametrów.

```
set.seed(42)
alphalist <- seq(0,1,by=0.1)

elasticnet <- lapply(alphalist, function(a){
  cv.glmnet(x.train, y.train, alpha=a, family="gaussian", type.measure ='mse', nfol
ds=10 )
})

results<-data.frame()
for (i in 1:11) {
  temp <- data.frame(alpha=(i-1)/10,cvm=min(elasticnet[[i]]$cvm), lambda=min(elasti
cnet[[i]]$lambda.1se))
  results <- rbind(results, temp)
  }
results
```

```
##    alpha       cvm      lambda
## 1    0.0 0.1429501 11.64562788
## 2    0.1 0.1241827  0.22335269
## 3    0.2 0.1229309  0.11167635
## 4    0.3 0.1226749  0.06475359
## 5    0.4 0.1208097  0.05330024
## 6    0.5 0.1251894  0.04902586
## 7    0.6 0.1229696  0.03553349
## 8    0.7 0.1225116  0.03501847
## 9    0.8 0.1235759  0.03064116
## 10   0.9 0.1216031  0.02261229
## 11   1.0 0.1233744  0.02035107
```

Parametr dotyczący ilości foldów został wybrany jako 10. Zapewnia on wystarczającą ilość, aby ocenić błąd i nie jest obciążający czasowo.

Najniższy błąd jest dla alfa równego 0.4. Walidacja krzyżowa modelu oraz dodatkowe sprawdzenie RMSE modelu dla danych testowych niebiorących udziału w walidacji.

```
set.seed(42)
alpha03.fit<- cv.glmnet(x.train,y.train, type.measure='mse', alpha=0.4, family='gau
ssian', nfolds=10)
alpha03.predicted<- predict(alpha03.fit, s=alpha03.fit$lambda.1se, newx=x.test)
cat("\nRMSE:")
```

```
##
## RMSE:
```

```
mean((y.test-alpha03.predicted)^2)
```

```
## [1] 0.1862567
```

#Elastic_Net_2 Drugie podejście z wybranymi parametrami alfa i lambda, sprawdzonymi przy pomocy walidacji krzyżowej (k=10). Zwiększono liczbę walidacji, aby poprawić jakość modelu.

```
set.seed(42)
custom <- trainControl(method = "cv",number = 10)


en <- train(x.train,
            y.train,
            method='glmnet',
            tuneGrid =expand.grid(alpha=c(0,0.1,0.4,0.7,1),
            lambda = c(0,0.1,0.3,0.4,0.5)),
            trControl=custom)
en
```

```
## glmnet
##
## 3414 samples
## 9000 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3073, 3071, 3073, 3073, 3072, 3073, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  RMSE       Rsquared   MAE
##   0.0    0.0     0.3772882  0.9257669  0.2672086
##   0.0    0.1     0.3772882  0.9257669  0.2672086
##   0.0    0.3     0.3772882  0.9257669  0.2672086
##   0.0    0.4     0.3772882  0.9257669  0.2672086
##   0.0    0.5     0.3772882  0.9257669  0.2672086
##   0.1    0.0     0.3529930  0.9329255  0.2515703
##   0.1    0.1     0.3526480  0.9330814  0.2513183
##   0.1    0.3     0.3663070  0.9301565  0.2630665
##   0.1    0.4     0.3784393  0.9271638  0.2725952
##   0.1    0.5     0.3909888  0.9242728  0.2825389
##   0.4    0.0     0.3537573  0.9325875  0.2524042
##   0.4    0.1     0.3739759  0.9276142  0.2690922
##   0.4    0.3     0.4672041  0.9059730  0.3422052
##   0.4    0.4     0.5182226  0.8944281  0.3841700
##   0.4    0.5     0.5653075  0.8865422  0.4270325
##   0.7    0.0     0.3539497  0.9325149  0.2526832
##   0.7    0.1     0.4055001  0.9191775  0.2931072
##   0.7    0.3     0.5681135  0.8816732  0.4281132
##   0.7    0.4     0.6443368  0.8712411  0.5029302
##   0.7    0.5     0.7210220  0.8636374  0.5793939
##   1.0    0.0     0.3541323  0.9324450  0.2528564
##   1.0    0.1     0.4433896  0.9075532  0.3206766
##   1.0    0.3     0.6520425  0.8676191  0.5105787
##   1.0    0.4     0.7576902  0.8584027  0.6171296
##   1.0    0.5     0.8727606  0.8428327  0.7289084
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.1.
```

```
cat('\nBłąd walidacyjny (uśredniony) wyniósł:')
```

```
##
## Błąd walidacyjny (uśredniony) wyniósł:
```

```
mean(en$resample$RMSE)
```

```
## [1] 0.352648
```

```
final.pred <- predict(en, as.data.frame(x.test))
final.error <- mean((y.test - final.pred)^2)
cat('\nBłąd testowy:')
```

```
##
## Błąd testowy:
```

```
final.error
```

```
## [1] 0.1866015
```

