Klasyfikacja danych: German Credit Data

Autor: Robert Okonek, e-mail: robertokonek47@gmail.com

1. Wprowadzenie/Opis problemu

Celem projektu jest zaprezentowanie analizy zbioru danych **German Credit Data** (dostępnym na stronie internetowej: "https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29") w celu stworzenia najbardziej optymalnego modelu uczenia maszynowego służącego do klasyfikacji nowych klientów na tych, którym: powinno się udzielić kredytu lub nie powinno udzielić się kredytu. Klient, któremu powinno się udzielić kredytu to taki, który jest wiarygodny w tym, że po ustalonym terminie go spłaci. Otrzymuje on tak jakby pozytywną zdolność kredytową.

Zbiór danych opisuje dane osób ubiegających się o kredyt w banku. Nasz zbiór danych zawiera 1000 obserwacji (czyli 1000 klientów) i 21 zmiennych opisujących poszczególne cechy dla każdego z klientów. Najważniejsza jest ostatnia zmienna (kolumna) response, która podaje informację, czy dany klient otrzymał pożyczkę (1-otrzymał pożyczkę, 0-nie otrzymał pożyczki).

2.Przyjęte założenia w projekcie

W omawianym projekcie będę starał się do minimum używać wzorów matematycznych i skomplikowanych pojęć. Spróbuję wytłumaczyć wyniki i stosowane instrukcje w sposób jak najbardziej zrozumiały i intuicyjny. Mogę używać również trochę bardziej potocznego języka niż to wypada, ale wtedy o wiele prościej analizuje mi się dane. Niemniej jednak, zawsze warto poznać dlaczego i jak omawiana metoda działa. Tutaj sugeruję już skorzystać z książek omawiających zagadnienia z obszaru Machine Learning/Statystyka, aby to wszystko lepiej zrozumieć.

3.Instalacja i wczytywanie pakietów

Największą zaletą jezyka programowania R są jego pakiety, dzięki którym można wykonywać wiele skomplikowanych analiz w łatwiejszy sposób. Poniżej jeśli dany pakiet nie jest zainstalowany na komputerze, to program go zainstaluje. Pożniej ten pakiet będzie wczytany.

```
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
if (!require('ggthemes')) install.packages('ggthemes'); library('ggthemes')
if (!require('rpart')) install.packages('rpart'); library('rpart')
if (!require('rpart.plot')) install.packages('rpart.plot'); library('rpart.plot')
if (!require('RColorBrewer')) install.packages('RColorBrewer'); library('RColorBrewer')
if (!require('rattle')) install.packages('rattle'); library('rattle')
if (!require('randomForest')) install.packages('randomForest'); library('randomForest')
if (!require('e1071')) install.packages('e1071'); library('e1071')
if (!require('adabag')) install.packages('adabag'); library('adabag')
if (!require('gbm')) install.packages('gbm'); library('gbm')
if (!require('data.table')) install.packages('data.table'); library('data.table')
if (!require('mltools')) install.packages('mltools'); library('mltools')
if (!require('xgboost')) install.packages('xgboost'); library('xgboost')
```

4.Wczytywanie danych i podstawowa analiza

Najpierw należy wczytać dane. Później zmieniamy nazwy kolumn na podstawie opisu ze strony internetowej (sekcja Attribute Information:).

```
dataset=read.table("german.data")
colnames(dataset) <- c("chk_acct", "duration", "credit_his", "purpose",
"amount", "saving_acct", "present_emp", "installment_rate", "sex", "other_debtor",
"present_resid", "property", "age", "other_install", "housing", "n_credits",
"job", "n_people", "telephone", "foreign", "response")
```

Możemy zauważyć, że faktycznie mamy 1000 obserwacji (wierszy) i 21 zmiennych (kolumn).

```
dim(dataset)

## [1] 1000   21
```

Teraz przetłumaczmy sobie jakoś (najłatwiej z użyciem google tłumacza) co opisują poszczególne kolumny w naszym zbiorze danych (tutaj powownie opisane jest to w sekcji Attribute Information:).

- chk_acct: Status istniejącego rachunku rozliczeniowego
- duration: okres w miesiącach
- credit_his: Historia kredytowa
- purpose: Cel (samochód,meble,edukacja)
- amount: Kwota kredytu
- saving_acct: Konto oszczędnościowe / obligacje
- present_emp: Obecne zatrudnienie od
- installment_rate: Stawka rat w procentach dochodu do dyspozycji
- sex: Stan cywilny i płeć
- other_debtor: Inni dłużnicy / poręczyciele
- present_resid: Obecne miejsce zamieszkania od
- property: Majątek (nieruchomości, ubezpieczenia na życie)
- age: Wiek w latach
- other_install: Inne plany ratalne (bank, sklepy, brak)
- housing: Mieszkania (czynsz, własne, bezpłatne)
- n_credit: Liczba istniejących kredytów w tym banku
- job: Praca/Zawód
- n_people: Liczba osób zobowiązanych do utrzymania
- telephone: Telefon
- foreign: Obcokrajowy pracownik
- response: tak/nie (chodzi o zdolność kredytową: pozytywną/negatywną)

Poniżej trzy pierwsze obserwacje omawianego zbioru danych.

```
head(dataset,3)
```

```
##   chk_acct duration credit_his purpose amount saving_acct present_emp
## 1      A11        6        A34     A43   1169         A65         A75
## 2      A12       48        A32     A43   5951         A61         A73
## 3      A14       12        A34     A46   2096         A61         A74
##   installment_rate sex other_debtor present_resid property age other_install
## 1                4 A93         A101             4     A121  67          A143
## 2                2 A92         A101             2     A121  22          A143
## 3                2 A93         A101             3     A121  49          A143
##   housing n_credits  job n_people telephone foreign response
## 1    A152         2 A173        1      A192    A201        1
## 2    A152         1 A173        1      A191    A201        2
## 3    A152         1 A172        2      A191    A201        1
```

Można zastanawiać się co oznaczają niektóre wartosci, np. A43 dla zmiennej purpose. Ponownie należy wtedy skorzystać ze strony internetowej: "https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29", na której jest to wytłumaczone (w sekcji Attribute Information:). Możemy odczytać, że zmienna A43 oznacza radio/television, czyli radio/telewizję.

Teraz opiszę w skrócie w jaki sposób mogą być zapisywane zmienne:

- numerical(quantitative,continous) - numeryczne, ciągłe. Intuicyjnie to może być, np. zmienna opisująca wzrost. Przyjmuje ona ciągłą wartość i każda z wartości może być (zwykle jest) różna.
- qualitative - jakościowe. Intuicyjnie to może być, np. zmienna opisująca płeć. Przyjmuje ona skończoną liczbę wartości (Kobieta lub Mężczyzna). Jest to przykład zmiennej binarnej, gdyż mamy dwie klasy (Kobieta i Mężczyzna). Możemy mieć też zmienną opisującą na przykład kolory (czerowny,niebieski,zielony). Wtedy taka zmienna przyjmuje 3 klasy. Aby przekształcić zmienną ciągłą na jakościową w programie R, należy użyć funkcji as.factor().

W naszym zbiorze danych zmienna response jest zmienną binarną przyjmującą wartości 1 lub 0. Jeśli dla jakiejś obserwacji zmienna response przyjmie wartość 1, to ta obserwacja wskazuje klienta, którego zdolność kredytowa jest pozytywna, czyli powinien otrzymać pożyczkę, gdyż jest to wiarygodne, że ją spłaci.

Pierwsze, co należy wykonać to zamienić wartości zmiennej response. Przyjmuje ona wartości 2 lub 1, a będziemy chcieli aby przyjmowała ona wartości 1 lub 0. Następnie z użyciem omówionej już funkcji as.factor() przekształcimy naszą zmienną response na jakościową. Kiedy będziemy budować modele uczenia maszynowego, to wartości tej zmiennej będziemy starali się przewidywać.

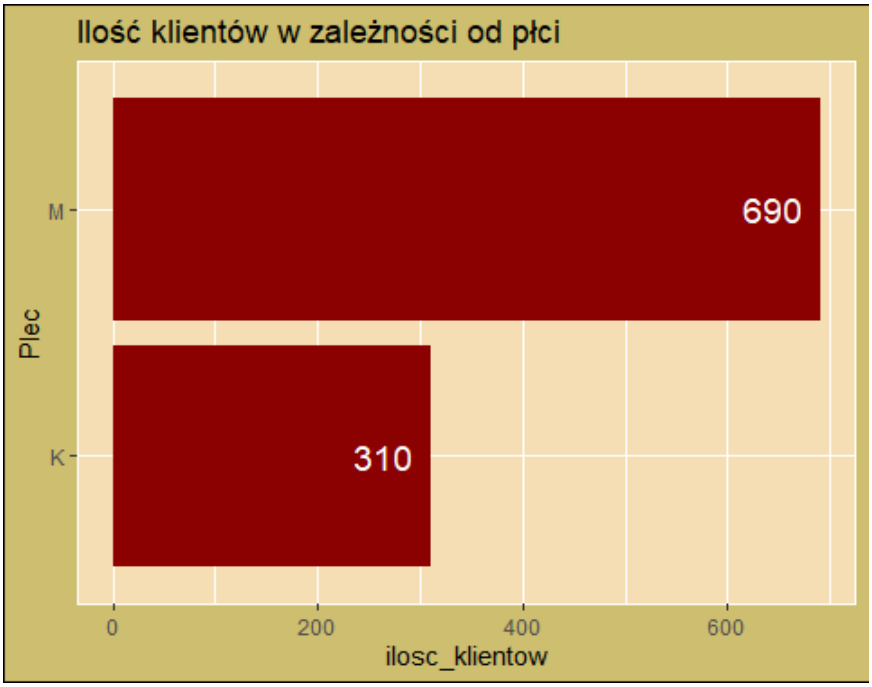
```
dataset$response = ifelse(dataset$response==2,1,0)
dataset$response <- as.factor(dataset$response)
```

Poniżej z wykorzystaniem pakietu ggplot2 można utworzyć taki elegancki (na pewno dla mnie) wykres kolumnowy. Widać, że 700 klientów miało negatywną zdolność kredytową (zmienna response wynosiła 0) i 300 klientów miało pozytywną zdolność kredytową (zmienna response wynosiła 1). W nazwach zmiennych unikam polskich znaków, aby nie było ewentualnych problemów z wykonywaniem poleceń.

```
data <- data.frame(
  zdolnosc_kredytowa=as.factor(0:1),
  ilosc_klientow=as.vector(table(dataset$response))
)
ggplot(data, aes(x=zdolnosc_kredytowa, y=ilosc_klientow)) +
  geom_bar(stat="identity", fill="steelblue")+
  geom_text(aes(label=ilosc_klientow), vjust=1.6, color="white", size=5)+
  ggtitle("Ilość klientów w zależności od zdolności kredytowej")+
  theme(panel.background = element_rect(fill = 'lightblue', colour = 'white'))+
  theme(plot.background = element_rect(fill = 'skyblue', colour = 'black'))
```

