Homework Assignment 2

Data Science and Machine Learning 1 - CEU 2018

Kristof Menyhert

2018-02-02

Load library and set the theme:

```
library(data.table)
library(caret)
theme_set(theme_bw()) #globally set ggplot theme to black & white
library(knitr)
```

1. Predicting mental health problems in the tech sector

Load the data and tidy it:

```
data <- fread("C:/Users/Chronos/OneDrive - Central European University/R/machine_learning1/hw2/survey_c

data <- data[,c("comments", "state","work_interfere") := NULL]
data[, age := as.numeric(age)]
data[ , treatment := factor(treatment, levels = c("Yes", "No"))]
```

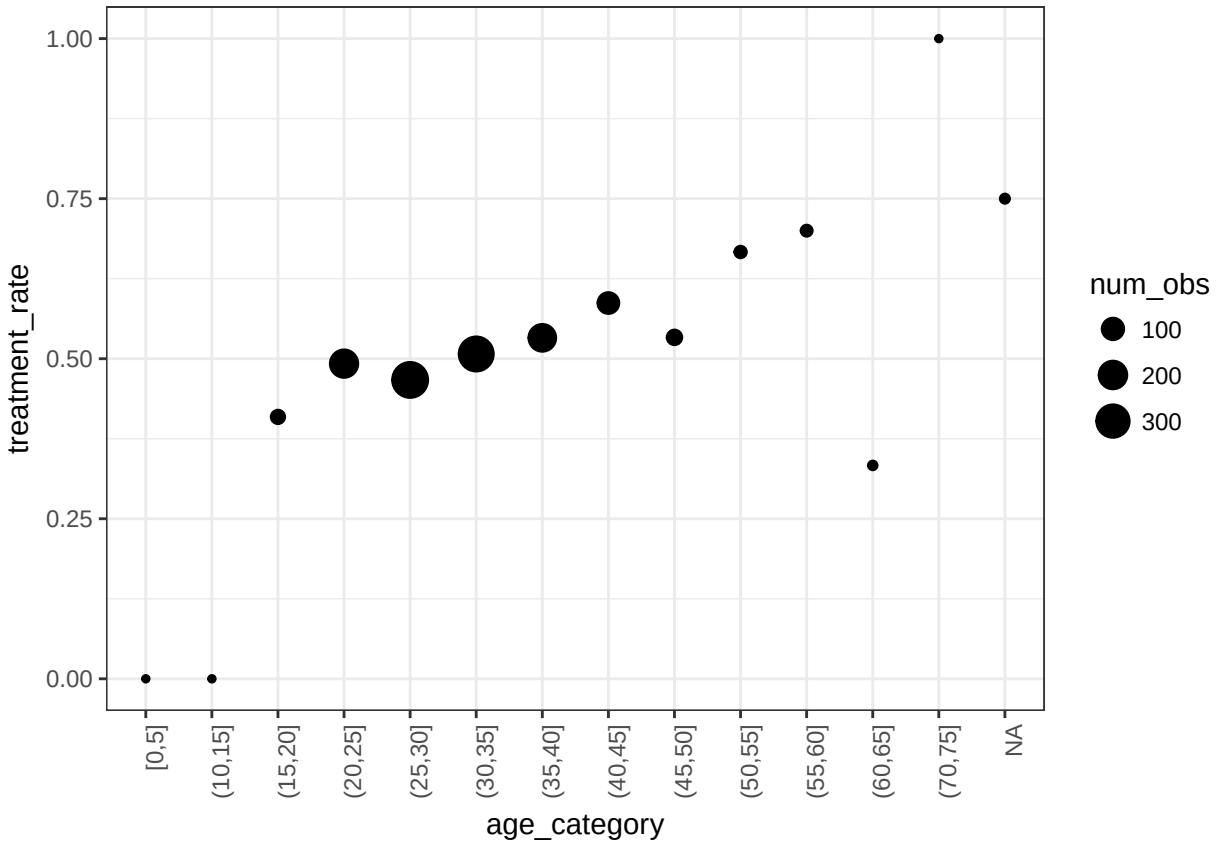
Create treatment_num variable:

```
data[, treatment_num := ifelse(treatment == "Yes", 1, 0)]
```

a) Explore some predictors that can be used to predict treatment.

Age vs. Treatment

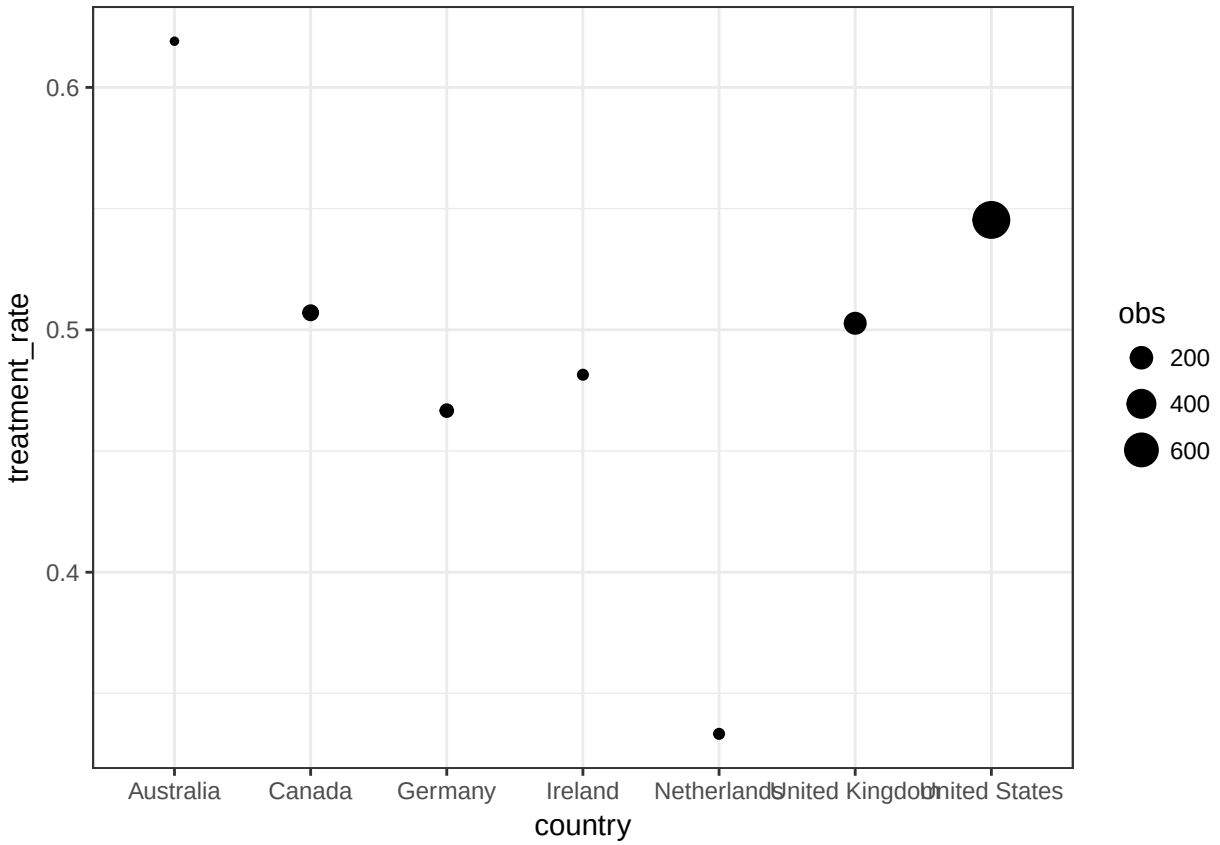
```
data_by_age <- data[,
  .(treatment_rate = mean(treatment_num), num_obs = .N),
  keyby = .(age_category = cut(age, breaks = seq(0, 100, by = 5),
  include.lowest = TRUE))]

ggplot(data = data_by_age, aes(x = age_category, y = treatment_rate, size = num_obs)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We can see a slightly increasing trend with age considering treatment ratio. (Maybe we could have dropped observations with age < 15, but now I am not dealing with them)

Country vs. Treatment

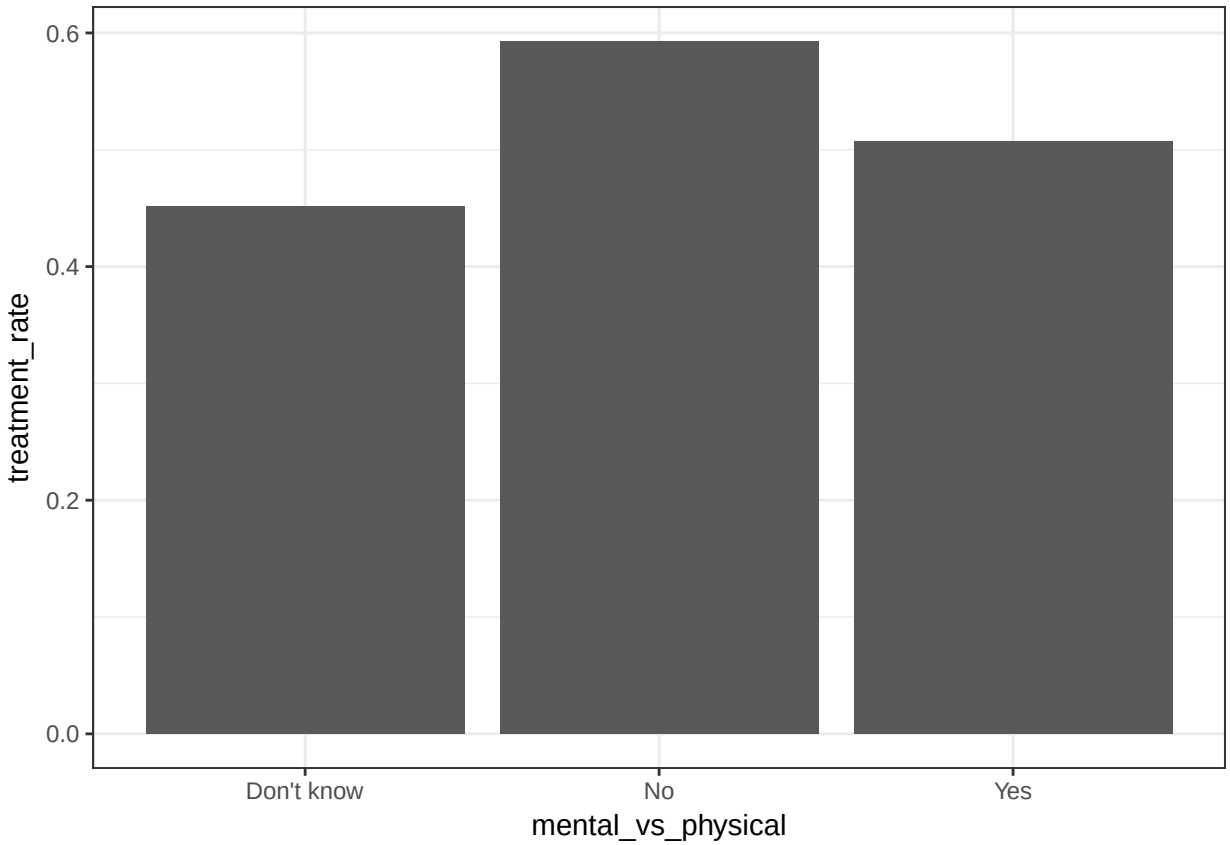
```
treament_percent <- data[, .(treatment_rate = mean(treatment_num), obs = .N), by = country]

