PROJEKT 1, STATYSTYKA 2, 2016Z

Piotr Podolski i Michał Skrzypecki

Spis Treści:

Celem projektu było znalezienie reguły decyzyjnej, która miałaby pomóc w ocenie, czy dany pacjent z nowotworem Glioblastoma przeżyje najbliższy rok.
Na podstawie otrzymanych wyników nie udało nam się ocenić dalszego przeciętnego trwania życia pacjenta.

```
install.packages("rmarkdown")
 library(rmarkdown)

load("~/Desktop/GlioblastomaWide.rda")

 data <- GlioblastomaWide

data_kaggle <- read.table(file = "~/Desktop/GlioblastomaWide.csv", sep=",", dec=".", header =
TRUE)
data_test_kaggle <- read.table(file = "~/Desktop/final.csv", sep=",", dec=".", header = TRUE)

#data_kaggle <- read.table(file = "https://inclass.kaggle.com/c/glioma-survival-predictions/
download/GlioblastomaWide.csv", sep=",", dec=".", header = TRUE)
#data_test_kaggle <- read.table(file = "https://inclass.kaggle.com/c/glioma-survival-predictions/
download/final.csv", sep=",", dec=".", header = TRUE)
```

Etap 1

```
-------------------------przygotowanie danych----------------------------------

data$Cluster<-factor(data$Cluster)
data$death1y <- 1-(as.numeric(as.factor(data$death1y)) - 1)

for(i in 1:16115){
  if(mean(is.na(data.frame[,16120-i])) > 0.2)
{     data[,16120-i] <- NULL
  } else {
    sr <- mean(na.omit(data[,16120 - i]))
    for(j in 1:125)
{       if(is.na(data[j, 16120 -
i])){
        data[j, 16120 - i] <- sr
      }
    } } }
```

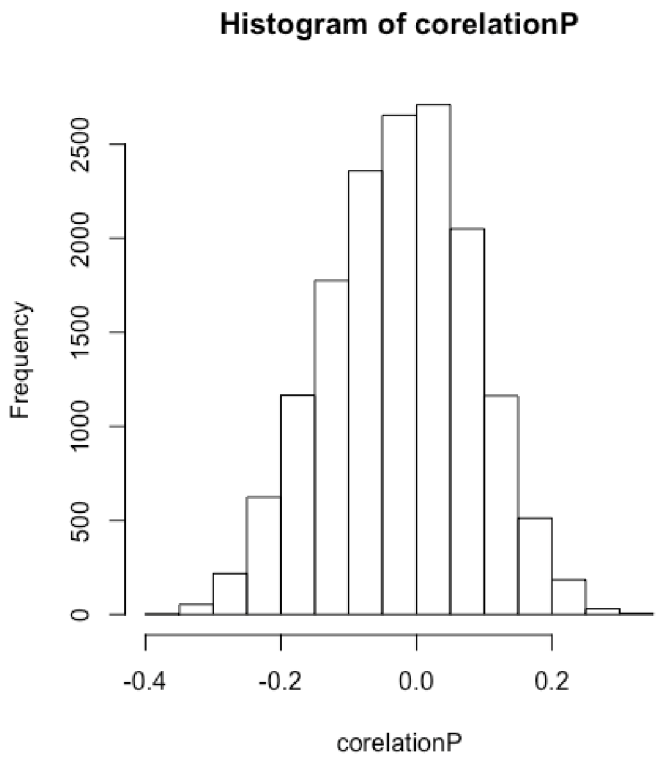
```
# dane o wieku

summary_age<-summary(data$age)
boxplot(data$age, col="grey", horizontal =
TRUE) hist(data$age, col = "grey")
plot(ecdf(data$age), las=1)
```

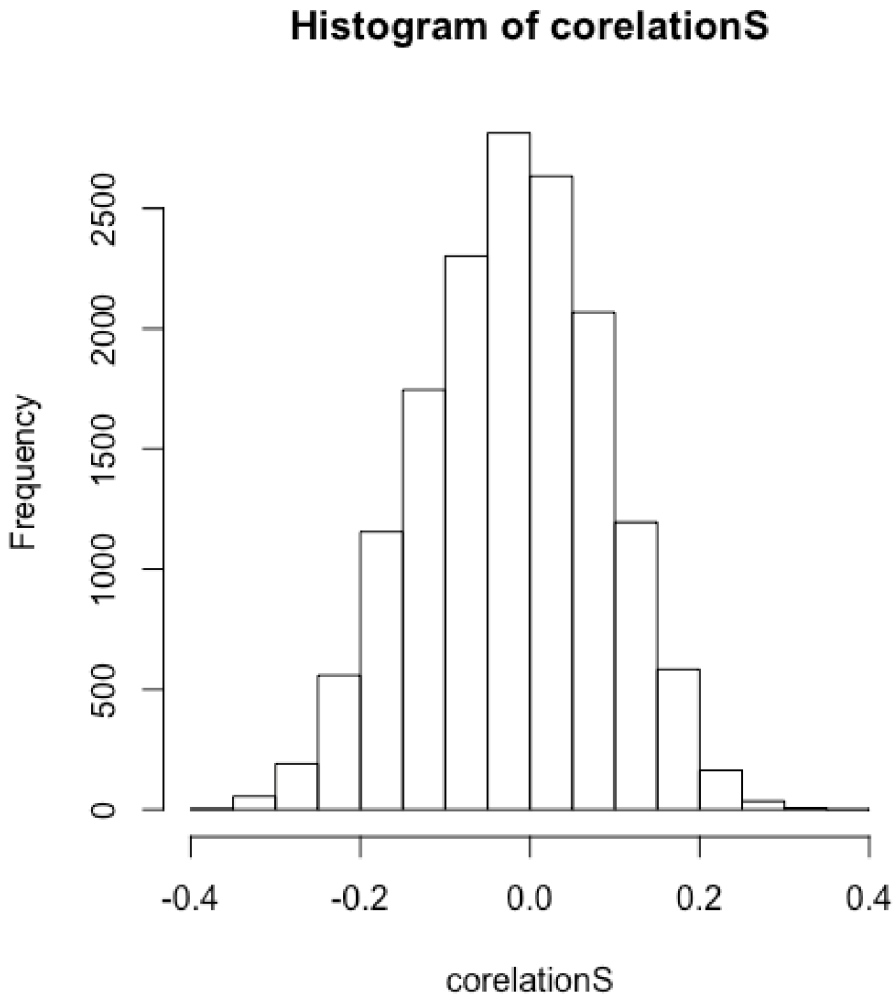
```
#dzielimy dane na dwa zbiory, alive i
dead alive <- data[data$death1y == 1,]
dead <- data[data$death1y == 0,]
```

Posłużyliśmy się **testem korelacji Pearsona** i **testem korelacji Spearmana** w celu zbadania, czy istnieje jakiś wpływ ekspresji genów na możliwość przeżycia przez pacjenta pierwszego roku od diagnozy.

```
#definiuje funkcje korelacji spearmana
corelation_function_spearman <- function(arg){
  return(cor(data$death1y, arg, use = "complete.obs",method ="spearman"))
}

#definiuje funkcje korelacji pearsona
corelation_function_pearson <- function(arg){
  return(cor(data$death1y, arg, use = "complete.obs",method ="pearson"))
}

#mapuje funkcje korelacji na danych
corelationS <- sapply(data[,5:ncol(data)], FUN=corelation_function_spearman)

corelationP <- sapply(data[,5:ncol(data)], FUN=corelation_function_pearson)

hist(corelationS)
hist(corelationP)
```

Poniżej przedstawiamy otrzymany histogram obu korelacji.





Na podstawie wyżej wspomnianych testów wybraliśmy **30 najbardziej istotnych genów**. Poniżej przedstawiamy ich listę.