Teraz dowiedzmy się ile jest kobiet i mężczyzn w naszych obserwacjach. Ta informacja jest zawarta w zmienej sex, ale niestety zawiera ona też informacje na temat stanu cywilnego danej osoby. Na podstawie opisu zbioru danych odszukujemy, że zmienna sex przyjmuje wartość A92 i A95 dla kobiet w zależności od stanu cywilnego. Można wykorzystać do tego pętlę for oraz instrukcję warunkową if jak poniżej. Na końcu narysowany jest wykres kolumnowy przestawiający tę informację.

```
Plec=c()
for (k in 1:dim(dataset)[1]) {
  if (dataset$sex[k]=="A92" || dataset$sex[k]=="A95"){
    Plec[k]="K"
  }else{
    Plec[k]="M"
  }
}
data2 <- data.frame(
  Plec=as.factor(c("K","M")),
  ilosc_klientow=as.vector(table(Plec))
)
  ggplot(data2, aes(x=Plec, y=ilosc_klientow)) +
    geom_bar(stat="identity", fill="darkred")+
    geom_text(aes(label=ilosc_klientow), hjust=1.3, color="white", size=5)+
    ggtitle("Ilość klientów w zależności od płci")+ coord_flip()+
    theme(panel.background = element_rect(fill = 'wheat', colour = 'white'))+
    theme(plot.background = element_rect(fill = 'lightgoldenrod3', colour = 'black'))
```



Na podstawie wykresu widać, że z 1000 klientów: 310 to kobiety i 690 to mężczyzni.

Generalnie podczas przeglądania danych warto skorzystać z funkcji: str() oraz summary(). Pierwsza z nich przedstawia strukturę danych, czyli informację o typach zmiennych. Druga instrukcja, wyświetla statystyki opisowe dla zmiennych znajdujących się w zbiorze danych.

```
str(dataset)
## 'data.frame':    1000 obs. of  21 variables:
##  $ chk_acct        : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ duration        : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_his      : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ purpose         : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ amount          : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ saving_acct     : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ present_emp     : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ installment_rate: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ sex             : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ other_debtor    : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ present_resid   : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ property        : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ age             : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ other_install   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ housing         : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ n_credits       : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job             : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ n_people        : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone       : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign         : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ response        : Factor w/ 2 levels "0","1": 1 2 1 1 2 1 1 1 1 2 ...
```

Powyżej widać typ każdej zmiennej w zbiorze danych. Factor oznacza, że dana zmienna jest typu jakościowego, a int - typu ciągłego. Oczywiście zawsze w zależności od potrzeby można przekształcić typ zmiennej na inny. Poniżej użyto funkcji summary().

```
summary(dataset)
##  chk_acct     duration    credit_his    purpose        amount      saving_acct
##  A11:274   Min.   : 4.0   A30: 40    A43    :280   Min.   :  250   A61:603
##  A12:269   1st Qu.:12.0   A31: 49    A40    :234   1st Qu.: 1366   A62:103
##  A13: 63   Median :18.0   A32:530    A42    :181   Median : 2320   A63: 63
##  A14:394   Mean   :20.9   A33: 88    A41    :103   Mean   : 3271   A64: 48
##            3rd Qu.:24.0   A34:293    A49    : 97   3rd Qu.: 3972   A65:183
##            Max.   :72.0              A46    : 50   Max.   :18424
##                                      (Other): 55
##  present_emp installment_rate  sex      other_debtor present_resid   property
##  A71: 62     Min.   :1.000    A91: 50   A101:907     Min.   :1.000   A121:282
##  A72:172     1st Qu.:2.000    A92:310   A102: 41     1st Qu.:2.000   A122:232
##  A73:339     Median :3.000    A93:548   A103: 52     Median :3.000   A123:332
##  A74:174     Mean   :2.973    A94: 92                Mean   :2.845   A124:154
##  A75:253     3rd Qu.:4.000                           3rd Qu.:4.000
##              Max.   :4.000                           Max.   :4.000
##
##       age        other_install housing      n_credits       job
##  Min.   :19.00   A141:139      A151:179   Min.   :1.000   A171: 22
##  1st Qu.:27.00   A142: 47      A152:713   1st Qu.:1.000   A172:200
##  Median :33.00   A143:814      A153:108   Median :1.000   A173:630
##  Mean   :35.55                            Mean   :1.407   A174:148
##  3rd Qu.:42.00                            3rd Qu.:2.000
##  Max.   :75.00                            Max.   :4.000
##
##     n_people     telephone  foreign    response
##  Min.   :1.000   A191:596   A201:963   0:700
##  1st Qu.:1.000   A192:404   A202: 37   1:300
##  Median :1.000
##  Mean   :1.155
##  3rd Qu.:1.000
##  Max.   :2.000
```

Dla zmiennych ciągłych są wyliczone statystyki opisowe, jak minimum, pierwszy i trzeci kwantyl, mediana, średnia i maksimum. Natomiast dla zmiennych jakościowych jest zliczone, ile razy występują konkretne wartości w tej zmiennej. Choć widać to powyżej, to zawsze przed dokonywaniem analiz, należy sprawdzić czy nie posiadamy żadnych brakujących wartości w zbiorze danych. Mam na myśli jakieś puste wartości lub przypadkowe wartości będące wynikiem błędu zapisu danych. Sprawdzenie czy w zbiorze danych występuje jakaś brakująca wartość możemy dokonać z użyciem poniższej instrukcji.

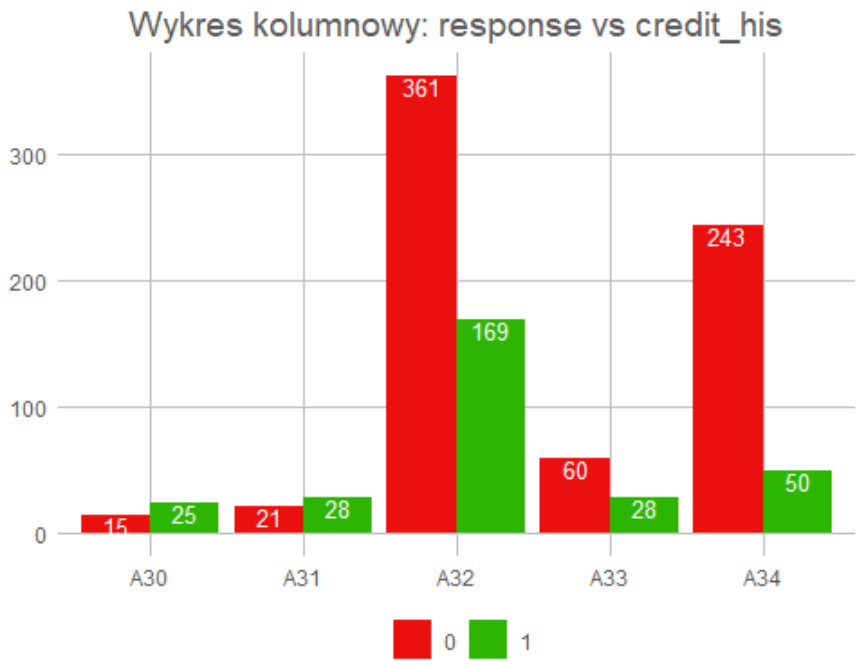
```
any(is.na(dataset)==TRUE)

## [1] FALSE
```

Przydatną rzeczą może być wykonanie kilku wykresów, które przedstawiają zależności pomiędzy dwiema zmiennymi. Kilka takich wykresów utworzę i spróbuję je zinterpretować.

Wykres 1: zależność pomiędzy zmienną credit_his a zmienną response.

```
dattt=as.data.frame(table(dataset$credit_his,dataset$response))
colnames(dattt)=c("credit_his","response","ilosc")
ggplot(data=dattt, aes(x=credit_his, y=ilosc, fill=response)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=ilosc), vjust=1, color="white",
            position = position_dodge(0.9), size=3.5)+
  scale_fill_manual(values=c('#eb1010','#2db504'))+
  ggtitle("Wykres kolumnowy: response vs credit_his")+
  ggthemes::theme_excel_new()
```



Najpierw poniżej wpiszę, co opisują wartości A30,A31, A32, A33 i A34 dla zmiennej credit_his (czyli historia kredytowa).

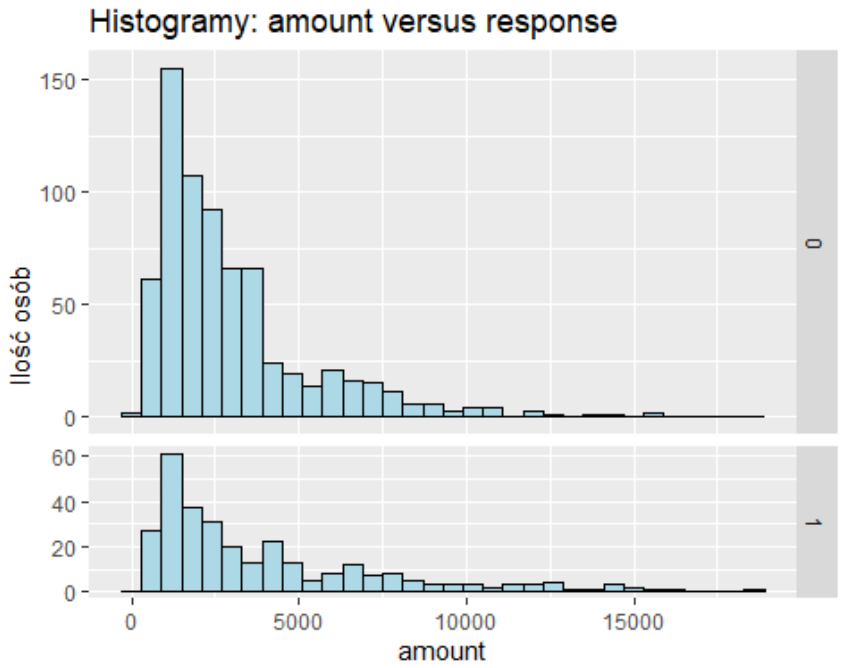
- A30: brak zaciągniętych kredytów / wszystkie kredyty spłacone należycie
- A31: wszystkie kredyty w tym banku spłacone należycie
- A32: dotychczasowe kredyty spłacane należycie
- A33: opóźnienie w spłacie w przeszłości
- A34: konto krytyczne / inne istniejące kredyty (nie w tym banku)