ggplot(treament_percent[obs > 20], aes(y=treatment_rate, x = country, size = obs)) + geom_point()
```



There are only 7 countries in the dataset with 20+ observations and among them US and UK have most of the observations, but from this graph we can see that the treatment rates are not the same.

Mental and Physical vs. Treatment

```
mental_ph_percent <- data[, .(treatment_rate = mean(treatment_num), obs = .N), by = mental_vs_physical]

ggplot(mental_ph_percent, aes(mental_vs_physical, treatment_rate)) + geom_bar(stat = "identity")
```



Those people whose workplace are not taking mental health as seriously have the highest treatment ratio on average.

b) Partition your data to 70% training and 30% test samples.

```
cut <- createDataPartition(y = data$treatment, times = 1, p = 0.7, list = FALSE)

data_train <- data[cut, ]

data_test <- data[-cut, ]

# check the cut
length(data$treatment) == (length(data_train$treatment) + length(data_test$treatment))
```

```
## [1] TRUE
```

c) Build models with glmnet and rpart that predict the binary outcome of treatment (you don't have to use all variables if you don't want to - experiment! Just use the same variables for both model families). Use cross-validation on the training set and use AUC as a selection measure (use metric = "ROC" in train and also don't forget to use classProbs = TRUE, summaryFunction = twoClassSummary in trainControl). Make sure to set the same seed before each call to train.

Set Cv:

```
set.seed(1234)
train_control <- trainControl(method = "cv",
```

```
                                number = 5,
                                classProbs = TRUE,
                                summaryFunction=twoClassSummary)
```

GLM model:

```
set.seed(1234)
glmnet_model <- train(treatment ~ age + gender + family_history + leave + supervisor + mental_vs_physica
                      method = "glmnet",
                      metric = "ROC",
                      data = data_train,
                      trControl = train_control,
                      tuneLength=5) # use just 5 lengh tuneLength, I could have set alpha or/and lambda ma
```

See the results and predict outcomes:

```
glmnet_model
```

```
## glmnet
##
## 880 samples
##   6 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 704, 704, 703, 705, 704
## Resampling results across tuning parameters:
##
##   alpha  lambda        ROC        Sens       Spec
##   0.100  0.0001852673  0.7250179  0.6374617  0.7545455
##   0.100  0.0008599347  0.7250179  0.6374617  0.7545455
##   0.100  0.0039914631  0.7250693  0.6374617  0.7545455
##   0.100  0.0185267307  0.7245795  0.6329673  0.7545455
##   0.100  0.0859934665  0.7201563  0.6171859  0.7682602
##   0.325  0.0001852673  0.7248629  0.6374617  0.7545455
##   0.325  0.0008599347  0.7248629  0.6374617  0.7545455
##   0.325  0.0039914631  0.7260107  0.6374617  0.7545455
##   0.325  0.0185267307  0.7226034  0.6217058  0.7636625
##   0.325  0.0859934665  0.7364551  0.5924413  0.7889498
##   0.550  0.0001852673  0.7249918  0.6374617  0.7545455
##   0.550  0.0008599347  0.7249918  0.6374617  0.7545455
##   0.550  0.0039914631  0.7253488  0.6329673  0.7545455
##   0.550  0.0185267307  0.7191465  0.6194586  0.7659613
##   0.550  0.0859934665  0.7289863  0.5901685  0.8003918
##   0.775  0.0001852673  0.7249401  0.6374617  0.7545455
##   0.775  0.0008599347  0.7249662  0.6374617  0.7545455
##   0.775  0.0039914631  0.7250888  0.6307201  0.7545455
##   0.775  0.0185267307  0.7205353  0.6149132  0.7705590
##   0.775  0.0859934665  0.6988721  0.5901685  0.8003918
##   1.000  0.0001852673  0.7249662  0.6374617  0.7545455
##   1.000  0.0008599347  0.7250945  0.6374617  0.7545455
##   1.000  0.0039914631  0.7253750  0.6284474  0.7613898
##   1.000  0.0185267307  0.7326535  0.6081716  0.7797544
##   1.000  0.0859934665  0.6952802  0.5901685  0.8003918
```

```
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.325 and lambda
##  = 0.08599347.
```

```
data_train$glm_model_prediction <- predict.train(glmnet_model, newdata = data_train)
glm_model_prediction_percent <- predict.train(glmnet_model, newdata = data_train, type = "prob")
data_train$glm_model_prediction_per <- glm_model_prediction_percent$Yes
data_test$glm_model_prediction <- predict.train(glmnet_model, newdata = data_test)
```

Rpart model:

```
set.seed(1234)
tune_grid <- data.frame("cp" = c(0.0001, 0.001, 0.01, 0.1, 0.2, 0.3))
rpart_model <- train(treatment ~ age + gender + family_history + leave + supervisor + mental_vs_physical
                     data = data_train,
                     method = "rpart",
                     trControl = train_control,
                     tuneGrid = tune_grid,
                     metric = "ROC")
```

See the results and predict outcomes:

```
rpart_model
```

```
## CART
##
## 880 samples
##   6 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 704, 704, 703, 705, 704
## Resampling results across tuning parameters:
##
##   cp     ROC        Sens       Spec
##   1e-04  0.6825781  0.6532431  0.6512800
##   1e-03  0.6929574  0.6757406  0.6534744
##   1e-02  0.6952279  0.5947140  0.7911964
##   1e-01  0.6952802  0.5901685  0.8003918
##   2e-01  0.6952802  0.5901685  0.8003918
##   3e-01  0.6952802  0.5901685  0.8003918
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.3.
```

```
data_train$rpart_model_prediction <- predict.train(rpart_model, newdata = data_train)
data_test$rpart_model_prediction <- predict.train(rpart_model, newdata = data_test)
```

d) Compare models based on their predictive performance based on the cross-validation information (you can just use the mean AUC to select the best model).