W przypadku korelacji Pearsona:
[1] “PLAUR” [2] ‘’LRRC61” [3] ‘’LSP1” [4] ‘’NRG1” [5] “AQP9” [6] “STAG2”
[7] “CTSB” [8] “ACTR3C” [9] “MYBPC2” [10] “SLC17A9” [11] “HK3” [12] “CTF1”
[13] “SPP1” [14] “MAP2K3” [15] “TBL1XR1” [16] “IL27RA” [17] “MXRA5” [18] “TMEM44”
[19] “BHLHE40” [20] “SLC11A1” [21] “SH3BP2” [22] “OSMR” [23] “RPL39L” [24] “C8orf58”
[25] “LOXL1” [26] “SLC16A3” [27] “TNFSF14” [28] “RASSF1” [29] “ZC3H12A”
[30] “CLCF1”
W przypadku korelacji Spearmana:
[1] “SLC17A9” [2] “PLAUR” [3] “TBL1XR1” [4] ”LSP1” [5] ”CSNK1D” [6] ‘’CLEC5A”
[7] “NRG1” [8] “CTSB” [9] “LRRC61” [10] “HK3” [11] “AQP9” [12] “MLPH”
[13] “ACTR3C” [14] “RPL39L” [15] “STC1” [16] “SH3BP2” [17] “SLC11A1” [18] “MAP2K3”
[19] “GPNMB” [20] “CTF1” [21] “IFI44L” [22] “TM7SF4” [23] “SLC6A6” [24] “C8orf58’
[25] “BHLHE40” [26] “MXRA5” [27] “CPPED1” [28] “BCL3” [29] “FCRLB” [30] “POM121L9P”

Rzućmy okiem na powyższe wyniki. Zauważmy, że następujące geny znajdują się w obu pierwszych trzydziestkach:

[1] ‘’PLAUR” [2] ‘’LRRC61” [3] ‘’NRG1” [4] ‘’AQP9” [5] ‘’CTSB” [6] ”ACTR3C”
[7] ‘’SLC17A9” [8] ‘’HK3” [9] ‘’MAP2K3” [10] ‘’TBL1XR1” [11] ‘’MXRA5” [12] „BHLHE40”
[13] ‘’SLC11A1” [14] ‘’SH3BP2” [15] „RPL39L” [16] ‘’C8orf58”

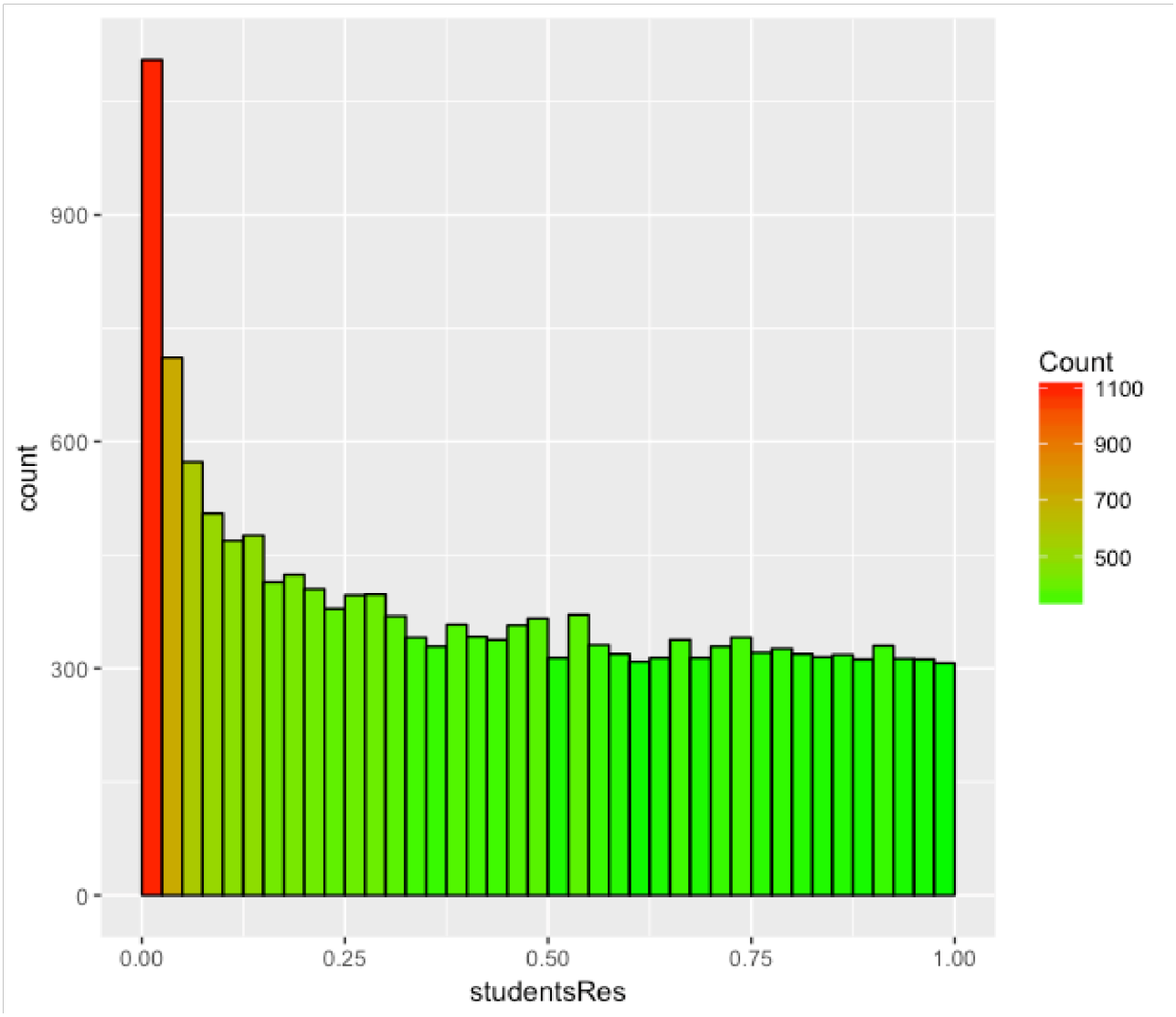
Niektóre geny z powyższej listy zajmują bardzo wysokie miejsca w obu ,,rankingach korelacyjnych”.

```
# wypisuje 20 genow o najwyzszych wartosciach topS<-
names(head(sort(abs(corelationS),decreasing = TRUE),30)) topP<-
names(head(sort(abs(corelationP),decreasing = TRUE),30))
plot(data[,topS[1:10]],col=c("red","green")[as.factor(data$death1y)], pch=18)
```

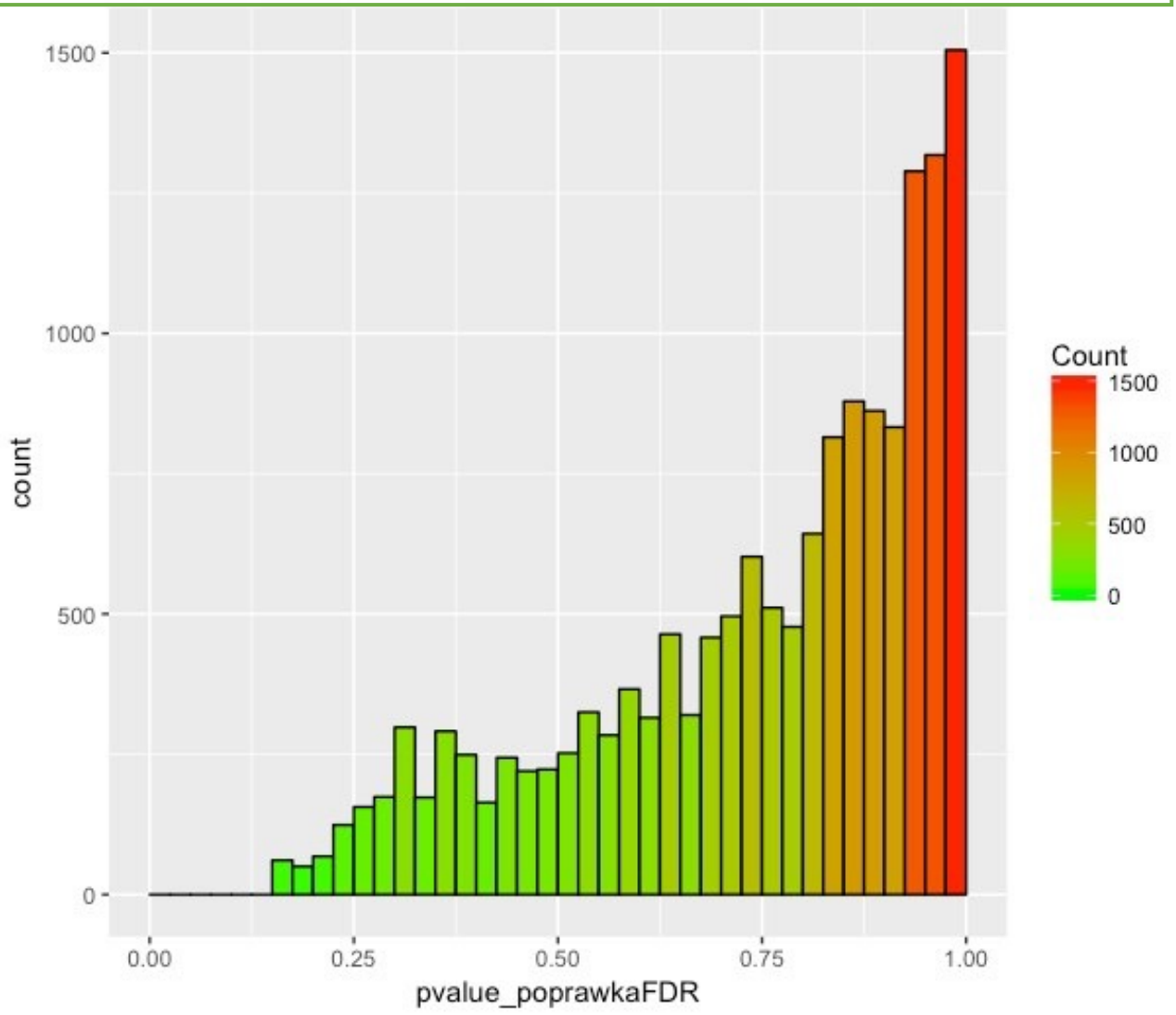
Użyliśmy też **testu t-Studenta**, przy którym dzieliliśmy ludzi na tych, którzy przeżyli rok i nie przeżyli roku. Rozpatrywaliśmy wynik tego testu dla każdego genu z osobna. Dokonaliśmy **poprawki FDR** i **poprawki Bonferroniego** p-wartości testów. p-wartości dla poprawki Bonferroniego są skupione w jedynce, a te FDR są bardziej jednostajnie rozłożone na odcinku (0,1).

```
# test t-studenta dla kazdej kolumny dla alive i dead
studentsRes<-NULL
cols <- ncol(data) for(i
in 5:cols){   a<-alive[,i]
d<-dead[,i]   t<-
t.test(d, a)$p.value
  studentsRes[(i-4)] <- t
}
pvalue_poprawkaFDR<-p.adjust(studentsRes, method = "fdr") pvalue_poprawkaBONF<-
p.adjust(studentsRes, method = "bonferroni") rpvalue_poprawkaFDR<-
as.data.frame(pvalue_poprawkaFDR) rpvalue_poprawkaBONF<-
as.data.frame(pvalue_poprawkaBONF) rpvalue<-as.data.frame(studentsRes)
```