Tak więc na podstawie powyższego wykresu można śmiało stwierdzić ile osób ubiegało się o kredyt w zależności od historii kredytowej oraz jaką otrzymało zdolność kredytową, czyli czy bank takiej osobie udzieliłby kredytu. Po pierwsze, widać, że bardzo dużo osób (nawet większość) ubiegało się o kredyt, nawet jeśli już jakiś miała do spłacenia. W sytuacji, kiedy brakuje środków na spłacenie bieżącego kredytu, wiele osób niestety zadłuża się i próbuje ratować jakoś swoją sytuację życiową decydując się na kolejny kredyt, nawet pewnie na wyższym procencie. Oczywiście zawsze można brać kredyt na kolejną inwestycję i nie musi ona oznaczać problemy finansowe w życiu, ale wydaje mi się, że przeważnie są to problemy finansowe. Niestety pomiędzy różnymi klasami (zmienna credit_his) mamy dosyć spore różnice w ilości osób ubiegających się o kredyt i w większym przypadku możemy mówić o analizowaniu zdolności kredytowej dla osób, które mają już jakiś kredyt do spłacenia. Warto zauważyć, że już na podstawie takiego wykresu możemy coś powiedzieć o tym czy nowemu klientowi ubiegającemu się o kredyt należałoby udzielić pożyczki na podstawie jego historii kredytowej.

Wykres 2: zależność pomiędzy zmienną amount a zmienną response.

```
dattt=data.frame(amount=dataset$amount,response=as.integer(dataset$response)-1)
colnames(dattt)=c("amount","response")
dattt.less10000=dattt[dattt$amount<10000,]

ggplot(dattt, aes(x=amount))+
  geom_histogram(binwidth=600,color="black", fill="lightblue")+
  facet_grid(response ~ ., scales="free", space="free")+
  ggtitle("Histogramy: amount versus response")+
  ylab("Ilość osób")
```



Na powyższym wykresie mamy widoczne dwa histogramy w zależności od tego czy klient miał pozytywną lub negatywną zdolność kredytową. Na osi poziomej mamy widoczne wysokości kredytów, a na osi pionowej ilość klientów ubiegających się o kredyt w zależności właśnie od wysokości kredytu. Można zauważyć, że przeważnie rozpatrywane były kredyty na kwotę do 2500 - 3000 zł (albo może Euro, nie wiem o jaką walutę chodzi, przyjmuję że polski zł). Oba histogramy mają nawet podobny kształt, ale pierwszy z nich jest większy, ponieważ więcej osób uzyskiwało negatywną zdolność kredytową.

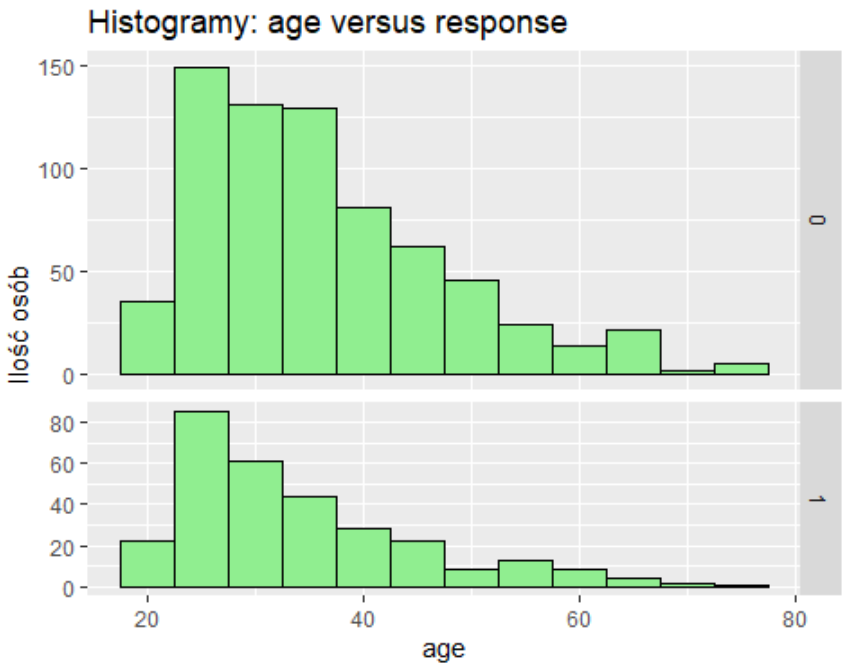
Poniżej wykres przedstawia to samo, ale nie uwzględnia czy kredyt został przyznany czy nie. Jednak, za to lepiej pokazane są przedziały rozpatrywanych wysokości kredytów wraz z ich liczebnością.

```
options(scipen=999)
przedzialy=seq(min(dattt$amount),max(dattt$amount),by=1000)
cc=cut(dattt$amount,breaks=przedzialy,dig.lab=6)
przedzial=data.frame(przedzialy=levels(cc))
ile=as.numeric(table(cc))
przedzial=cbind(przedzial,ile)
ggplot(przedzial, aes(x=przedzialy, y=ile)) +
  geom_bar(stat="identity", fill="darkred")+
  geom_text(aes(label=ile), hjust=-0.5, color="black", size=3)+
  scale_y_continuous(limits = c(0,300))+
  ggtitle("Jakiej wysokości były rozpatrywane kredyty?") + coord_flip()+
  ylab("Ile było takich rozpatrywanych kredytów")
```

Powyższy wykres daje nam lepszą informację o wysokościach rozpatrywanych wykresów. Można chociażby wyliczyć, że (193+172+292)/1000 = 0.657 = 65.7% kredytów było rozpatrywanych do kwoty 3250 zł. A do kwoty 4250 zł było to aż (193+172+292+112)/1000 = 0.769 = 76.9% wszystkich kredytów.

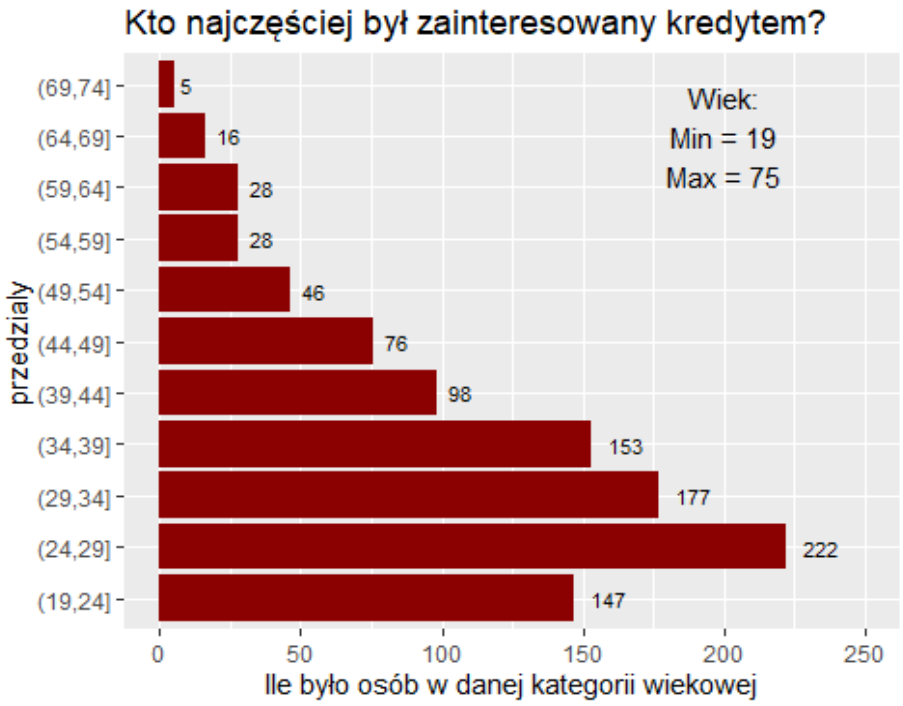
Wykres 3: zależność pomiędzy zmienną age a zmienną response.

```
datt3=data.frame(age=dataset$age,response=as.integer(dataset$response)-1)
colnames(datt3)=c("age","response")
ggplot(datt3, aes(x=age))+
  geom_histogram(binwidth=5,color="black", fill="lightgreen")+
  facet_grid(response ~ ., scales="free", space="free")+
  ggtitle("Histogramy: age versus response")+
  ylab("Ilość osób")
```



Na obu histogramach można zauważyć, że o kredyt najczęściej ubiegają się osoby w wieku 20 - 35 lat bez względu na to czy mają pozytywną czy negatywną zdolność kredytową. Poniższy wykres będzie trochę lepszy, bo będą określone przedziały dla wieku, ale nieuwzględniający czy kredyt został przyznany czy nie.