```
compare <- data.table("mean of the ROC of the best glmnet_models" = max(glmnet_model$results$ROC),
                      "VS" = " "  ,
                      "mean of the ROC of the best rpart_models" = max(rpart_model$results$ROC))
```

```
kable(compare)
```

mean of the ROC of the best glmnet_models	VS	mean of the ROC of the best rpart_models
0.7364551		0.6952802

glmnet model seems better, but let's observe the real predictions in a confusion matrix:

For the glmnet model:

```
confusionMatrix(data_train$treatment, data_train$glm_model_prediction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 262 182
##        No   89 347
##
##                Accuracy : 0.692
##                  95% CI : (0.6604, 0.7224)
##     No Information Rate : 0.6011
##     P-Value [Acc > NIR] : 1.357e-08
##
##                   Kappa : 0.3852
##  Mcnemar's Test P-Value : 2.289e-08
##
##             Sensitivity : 0.7464
##             Specificity : 0.6560
##          Pos Pred Value : 0.5901
##          Neg Pred Value : 0.7959
##              Prevalence : 0.3989
##          Detection Rate : 0.2977
##    Detection Prevalence : 0.5045
##       Balanced Accuracy : 0.7012
##
##        'Positive' Class : Yes
##
```

For the rpart model:

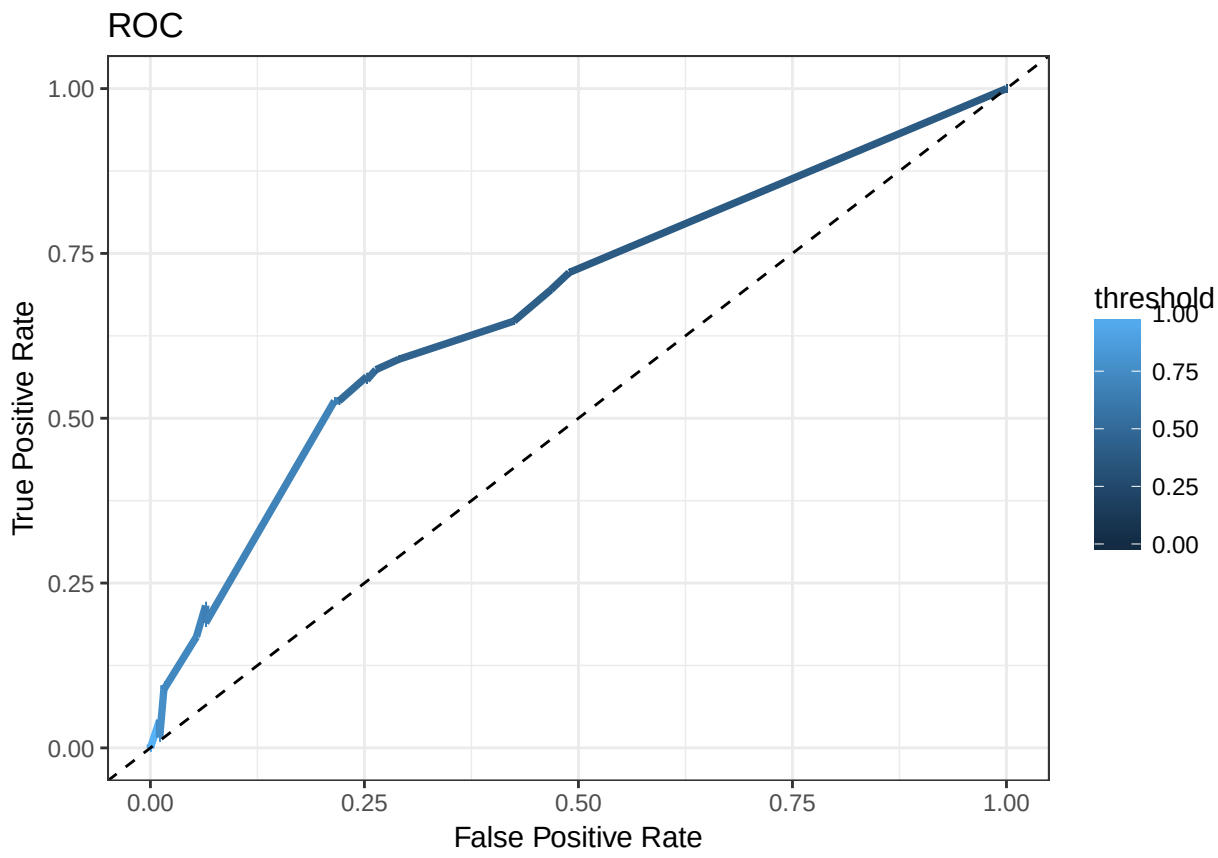
```
confusionMatrix(data_train$treatment, data_train$rpart_model_prediction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 262 182
##        No   87 349
##
##                Accuracy : 0.6943
##                  95% CI : (0.6627, 0.7246)
##     No Information Rate : 0.6034
##     P-Value [Acc > NIR] : 1.301e-08
##
```

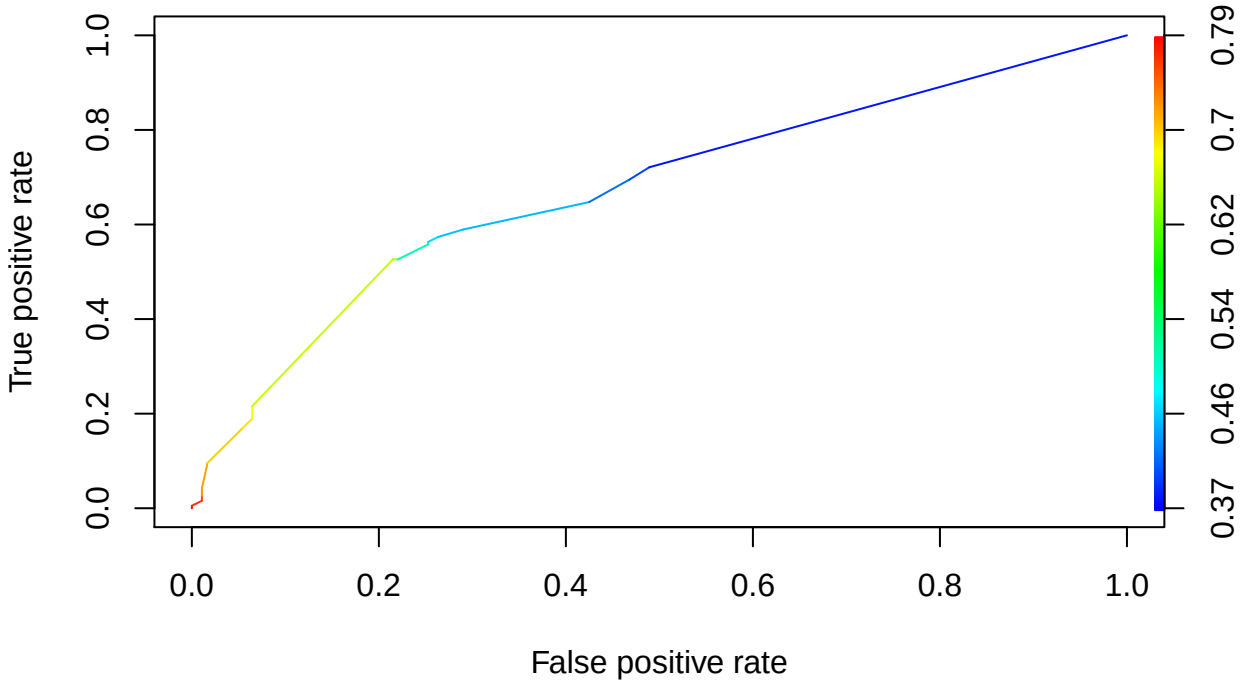
```
##                   Kappa : 0.3898
##  Mcnemar's Test P-Value : 9.967e-09
##
##             Sensitivity : 0.7507
##             Specificity : 0.6573
##          Pos Pred Value : 0.5901
##          Neg Pred Value : 0.8005
##              Prevalence : 0.3966
##          Detection Rate : 0.2977
##    Detection Prevalence : 0.5045
##       Balanced Accuracy : 0.7040
##
##        'Positive' Class : Yes
##
```

e) Evaluate the best model on the test set: draw an ROC curve and calculate and interpret the AUC.

