WstÄ™p do Uczenia Maszynowego 2020: projekt I, kamieÅ„ milowy III - drzewa klasyfikacyjne

Jakub Kosterna

09/04/2020

1. Odczyt

W II kamieniu milowym dokonaliśmy porządnej inżynierii cech. Wczytajmy rezultat naszej pracy.

```
library(dplyr)
X_train <- read.csv("x_train.csv")
X_test <- read.csv("x_test.csv")
y_train <- read.csv("y_train.csv")
y_test <- read.csv("y_test.csv")

X_test <- select(X_test, -X)
X_train <- select(X_train, -X)
y_test <- select(y_test, -X)
y_train <- select(y_train, -X)

colnames(y_train) <- "is_good_customer_type"
colnames(y_test) <- "is_good_customer_type"

knitr::kable(sample_n(X_train, 10))
```

duration	credit_amount	installment_rate	present_residence	age	existing_credits	dependents	has_telephone
24	2538	4	4	47	2	2	0
24	1231	4	4	57	2	1	1
12	1282	2	4	20	1	1	0
12	1262	3	2	25	1	1	0
9	2507	2	4	51	1	1	0
60	15653	2	4	21	2	1	1
48	6999	1	1	34	2	1	1
24	2679	4	1	29	1	1	1
36	2746	4	4	31	1	1	0
12	741	4	3	22	1	1	0

```
knitr::kable(sample_n(y_test, 10))
```

is_good_customer_type
1
0
0
1
1
1
0
1
0
1

Wszystko jest w porządku! Możemy zacząć implementować nasze drzewo klasyfikacyjne.

2. Miary oceny klasyfikatora

W ocenie kolejnych modeli posłużę się czterema najbardziej klasycznymi miarami:

- *accuracy* - $\frac{TP+TN}{TP+FP+FN+TN}$
- *precision* - $\frac{TP}{TP+FP}$
- *recall* - $\frac{TP}{TP+FN}$
- *f1* - $2 * \frac{Recall*Precision}{Recall+Precision}$

```
confusion_matrix_values <- function(confusion_matrix){
TP <- confusion_matrix[2,2]
TN <- confusion_matrix[1,1]
FP <- confusion_matrix[1,2]
FN <- confusion_matrix[2,1]
return (c(TP, TN, FP, FN))
}

accuracy <- function(confusion_matrix){
conf_matrix <- confusion_matrix_values(confusion_matrix)
return((conf_matrix[1] + conf_matrix[2]) / (conf_matrix[1] + conf_matrix[2] + conf_matrix[3] + conf_mat
}

precision <- function(confusion_matrix){
conf_matrix <- confusion_matrix_values(confusion_matrix)
return(conf_matrix[1]/ (conf_matrix[1] + conf_matrix[3]))
}

recall <- function(confusion_matrix){
conf_matrix <- confusion_matrix_values(confusion_matrix)
return(conf_matrix[1] / (conf_matrix[1] + conf_matrix[4]))
}

f1 <- function(confusion_matrix){
conf_matrix <- confusion_matrix_values(confusion_matrix)
rec <- recall(confusion_matrix)
prec <- precision(confusion_matrix)
```

```
return(2 * (rec * prec) / (rec + prec))
}
```

3. Najprostszy z najprostszych

Żeby trochę ugryźć temat, w pierwszej kolejności zajmę się najbardziej podstawowym wczytaniem i odtworzeniem modelu. Na kolejnych etapach będę próbował bawić się coraz to bardziej zaawansowanymi strukturami i wyborem hiperparametrów, ale idąc od rdzenia i będziemy mogli zaobserwować ewolucję skomplikowania kodu, ale także i porównań wyników, czy w praktyce będziemy mieli wyraźnie lepsze rozwiązanie.

Do konstrukcji classification trees użyję przydatnego pakietu **rpart**.

```
library(rpart)
```

Wygenerujmy naszą pierwszą roślinkę.

```
X <- cbind(X_train, y_train)

primitive_model <- rpart(is_good_customer_type ~ ., data = X, method = "class", control = rpart.control
```

. . . i ją z wizualizujmy. Możemy to zrobić na trzy sposoby:

1. Po prostu wyczytując wartości z powstałej wygenerowanej zmiennej, która zawiera kluczowe przedziały
2. Użyć podstawowej wizualizacji, jaką oferuje **rpart.plot**.
3. Także skorzystać z pakietu *rpart.plot*, lecz z ułatwiającymi bonusami.