*3. Lasy losowe*

Hiperparametrami wybieranymi w tym modelu są ntree- liczba drzew i mtry-ilość zmiennych losowo samplowanych jako kandydaci na każdy podział. Dodatkowym sposobem na wybór lepszych parametrów do modelu zastosowango funkcje tuneRF, która wybiera najlepszą wartość mtry, dzięki czemu zamiast wybierać tę wartość samemu wykorzystano te funkcję. Pozwoliło to również oszczędzić obliczeń dla modeli o podanych parametrach mtry przez badacza.

```
library(caTools)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

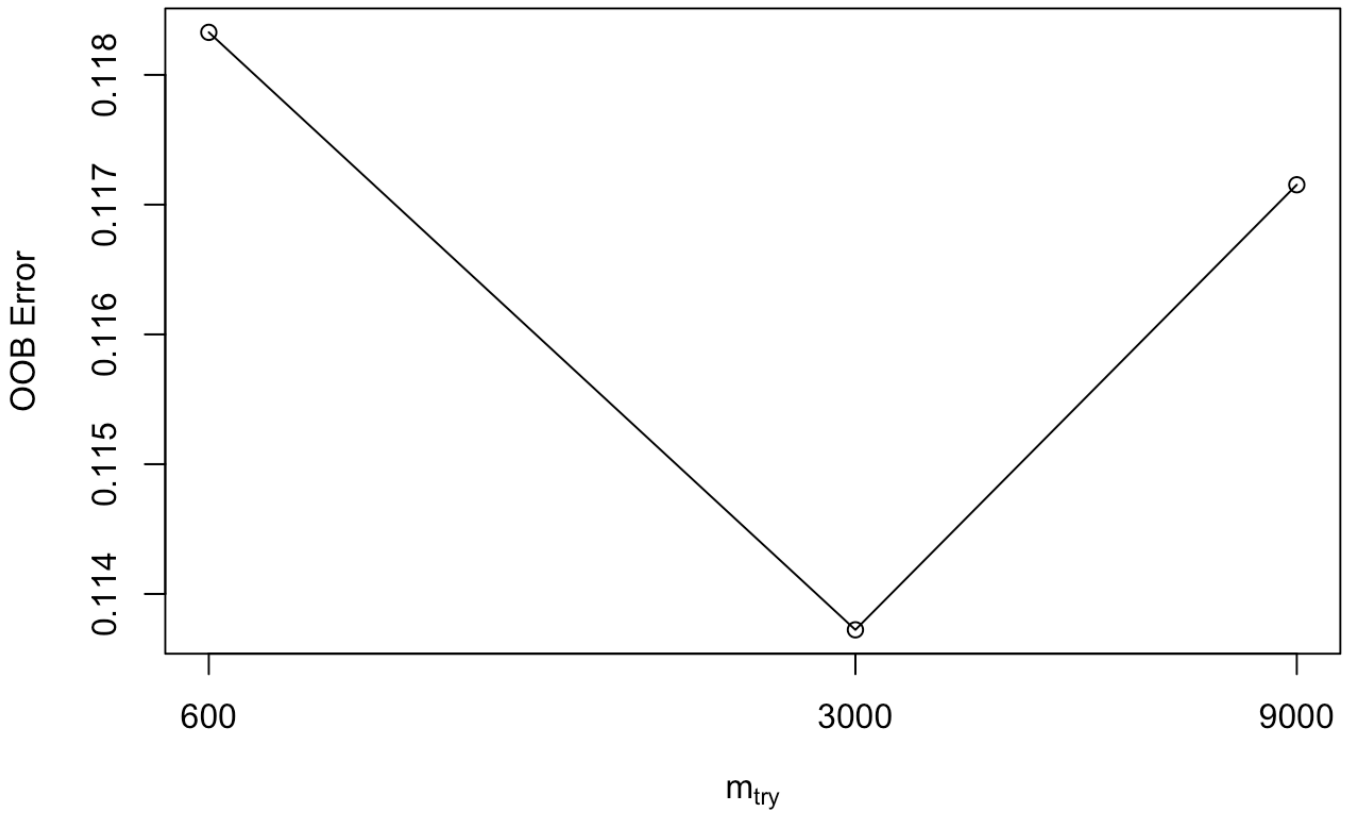
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
bestmtry <- tuneRF(x.train, y.train, stepFactor=5, improve=1e-3)
```

```
## mtry = 3000  OOB error = 0.1137236
## Searching left ...
## mtry = 600   OOB error = 0.1183278
## -0.04048644 0.001
## Searching right ...
## mtry = 9000  OOB error = 0.1171532
## -0.03015806 0.001
```




```
print(bestmtry)
```

```
##      mtry  OOBError
## 600   600 0.1183278
## 3000 3000 0.1137236
## 9000 9000 0.1171532
```