```
test_prediction_probs <- predict.train(glmnet_model,
                                       newdata = data_test,
                                       type = "prob")
thresholds <- seq(0, 1, by = 0.001)

test_truth <- data_test$treatment

true_positive_rates <- rep(0, length(thresholds))
false_positive_rates <- rep(0, length(thresholds))

for (ix in 1:length(thresholds)) {
  thr <- thresholds[ix]
  test_prediction <- ifelse(test_prediction_probs$Yes > thr, "Yes", "No")
  test_prediction <- factor(test_prediction, levels = c("Yes", "No"))
  cm <- as.matrix(confusionMatrix(test_prediction, test_truth))
  true_positive_rates[ix] <- cm[1, 1] / (cm[1, 1] + cm[2, 1])
  false_positive_rates[ix] <- cm[1, 2] / (cm[1, 2] + cm[2, 2])
}

manual_roc <- data.table("threshold" = thresholds,
                         "true_positive_rate" = true_positive_rates,
                         "false_positive_rate" = false_positive_rates)
ggplot(data = manual_roc,
       aes(x = false_positive_rate,
           y = true_positive_rate,
           color = threshold)) +
  geom_line(size = 1.3) + geom_abline(slope = 1, color = "black", linetype = 2) +
  labs(x="False Positive Rate", y="True Positive Rate")+ ggtitle("ROC")
```




```
library(ROCR)
rocr_prediction <- prediction(test_prediction_probs$Yes,
                              data_test$treatment)
# built-in plot method
plot(performance(rocr_prediction, "tpr", "fpr"), colorize=TRUE)
```




```
AUC <- performance(rocr_prediction, "auc")
print(AUC@y.values[[1]])
```

```
## [1] 0.6719864
```

f) If you have to choose a probability threshold to predict the outcome, what would you choose? At this threshold, how large are the true positive rate and the false positive rate? How many false positives and false negatives there are in the test sample?

I would choose the north-weest corner of the curve around the threshold of around 0.4.
* the true positive rate would be around 0.63
* the false positive rate would e around 0.28

```
confusionMatrix(data_test$treatment, predict.train(glmnet_model, newdata = data_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 100  90
##        No   41 145
##
##                Accuracy : 0.6516
##                  95% CI : (0.6011, 0.6997)
##     No Information Rate : 0.625
##     P-Value [Acc > NIR] : 0.1557
##
```

```
##                   Kappa : 0.305
##  Mcnemar's Test P-Value : 2.743e-05
##
##             Sensitivity : 0.7092
##             Specificity : 0.6170
##          Pos Pred Value : 0.5263
##          Neg Pred Value : 0.7796
##              Prevalence : 0.3750
##          Detection Rate : 0.2660
##    Detection Prevalence : 0.5053
##       Balanced Accuracy : 0.6631
##
##        'Positive' Class : Yes
##
```

2. Transformed scores

Take the medical appointment no-show dataset we used in class and apply all the cleaning steps we did, then create a training and a test set. Estimate a predictive model of your choice for no_show as a target variable. Get predicted scores (probabilities). Then calculate two transformations of the scores: take the square root and the square of the probabilities. These are valid scores as well, they are also between 0 and 1 so they can be used for classification.

Load the required packages and set the theme black and white:

```
library(ggplot2)
library(data.table)
library(caret)
library(glmnet)
library(ROCR)

theme_set(theme_bw())
```

Load the data and do the transformations what we did in the class:

```
data <- fread("C:/Users/Chronos/OneDrive - Central European University/R/machine_learning1/hw2/no-show-

# [... apply the cleaning steps we did in class ...]:
# some data cleaning
data[, c("PatientId", "AppointmentID", "Neighbourhood") := NULL]
setnames(data,
         c("No-show",
           "Age",
           "Gender",
           "ScheduledDay",
           "AppointmentDay",
           "Scholarship",
           "Hipertension",
           "Diabetes",
           "Alcoholism",
           "Handcap",
           "SMS_received"),
         c("no_show",
           "age",
           "gender",
```

```
            "scheduled_day",
            "appointment_day",
            "scholarship",
            "hypertension",
            "diabetes",
            "alcoholism",
            "handicap",
            "sms_received"))

# clean up a little bit
data <- data[age %between% c(0, 95)]
# for binary prediction with caret, the target variable must be a factor
data[, no_show := factor(no_show, levels = c("Yes", "No"))]
data[, no_show_num := ifelse(no_show == "Yes", 1, 0)]
data[, handicap := ifelse(handicap > 0, 1, 0)]

# create new variables
data[, scheduled_day := as.Date(scheduled_day)]
data[, appointment_day := as.Date(appointment_day)]
data[, days_since_scheduled := as.integer(appointment_day - scheduled_day)]
data <- data[days_since_scheduled > -1]


#one more:
data[, days_category := cut(
  days_since_scheduled,
  breaks = c(-1, 0, 1, 2, 5, 10, 30, Inf),
  include.lowest = TRUE)]
```

Split the data:

```
# [... create train and test sets ... ]
cut <- createDataPartition(y = data$no_show, times = 1, p = 0.7, list = FALSE)

data_train <- data[cut, ]

data_test <- data[-cut, ]

# check the cut
length(data$no_show) == (length(data_train$no_show) + length(data_test$no_show))
```

```
## [1] TRUE
```

Create a model:

```
train_control <- trainControl(method = "cv",
                              number = 5,
                              classProbs = TRUE,
                              verboseIter = TRUE,
                              summaryFunction = twoClassSummary)

tune_grid <- expand.grid("alpha" = c(0, 0.25, 0.5, 0.75, 1),
                         "lambda" = c(0.01, 0.001, 0.0001))

model <- train(no_show ~ days_category +
```

```
                  poly(age, 3) +
                  scholarship +
                  gender +
                  alcoholism +
                  diabetes,
                data = data_train, method = "glmnet", preProcess = c("center", "scale"),
                trControl = train_control, tuneGrid = tune_grid, metric = "ROC")
```

```
## + Fold1: alpha=0.00, lambda=0.01
## - Fold1: alpha=0.00, lambda=0.01
## + Fold1: alpha=0.25, lambda=0.01
## - Fold1: alpha=0.25, lambda=0.01
## + Fold1: alpha=0.50, lambda=0.01
## - Fold1: alpha=0.50, lambda=0.01
## + Fold1: alpha=0.75, lambda=0.01
## - Fold1: alpha=0.75, lambda=0.01
## + Fold1: alpha=1.00, lambda=0.01
## - Fold1: alpha=1.00, lambda=0.01
## + Fold2: alpha=0.00, lambda=0.01
## - Fold2: alpha=0.00, lambda=0.01
## + Fold2: alpha=0.25, lambda=0.01
## - Fold2: alpha=0.25, lambda=0.01
## + Fold2: alpha=0.50, lambda=0.01
## - Fold2: alpha=0.50, lambda=0.01
## + Fold2: alpha=0.75, lambda=0.01
## - Fold2: alpha=0.75, lambda=0.01
## + Fold2: alpha=1.00, lambda=0.01
## - Fold2: alpha=1.00, lambda=0.01
## + Fold3: alpha=0.00, lambda=0.01
## - Fold3: alpha=0.00, lambda=0.01
## + Fold3: alpha=0.25, lambda=0.01
## - Fold3: alpha=0.25, lambda=0.01
## + Fold3: alpha=0.50, lambda=0.01
## - Fold3: alpha=0.50, lambda=0.01
## + Fold3: alpha=0.75, lambda=0.01
## - Fold3: alpha=0.75, lambda=0.01
## + Fold3: alpha=1.00, lambda=0.01
## - Fold3: alpha=1.00, lambda=0.01
## + Fold4: alpha=0.00, lambda=0.01
## - Fold4: alpha=0.00, lambda=0.01
## + Fold4: alpha=0.25, lambda=0.01
## - Fold4: alpha=0.25, lambda=0.01
## + Fold4: alpha=0.50, lambda=0.01
## - Fold4: alpha=0.50, lambda=0.01
## + Fold4: alpha=0.75, lambda=0.01
## - Fold4: alpha=0.75, lambda=0.01
## + Fold4: alpha=1.00, lambda=0.01
## - Fold4: alpha=1.00, lambda=0.01
## + Fold5: alpha=0.00, lambda=0.01
## - Fold5: alpha=0.00, lambda=0.01
## + Fold5: alpha=0.25, lambda=0.01
## - Fold5: alpha=0.25, lambda=0.01
## + Fold5: alpha=0.50, lambda=0.01
## - Fold5: alpha=0.50, lambda=0.01
```