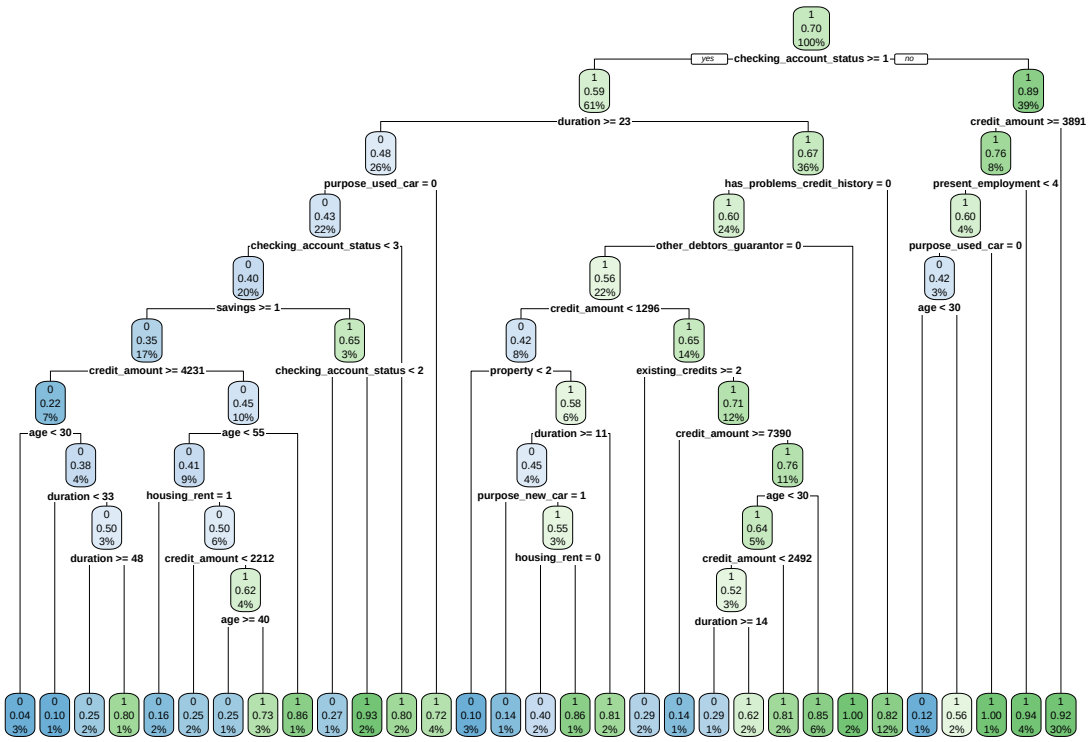
W pierwszej kolejności skorzystam ze wszystkich trzech opcji, dla kultury.

```
library(rpart.plot)

primitive_model
```