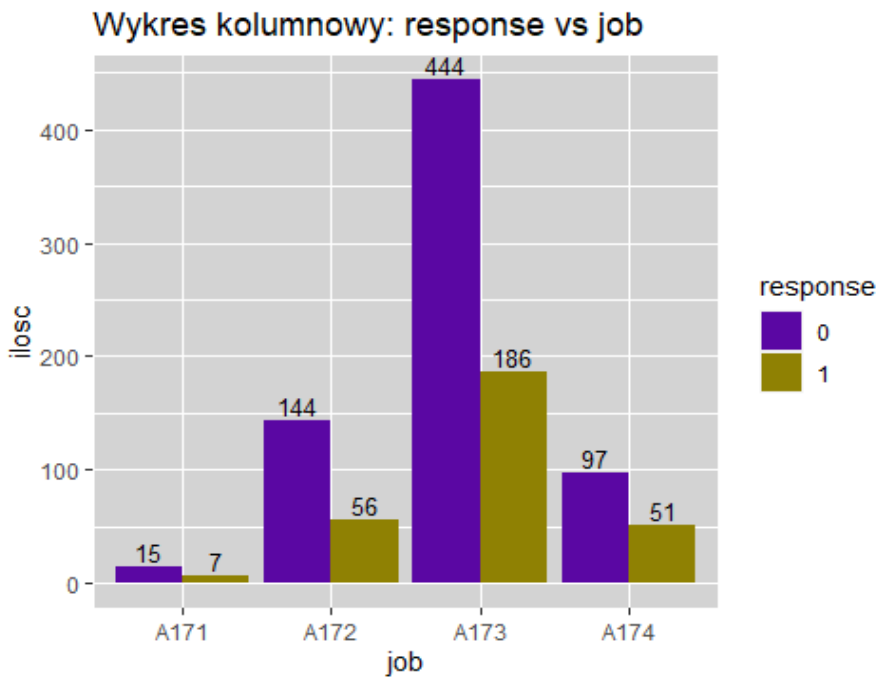
```
przedzialy=seq(min(datt3$age),max(datt3$age), by = 5)
cc2=cut(datt3$age,breaks=przedzialy,dig.lab=6)
przedzial2=data.frame(przedzialy=levels(cc2))
ile2=as.numeric(table(cc2))
przedzial2=cbind(przedzial2,ile2)
ggplot(przedzial2, aes(x=przedzialy, y=ile2)) +
  geom_bar(stat="identity", fill="darkred")+
  geom_text(aes(label=ile2), hjust=-0.5, color="black", size=3)+
  scale_y_continuous(limits = c(0,250))+
  ggtitle("Kto najczęściej był zainteresowany kredytem?") + coord_flip()+
  ylab("Ile było osób w danej kategorii wiekowej")+
  annotate("text", x=10, y=200, label=paste0("Wiek:","\n","Min = ",min(datt3$age),"\n",
  "Max = ",max(datt3$age)))
```



Powyższy wykres daje nam nieco szersze spojrzenie na to jaka grupa wiekowa jest najbardziej zainteresowana wzięciem pożyczki w banku. Wyniki nie są zaskakujące. Większość, bo aż (153+177+222+147)/1000=0.699=69.9% wszystkich osób ubiegających się o kredyt ma mniej niż 40 lat. Najmłodsza osoba ubiegająca się o kredyta miała 19 lat, a najstarsza odpowiednio 75 lat. Można tego się też dowiedzieć z użyciem funkcji summary().

Wykres 4: zależność pomiędzy zmienną job a zmienną response

```
datt4=as.data.frame(table(dataset$job,dataset$response))
colnames(datt4)=c("job","response","ilosc")
ggplot(data=datt4, aes(x=job, y=ilosc, fill=response)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=ilosc), vjust=-0.25, color="black",
            position = position_dodge(0.9), size=3.5)+
  scale_fill_manual(values=c('#5a07a3','#8f8103'))+
  ggtitle("Wykres kolumnowy: response vs job")+
  theme(panel.background = element_rect(fill = 'lightgrey', colour = 'white'))
```



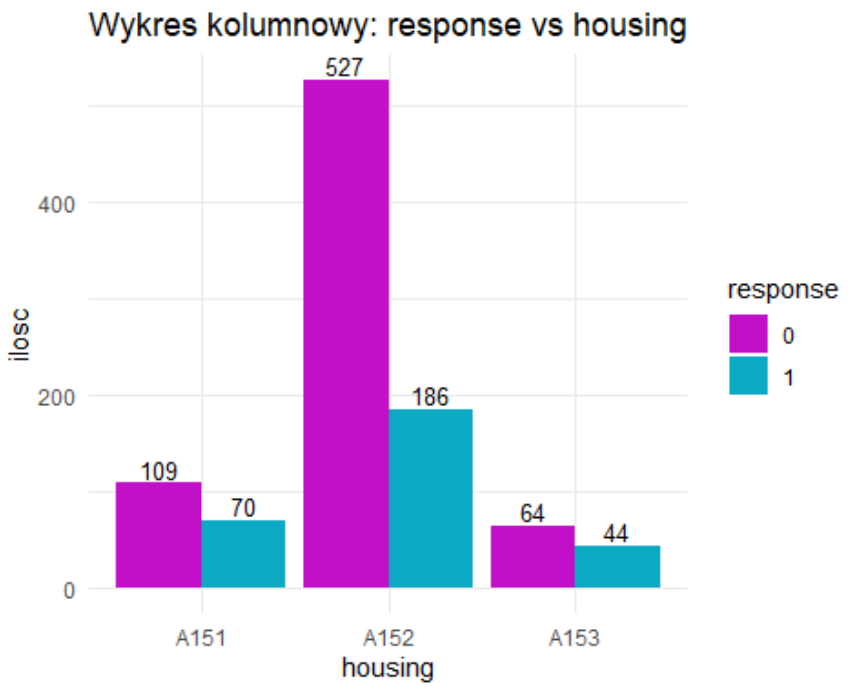
Najpierw poniżej wpiszę, co opisują wartości A171, A172, A173 i A174 dla zmiennej job (czyli Praca/Zawód).

- A171: bezrobotny / niewykwalifikowany - nierezydent
- A172: niewykwalifikowany-mieszkaniec
- A173: wykwalifikowany pracownik / urzędnik
- A174: kierownictwo / samozatrudnienie / wysoko wykwalifikowany pracownik / oficer

Co można zatem odczytać z powyższego wykresu kolumnowego? Przede wszystkim to, że o kredyt ubiegają się w największym stopniu pracownicy wykwalifikowani, bo stanowią oni aż (444+186)/1000=0.63=63% wszystkich osób.

Wykres 5: zależność pomiędzy zmienną housing a zmienną response

```
datt5=as.data.frame(table(dataset$housing,dataset$response))
colnames(datt5)=c("housing","response","ilosc")
ggplot(data=datt5, aes(x=housing, y=ilosc, fill=response)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=ilosc), vjust=-0.25, color="black",
            position = position_dodge(0.9), size=3.5)+
  scale_fill_manual(values=c('#c110c7','#0ca9c4'))+
  ggtitle("Wykres kolumnowy: response vs housing")+
  theme_minimal()
```



Najpierw poniżej wpiszę, co opisują wartości A151, A152 i A153 dla zmiennej housing (czyli Mieszkanie).

- A151: czynsz (czyli na wynajem)
- A152: własne mieszkanie
- A153: darmowe mieszkanie(czyli pewnie socjalne)

Z powyższego wykresu wynika, że najwięcej osób, bo aż (527+186)/1000=0.713=71.3% posiada własne mieszkanie. Jednak, dosyć sporo z tych klientów, nie otrzymuje kredytu. Może to być spowodowane, że właśnie to na własne mieszkanie klient ma już zaciągnięty kredyt i bank nie jest chętny udzielić kolejnej pożyczki. Osoby, które mają mieszkanie na wynajem bądź darmowe, mają znacznie większe szanse na otrzymanie pożyczki. Moim zdaniem, większość osób, jeśli kiedykolwiek decyduje się na kredyt, to jest to kredyt na mieszkanie. Informację o tym, w jakim celu miała być udzielona pożyczka, zawiera zmienna purpose, którą teraz sprawdzimy.

```
sort(base::table(dataset$purpose),decreasing=TRUE)
```

```
##
##  A43  A40  A42  A41  A49  A46  A45 A410  A44  A48
##  280  234  181  103   97   50   22   12   12    9
```

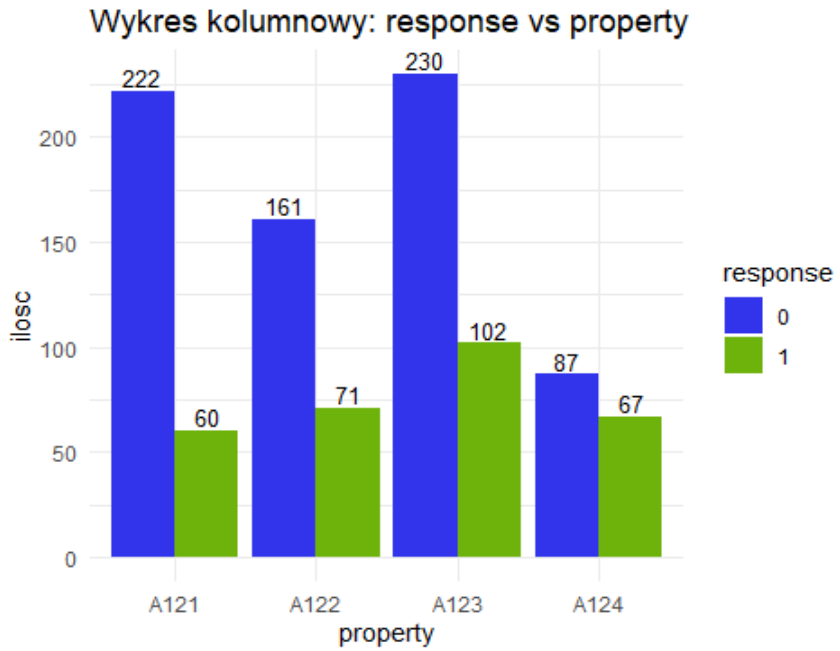
Oczywiście, tak jak ostatnio wypiszę co oznaczaję te niewiadome wartości ze zmiennej purpose.

- A40: samochód (nowy)
- A41: samochód (używany)
- A42: meble / wyposażenie
- A43: radio / telewizja
- A44: sprzęt gospodarstwa domowego
- A45: naprawy
- A46: edukacja
- A47: (urlop - nie istnieje?)
- A48: przekwalifikowanie
- A49: biznes
- A410: inne

Jak widać nie ma wyraźej informacji o tym czy pożyczka miała być udzielona na mieszkanie. Najwidoczniej nie zostało to uwzględnione w zbiorze danych. Można tego też się domyślić po wysokości kredytów. Rozpatrywany Kredyt o najwyższej wartości sięgał kwoty około 18 000 zł. Wysokości kredytów zależą też od tego, kiedy i gdzie dane były zapisywane. Ze strony internetowej można odczytać, że raczej w 2000 r. w Niemczech. Zresztą, ceny mieszkań są dużo większe niż 18 000 zł, więc analizowany zbiór danych przedstawia raczej informacje o zdolności kredytowej klientów na uzyskanie kredytu gotówkowego, a nie kredytu hipotecznego. Wracając do wyników, to widać, że najczęściej o kredyt klienci ubiegali się w celu kupna radia/telewizji lub nowego samochodu. Też sporo osób chciało kupić używany samochód lub meble.