```
## + Fold5: alpha=0.75, lambda=0.01
## - Fold5: alpha=0.75, lambda=0.01
## + Fold5: alpha=1.00, lambda=0.01
## - Fold5: alpha=1.00, lambda=0.01
## Aggregating results
## Selecting tuning parameters
## Fitting alpha = 0.75, lambda = 0.001 on full training set
```

```
model
```

```
## glmnet
##
## 77333 samples
##     6 predictor
##     2 classes: 'Yes', 'No'
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 61866, 61867, 61866, 61866, 61867
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens          Spec
##   0.00   1e-04   0.7264083  0.0001280820  1.0000000
##   0.00   1e-03   0.7264083  0.0001280820  1.0000000
##   0.00   1e-02   0.7263694  0.0000000000  1.0000000
##   0.25   1e-04   0.7263747  0.0009606148  0.9999514
##   0.25   1e-03   0.7263980  0.0008965738  0.9999514
##   0.25   1e-02   0.7257024  0.0000000000  1.0000000
##   0.50   1e-04   0.7263894  0.0009606148  0.9999514
##   0.50   1e-03   0.7263968  0.0007044508  0.9999676
##   0.50   1e-02   0.7241666  0.0000000000  1.0000000
##   0.75   1e-04   0.7263946  0.0009606148  0.9999514
##   0.75   1e-03   0.7264142  0.0005123279  0.9999838
##   0.75   1e-02   0.7222777  0.0000000000  1.0000000
##   1.00   1e-04   0.7263979  0.0009606148  0.9999514
##   1.00   1e-03   0.7264039  0.0001921230  1.0000000
##   1.00   1e-02   0.7199861  0.0000000000  1.0000000
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.75 and lambda = 0.001.
```

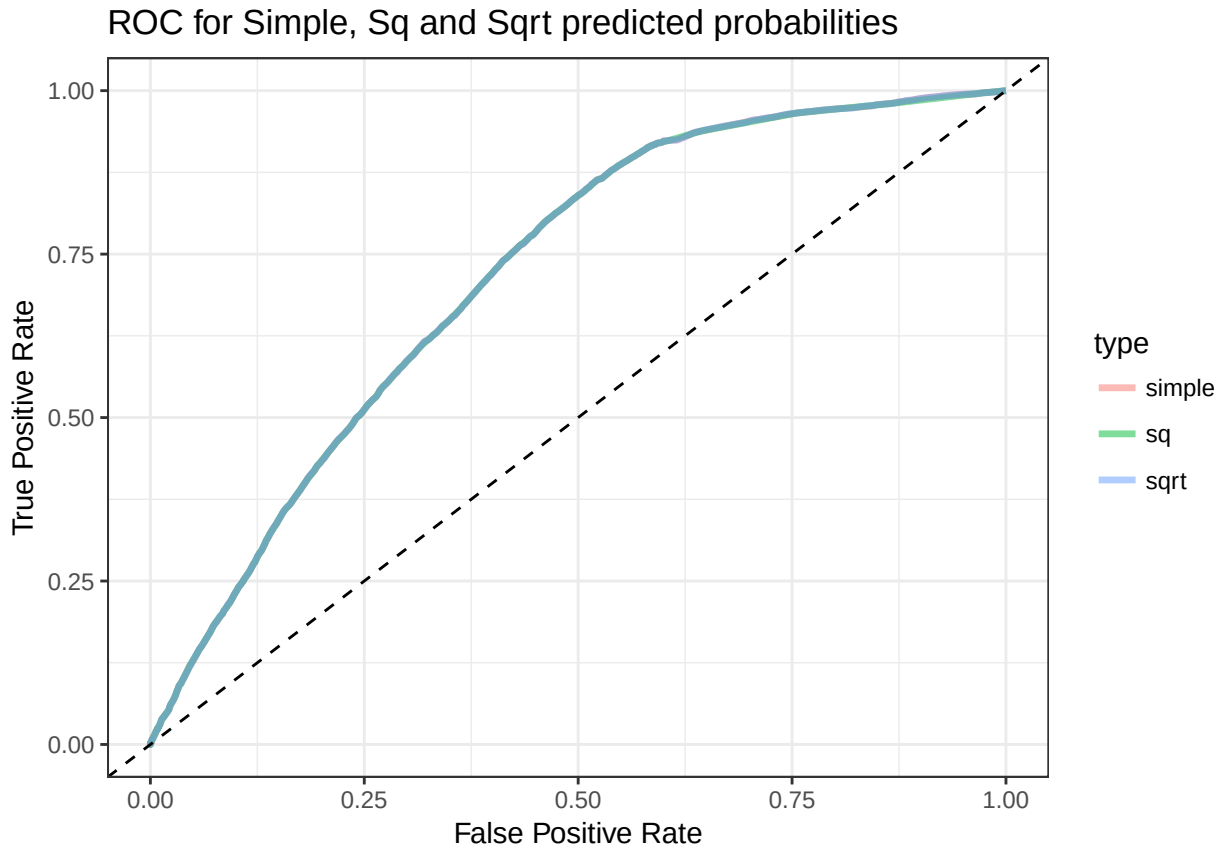
```
prediction <- predict.train(model, newdata = data_test, type = "prob")
prediction_sqrt <- sqrt(prediction)
prediction_sq <- prediction^2

prediction_sq <- data.table(prediction_sq)
```

a) Draw ROC curves for all three scores and calculate the AUC. How do they compare? Is it surprising in light of the interpretation of the AUC?

```
thresholds <- seq(0, 1, by = 0.001)

test_truth <- data_test$no_show

true_positive_rates <- rep(0, length(thresholds))
```

```r
false_positive_rates <- rep(0, length(thresholds))

true_positive_rates_sqrt <- rep(0, length(thresholds))
false_positive_rates_sqrt <- rep(0, length(thresholds))

true_positive_rates_sq <- rep(0, length(thresholds))
false_positive_rates_sq <- rep(0, length(thresholds))

for (ix in 1:length(thresholds)) {
  thr <- thresholds[ix]
  test_prediction <- ifelse(prediction$Yes > thr, "Yes", "No")
  test_prediction <- factor(test_prediction, levels = c("Yes", "No"))

  test_prediction_sqrt <- ifelse(prediction_sqrt$Yes > thr, "Yes", "No")
  test_prediction_sqrt <- factor(test_prediction_sqrt, levels = c("Yes", "No"))

  test_prediction_sq <- ifelse(prediction_sq$Yes > thr, "Yes", "No")
  test_prediction_sq <- factor(test_prediction_sq, levels = c("Yes", "No"))

  cm <- as.matrix(confusionMatrix(test_prediction, test_truth))
  cm2 <- as.matrix(confusionMatrix(test_prediction_sqrt, test_truth))
  cm3 <- as.matrix(confusionMatrix(test_prediction_sq, test_truth))

  true_positive_rates[ix] <- cm[1, 1] / (cm[1, 1] + cm[2, 1])
  false_positive_rates[ix] <- cm[1, 2] / (cm[1, 2] + cm[2, 2])

  true_positive_rates_sqrt[ix] <- cm2[1, 1] / (cm2[1, 1] + cm2[2, 1])
  false_positive_rates_sqrt[ix] <- cm2[1, 2] / (cm2[1, 2] + cm2[2, 2])