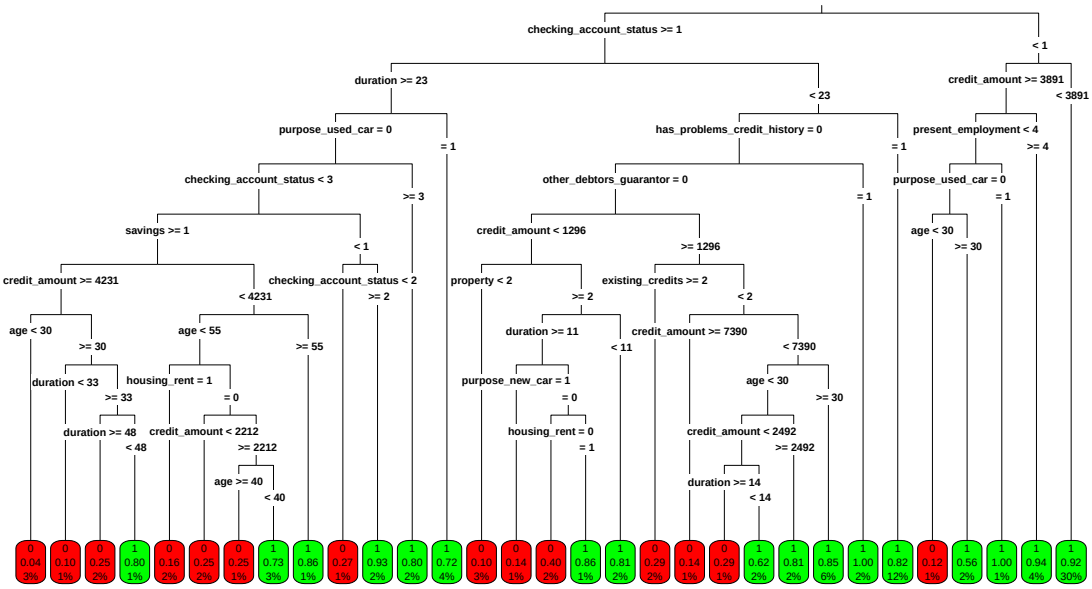
```
## n= 800
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 800 236 1 (0.29500000 0.70500000)
##      2) checking_account_status>=0.5 491 201 1 (0.40936864 0.59063136)
##        4) duration>=22.5 204  97 0 (0.52450980 0.47549020)
##          8) purpose_used_car< 0.5 175  76 0 (0.56571429 0.43428571)
##           16) checking_account_status< 2.5 160  64 0 (0.60000000 0.40000000)
##             32) savings>=0.5 134  47 0 (0.64925373 0.35074627)
##               64) credit_amount>=4231 58  13 0 (0.77586207 0.22413793)
##                128) age< 29.5 26   1 0 (0.96153846 0.03846154) *
##                129) age>=29.5 32  12 0 (0.62500000 0.37500000)
##                  258) duration< 33 10   1 0 (0.90000000 0.10000000) *
##                  259) duration>=33 22  11 0 (0.50000000 0.50000000)
##                    518) duration>=47.5 12   3 0 (0.75000000 0.25000000) *
##                    519) duration< 47.5 10   2 1 (0.20000000 0.80000000) *
```

```
##               65) credit_amount< 4231 76  34 0 (0.55263158 0.44736842)
##                130) age< 54.5 69  28 0 (0.59420290 0.40579710)
##                  260) housing_rent>=0.5 19   3 0 (0.84210526 0.15789474) *
##                  261) housing_rent< 0.5 50  25 0 (0.50000000 0.50000000)
##                    522) credit_amount< 2211.5 16   4 0 (0.75000000 0.25000000) *
##                    523) credit_amount>=2211.5 34  13 1 (0.38235294 0.61764706)
##                     1046) age>=40 8   2 0 (0.75000000 0.25000000) *
##                     1047) age< 40 26   7 1 (0.26923077 0.73076923) *
##                131) age>=54.5 7   1 1 (0.14285714 0.85714286) *
##              33) savings< 0.5 26   9 1 (0.34615385 0.65384615)
##                66) checking_account_status< 1.5 11   3 0 (0.72727273 0.27272727) *
##                67) checking_account_status>=1.5 15   1 1 (0.06666667 0.93333333) *
##            17) checking_account_status>=2.5 15   3 1 (0.20000000 0.80000000) *
##           9) purpose_used_car>=0.5 29   8 1 (0.27586207 0.72413793) *
##         5) duration< 22.5 287  94 1 (0.32752613 0.67247387)
##          10) has_problems_credit_history< 0.5 194  77 1 (0.39690722 0.60309278)
##            20) other_debtors_guarantor< 0.5 176  77 1 (0.43750000 0.56250000)
##              40) credit_amount< 1296 66  28 0 (0.57575758 0.42424242)
##                