Wykres 6: zależność pomiędzy zmienną property a zmienną response

```
datt6=as.data.frame(table(dataset$property,dataset$response))
colnames(datt6)=c("property","response","ilosc")
ggplot(data=datt6, aes(x=property, y=ilosc, fill=response)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=ilosc), vjust=-0.25, color="black",
            position = position_dodge(0.9), size=3.5)+
  scale_fill_manual(values=c('#3134eb','#6db30c'))+
  ggtitle("Wykres kolumnowy: response vs property")+
  theme_minimal()
```



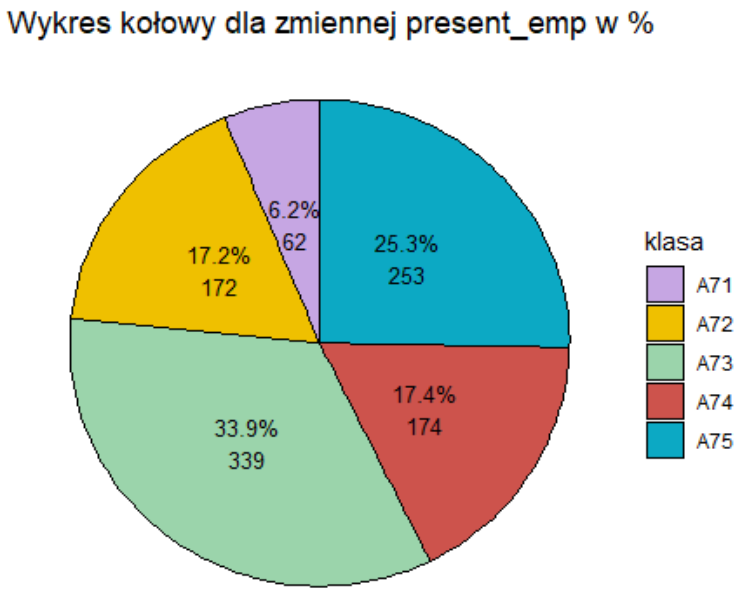
Wypiszmy co oznaczają te niewiadome wartości ze zmiennej property.

- A121: nieruchomość
- A122: umowa oszczędności na spółkę budowlaną lub ubezpieczenie na życie
- A123: samochód lub inny
- A124: nieznane lub brak informacji

Powyższy wykres przedstawia jak rozkłada się zdolność kredytowa klientów w zależności od ich majątku. Oszczędności lub też ubezpieczenie na życie wydają się dawać większe szanse na to, iż otrzymamy pozytywną zdolność kredytową. Ma to sens, gdyż jeśli klient posiada takie coś, to jest przygotowany na ewentualne przykre lub niespodziewane wydarzenie w życiu i będzie miał jak spłacić otrzymany kredyt. Najgorzej w tym zestawieniu wygląda nieruchomość, która może być zwykłym mieszkaniem, które każdy klient raczej posiada.

Wykres 7: Wykres kołowy dla zmiennej present_emp

```
ramka=as.data.frame(table(dataset$present_emp))
colnames(ramka)<-c("klasa","ilosc")
count.data <- data.frame(
  klasa = ramka$klasa,
  ilosc = ramka$ilosc,
  prop = ramka$ilosc/sum(ramka$ilosc)
)
count.data <- count.data %>%
  arrange(desc(klasa)) %>%
  mutate(lab.ypos = cumsum(prop) - 0.5*prop)
mycols <- c("#c6a6e3", "#EFC000FF", "#9bd4ab", "#CD534CFF","#0ca9c4")
ggplot(count.data, aes(x = "", y = prop, fill = klasa))+
  geom_bar(width = 1, stat = "identity", color = "black")+
  coord_polar("y", start = 0)+
  geom_text(aes(y = lab.ypos, label = paste0(prop*100,"%","\n",ilosc)),
            color = "black",size=3.5)+
  scale_fill_manual(values = mycols)+
  theme_void()+ggtitle("Wykres kołowy dla zmiennej present_emp w %")
```



Wypiszmy najpierw co oznaczają niewiadome wartości ze zmiennej present_emp opisującej obecne zatrudnienie w latach.

- A71: bezrobotni
- A72: mniej niż 1 rok
- A73: [1,4) lata
- A74: [4,7) lat
- A75: powyżej 7 lat

Tak więc, na podstawie powyższego wykresu kołowego widać, że około 33% wszystkich klientów to osoby pracujące powyżej 1 roku i zarazem poniżej 4 lat. Druga najbardziej liczna grupa ludzi, bo ponad 25% to osoby mające powyżej 7 lat doświadczenia zawodowego.

Ogólne wnioski:

Uzyskane wyniki na podstawie otrzymanych kilku wykresów pozwalają nam rozpoznać z jakimi danymi mamy do czynienia. Oczywiście nie uwzględniłem jeszcze wielu innych zmiennych, ale chodziło bardziej o to, aby pokazać, że można wyciągnąć jakieś wnioski i je zinterpretować. W mojej ocenie najbardziej ciekawymi informacjami jest to jak dużo osób otrzymało negatywną zdolność kredytową. Było to dokładnie 70% wszystkich klientów. Drugą rzeczą, która wydaje się być interesująca to wysokość kredytów o jaką starali się klienci. Można było odnotować ile łącznie osób ubiegało się o kredyt w wysokości mniejszej niż 3000 zł. Z pewnością można rozpatrzyć więcej zmiennych i dokonywać jeszcze większej ilości analiz, ale na tym zakończę.

5.Uczenie maszynowe - Na czym ono polega i po co je stosować?

Celem uczenia maszynowego jest stworzenie modelu, który na podstawie danych będzie potrafił przewidywać wartości. W kontekście zagadnienia klasyfikacji i naszego zbioru danych, będziemy chcieli, aby na podstawie nowych obserwacji (czyli zawartych informacji o historii kredytowej, kwoty kredytu, płci i wielu innych zmiennych opisujących danego klienta) model z wysoką i zadowalającą dokładnością przewidywał czy należy temu klientowi udzielić kredytu lub nie, to znaczy ustalić czy powinien mieć on pozytywną lub negatywną zdolność kredytową. Zauważmy, że informację o tym czy danemu klientowi udzielić kredytu lub nie, zawiera w sobie zmienna response. Jeśli przyjmuje ona wartość 1, to takiemu klientowi należy udzielić pozytywną zdolność kredytową, a jak przyjmuje wartość 0, to należy udzielić negatywną zdolność kredytową, czyli że nie powinniśmy udzielać mu pożyczki. Czyli podsumowując jeszcze raz, przychodzi do banku nowy klient i znamy jego historię kredytową, płeć, itd., ale nie mamy o nim informacji czy udzielić mu pożyczki lub nie. Dokładnie tego chcemy się dowiedzieć, a ściślej mówiąc, zależy nam na stworzeniu pewnego rodzaju klasyfikatora, który będzie podejmował decyzję, co powinniśmy zrobić z takim klientem. Moim celem będzie utworzenie takich kilku klasyfikatorów i wybranie tego, który najlepiej poradzi sobie z klasyfikowaniem nowych klientów.

Idea jest taka, że na podstawie 1000 obserwacji, bo z tylu wierszy składa się nasz zbiór danych, część bedziemy chcieli wykorzystać do zbudowania modelu, a pozostałą część do przetestowania, w jaki sposób i czy dobrze model klasyfikuje nowe obserwacje, czyli tutaj zdolności kredytowej klientów ubiegających się o kredyt w banku. Jak sama nazwa na to wskazuje, model uczenia maszynowego “uczy się” danych, stąd pojawia się taka nazwa jak Machine Learning, czyli “maszyna ucząca się”. Musimy dostarczyć jej dane, aby lepiej potrafiła “zrozumieć dane”.

Pojawia się pytanie w jaki sposób wykorzystać obserwacje ze zbioru danych. Czy nasza “maszyna” powinna nauczyć się danych na podstawie wszystkich 1000 obserwacji? Nie za bardzo, ponieważ jak już wspomniałem wcześniej, celem jest stworzenie klasyfikatora, który ma dobrze klasyfikować nowe obserwacje, czyli te, dla których nie ma informacji czy danemu klientowi powinniśmy udzielić kredytu. Tę informację model ma ustalić dokładnie na podstawie otrzymanych danych o historii kredytowej klienta, jego płci, wykonywanej pracy i pozostałych zmiennych. Czyli zatem, jak należy wykorzystać te obserwacje ze zbioru danych? Metod jest wiele, ale najprostsza polega na tym, aby podzielić cały zbiór danych na dwie części: na zbiór uczący i zbiór testowy. Zbiór uczący jest używany do tego, aby model na podstawie niego “nauczył się” danych, czyli uchwycił zależności jakie panują pomiędzy zmiennymi i zrozumiał w jaki sposób klasyfikować nowych klientów. Ten drugi zbiór, czyli zbiór testowy, jak sama nazwa wskazuje służy do sprawdzenia, w jaki sposób nasz model radzi sobie z klasyfikowaniem nowych obserwacji, których wcześniej “nie widział”. Być może tłumaczę to wszystko w sposób zbyt potoczny i się powtarzam, ale zakładam, że jeśli osoba, która się na tych zagadniniach zna, to i tak będzie wiedziała o co mi chodzi. Chodzi przede wszystkim o osoby, które nie mają styczności z omawianym zagadnieniem i ważne jest dla mnie to, aby miały możliwość zrozumieć go przynajmniej w pewnym stopniu, tak aby wiedziały o co w tym wszystkim mniej więcej chodzi. Na pewno chodzi też o to, aby nie używać cały czas słowa “chodzi”, ale już uprzedziłem, że będę używał potoczny i prosty (może nawet zbyt prosty) język.

Teraz poniżej zajmiemy się utworzeniem zbioru treningowego (inaczej uczącego) i zbioru testowego. Potem zostanie użytych kilka metod uczenia maszynowego w celu stworzenia najbardziej optymalnego klasyfikatora do ustalania zdolności kredytowej nowych klientów.