Poniżej przedstawiamy: histogram otrzymanych wyników testu t-Studenta oraz kolejno histogramy przyjmowanych p-wartości przy poprawce FDR i poprawce Bonferroniego.



```
library(ggplot2)
ggplot(data=rpvalue, aes(studentsRes)) +
geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
 scale_fill_gradient("Count", low = "green", high = "red") ggplot(data=rpvalue_poprawkaFDR,
 aes(pvalue_poprawkaFDR)) +
 geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
 scale_fill_gradient("Count", low = "green", high = "red") ggplot(data=rpvalue_poprawkaBONF,
 aes(pvalue_poprawkaBONF)) +
 geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
 scale_fill_gradient("Count", low = "green", high = "red")
```



Etap 2

Aby ocenić, czy dany pacjent przeżyje rok posłużyliśmy się metodami: regresji logistycznej, Naive Bayes, Random Forest, drzewami klasyfikacyjnymi i Support Vector Machine. W tym celu podzieliliśmy zbiór danych **zbiór testowy** i **zbiór uczący**.

```
# zbior testowy i treningowy

#install.packages("caret", dependencies =
c("Depends")) library("caret") set.seed(1313)
```

```
data$death1y <- as.factor(data$death1y)

inds <- createDataPartition(data$death1y, p = 0.75)
train <- data[inds[[1]],]
test  <- data[-inds[[1]],]
wybrane <- c("death1y",  topS)
```

Rozpoczynamy od analizy wyników **regresji logistycznej probit**.

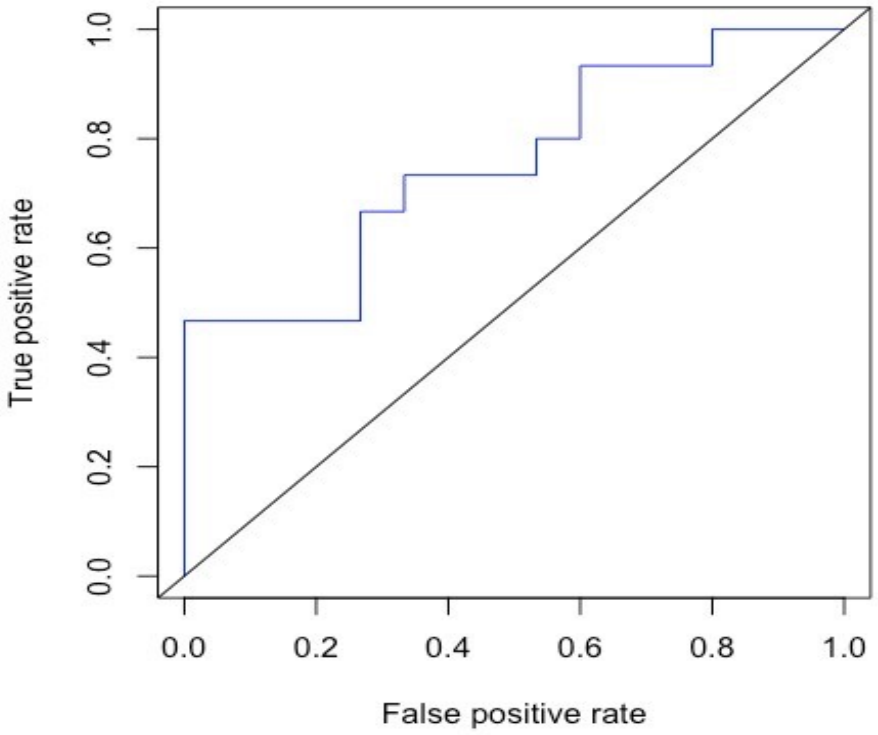
```
 # ------------------------regresja logistyczna probit-----------------------------------

modelProbit <- glm(death1y ~ .,
data=train[,wybrane],family=binomial(link="probit")) poprawionyProbit <-
step(modelProbit, method="backward") summary(modelProbit)

foldy <- createFolds(data$death1y, k = 10)
errorsProbit <- lapply(foldy, function(ind) {
  model<-step(glm(death1y ~ .,
data=train[,wybrane],family=binomial(link="probit")),method="backward")
predykcja1 <- predict(model, data[ind,wybrane],type="response")
mean(abs(predykcja1-as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsProbit)) hist(unlist(errorsProbit), col="grey")
```

Na naszych danych algorytm scorów Fishera działa bardzo dobrze (używa bardzo małej liczby iteracji). Nie wnosi to jednak nic do rozwiązania naszego problemu.
Nasz model ma **AIC=116.58**. Porównamy go potem z innymi. Ten, który będzie miał najmniejszą wartość tego kryterium będzie modelem najlepszym do zadanej nam predykcji.

```
# --------------------regresja logisyczna dla dobranego zbioru testowego i treningowego -----

model <- glm(death1y ~ ., data=train[,wybrane],family = binomial(logit))
summary(model)
poprawiony <- step(model, method="backward")


# dla k fold k = 10


foldy <- createFolds(data$death1y, k = 10)
errors <- lapply(foldy, function(ind) {
  model<-step(glm(death1y~.,data[-ind,wybrane],family="binomial"),method="backward")
predykcja1 <- predict(model, data[ind,wybrane],type="response")   mean(abs(predykcja1-
as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errors))
```

W tym przypadku **AIC** pokazuje wartość **117.22**, co świadczy, że to podejście jest gorsze od regresji logistycznej probit. Mniejsza liczba iteracji świadczy wyłącznie o dobrej zbieżności tej metody dla tego zbioru danych.

```
hist(unlist(errors), col="grey")

library (“ROCR”)

# rysowanie krzywej ROCR

library("plotROC")

pr edictionRegr<-predict(poprawiony, test)
```