80) property< 1.5 21   2 0 (0.90476190 0.09523810) *
##                81) property>=1.5 45  19 1 (0.42222222 0.57777778)
##                 162) duration>=11 29  13 0 (0.55172414 0.44827586)
##                   324) purpose_new_car>=0.5 7   1 0 (0.85714286 0.14285714) *
##                   325) purpose_new_car< 0.5 22  10 1 (0.45454545 0.54545455)
##                     650) housing_rent< 0.5 15   6 0 (0.60000000 0.40000000) *
##                     651) housing_rent>=0.5 7   1 1 (0.14285714 0.85714286) *
##                 163) duration< 11 16   3 1 (0.18750000 0.81250000) *
##              41) credit_amount>=1296 110  39 1 (0.35454545 0.64545455)
##                82) existing_credits>=1.5 17   5 0 (0.70588235 0.29411765) *
##                83) existing_credits< 1.5 93  27 1 (0.29032258 0.70967742)
##                 166) credit_amount>=7390 7   1 0 (0.85714286 0.14285714) *
##                 167) credit_amount< 7390 86  21 1 (0.24418605 0.75581395)
##                   334) age< 29.5 39  14 1 (0.35897436 0.64102564)
##                     668) credit_amount< 2492 23  11 1 (0.47826087 0.52173913)
##                      1336) duration>=13.5 7   2 0 (0.71428571 0.28571429) *
##                      1337) duration< 13.5 16   6 1 (0.37500000 0.62500000) *
##                     669) credit_amount>=2492 16   3 1 (0.18750000 0.81250000) *
##                   335) age>=29.5 47   7 1 (0.14893617 0.85106383) *
##            21) other_debtors_guarantor>=0.5 18   0 1 (0.00000000 1.00000000) *
##           11) has_problems_credit_history>=0.5 93  17 1 (0.18279570 0.81720430) *
##       3) checking_account_status< 0.5 309  35 1 (0.11326861 0.88673139)
##         6) credit_amount>=3891 68  16 1 (0.23529412 0.76470588)
##          12) present_employment< 3.5 35  14 1 (0.40000000 0.60000000)
##            24) purpose_used_car< 0.5 24  10 0 (0.58333333 0.41666667)
##              48) age< 30 8   1 0 (0.87500000 0.12500000) *
##              49) age>=30 16   7 1 (0.43750000 0.56250000) *
##            25) purpose_used_car>=0.5 11   0 1 (0.00000000 1.00000000) *
##          13) present_employment>=3.5 33   2 1 (0.06060606 0.93939394) *
##         7) credit_amount< 3891 241  19 1 (0.07883817 0.92116183) *
```

```
rpart.plot(primitive_model)
```




```
rpart.plot(primitive_model, type = 3, box.palette = c("red", "green"), fallen.leaves = TRUE)
```