  true_positive_rates_sq[ix] <- cm3[1, 1] / (cm3[1, 1] + cm3[2, 1])
  false_positive_rates_sq[ix] <- cm3[1, 2] / (cm3[1, 2] + cm3[2, 2])
}

manual_roc <- data.table("threshold" = thresholds,
                         "true_positive_rate" = true_positive_rates,
                         "false_positive_rate" = false_positive_rates,
                         "true_positive_rate_sqrt" = true_positive_rates_sqrt ,
                         "false_positive_rate_sqrt" = false_positive_rates_sqrt ,
                         "true_positive_rate_sq"= true_positive_rates_sq,
                         "false_positive_rate_sq" = false_positive_rates_sq)

roc1 <- data.table("threshold" = thresholds,
                   "true_positive_rate" = true_positive_rates,
                   "false_positive_rate" = false_positive_rates,
                   "type" = "simple")

roc2 <- data.table("threshold" = thresholds,
                   "true_positive_rate" = true_positive_rates_sqrt,
                   "false_positive_rate" = false_positive_rates_sqrt,
                   "type" = "sqrt")

roc3 <- data.table("threshold" = thresholds,
                   "true_positive_rate"= true_positive_rates_sq,
```

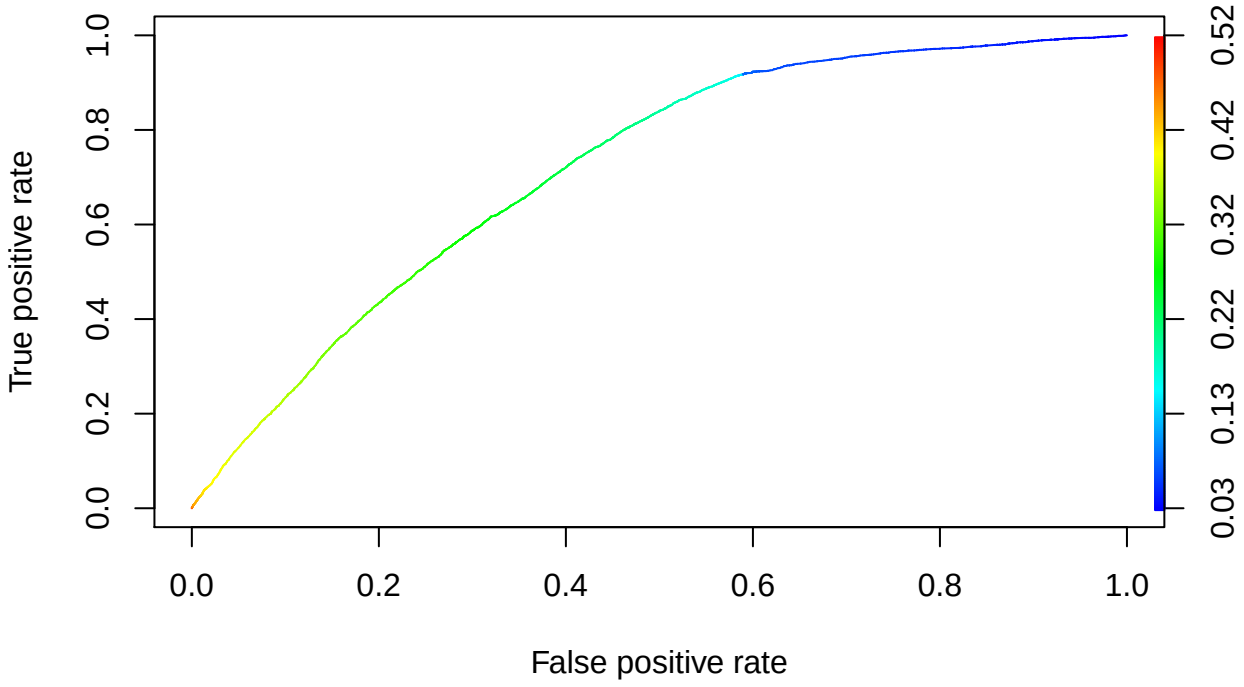
```
                    "false_positive_rate" = false_positive_rates_sq,
                    "type" = "sq")

roc_all <- rbind(roc1, roc2, roc3)


ggplot(data = roc_all,
       aes(x = false_positive_rate,
           y = true_positive_rate,
           color = type)) +
  geom_line(size = 1.2, alpha = 0.5) +
  geom_abline(slope = 1, color = "black", linetype = 2) +
  labs(x="False Positive Rate",
       y="True Positive Rate") +
  ggtitle("ROC for Simple, Sq and Sqrt predicted probabilities")
```



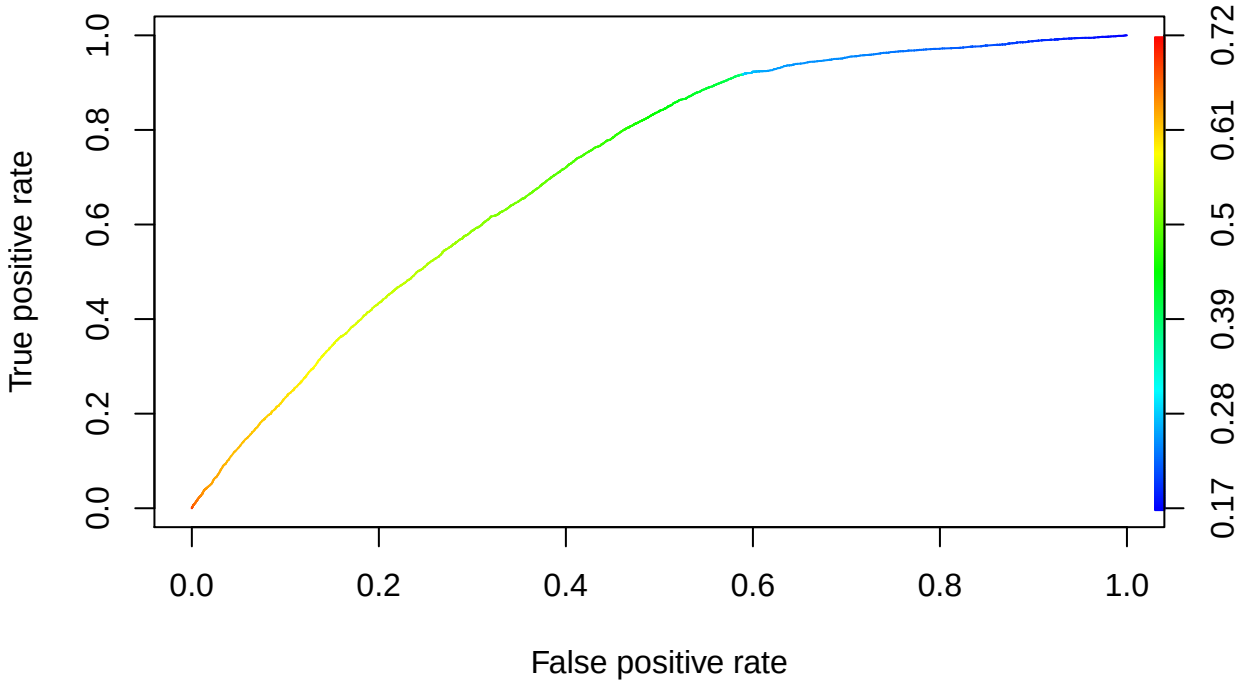
All the ROC cuvres are overlaping/same.

We can do the same as above just with the ROCR library separately:

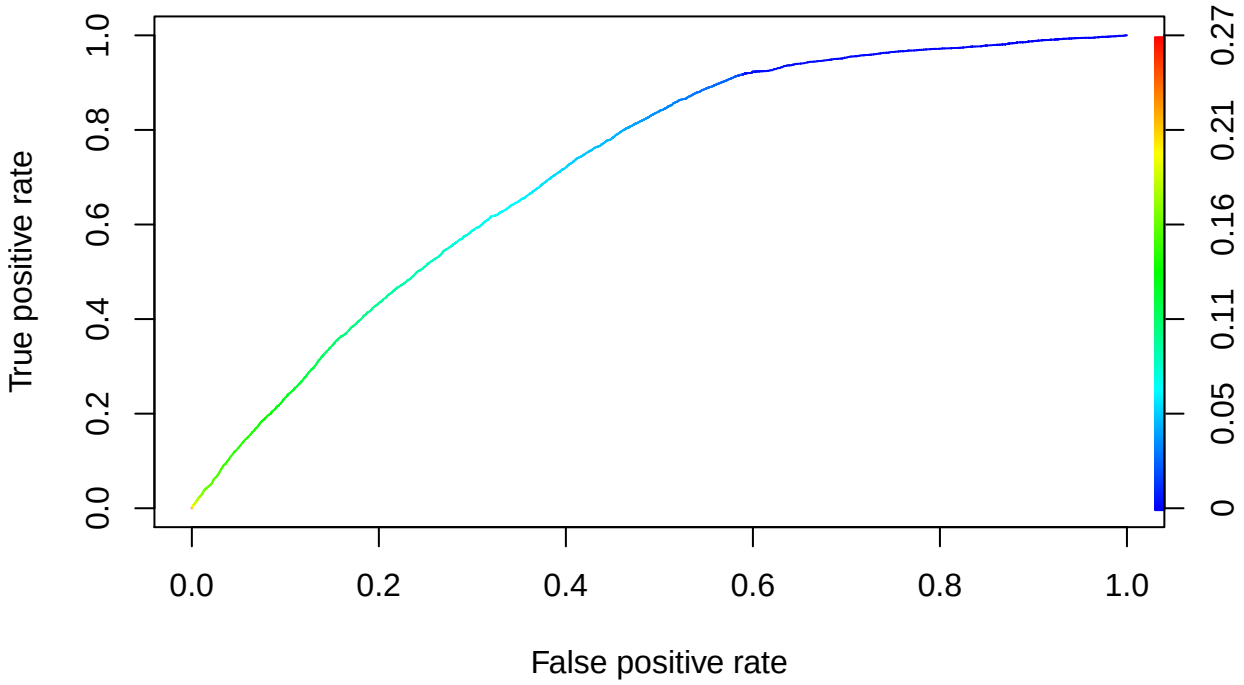
```
library(ROCR)
rocr_prediction <- prediction(prediction$Yes, test_truth)
# built-in plot method
plot(performance(rocr_prediction, "tpr", "fpr"), colorize=TRUE)
```