Zatem dzielimy zbiór danych na treningowy i testowy w proporcji 70/30, czyli 70% wszystkich obserwacji znajdzie się w zbiorze treningowym, a pozostałe 30% w zbiorze testowym. Oczywiście obserwacje będą wybierane w sposób losowy (odpowiada za to funkcja sample())

```
index=sample(nrow(dataset),nrow(dataset)*0.70)
dataset.train=dataset[index,]
dataset.test=dataset[-index,]
```

Możemy faktycznie zobaczyć, że 700 obserwacji trafiło do zbioru treningowego, a pozostałe 300 do zbioru testowego. W naszych zbiorach danych mamy po 21 zmiennych, w tym jedną o nazwie response, której wartość będziemy chcieli przewidzieć jako 1, czyli pozytywna zdolność kredytowa lub 0-negatywna zdolność kredytowa. Wszystko jest już zatem przygotowane.

```
dim(dataset.train)

## [1] 700  21

dim(dataset.test)

## [1] 300  21
```

6.Metody uczenia maszynowego:

W celu zbudowania klasyfikatora skorzystam z: regresji logistycznej, drzew decyzyjnych, bagging, random forest (las losowy), maszyny wektorów nośnych, naiwnego klasyfikatora baysowskiego i jeśli się uda to też ze wzmocnionych drzew decyzyjnych (czyli tzw. boosting decision trees). Mógłbym spróbować też metody najbliższych sąsiadów, ale ma ona taką wadę, iż obsługuje tylko zmienne ciągłe, a ewentualnie zmienne jakościowe musiałbym jakoś przekształcić na typ liczbowy, co jest trudnym zadaniem. W każdej z tych metod uczenia maszynowego spróbuję chociaż intuicyjnie omówić w jaki sposób ona działa i w jaki sposób wykorzystuje się ją do klasyfikacji danych. Będę jednak stosował wiele uproszczeń i omijał skomplikowane wyjaśnienia, gdyż nie to jest celem tego projektu. W poniżej utworzonej tabelce będę umieszczał uzyskane wyniki przez poszczególne modele.

```
nazwa.metody=c("Regresja logistyczna","Drzewo decyzyjne","Bagging","Las losowy",
               "Maszyna wektorów nośnych,kernel=linear","Maszyna wektorów
nośnych,kernel=polynomial",
               "Maszyna wektorów nośnych,kernel=radial","Naiwny klasyfikator bayesowski",
               "Adaboost","Gradient Boosting","Extreme Gradient Boosting")
dokladnosc=rep(0,length(nazwa.metody))
metody.ML=data.frame(nazwa.metody,dokladnosc)
```

6.1.Regresja logistyczna

Metoda regresji logistycznej polega na wyznaczeniu parametrów równiania regresji logistycznej opisującej zależność pomiędzy zmienną objaśnianą response a pozostałymi zmiennymi objaśniającymi z naszego zbioru danych. Chcemy, aby na podstawie podanych wartości zmiennych objaśniających model wyznaczał prawdopodobieństwo sukcesu, czyli że zmienna response przyjmie wartość 1, a zatem, że dany klient powinien mieć pozytywną zdolność kredytową. Otrzymany model wyznacza prawdopodobieństwo tego, na ile dany klient będzie miał pozytywną zdolność kredytową. Nazywać będziemy to zjawisko sukcesem (czyli że zmienna response przyjmie wartość 1). Regresję logistyczną można wykonać za pomocą funkcji glm() i podając w niej parametr family="binomial" oznaczający, że chcemy zastosować właśnie regresję logistyczną. Poniżej została utworzona też tabela z wyznaczonymi współczynnikami modelu regresji logistycznej oraz odpowiadającymi p-wartościami oznaczającymi istotność tych współczynników. Przy okazji zapis "response~." oznacza, że chcemy przewidzieć wartość zmiennej response używając wszystkich pozostałych zmiennych ze zbioru danych. Wyświetliłem pierwsze 6 wierszy, aby wynik nie zajmował za dużo miejsca w raporcie.

```
m.reglog=glm(response~.,data=dataset.train,family="binomial")
tabela1=data.frame(coefficient=m.reglog$coefficients,
                p.value=summary(m.reglog)$coefficients[,4])
tabela1[1:6,]#tabela1 -aby całą tabelę wyświetlić

##                coefficient           p.value
## (Intercept)     0.53174200 0.697713901098215
## chk_acctA12    -0.44883014 0.097731549100848
## chk_acctA13    -0.68659498 0.153534801879425
## chk_acctA14    -1.64909159 0.000000006526954
## duration        0.01777135 0.121233546038130
## credit_hisA31   0.25760260 0.700588696187323
```

Jak widać jest więcej współczynników niż zmiennych objaśniających. Jest to spowodowane tym, że istnieje współczynnik beta_0, który występuje samodzielnie bez żadnej zmiennej oraz dla zmiennych jakościowych są tworzone zmienne pomocnicze, które uwzględniają klasy występujące w tych zmiennych. Jeśli dana zmienna jakościowa ma n klas, to zamiast jednej zmiennej, w modelu jest utworzonych n-1 zmiennych opisujących tę zmienną jakościową. Są utworzone, tzw. dummy variables, o których nie będę już się rozpisywał. Pierwsza kolumna przedstawia oszacowane współczynniki, a druga kolumna zawiera informację o tzw. p-wartościach dla każdego ze współczynników. W dużym uproszczeniu, jeśli p-wartość jest mniejsza niż 0.05, to uznajemy daną zmienną za istotną w modelu, a więc jest ona ważna przy wyznaczaniu prawdopodobieństwa sukcesu, jakim jest pozytywna zdolność kredytowa. Zatem usuniemy z modelu zmienne objaśniające, dla których p-wartość jest trochę większa od wartości 0.05 lub ewentualnie 0.1. Wiele rzeczy pomijam i nie tłumaczę. Dlaczego podejmuję taką a nie inną decyzję, bo nie taki jest jednak cel tego projektu. Aby w większym stopniu zrozumieć mechanizmy za tym wszystkim stojące, ponownie odsyłam do książek ze statystyki i uczenia maszynowego.

```
m.reglog2=glm(response~chk_acct+duration+credit_his+purpose+amount+saving_acct+installment_rate+ag
e+
                telephone+foreign,
              data=dataset.train,family="binomial")
tabela2=data.frame(coefficient=m.reglog2$coefficients,
                p.value=summary(m.reglog2)$coefficients[,4])
tabela2[1:6,]

##                coefficient           p.value
## (Intercept)     0.07327915 0.918243382108738
## chk_acctA12    -0.48159122 0.054316195455214
## chk_acctA13    -0.72799272 0.118901617530970
## chk_acctA14    -1.61038693 0.000000001329271
## duration        0.01659737 0.109087880626588
## credit_hisA31   0.24664598 0.685703359724059
```

Jak już mamy utworzony model regresji logistycznej zawierający te zmienne objaśniające, które mają istotny wpływ w przewidywaniu zdolności kredytowej klienta, to możemy zastosować ten model do predykcji. Ale o co chodzi z tą predykcją? Chodzi o to, że możemy sprawdzić w jaki sposób nasz model radzi sobie jako klasyfikator służący do klasyfikowania nowych klientów na tych, którym powinniśmy przyznać pozytywną lub negatywną zdolność kredytową. Generalnie model regresji logistycznej wyznacza prawdopodobieństwo, które jest liczbą z przedziału [0,1] i wtedy tę wartość należy zaokrąglić w górę do jedynki lub w dół do zera. Jeśli wyznaczone prawdopodobieństwo p jest większe lub równe od ustalonej wcześniej wartości progowej, to daną obserwację klasyfikujemy jako sukces (pozytywna zdolność kredytowa), a jeśli mniejsze od tej wartości progowej, to to daną obserwację klasyfikujemy jako porażkę (negatywna zdolność kredytowa). Najpierw sprawdźmy w jaki sposób model regresji logistycznej radzi sobie z klasyfikowaniem obserwacji ze zbioru uczącego, czyli na podstawie tych, których został utworzony. Ustalmy wartość progową na 0.5.

```
pre1=predict(m.reglog2,newdata=dataset.train,type="response")
pre1=ifelse(pre1>=0.5,1,0)
head(pre1)

## 404 892 115 236 324 999
##   0   0   0   1   0   1
```

Teraz zobaczmy, jaki wyglądają przewidziane wartości zmiennej response z ich faktycznymi wartościami na zbiorze uczącym. Wyliczymy również dokładność (w skrócie acc od słowa accuracy), czyli jaka część obserwacji została poprawnie sklasyfikowana.

```
tt1=table(predicted=pre1,actual=dataset.train$response)
tt1

##          actual
## predicted   0   1
##         0 440 106
##         1  53 101

sum(diag(tt1))/sum(tt1)

## [1] 0.7728571
```

Widać zatem, że pierwszy wiersz w tabelce (nazywa się ją też tabelą pomyłek) odpowiada za obserwacje, które zostały sklasyfikowane jako porażka (negatywna zdolność kredytowa). Przykładowo, wartość z pierwszego wiersza i drugiej kolumny wskazuje liczbę obserwacji, które zostały sklasyfikowane jako porażkę (zmienna response przyjmuje wtedy wartość 0), a w rzeczywistości (faktycznie) powinny zostać sklasyfikowane jako sukces (wartość 1). Wartości, które są poza głowną przekątną, oznaczają błędnie sklasyfikowane obserwacje. U nas dokładność wynosi około 0.7728571, czyli acc=77.2857143% wszystkich obserwacji zostało poprawnie sklasyfikowanych. Ale nas powinna bardziej interesować informacja jak model radzi sobie z klasyfikowaniem nowych obserwacji, których jeszcze “nie widział”. Sprawdzanie dokładności klasyfikacji obserwacji na zbiorze uczącym ma tylko charakter poglądowy, aby sprawdzić czy model w ogóle radzi sobie z klasyfikowaniem obserwacji. Obliczmy poniżej predykcje na zbiorze testowym i utwórzmy tabelę pomyłek oraz wyliczmy też miarę dokładności.

```
pre2=predict(m.reglog2,newdata=dataset.test,type="response")
pre2=ifelse(pre2>=0.5,1,0)
tt2=table(predicted=pre2,actual=dataset.test$response)
tt2
```

```
##          actual
## predicted   0   1
##         0 185  60
##         1  22  33
```

```
(metody.ML$dokladnosc[1]=sum(diag(tt2))/sum(tt2))
```

```
## [1] 0.7266667
```

Widzimy zatem, że dokładność dla zbioru testowego jest odrobinę niższa niż dla zbioru treningowego (uczącego), lecz nadal na zadowalającym poziomie. Sprawdźmy jeszcze jak dobrze poradziłby sobie model ze wszystkimi dostępnymi zmiennymi objaśniającymi w modelu w klasyfikowaniu obserwacji ze zbioru testowego.

```
pre3=predict(m.reglog,newdata=dataset.test,type="response")
pre3=ifelse(pre3>=0.5,1,0)
tt3=table(predicted=pre3,actual=dataset.test$response)
tt3
```

```
##          actual
## predicted   0   1
##         0 180  58
##         1  27  35
```

```
sum(diag(tt3))/sum(tt3)
```