Woah! Jak widać, dużo cech poskutkowało także niewytłumaczalnym banalnie classification tree.

Sprawdźmy jeszcze, jak sobie ono radzi w akcji.

```
y_pred <- predict(primitive_model, X_test, type = "class")
y_pred <- as.data.frame(y_pred)

confusion_matrix_primitive <- table(Truth = y_test$is_good_customer_type, Prediction = y_pred$y_pred)

knitr::kable(confusion_matrix_primitive)
```

	0	1
0	24	40
1	17	119

```
accuracy_primitive <- accuracy(confusion_matrix_primitive)
precision_primitive <- precision(confusion_matrix_primitive)
recall_primitive <- recall(confusion_matrix_primitive)
f1_primitive <- f1(confusion_matrix_primitive)

classification_report_primitive <- data.frame(accuracy_primitive, precision_primitive,
recall_primitive, f1_primitive)
colnames(classification_report_primitive) <- c("accuracy", "precision",
"recall", "f1")
```

```
knitr::kable(classification_report_primitive)
```

accuracy	precision	recall	f1
0.715	0.7484277	0.875	0.8067797

Jakby nie patrzeć całkiem satysfakcjonujące wyniki.

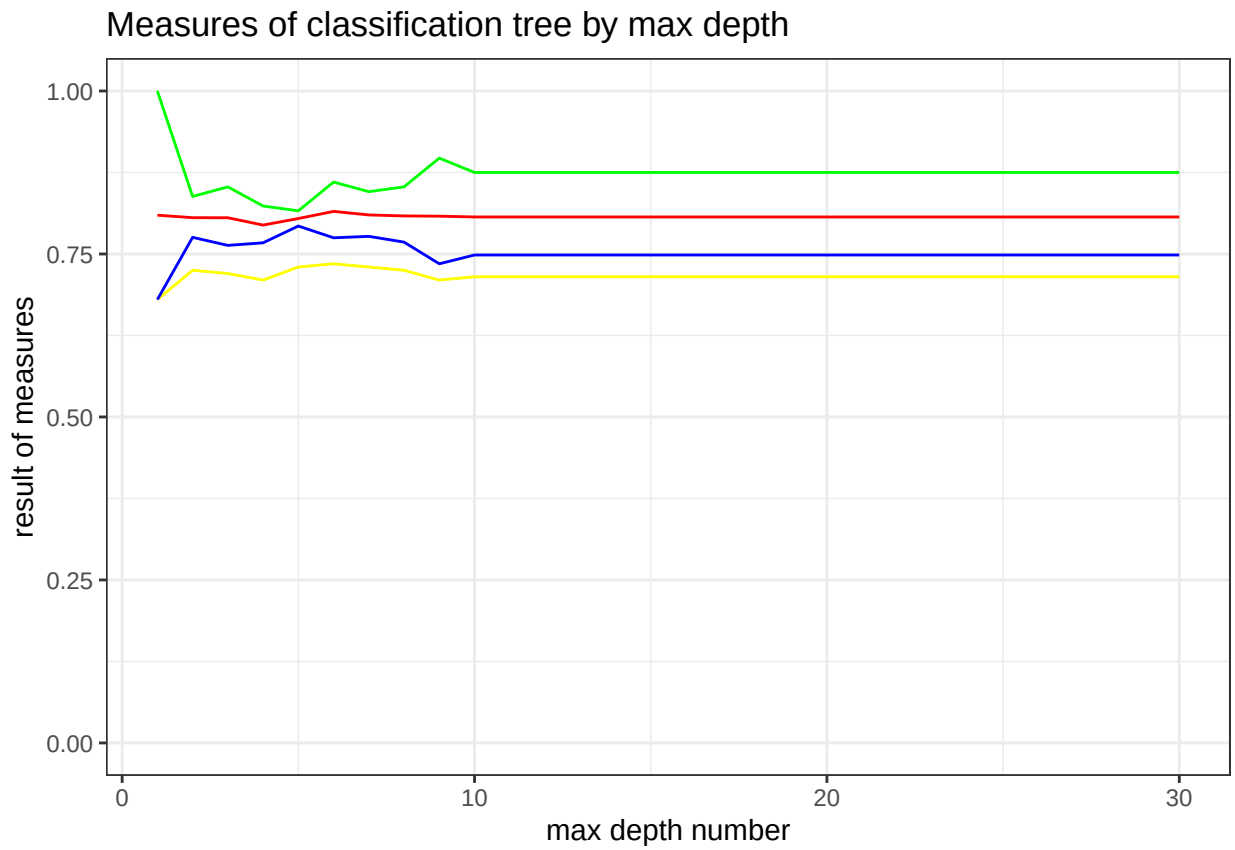
4. Drzewa z ustawionymi maksymalnymi poziomami wysokości

Może warto ograniczyć wysokość drzewa? Sprawdźmy to!

Za najlepszą miarę uznam **f1** i myślę, że jest to dobry krok, łączymy w jedno wszystkie cztery przypadki TP, FP, TN, FN i dla balansu takiego jak w naszej ramce powinniśmy otrzymać satysfakcjonujący rezultat.

```
indexes <- 1:30
with_max_depths_models <- list()
y_preds <- list()
acccuracy_scores <- list()
precision_scores <- list()
recall_scores <- list()
f1_scores <- list()

for (i in indexes){
with_max_depths_model <- rpart(is_good_customer_type ~ ., data = X, method = "class", control = rpart.c
with_max_depths_models[[i]] <- with_max_depths_model
y_preds[[i]] <- predict(with_max_depths_model, X_test, type = "class")
confusion_matrix <- table(Truth = y_test$is_good_customer_type, Prediction = y_preds[[i]])

acccuracy_scores[[i]] <- accuracy(confusion_matrix)
precision_scores[[i]] <- precision(confusion_matrix)
recall_scores[[i]] <- recall(confusion_matrix)
f1_scores[[i]] <- f1(confusion_matrix)
}

acccuracy_scores <- unlist(acccuracy_scores)
precision_scores <- unlist(precision_scores)
recall_scores <- unlist(recall_scores)
f1_scores <- unlist(f1_scores)

measures <- data.frame(1:30, acccuracy_scores, precision_scores, recall_scores, f1_scores)

library(ggplot2)
ggplot(measures, aes(x = X1.30)) +
geom_line(aes(y = acccuracy_scores), color = "yellow") +
geom_line(aes(y = precision_scores), color = "blue") +
geom_line(aes(y = recall_scores), color = "green") +
geom_line(aes(y = f1_scores), color = "red") +
theme_bw() +
xlab("max depth number") +
ylab("result of measures") +
```