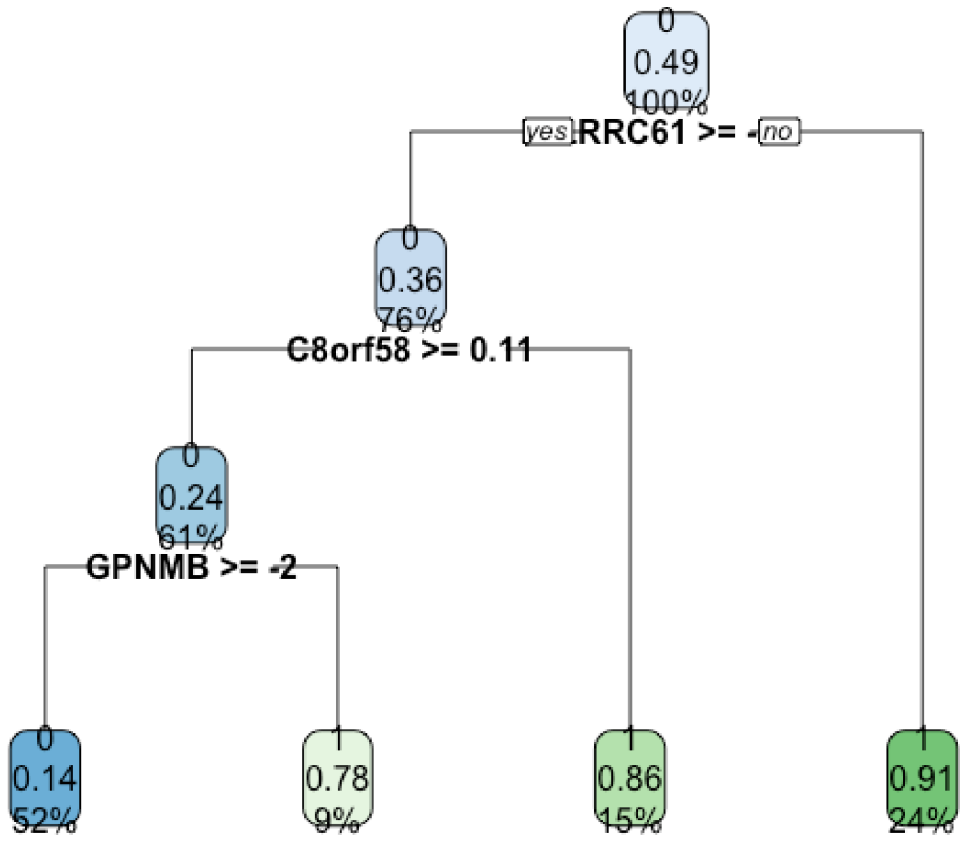
Powyżej przedstawiliśmy otrzymaną dla regresji logistycznej probit **krzywą ROC**.

Krzywa ROC jest bardziej przesunięta w lewo, co świadczy o tym, że nasz test dokonuje całkiem skutecznej klasyfikacji.

```
tabela kontyngencji

table(pred=predictionRegr, true=test$death1y)



predRegr <- prediction( predictionRegr, test$death1y) perfRegr <-
performance(predRegr,"tpr","fpr") plot(perfRegr,col="blue") abline(0,1)

library("OptimalCutpoints")
pref_df_regr <- data.frame(pred = predictionRegr, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden",
data=pref_df_regr, tag.healthy = "0") summary(oc) plot(oc, which=1)
```

Kolejną używaną metodą były **drzewa klasyfikacyjne rpart**. Poniżej przedstawiamy jej krótką analizę.

```
library("rpart")
library("rpart.plot")

# model dla wybranego zbioru testowego

drzewo <- rpart(as.factor(death1y) ~ . , data=train[,wybrane],
method="class", control=rpart.control(maxdepth = 10)) rpart.plot(drzewo)
```



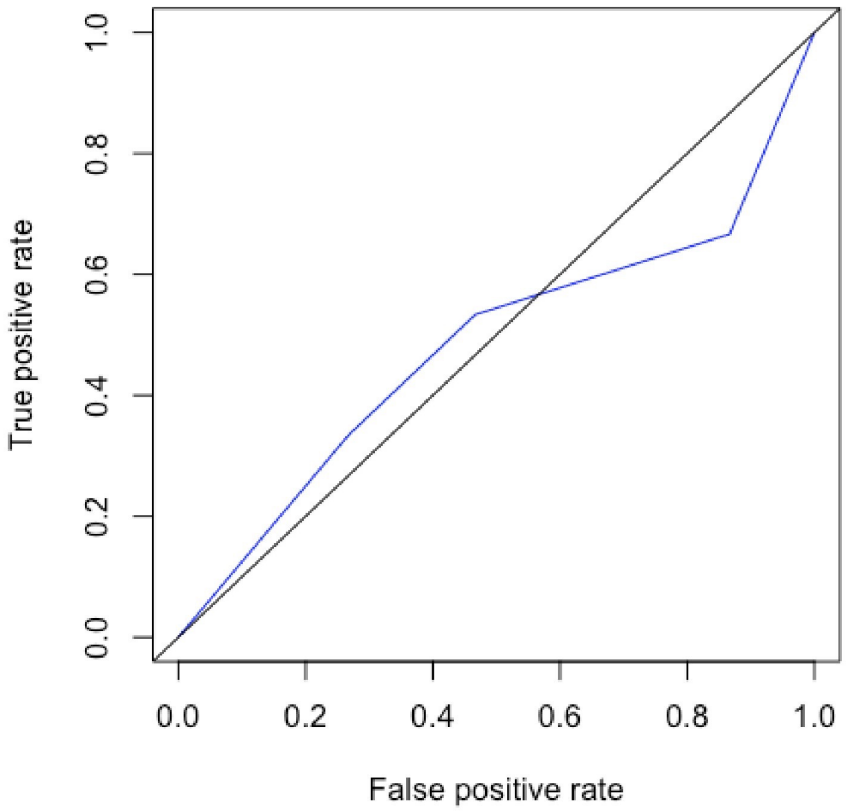
Powyżej zamieściliśmy otrzymane drzewo klasyfikacyjne.

Narysowaliśmy też podobnie jak w poprzednim przypadku krzywą ROC.

```
# rysowanie krzywej ROCR

predictionRpartTree<-predict(drzewo, test)[,2]
#tabela kontyngencji
table(pred=predictionRpartTree, true=test$death1y)
```

Krzywa ROC położona jest bliżej diagonali, zatem drzewa klasyfikacyjne łatwiej dokonują błędnego zakwalifikowania pacjentów niż metoda regresji liniowej.
Poniżej wspomniana krzywa:

```
ROC
predRpartTree <- prediction( predictionRpartTree,
test$death1y) perfRpartTree <-
performance(predRpartTree,"tpr","fpr")
plot(perfRpartTree,col="blue") abline(0,1)
```



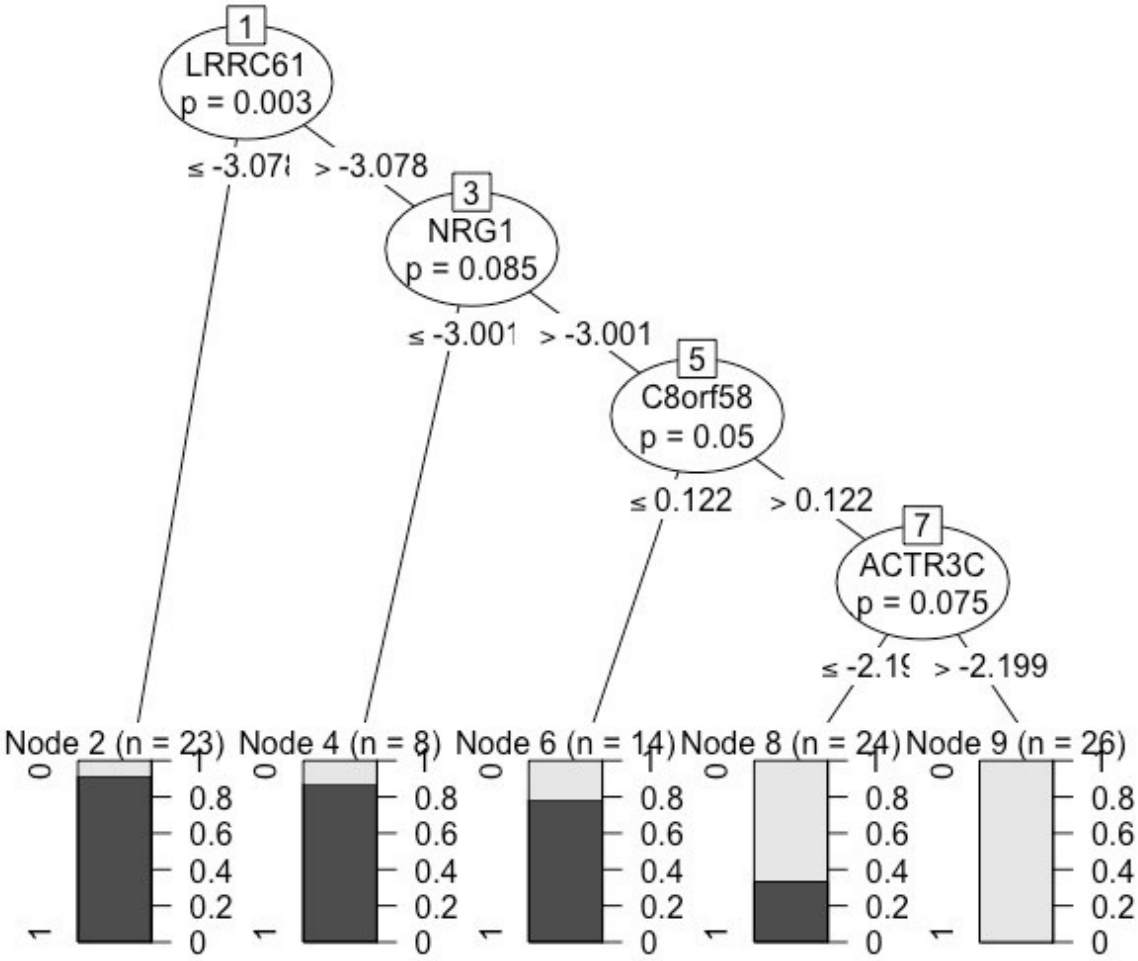
Zastosowaliśmy dziesięciokrotnie **createFolds**.

