

# KM3NaiveBayes

Mikołaj Malec

April 21, 2020

## Czym jest algorytm Naiwnego Bayesa?

Jest to klasyfikator, a raczej rodzina klasyfikatorów, która opiera swoje działanie na wzorze Bayesa. Podstawowym założeniem jest tutaj niezależność wszystkich zmiennych. Sprawdźmy czy jest to prawdą

## Przygotowanie danych

Nasza ramka danych nie jest przygotowana na takie algorytmy. Wiele kolumn ma wartości ciągłe jak np.: wzrost czy wynagrodzenie. Musimy podzielić takie wartości na przedziały tak aby nasze prawdopodobieństwa warunkowe nie miały tylko jednego przypadku.

```
data <- read.csv("german_credit_data_weka_dataset.csv")

levels(data[,1]) <- c("low", "fair", "high", "not_have") #DM low<0<fair<200<high
levels(data[,3]) <- c("all_paid", "all_paid_here", "paid_till_now", "delay", "critical")
levels(data[,4]) <- c("new_car", "used_car", "furniture/equipment", "radio/television", "domestic", "repairs")
levels(data[,6]) <- c("low", "normal", "high", "very_high", "not_have/unknown") #DM low<100<normal<500<high
levels(data[,7]) <- c("unemployed", "less_than_year", "1-3_years", "4-6_yeras", "7+_years")
levels(data[,9]) <- c("male_d/s", "female_d/s/m", "male_single", "male_m/w") #d = divorced, s = separat
levels(data[,10]) <- c("none", "co-applicant", "guarantor")
levels(data[,12]) <- c("real_estate", "building_savings", "car", "not_have/unknown")
levels(data[,14]) <- c("bank", "stores", "none")
levels(data[,15]) <- c("rent", "own", "for_free")
levels(data[,17]) <- c("unskilled_non_resident", "unskilled_resident", "skilled_employee", "highly_qual")
levels(data[,19]) <- c("no", "yes")
levels(data[,20]) <- c("yes", "no")
data[,21] <- as.factor(as.character(data[,21]))
levels(data[,21]) <- c("Good", "Bad")

#podział ciągłych danych.
data$age <- cut( data$age, breaks = seq( 10, 80, by = 10))
data$duration <- cut( data$duration, breaks = c(0,12,24,36,48,60,72))

for (column in names(data)) {data[,column] <- as.factor( data[,column])}
n <-which( names( data) == "customer_type")
set.seed(3114)
rows <- sample(nrow(data))
num_data <- data[rows, ]

test_data <- head(data,n = 200)
train_data <- tail(data,n = 800)
```

## Metryka Dokładności

Do przetestowania modelu użyjemy czterech metryk.

```
accuracy <- function( table_in){
  sum( diag( table_in)) / sum( table_in)
}

f1 <- function( table_in) {
  recall <- table_in[2,2] / sum( table_in[2,])
  precicion <- table_in[2,2] / sum( table_in[,2])
  ( 2*recall*precicion) / (recall + precicion)
}

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))
}

accuracy2 <- 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_m
})

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]))
}

f2 <- function(confusion_matrix){
  conf_matrix <- confusion_matrix_values(confusion_matrix)
  rec <- recall(confusion_matrix)
  prec <- precision(confusion_matrix)
  return(2 * (rec * prec) / (rec + prec))
}
```