```
rocr_prediction_sqrt <- prediction(prediction_sqrt$Yes, test_truth)
# built-in plot method
plot(performance(rocr_prediction_sqrt, "tpr", "fpr"), colorize=TRUE)
```




```
rocr_prediction_sq <- prediction(prediction_sq$Yes, test_truth)
# built-in plot method
plot(performance(rocr_prediction_sq, "tpr", "fpr"), colorize=TRUE)
```



We also should notice that the plotted ROCs are the same, but the thresholds are changed on each of the graphs.

b) What is the key, common property of both the square root and the square functions that leads to this finding?

Both (sqrt and sq) of them are not linear transformations. So the big values and the small values are not getting the same effect by executing sqrt and sq function on them. That is the reason why the thresholds are different.

However we are not able to get a different separation, since the ROC function is monotone and the transformation does not change the 'order' of the predicted probabilities.

c) Draw a calibration plot for all three scores separately:

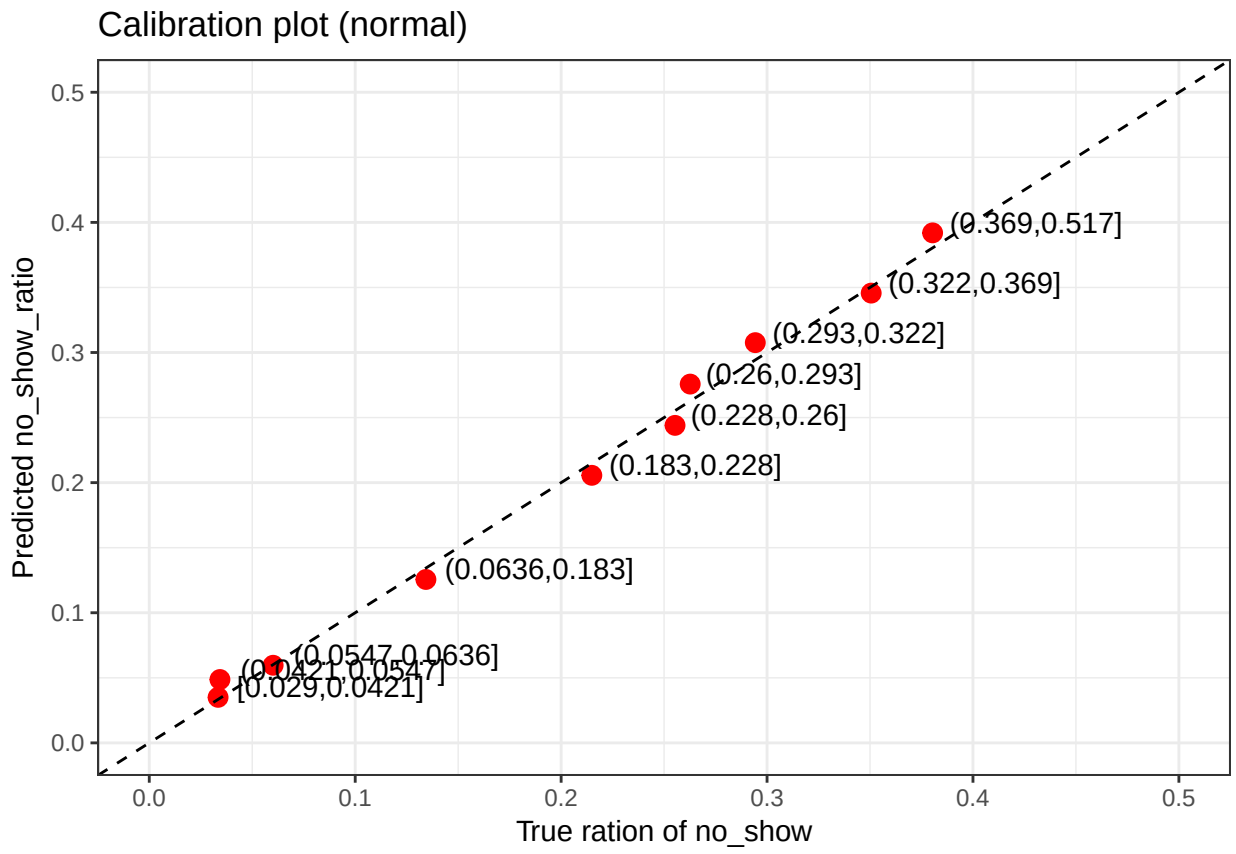
- group people into bins based on predicted scores
- display on a scatter plot the mean of the predicted scores versus the actual share of people surviving

```
data_test[, prediction_normal := prediction$Yes]
data_test[, prediction_sqrt := prediction_sqrt$Yes]
data_test[, prediction_sq := prediction_sq$Yes]
```

Normal probabilities:

```
deciles <- quantile(data_test$prediction_normal, prob = seq(0, 1, length = 11))
```

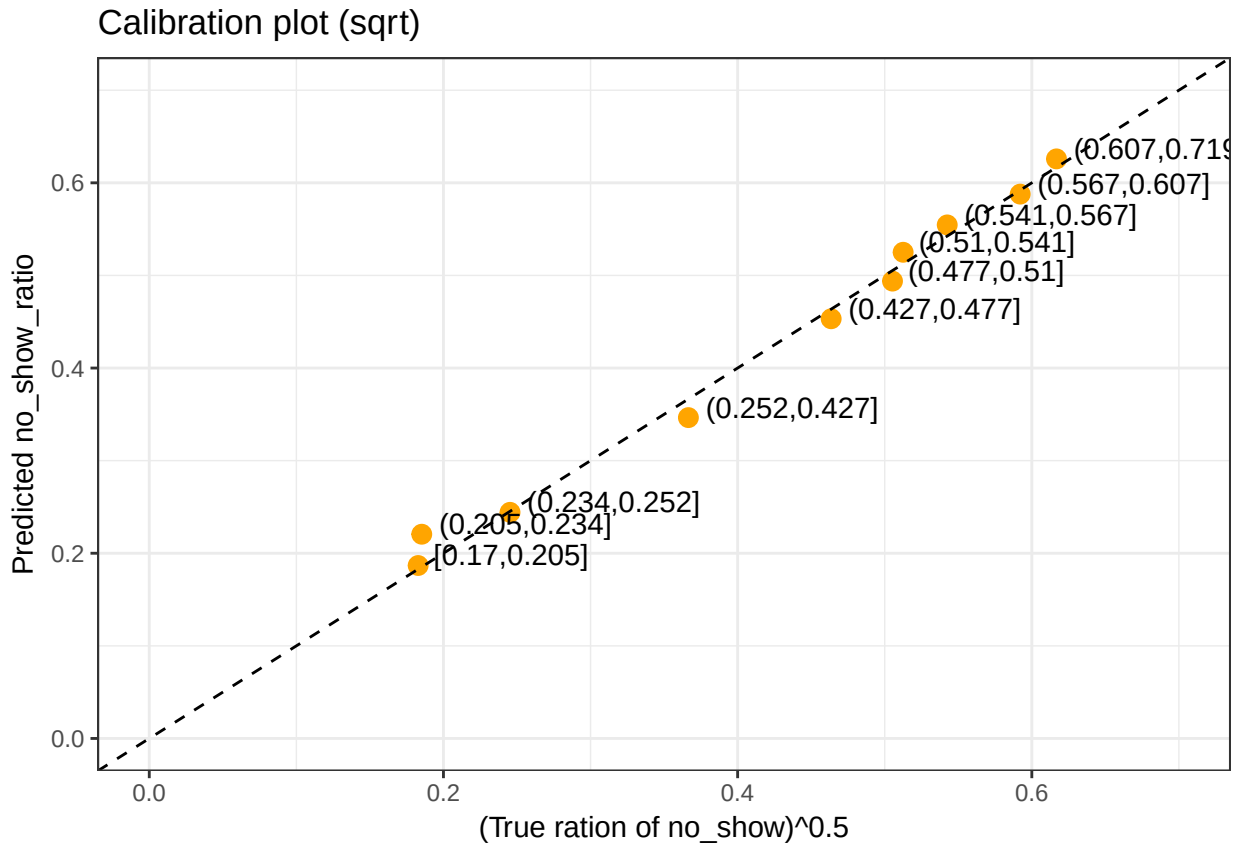
```
group <- cut(data_test$prediction_normal, deciles, include.lowest = TRUE)

data_test[, prediction_normal_group := group]

calibrateplot <- data_test[, .(N = .N, true_mean_no_show = mean(no_show_num), predicted_mean_no_show = m

ggplot(calibrateplot, aes(x= true_mean_no_show, y=predicted_mean_no_show)) +
  geom_point(size = 3, color = "red") +
  geom_text(aes(label=prediction_normal_group),hjust=-0.1, vjust=0) +
  geom_abline(slope = 1, linetype = 2) +
  coord_cartesian(xlim = c(0, 0.5), ylim = c(0,0.5)) +
  labs(x="True ration of no_show", y="Predicted no_show_ratio") +
  ggtitle("Calibration plot (normal)")
```



Sqrt probabilities:

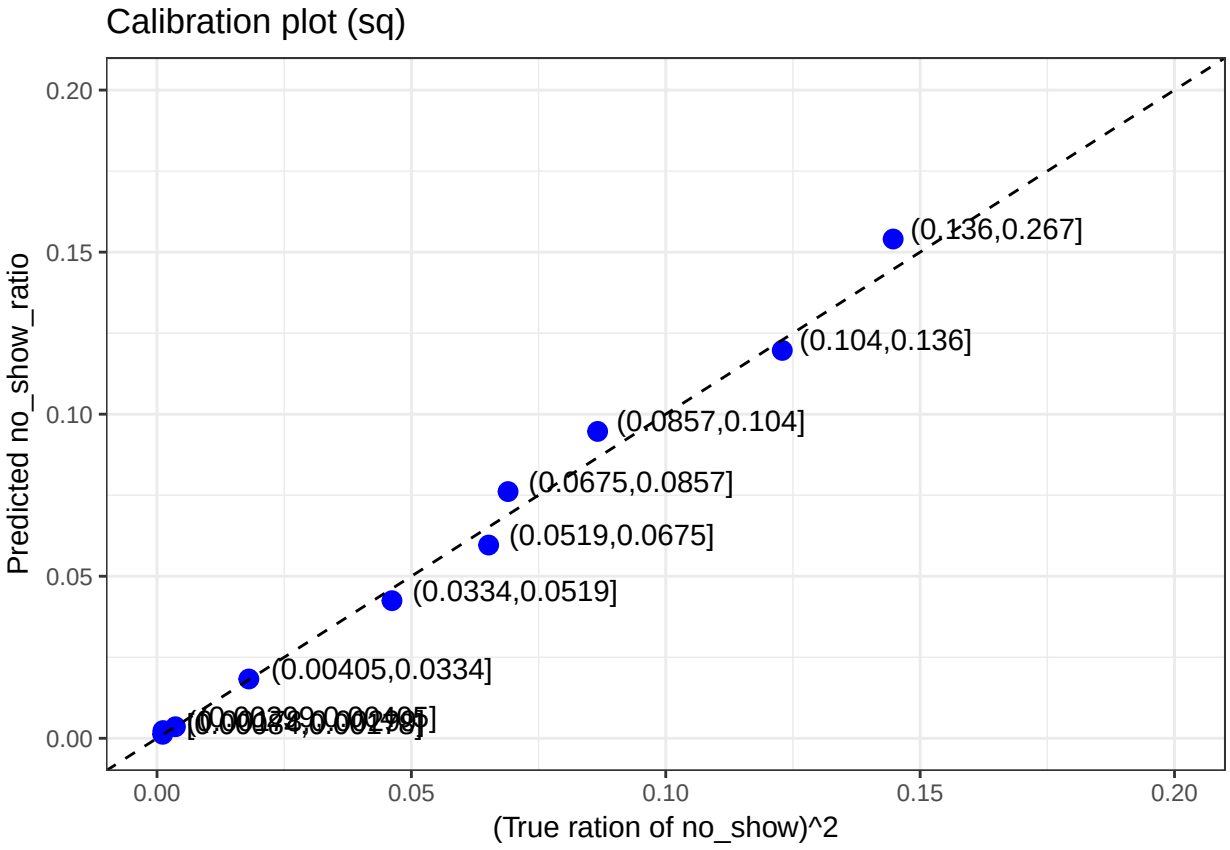
```
deciles <- quantile(data_test$prediction_sqrt, prob = seq(0, 1, length = 11))

group <- cut(data_test$prediction_sqrt, deciles, include.lowest = TRUE)

data_test[, prediction_sqrt_group := group]

calibrateplot_sqrt <- data_test[, .(N = .N, true_mean_no_show = mean(no_show_num), predicted_mean_no_sho

ggplot(calibrateplot_sqrt, aes(x= true_mean_no_show^0.5, y=predicted_mean_no_show)) +
  geom_point(size = 3, color = "orange") +
  geom_abline(slope = 1, linetype = 2) +
```

```
  geom_text(aes(label=prediction_sqrt_group),hjust=-0.1, vjust=0) +
  coord_cartesian(xlim = c(0, 0.7), ylim = c(0,0.7)) +
  labs(x="(True ration of no_show)^0.5", y="Predicted no_show_ratio") +
  ggtitle("Calibration plot (sqrt)")
```



Sq probabilities:

```
deciles <- quantile(data_test$prediction_sq, prob = seq(0, 1, length = 11))

group <- cut(data_test$prediction_sq, deciles, include.lowest = TRUE)

data_test[, prediction_sq_group := group]

calibrateplot_sq <- data_test[, .(N = .N, true_mean_no_show = mean(no_show_num), predicted_mean_no_show

ggplot(calibrateplot_sq, aes(x= true_mean_no_show^2, y=predicted_mean_no_show)) +
  geom_point(size = 3, color = "blue") +
  geom_abline(slope = 1, linetype = 2) +
  geom_text(aes(label=prediction_sq_group),hjust=-0.1, vjust=0) +
  coord_cartesian(xlim = c(0, 0.2), ylim = c(0,0.2)) +
  labs(x="(True ration of no_show)^2", y="Predicted no_show_ratio") +
  ggtitle("Calibration plot (sq)")
```



How do they compare? Which score(s) can be regarded as well-calibrated probabilites?

I would say that all the methods are well-calibrated and there are only small differences. Since the difference between the results is negligible I found that there is no point in deciding which is the most well-calibrated method.

But, if I had to make a decision I would choose the sqrt probabilities method for the calibration.