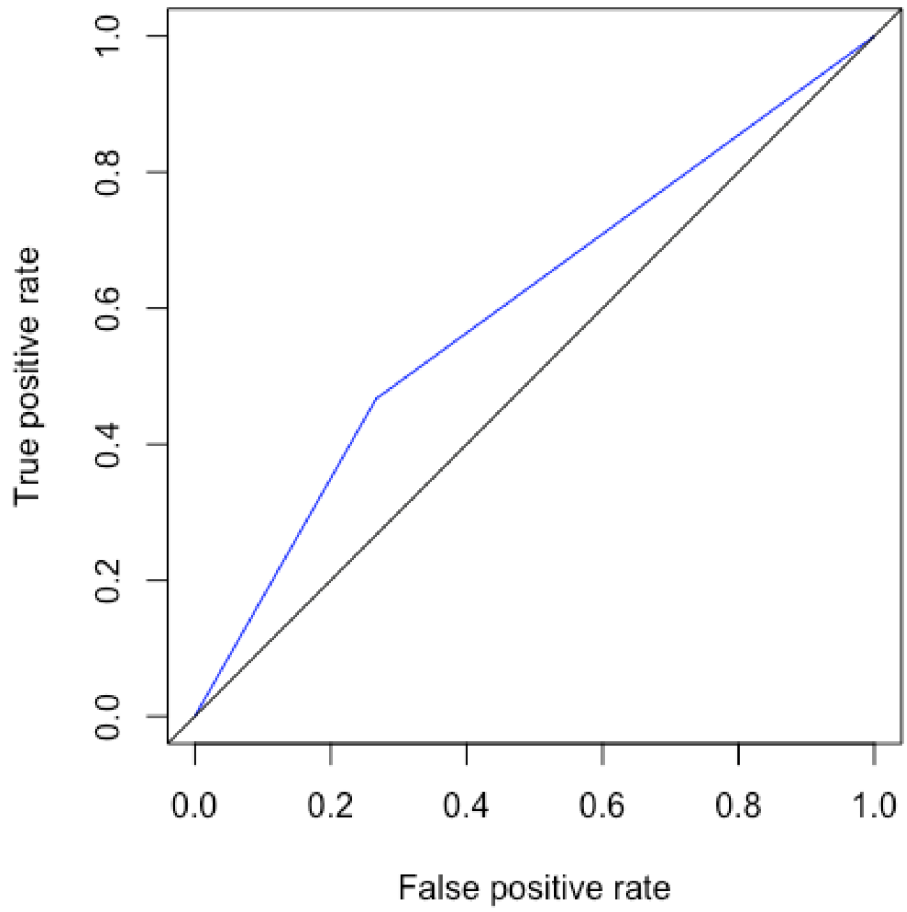
```
#optimal cutpoints
pref_df_RpartTree <- data.frame(pred = predictionRpartTree, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden",
data=pref_df_RpartTree, tag.healthy = "1") summary(oc) plot(oc, which=1) # k fold dla k=10

foldy <- createFolds(data$death1y, k = 10)
errorsRTREE <- lapply(foldy, function(ind) {
  drzewo <- rpart(death1y ~ . , data=data[-ind,wybrane],
method="class", control=rpart.control(maxdepth = 10))   predykcja1 <-
predict(drzewo, newData=data[ind,wybrane])   mean(abs(predykcja1-
as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsRTREE))
hist(unlist(errorsRTREE), col="grey")
```

Poniżej przedstawiamy wyniki użycia **metody drzewa klasyfikacyjnego ctree**.
Otrzymana krzywa ROC jest podobna do tej z drzewa klasyfikacyjnego cpart. Zatem jak na razie model regresji liniowej wychodzi najlepiej.

Zastosowaliśmy ponownie dziesięciokrotny createFolds.

```
# ---------------------------------drzewo klasyfikacyjne ctree---------------------------------


install.packages("party")
library("party")
nextTree <- ctree(as.factor(death1y) ~ ., data=train[,wybrane],
controls=ctree_control(maxdepth=10, testtype = "Bonferroni", mincriterion = 0.75))
plot(nextTree)

# rysowanie krzywej ROCR

predictionTree<-predict(nextTree, test)
#tabela kontyngencji
table(pred=predictionTree, true=test$death1y)

#krzywa ROC
predTree <- prediction( as.numeric(predictionTree),
test$death1y) perfTree <- performance(predTree,"tpr","fpr")
plot(perfTree,col="blue") abline(0,1)

# k fold dla k=10
```


1
LRRC61
p = 0.003
≤ -3.078
> -3.078
3
NRG1
p = 0.085
≤ -3.001
> -3.001
5
C8orf58
p = 0.05
≤ 0.122
> 0.122
7
ACTR3C
p = 0.075
≤ -2.199
> -2.199
Node 2 (n = 23)
Node 4 (n = 8)
Node 6 (n = 14)
Node 8 (n = 24)
Node 9 (n = 26)
0
1
0.8
0.6
0.4
0.2
0

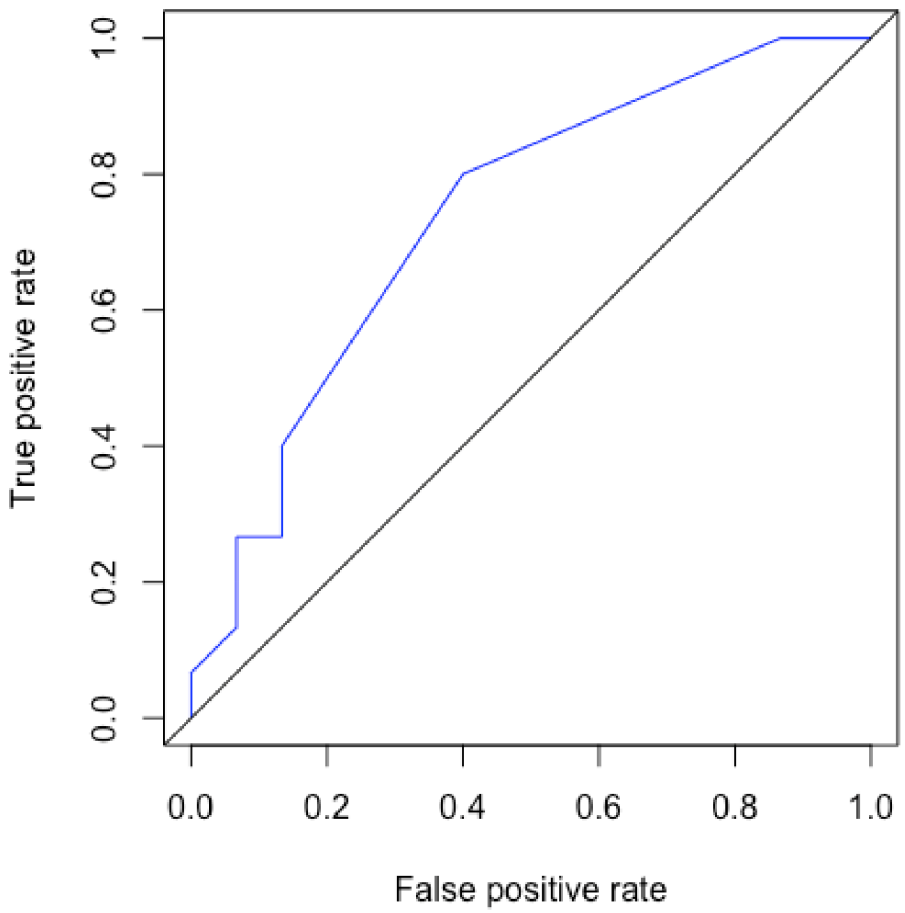



```
foldy <- createFolds(data$death1y, k = 10)
errorsCTREE <- lapply(foldy, function(ind) {
  drzewo <- ctree(as.factor(death1y) ~ ., data=train[-ind,wybrane],
controls=ctree_control(maxdepth=10, testtype = "Bonferroni", mincriterion = 0.75))
predykcja <- predict(drzewo, data[ind,wybrane],type="response")
mean(abs(as.numeric(predykcja) - as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsCTREE))
hist(unlist(errorsCTREE), col="grey")
```