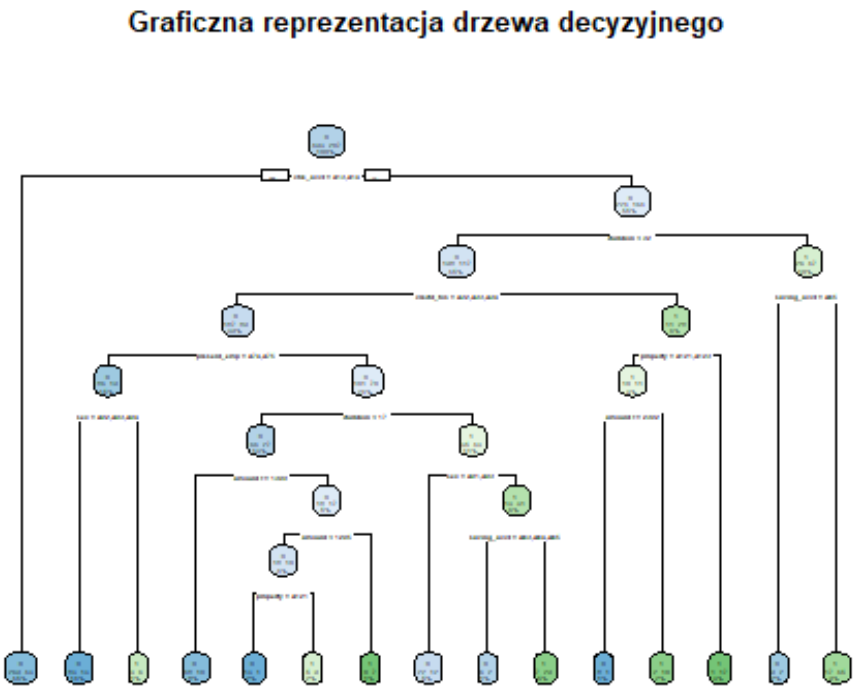
```
## [1] 0.7166667
```

Widać, że dokładność jest nawet odrobinę wyższa i nasuwa się pytanie czy model ze wszystkimi zmiennymi objaśniającymi powinien być lepszym klasyfikatorem od tego drugiego modelu. Moim zdaniem nie, ponieważ warto zauważyć, że różnica w dokładności jest naprawdę mała, a model drugi korzysta tylko z 10 zmiennych objaśniających, w przeciwieństwie do pierwszego, który ma w sobie wszystkie zmienne objaśniające (dokładnie 20 zmiennych objaśniających, czyli 2 razy więcej). Ostatecznie najbardziej optymalnym modelem regresji logistycznej jest ten zawierający 10 zmiennych objaśniających.

6.2.Drzewa decyzyjne

Drzewa decyzyjne są jedną z najprostszych i najbardziej intuicyjnych metod uczenia maszynowego. Nie wierzysz w to? Zaraz to udowodnię. Nasz model uczenia maszynowego służący do klasyfikacji będzie przypominał budową zwykłe drzewo, które na każdym kroku będzie oddzielało obserwacje, w taki sposób, aby klasyfikacja obserwacji była coraz lepsza. Podczas budowania drzewa decyzyjnego porównywana jest każda zmienna objaśniająca ze zmienną objaśnianą (u nas jest to zmienna response), po to aby dowiedzieć się, która zmienna najlepiej nadaje się do rozdzielania obserwacji w takim celu, aby podzbiór obserwacji był jak najbardziej czysty. Co to znaczy, że czysty? Chodzi o to, że na przykład rozdzielamy obserwacje na podstawie kryterium, że do lewego podzbioru mają trafić obserwacje, dla których wartość zmiennej age jest większa od 40, a do prawego podzbioru reszta obserwacji, czyli te obserwacje dla których wartość zmiennej age jest mniejsza lub równa 40. Wtedy mamy dwa podzbiory i cała sztuczka polega na tym, aby w każdym podzbiorze były obserwacje należące do jednej klasy (czyli dla których wartość zmiennej response przyjmuje wartość 1 lub 0). Oczywiście przeważnie nie zawsze uda nam się tak idealnie porozdzielać obserwacje, ale chodzi o to, aby w każdym podzbiorze jedna klasa była coraz bardziej liczna, czyli nasz podzbiór był coraz bardziej czysty. Wszystko będzie o wiele bardziej zrozumiałe jak spojrzy się na graficzną reprezentację drzewa decyzyjnego. Do budowy modelu drzewa decyzyjnego skorzystamy z pakietu rpart, a do jego graficznej reprezentacji pakietu rpart.plot.

```
m.decisiontree=rpart(response~.,data=dataset.train,method="class")
rpart.plot(m.decisiontree, extra=101, main="Graficzna reprezentacja drzewa decyzyjnego")
```



Zatem powyższy wykres przedstawia dość intuicyjnie (przepraszam za słabą jakość wykresu, odsyłam wtedy do pliku html-powinno być lepiej widoczne), w jaki sposób działa nasze drzewo decyzyjne w klasyfikowaniu obserwacji. Każdy prostokąt zawiera w sobie informacje, ile jest obserwacji w każdej z klas w poszczególnym etapie drzewa. Podana jest też informacja o tym ile procent wszystkich obserwacji znajduje się w danym momencie rozdzielania obserwacji oraz do jakiej klasy przydzielana jest obserwacja (kolor niebieski-klasa 0, kolor zielony-klasa 1). Klasa 0 oznacza, iż danej obserwacji przypisujemy negatywną zdolność kredytową. Na samym górze, pierwszy prostokąt nazywany jest korzeniem (węzłem początkowym, startowym). Możemy zauważyć, że w zbiorze uczącym jest 700 obserwacji, z których w 690 wartość zmiennej response wynosi 0, czyli tyle klientów miało ustaloną negatywną zdolność kredytową. Prostokąty leżące po środku, czyli nie w ostatnim rzędzie oraz nie na samej górze (czyli korzeń) są nazywane węzłami wewnętrznymi. Natomiast prostokąty leżące na samym dole nazywa się węzłami końcowymi lub też po prostu liściami.

To na podstawie liści podejmowana jest decyzja, do której klasy należy przypisać nową obserwację. Jeśli w liściu klasa 1 przewyższa klasę 0 (czyli więcej obserwacji ma dla zmiennej response wartość 1), to nowa obserwacja jak tam trafi, to zostanie sklasyfikowana do klasy 1. Analogicznie jest, gdy klasa 0 przewyższa klasę 1. Zatem jak mamy już zbudowany model drzewa decyzyjnego, to nie pozostaje nam nic innego jak dokonać klasyfikacji nowych obserwacji. Wykonamy to osobno dla zbioru uczącego i zbioru testowego.

```
pre4=predict(m.decisiontree,dataset.train, type="class")
tt4=table(predicted=pre4,actual=dataset.train$response)
tt4
```

```
##          actual
## predicted   0   1
##         0 458  84
##         1  35 123
```

```
sum(diag(tt4))/sum(tt4)
```

```
## [1] 0.83
```

Widzimy zatem, że dla zbioru trenigowego mamy dokładność na poziomie około 83%. Jest to całkiem niezły wynik. Zobaczmy teraz jak poradzi sobie nasze drzewo z klasyfikowaniem obserwacji ze zbioru testowego.

```
pre5=predict(m.decisiontree,dataset.test, type="class")
tt5=table(predicted=pre5,actual=dataset.test$response)
tt5
```

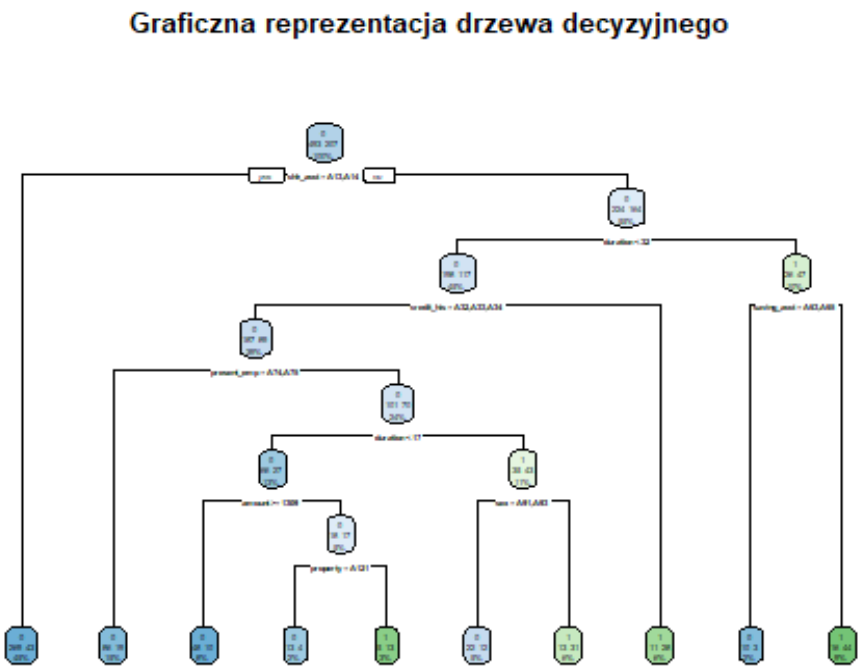
```
##          actual
## predicted   0   1
##         0 182  61
##         1  25  32
```

```
sum(diag(tt5))/sum(tt5)
```

```
## [1] 0.7133333
```

Widać, że wynik jest wyraźnie niższy. Dokładność dla zbioru testowego zdecydowanie spadła. Dlaczego tak się stało? Otóż, odpowiedź na to pytanie jest prosta. Drzewa decyzyjne są niemal idealną metodą uczenia maszynowego, ich graficzna reprezentacja i prostota są największym atutem, czego mogą zazdrościm im inne metody uczenia maszynowego. Jednakże mają jedną zasadniczą wadę. A mianowicie mają one nadmierną tendencję do zbyt dokładnego dopasowania do danych ze zbioru treningowego. A naszym celem jest mieć klasyfikator, który będzie świetnie radził sobie z klasyfikowaniem nowych obserwacji i tutaj pojawia się problem. Nasze drzewo decyzyjne jest zbyt dokładne i obszerne, co wcale nie musi i rzeczywiście nie przekłada się na lepszą klasyfikację nowych obserwacji, a nawet zdecydowanie ją pogarsza. Tak więc chcielibyśmy raczej jakoś uprościć nasze drzewo. Ale czy tak się w ogóle robi i czy warto to robić? Zdecydowanie warto zdecydować się na taki krok. Nowo utworzone drzewo będzie dużo prostsze w interpretacji, bo będzie mniejszych rozmiarów, a przede wszystkim powinna poprawić się dokładność klasyfikacji obserwacji ze zbioru testowego, a to jest coś nam czym nam najbardziej zależy przecież. Ta technika zmniejszania drzewa nazywa się przycinaniem drzewa. To ostateczne drzewo będzie miało o wiele mniej użytych węzłów (czyli prostokątów na wykresie). Przycinać drzewo można oczywiście na wiele sposobów. Niektóre z nich często prowadzą do takiego samego wyglądu drzewa, więc pole manewru podczas przycinania drzewa jest dosyć spore. Zastosuję jeden z łatwiejszych parametrów do przycinania drzewa, jakim jest minsplit, oznaczający wymaganą minimalną liczbę obserwacji w korzeniu, aby dokonać dalszego podziału obserwacji. Im większa wartość minsplit przy budowaniu drzewa, tym otrzymane drzewo będzie mniejszych rozmiarów. Ustalmy więc wartość parametru minsplit na 35 i narysujmy nowe uproszczone drzewo.

```
m.decisiontree2=rpart(response~.,data=dataset.train,method="class",minsplit=35)
rpart.plot(m.decisiontree2, extra=101, main="Graficzna reprezentacja drzewa decyzyjnego")
```