Dodatkowo trenowanie dla siatki parametrów składających się z różnych wartości. Dla większych wartości parametru nTree drzewo ma mniejszy błąd. Organiczenie ilości drzew w tym wypadku spowodowane jest ograniczeniami sprzętowymi.

```
mtry<-3000
treelist <- c(1,5,10)
control <- trainControl(method="cv", number=5)
tunegrid <- expand.grid(.mtry=mtry)

tree_search<-lapply(treelist, function(a){
rf_gridsearch <- train(x.train,y.train  ,method="rf", tuneGrid=tunegrid, trControl=
control, ntree=a)
print(rf_gridsearch)})
```

```
## Random Forest
##
## 3414 samples
## 9000 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2730, 2733, 2730, 2731, 2732
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.5936183  0.8166624  0.3516688
##
## Tuning parameter 'mtry' was held constant at a value of 3000
## Random Forest
##
## 3414 samples
## 9000 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2732, 2731, 2731, 2731, 2731
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.3656051  0.9276392  0.2509552
##
## Tuning parameter 'mtry' was held constant at a value of 3000
## Random Forest
##
## 3414 samples
## 9000 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2731, 2731, 2731, 2732, 2731
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.3513677  0.9336237  0.245097
##
## Tuning parameter 'mtry' was held constant at a value of 3000
```

Model z wybranymi najlepszymi parametram został wytrenowany na k=10, tak jak w przypadku ElasticNet.

```
control <- trainControl(method="cv", number=10)
mtry <- 3000 #wartość z funkcji tune RF
tunegrid <- expand.grid(.mtry=mtry)
rf_best <- train(x.train,y.train ,method="rf", tuneGrid=tunegrid, trControl=control
, ntree=10)
print(rf_best)
```

```
## Random Forest
##
## 3414 samples
## 9000 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3074, 3071, 3071, 3074, 3073, 3073, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.3437556  0.9365986  0.2389626
##
## Tuning parameter 'mtry' was held constant at a value of 3000
```

```
p<-predict(rf_best, x.test)
sqrt(mean((y.test-p)^2))
```

```
## [1] 0.3602593
```

Model referencyjny dla ElasticNet.

```
set.seed(42)
predictions <- predict(en, x.test)
cat('\nRMSE dla modelu referencyjnego ElasticNet:')
```

```
##
## RMSE dla modelu referencyjnego ElasticNet:
```

```
sqrt(mean((mean(y_train$CD36) - predictions)^2))
```

```
## [1] 1.248239
```

Model referencyjny dla Regression Tree.

```
set.seed(42)
predictions_tree <- predict(rf_best, x.test)
cat('\nRMSE dla modelu referencyjnego Regression Tree:')
```

```
##
## RMSE dla modelu referencyjnego Regression Tree:
```

```
sqrt(mean((mean(y_train$CD36) - predictions_tree)^2))
```

```
## [1] 1.237854
```

Tabela podsumuwująca wyniki ze wszystkich modeli, gdzie porównane są wartości RMSE w każdym etapie walidacji krzyżowej.

```
comp_table<-merge(en$resample, rf_best$resample, by='Resample', all.x=FALSE)
comp_table<-comp_table[c('Resample', 'RMSE.x', 'RMSE.y')]
colnames(comp_table) <- c('Sample', 'ElasticNet', 'Tree')
comp_table
```

```
##    Sample ElasticNet      Tree
## 1  Fold01  0.3558847 0.3493946
## 2  Fold02  0.3400802 0.3369832
## 3  Fold03  0.3303341 0.3596426
## 4  Fold04  0.3539005 0.3432413
## 5  Fold05  0.3176907 0.3465751
## 6  Fold06  0.3946314 0.3703726
## 7  Fold07  0.3372118 0.3273301
## 8  Fold08  0.3586257 0.3232253
## 9  Fold09  0.3482004 0.3110877
## 10 Fold10  0.3899205 0.3697033
```

*4. Predykcja na zbiorze testowym*

Ze względu na wyniki osiągnięte w poprzednich podpunktach oraz wydajność obliczeniową wybrany został model ElasticNet, który dla danych zakresów parametrów wybrał optymalne i został poddany walidacji krzyżowej o parametrze k=10.

```
predictions_final<- predict(en, as.data.frame(X_test))
predictions_final<-as.data.frame(predictions_final)
predictions_final$Id <- 0:(nrow(predictions_final)-1)
colnames(predictions_final) <- c('Expected', 'Id')
predictions_final <- as.data.frame(predictions_final[, c(2,1)])
write.csv(predictions_final,"/Users/ads/Desktop/SAD/projekt2/pred_1.csv", row.names
= FALSE)
```