Użyliśmy także **metody Naive Bayes**. Krzywa ROC w przypadku Naive Bayes jest trochę inna niż przy drzewach klasyfikacyjnych, ale wciąż skupiona w sąsiedztwie diagonali. Wciąż najlepszym sposobem według tego kryterium jest regresja liniowa.

```
# ------------------------------ naive bayes ----------------------------------------


install.packages("klaR")
install.packages("e1071"
) library("e1071")
library("klaR") tempData

<- data
```

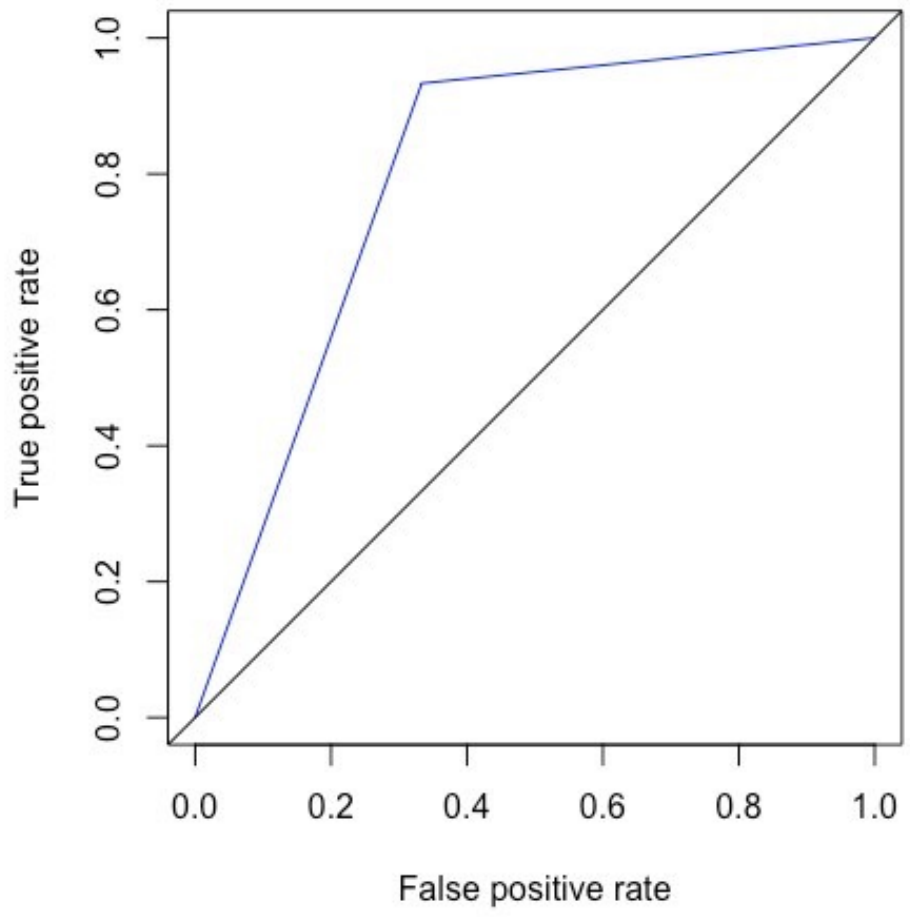
```
for(i in 1:15513){tempData[,i] <- factor(tempData[,i])}
indsTemp <- createDataPartition(tempData$death1y, p =
0.75) tempTrain <- tempData[inds[[1]],] tempTest <-
tempData[-inds[[1]],]

nb <- NaiveBayes(death1y~.,tempTrain[,wybrane])
partimat(death1y~CTSB+NRG1,data=tempTrain[,wybrane],method="naiveBayes",col.correct =
"black", col.wrong = "red",pch=15,image.colors=c("grey","blue1"))

# rysowanie krzywej ROCR

predictionNB<- as.data.frame(predict(nb, tempTest,threshold = 0.001,eps=0,type = c("class",
"raw")))[,3]
```




```
# tablica kontyngencji

table(pred=predictionNB, true=test$death1y)

#krzywa ROC
predNB <- prediction( predictionNB,
test$death1y) perfNB <-
performance(predNB,"tpr","fpr")
plot(perfNB,col="blue") abline(0,1)
```

```
# k fold dla k=10 foldy <-
createFolds(tempData$death1y, k = 10) errorsNB
<- lapply(foldy, function(ind) {   nb <-
NaiveBayes(death1y~.,tempData[-ind,wybrane])
predykcja1 <- predict(nb,
tempData[ind,wybrane],type="response")
mean(abs(as.numeric(predykcja1)-
as.numeric(tempData[ind, "death1y"])))

}) mean(as.numeric(errorsNB))
hist(unlist(errorsNB), col=„grey")
```

Użyliśmy do predykcji również **metody Random Forest**. Z otrzymanych wyników w varImpPlot możemy wnioskować o skuteczności **cross-validacji out-of-bag** po nauczeniu, ale przed wyborem próbki bootstrapowej.
Krzywa ROC pokazała, że metoda Random Forest dokonuje skuteczniejszej klasyfikacji niż drzewa klasyfikacyjne, ale nie lepszej niż model regresji liniowej.

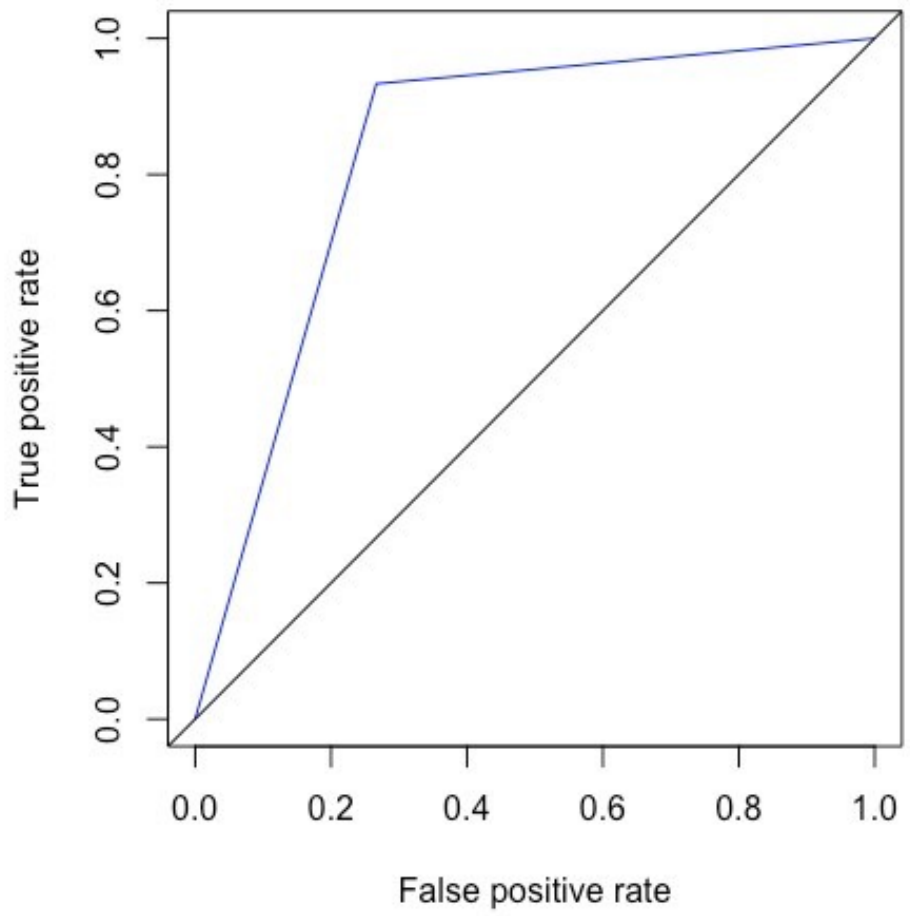


```
# ----------------------------------------random forest--------------------------

library("randomForest")
randomForestData<-
randomForest(as.factor(death1y)~.,data=train[,wybrane],importance=TRUE,proximity=TRUE)

library("plotROC")
predictionRF<-predict(randomForestData, test)
roc.estimate <- calculate_roc(predictionRF, test$death1y)

#tabela kontyngencji
table(pred=predictionRF, true=test$death1y)
```

```
varImpPlot(randomForestData)
importance(randomForestData)
library("ROCR")
pred <- prediction( as.numeric(predictionRF),
test$death1y) perf <- performance(pred,"tpr","fpr")
plot(perf,col="blue") abline(0,1)

library("OptimalCutpoints")
pref_df <- data.frame(pred = predictionRF, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden", data=pref_df, tag.healthy =
"0")
summary(oc)
plot(oc, which=1)

# k fold dla k=10
foldy <- createFolds(tempData$death1y, k = 10)
errorsRF <- lapply(foldy, function(ind) {
  randomForestData<-
randomForest(death1y~.,data=train[ind,wybrane],importance=TRUE,proximity
=TRUE)   predykcja1 <- predict(randomForestData,
data[ind,wybrane],type="response")   mean(abs(as.numeric(predykcja1)-
as.numeric(tempData[ind, "death1y"])))
})
mean(as.numeric(errorsRF))
hist(unlist(errorsRF), col=„grey")
```