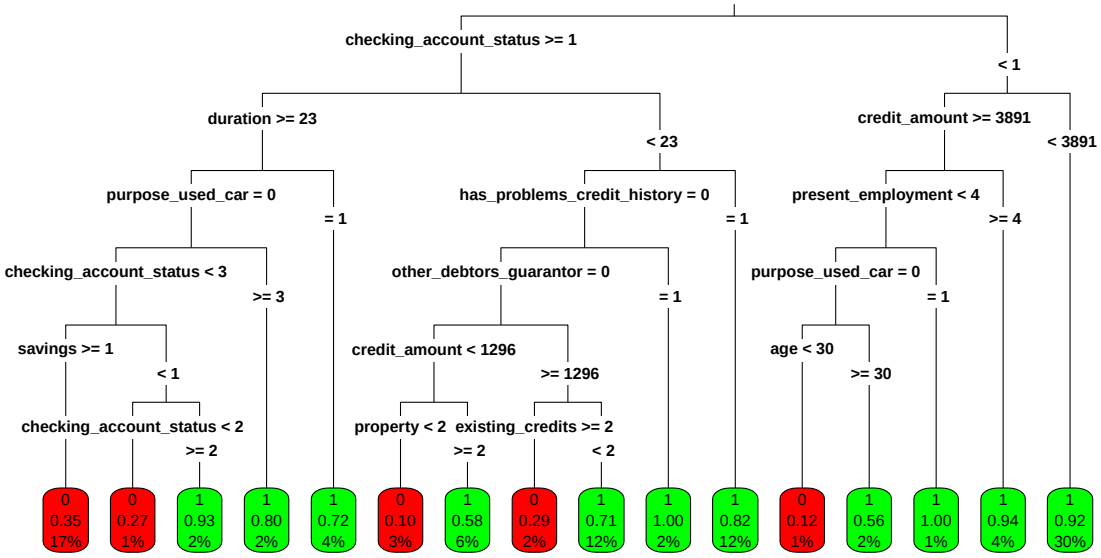
```
ylim(c(0, 1)) +
ggtitle("Measures of classification tree by max depth")
```



Co możemy stwierdzić po wykresie?

1) od głębokości 11 w górę wynik jest zawsze taki sam, czyli nie generują się już sensowniejsze głębsze drzewa.
2) mając kilka miar ciężko jednoznacznie stwierdzić, która jest najlepsza, jak dla mnie jednak będzie to ta **dla parametru 6** - chyba najważniejsza miara, biorąc pod uwagę nasz zbiór *f1* przyjmuje maksymalną wartość, tak samo *accuracy*, *recall* jest lepsze niż dla dwóch sąsiadujących wartości i *precision* również niczego sobie. No i wiadomo, prostota na plus!

```
maxd6_model <- rpart(is_good_customer_type ~ ., data = X, method = "class", control = rpart.control(cp =

rpart.plot(maxd6_model, type = 3, box.palette = c("red", "green"), fallen.leaves = TRUE)
```




```
y_pred <- predict(maxd6_model, X_test, type = "class")
y_pred <- as.data.frame(y_pred)

confusion_matrix_maxd6 <- table(Truth = y_test$is_good_customer_type, Prediction = y_pred$y_pred)

knitr::kable(confusion_matrix_maxd6)
```

	0	1
0	30	34
1	19	117

```
accuracy_maxd6 <- accuracy(confusion_matrix_maxd6)
precision_maxd6 <- precision(confusion_matrix_maxd6)
recall_maxd6 <- recall(confusion_matrix_maxd6)
f1_maxd6 <- f1(confusion_matrix_maxd6)

classification_report_maxd6 <- data.frame(accuracy_maxd6, precision_maxd6,
recall_maxd6, f1_maxd6)
colnames(classification_report_maxd6) <- c("accuracy", "precision",
"recall", "f1")
knitr::kable(classification_report_maxd6)
```

accuracy	precision	recall	f1
0.735	0.7748344	0.8602941	0.815331

I już mamy miary fajniejsze!

5. Las losowy

```
library(randomForest)

X$is_good_customer_type <- as.factor(X$is_good_customer_type)
random_forest <- randomForest(is_good_customer_type ~ ., data = X, proximity=T)

y_pred_forest <- predict(random_forest, X_test, method = "class")

confusion_matrix_forest <- table(Truth = y_test$is_good_customer_type, Prediction = y_pred_forest)

knitr::kable(confusion_matrix_forest)
```