## Trening Naszego Modelu

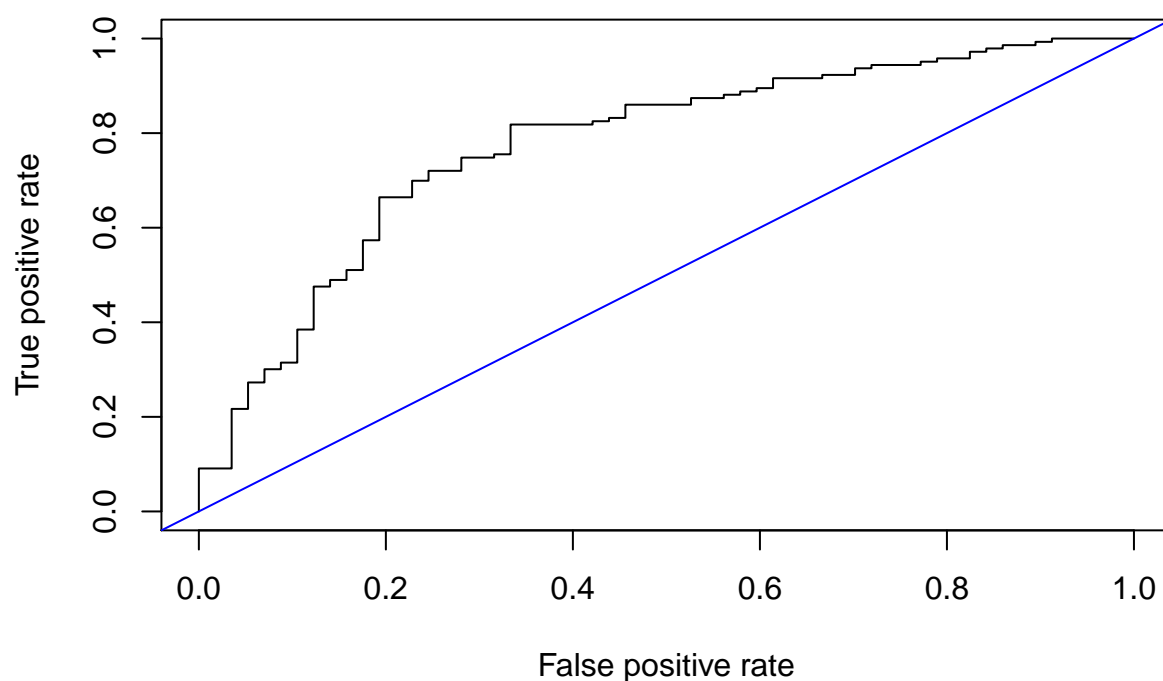
Do modelowania użyjemy funkcji `naiveBayes()` z pakietu "e1071". Oprócz danych i zmiennej celu nasz model ma jeszcze parametr **laplace**. Parametr ten jest głównie używany w NLP i po krótkiej analizie wnioskuje że nie wpływa on dobrze na model dlatego zostawimy go na domyślnym parametrze 0.

```
true_labels <- test_data[,n]
#training / no need for higher laplace;
nB <- naiveBayes( customer_type~. , data = train_data, laplace = 0)
pred_nB_raw <- predict( nB, test_data[-n], type = "raw")
```

## Krzywa ROC

Pokażmy teraz krzywą ROC

```
library( ROCR)
x <- prediction( pred_nB_raw[,1], true_labels)
ROC <- performance( x, "tpr", "fpr")
plot(ROC, col = as.list(1:10))
abline( 0 ,1, col = "blue")
```



## Najlepsze Parametry Odcięcia

Nasz model nie klasyfikuje binarnie tylko wyznacza prawdopodobieństwa dla każdej klasy zmiennej celu. Nasz model ma tylko dwie wartości 0 i 1. Można więc spróbować ustawić taki punkt odcięcia dla którego nasz model okaże się najlepszy według naszych metryk.

```
acc <- rep(0,9)
f<-rep(0,9)
rec<-rep(0,9)
pre<-rep(0,9)
for (i in 1:9) {
  pred_nB <- factor( ifelse( pred_nB_raw[,1] > i/10, "Good", "Bad"), levels = c("Good","Bad"))
  tab <- table( true_labels, pred_nB)
  acc[i]<-accuracy(tab)
```

```
f[i]<- f1(tab)
rec[i] <- recall(tab)
pre[i] <- precision(tab)
}
acc_max <- which.max(acc)
f1_max <- which.max(f)
```

Najlepsze parametry odcięcia otrzymujemy dla metryki celność w stosunku 60%/40%.

Najlepsze parametry odcięcia otrzymujemy dla metryki f1 w stosunku 60%/40%.

## Dokładne wyniki naszego modelu

Podumujmy teraz prace naszego modelu:

```
i=f1_max
pred_nB <- factor( ifelse( pred_nB_raw[,1] > i/10, "Good", "Bad"), levels = c("Good","Bad"))
tab <- table( true_labels, pred_nB)
knitr::kable(tab)
```

	Good	Bad
Good	117	26
Bad	19	38

Dla punktu odcięcia 0.6 otrzymujemy następujące wyniki:

Celność : 0.775

F1 : 0.628099173553719

Recall : 0.666666666666667

Precision : 0.59375