Użyliśmy także metody **Support Vector Machine**, czyli **maszyny wektorów nośnych**.

```
#------------------------------------ support vector machine ----------------------

svmModel <- svm(death1y~.,data=train[,wybrane])
summary(svmModel)
predictionSVM<-predict(svmModel, test)
predSVM <- prediction( as.numeric(predictionSVM),
test$death1y) perfSVM <- performance(predSVM,"tpr","fpr")
plot(perfSVM,col="blue") abline(0,1)
```




```
#tabela kontyngencji
table(pred=predictionSVM, true=test$death1y)

# k fold dla k=10 foldy <- createFolds(tempData$death1y, k = 10)
errorsSVM <- lapply(foldy, function(ind) {   svmModel <-
svm(death1y~.,data=train[-ind,wybrane])   predykcja1 <-
predict(svmModel, data[ind,wybrane],type="response")
mean(abs(as.numeric(predykcja1)-as.numeric(tempData[ind, "death1y"])))
})
mean(as.numeric(errorsSVM))
hist(unlist(errorsSVM), col="grey")
```

Etap 3

Klasyfikatory oparte o stacking

```
# random forest + logistic regression + svm

#randomForestData<-
randomForest(death1y~.,data=train[,wybrane],importance=TRUE,proximity=TRUE)
#predictionRandomForest<-predict(randomForestData, newdata=test)
#errRF <- sqrt((sum (test$death1y-predictionRandomForest)^2)/nrow(test))

#modelLogisticRegr <- glm(death1y ~ ., data=train[,wybrane],family="binomial")
#poprawionyLR <- step(modelLogisticRegr, method="backward")
#predictionLR<-predict(poprawionyLR, test[,wybrane])
#errLR <- sqrt((sum(test$death1y-predictionLR)^2)/nrow(test))

#svmFit <- svm(death1y~.,data=train[,wybrane])
#svm_predict <- predict(svmFit, newdata=test)
#errSvm <- sqrt((sum(test$death1y-svm_predict)^2)/nrow(test))

#preds <- (predictionRandomForest + predictionLR) / 2
#errSum <- sqrt((sum(test$death1y - preds)^2)/nrow(test))

#boosting with adabag
install.packages("adabag")
library("adabag") library("mlbench")
adaboost <- boosting(death1y ~., data=train[,wybrane], boos=TRUE, mfinal=10, control = (minsplit
= 0))
summary(adaboost)

adaboost$trees
adaboost$weights
adaboost$importance
predict(adaboost,test) t1<-
adaboost$trees[[1]] rpart.plot(t1)

resAda <- predict(adaboost, newdata=data_test_kaggle) predictionADA<-
predict(adaboost, test)
predADA <- prediction( as.numeric(predictionADA),
test$death1y) perfADA <- performance(predADA,"tpr","fpr")
plot(perfADA,col="blue") abline(0,1) t2<-adaboost$trees[[2]]
rpart.plot(t2)

t3<-adaboost$trees[[3]]
rpart.plot(t3)
#boosting with xgboost
library("xgboost")
y = data$death1y == "1" X =
model.matrix(death1y~.,data)
gb <- xgboost(X,y,objective="binary:logistic", nrounds=2, max.depth=2)
gb

#laczenie modeli
library(ElemStatLearn)
library(randomForest)
library(caret)
```

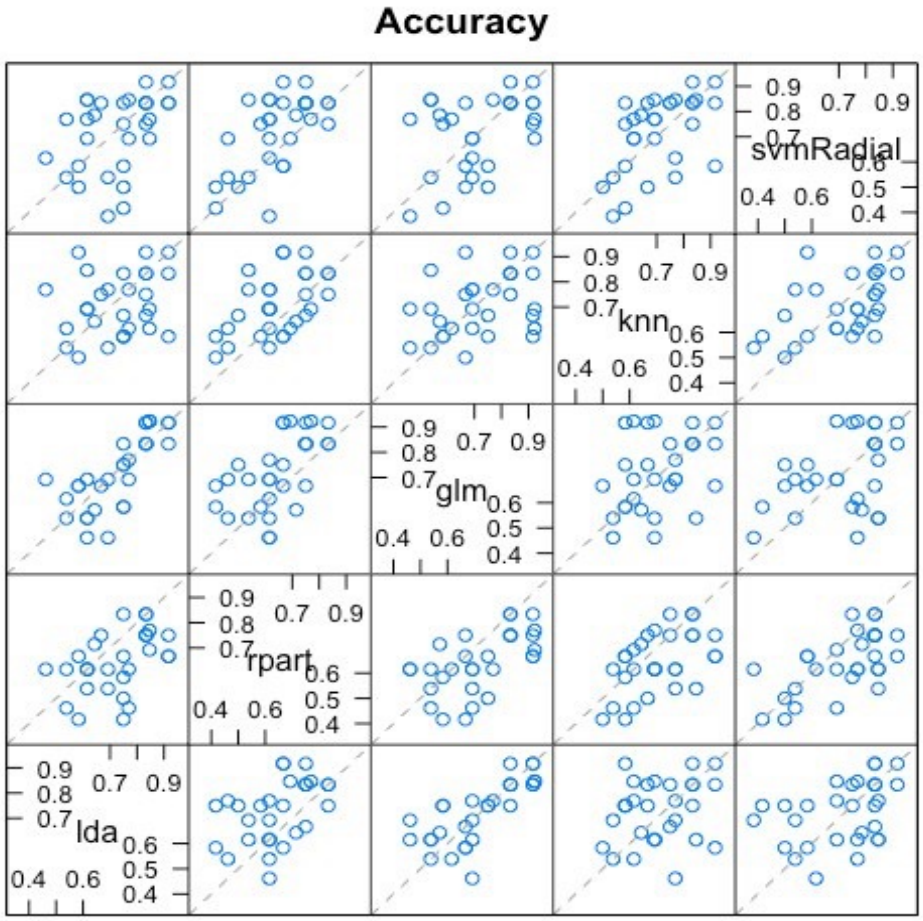
```
set.seed(33833) train$death1y <-
as.factor(train$death1y) test$death1y <-
as.factor(test$death1y)

fit1 <- train(death1y ~., data=train[,wybrane], method='rf')
fit2 <- train(death1y ~., data=train[,wybrane],
method='gbm')

results1 <- predict(fit1, newdata=test[,wybrane])
results2 <- predict(fit2, newdata=test[,wybrane])

combo <- data.frame(results1, results2, y =
test$death1y) fit3 <- train(y ~ ., data = combo, method =
"rf") results3 <- predict(fit3, newdata = test[,wybrane])

c1 <- confusionMatrix(results1, test$death1y) c2 <-
confusionMatrix(results2, test$death1y) c3 <-
confusionMatrix(results3, combo$death1y) result_final <-
predict(fit1, data_test_kaggle[,wybrane[-1]])

caretStack(models, method="glm")

library(gbm)

library(AppliedPredictiveModeling)

set.seed(62433)

# Train using 3 different models. fit1 <- train(death1y ~.,
data=train[,wybrane], method='rf') fit2 <- train(death1y ~.,
data=train[,wybrane], method='gbm') fit3 <- train(death1y
~., data=train[,wybrane], method='lda')

# Run models on testing data.
results1 <- predict(fit1, newdata=test)
results2 <- predict(fit2, newdata=test)
results3 <- predict(fit3, newdata=test)

# Stack models together and combine with random forests. combo <-
data.frame(results1, results2, results3, death1y = test$death1y) fit4 <-
train(death1y ~ ., data = combo, method = "rf") result_final_2 <-
predict(fit4, newdata = data_test_kaggle)

# Run stacked model on testing data.
results4 <- predict(fit4, newdata = test) #
random forests
c1 <- confusionMatrix(results1, test$death1y)
# boosting
c2 <- confusionMatrix(results2, test$death1y)
# lda
c3 <- confusionMatrix(results3, test$death1y)
# Stacked models
c4 <- confusionMatrix(results4, combo$diagnosis)



#----------------------- stacking ver 2 ----------------------------------
library(mlbench) library(caret) library(caretEnsemble) # create
submodels data$death1y[data$death1y=="0"]<-"dead"
data$death1y[data$death1y=="1"]<-"alive"
```

```
for(i in 1:16116){
  if(mean(is.na(data_test_kaggle[,16118-i])) > 0.2){     data_test_kaggle[,
16118-i] <- NULL
  } else {
    sr <- mean(na.omit(data_test_kaggle[,16118 -
i]))     for(j in 1:250){       if(is.na(data_test_kaggle[j,
16118 - i])){
        data_test_kaggle[j, 16118 - i] <- sr
      }
    }
  }
}

#Linear Discriminate Analysis (LDA)
#Classification and Regression Trees (CART)
#Logistic Regression (via Generalized Linear Model or GLM)
#k-Nearest Neighbors (kNN)
#Support Vector Machine with a Radial Basis Kernel Function (SVM)
#Generalized Boosted Regression Modeling

control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE,
classProbs=TRUE)
algorithmList <- c('lda', 'rpart', 'glm', 'knn', 'svmRadial', 'gbm', 'rf')

seed <- 7
set.seed(seed)


models <- caretList(as.factor(death1y)~., data=data[,wybrane],
trControl=control, methodList=algorithmList) results <- resamples(models)
summary(results) dotplot(results)
```