	0	1
0	20	44
1	8	128

```
accuracy_forest <- accuracy(confusion_matrix_forest)
precision_forest <- precision(confusion_matrix_forest)
recall_forest <- recall(confusion_matrix_forest)
f1_forest <- f1(confusion_matrix_forest)

classification_report_forest <- data.frame(accuracy_forest, precision_forest,
recall_forest, f1_forest)
colnames(classification_report_forest) <- c("accuracy", "precision",
"recall", "f1")
knitr::kable(classification_report_forest)
```

accuracy	precision	recall	f1
0.74	0.744186	0.9411765	0.8311688

6. Podsumowanie i najlepszy model

Porównajmy nasze miary.

```
comparison_frame <- data.frame(rbind(classification_report_primitive, classification_report_maxd6, class
comparison_frame <- cbind(model = c("primitive", "max depth: 6", "random forest"), comparison_frame)
comparison_frame
```

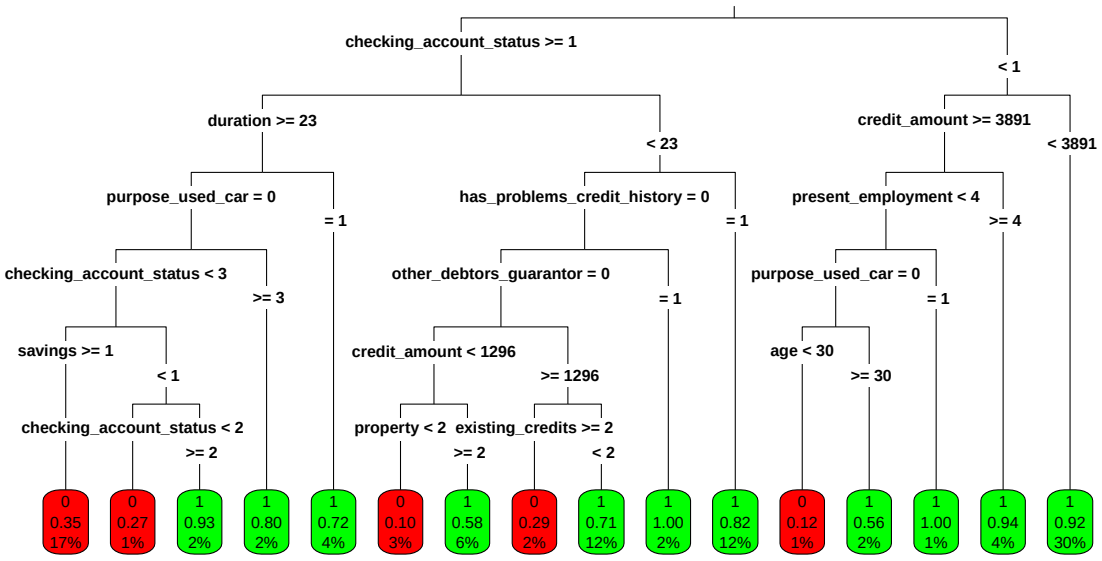
```
##           model accuracy precision    recall        f1
## 1     primitive    0.715 0.7484277 0.8750000 0.8067797
## 2  max depth: 6    0.735 0.7748344 0.8602941 0.8153310
## 3 random forest    0.740 0.7441860 0.9411765 0.8311688
```

Sprawa wyboru najlepszego modelu nie jest taka prosta. Biorąc pod uwagę **recall**, zdecydowanie najlepiej wypada *random forest* i także dla **f1** jest całkiem całkiem (choć tam bardzo małe różnice). Model ten radzi sobie jednak najgorzej w przypadku **precision**, gdzie dominuje ten, gdzie głębokość drzewa ograniczamy do 6. Ten sam model radzi sobie też najlepiej w przypadku **accuracy**, choć nie odstaje od swoich kolegów.

Na pewno **najgorzej wypadł model prymitywny**. Biorąc pod uwagę logikę biznesową i problem, uważam jednak **max depth = 6** za najlepszy - *precision* jest o tyle istotne, że nie chcemy dać zdolności kredytowej klientowi, który może zawalić i sprawić problemy. Niedanie pożyczki osobie, która prawdopodobnie nie zrobiłaby nic złego to potencjalnie mniejsza szkoda dla firmy.

Przyjrzyjmy się jeszcze raz naszemu najlepszemu drzewku.

```
rpart.plot(maxd6_model, type = 3, box.palette = c("red", "green"), fallen.leaves = TRUE)
```



Dodatkowym walorem modelu jest fakt, że jest dość prosty i łatwo wytłumaczalny. Prosto możemy przekazać, kto nie dostanie pożyczki.