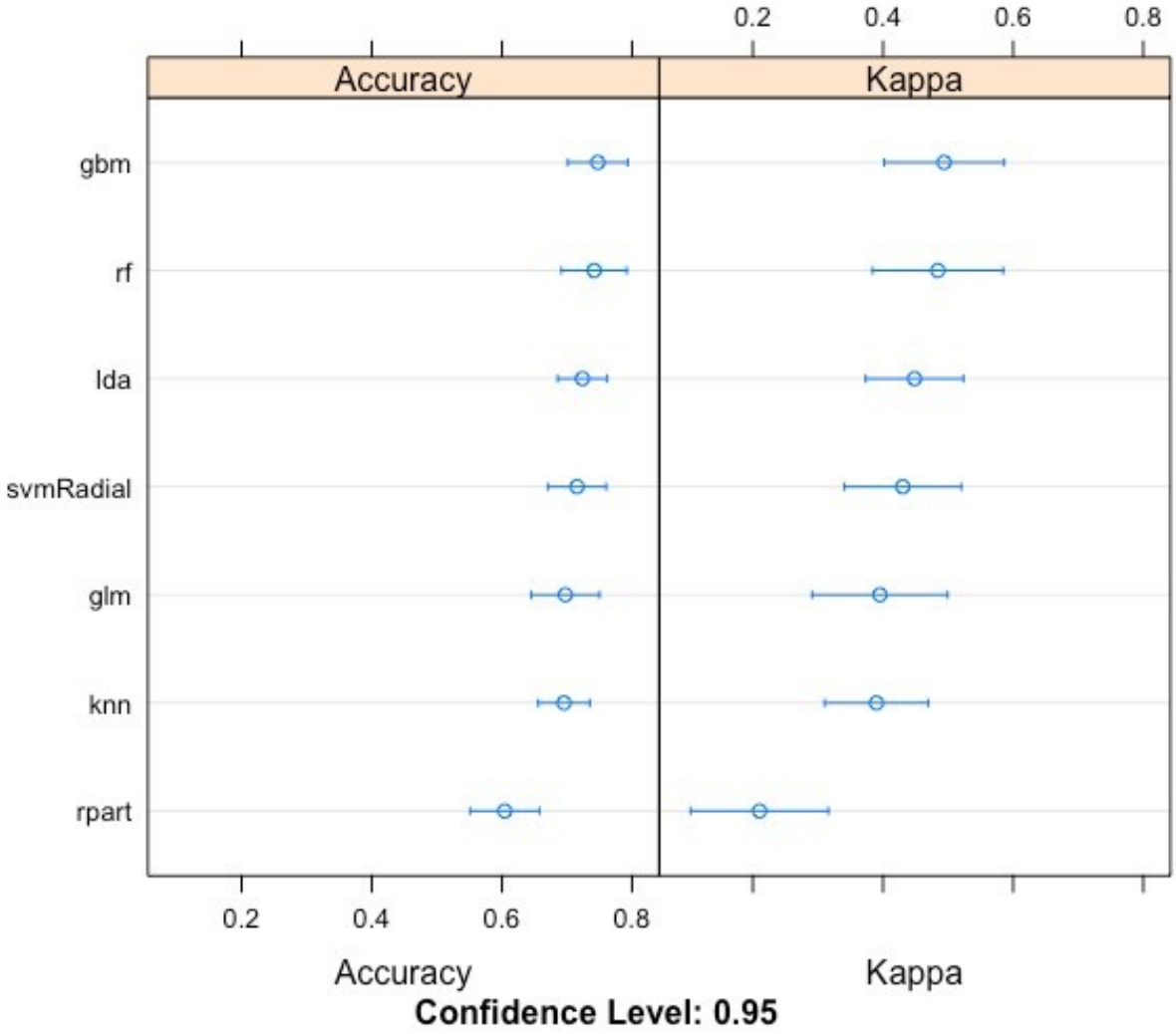
Poniżej przedstawiamy część z otrzymanych w Etapie 3. wyników.

```
                                                                    #Models
: lda, rpart, glm, knn, svmRadial, gbm, rf #Number of resamples: 30

#Accuracy
#Min. 1st Qu. Median    Mean 3rd Qu.    Max. NA's
#lda       0.5000  0.6667 0.7500 0.7238  0.7692 0.9167    0
#rpart     0.3333  0.5096 0.5833 0.6045  0.6859 1.0000    0
#glm       0.3846  0.5833 0.7212 0.6975  0.7692 1.0000    0
 #knn       0.5000  0.6282 0.6923 0.6955  0.7692 0.8571    0
#svmRadial 0.4615  0.6282 0.7500 0.7155  0.7816 0.9167    0
#gbm       0.5385  0.6667 0.7500 0.7474  0.8462 0.9231    0
#rf        0.4167  0.6667 0.7500 0.7418  0.8429 1.0000    0

#Kappa
#Min. 1st Qu. Median    Mean 3rd Qu.    Max. NA's
#lda        0.00000 0.33330 0.5000 0.4484  0.5412 0.8333    0
#rpart     -0.33330 0.01786 0.1667 0.2102  0.3690 1.0000    0
#glm       -0.20930 0.16670 0.4405 0.3952  0.5384 1.0000    0
#knn        0.00000 0.24600 0.3734 0.3900  0.5412 0.7143    0
#svmRadial -0.07059 0.25980 0.5000 0.4304  0.5639 0.8333    0
#gbm        0.07143 0.33330 0.5000 0.4939  0.6905 0.8471    0
#rf        -0.16670 0.33330 0.5000 0.4842  0.6845 1.0000    0
# correlation between results
 modelCor(results)
 splom(results)
```

```
#           lda       rpart       glm       knn      svmRadial   gbm       rf
#lda       1.0000000 0.4731959 0.7796056 0.5449545 0.3988154 0.3805589 0.4157637
#rpart     0.4731959 1.0000000 0.3628292 0.3693462 0.4375850 0.4963381 0.5671968
#glm       0.7796056 0.3628292 1.0000000 0.3849873 0.3473833 0.4984669 0.3813019
#knn       0.5449545 0.3693462 0.3849873 1.0000000 0.3786169 0.3937915 0.4517406
#svmRadial 0.3988154 0.4375850 0.3473833 0.3786169 1.0000000 0.6196128 0.7723533
#gbm       0.3805589 0.4963381 0.4984669 0.3937915 0.6196128 1.0000000 0.6925319
#rf        0.4157637 0.5671968 0.3813019 0.4517406 0.7723533 0.6925319 1.0000000
```



Macierzowy wykres punktowy




```
                                                                    # stack using glm
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3,
savePredictions=TRUE, classProbs=TRUE)
set.seed(seed)
stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=stackControl)
print(stack.glm)


# stack using random forest
set.seed(seed)
stack.rf <- caretStack(models, method="rf", metric="Accuracy", trControl=stackControl)
print(stack.rf)

res <- predict(stack.rf, data_test_kaggle)
toSave <- res
toSave<-as.data.frame(toSave) toSave$toSave<-
unfactor(toSave$toSave)
for(i in 1:250){   if(toSave[i,
1]=="alive"){     toSave[i,1]="1"
 }else{
   toSave[i,1]="0"
 }
}
```

```
write.table(toSave, sep=",")s
```