Widać z powyższego wykresu, że nowo utworzone drzewo jest o wiele mniejsze i łatwiej możnaby dokonywać jego interpretacji. Teraz wyznaczmy tabelę pomyłek wraz z obliczona dokładnością dla zbioru uczącego i zbioru testowego.

```
pre6=predict(m.decisiontree2,dataset.train, type="class")
tt6=table(predicted=pre6,actual=dataset.train$response)
tt6
```

```
##          actual
## predicted   0   1
##         0 448  91
##         1  45 116
```

```
sum(diag(tt6))/sum(tt6)
```

```
## [1] 0.8057143
```

Dokładność na zbiorze uczącym jest odrobinę gorsza niż w przypadku początkowego drzewa.

```
pre7=predict(m.decisiontree2,dataset.test, type="class")
tt7=table(predicted=pre7,actual=dataset.test$response)
tt7
```

```
##          actual
## predicted   0   1
##         0 176  56
##         1  31  37
```

```
sum(diag(tt7))/sum(tt7)
```

```
## [1] 0.71
```

Ale na zbiorze testowym nasze przycięte drzewo radzi sobie jednak lepiej. Widać zatem, że przycinanie drzewa ma sens, czego można było się domyślić wcześniej. Poniżej napiszę funkcję, która sama poda wartość parametru minsplit, dla którego dokładność na zbiorze testowym będzie najwyższa. Zaznaczam, że jest wiele innych metod przycinania drzew, ale są one niestety nieco bardziej złożone i dlatego zdecydowałem się na jedną z łatwiejszych metod, aby można było łatwiej wytłumaczyć jak działa przycinanie drzew. Na samym końcu projektu będą umieszczone moje uwagi opisujące, co można jeszcze poprawić, aby projekt był lepszy i bardziej kompletny. Mam zamiar później, co jakiś czas stopniowo go ulepszać. Powracając do drzew decyzyjnych, poniżej rozpatrywane jest kilka wartości dla parametru minsplit.

```
v_minsplit=c(5,15,25,35,45,55,65,75,85,95,105,115)
best_acc=c()
for (j in v_minsplit) {
modelTree=rpart(response~.,data=dataset.train,method="class",minsplit=j)
pre=predict(modelTree,dataset.test, type="class")
tt=table(predicted=pre,actual=dataset.test$response)
best_acc=c(best_acc,sum(diag(tt))/sum(tt))
}
tab=round(rbind(v_minsplit,best_acc),3)
tab
```

```
##             [,1] [,2]   [,3]  [,4]   [,5]   [,6]   [,7]   [,8]  [,9] [,10]
## v_minsplit 5.000 15.0 25.000 35.00 45.000 55.000 65.000 75.000 85.00 95.00
## best_acc   0.703  0.7  0.713  0.71  0.707  0.707  0.707  0.717  0.73  0.73
##             [,11]  [,12]
## v_minsplit 105.00 115.00
## best_acc     0.73   0.73
```

```
cat("\n","Najlepsza wartość dla minsplit i odpowiadająca jej dokładność:","\n")
```

```
##
##  Najlepsza wartość dla minsplit i odpowiadająca jej dokładność:
```

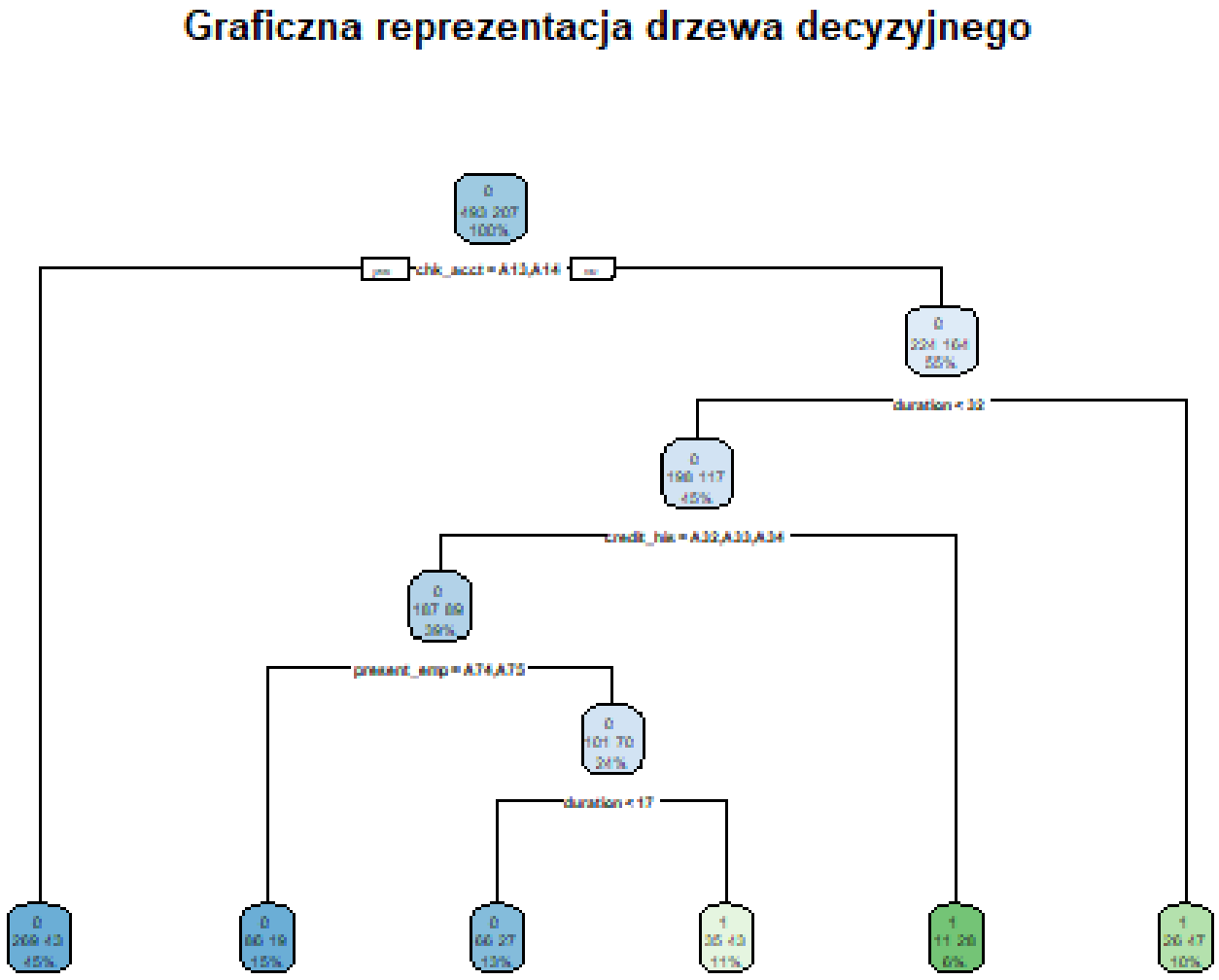
```
tab[,which.max(tab[2,])]
```

```
## v_minsplit   best_acc
##      85.00       0.73
```

```
metody.ML$dokladnosc[2]=tab[2,which.max(tab[2,])]
```

Widzimy już zatem dla jakiej wartości parametru minsplit otrzymujemy najwyższą dokładność na zbiorze testowym. Poniżej graficzna prezentacja najbardziej optymalnego drzewa z najlepszą wartością dla parametru minsplit. Poniższe drzewo będzie zatem naszym najbardziej optymalnym drzewem służącym do klasyfikacji nowych klientów na tych z pozytywną lub negatywną zdolnością kredytową.

```
b_minsplit=tab[1,which.max(tab[2,])]
m.decisiontree3=rpart(response~.,data=dataset.train,method="class",minsplit=b_minsplit)
rpart.plot(m.decisiontree3, extra=101, main="Graficzna reprezentacja drzewa decyzyjnego")
```



6.3.Lasy losowe i bagging

Korzystając z drzew decyzyjnych nie dało się nie zauważyć, że mają one tendencję do nadmiernego dopasowania do danych ze zbioru trenigowego, co prawie zawsze przekłada się na gorszą dokładność na zbiorze testowym. Dlatego też powstało wiele innych metod korzystających z drzew decyzyjnych, które rozwiązują omówiony problem. Jedną z bardziej znanych metod jest bagging (nie tłumaczę na język polski, bo forma angielska wydaje się lepsza) oraz szczególny ich przypadek, zwany lasy losowe (z angielskiego random forest). Najpierw wytłumaczę na czym polega metoda bagging. Polega ona na tym, że zamiast jednego drzewa, tworzymy ich kilkadziesiąt, a przeważnie od 100 do nawet 1000, wszystko w zależności od wielkości zbioru danych. Każde drzewo nie jest jednak tworzone na podstawie obserwacji ze zbioru uczącego. Dla każdego drzewa będą losowane obserwacje ze zbioru uczącego tak, aby ostatecznie miało ono do dyspozycji tyle samo obserwacji. Nasuwa się pytanie, czy nie będziemy mieli wtedy takich samych drzew ze względu na takie same obserwacje? Otóż nie, bo wszystko zależy od sposobu losowania obserwacji. Cały sztuczka polega na tym, że obserwacje losowane są ze zwracaniem. Wynika z tego to, że zbiór danych dla danego drzewa może mieć powtarzające się obserwacje. Okazuje się właśnie, że taka metoda daje całkiem dobre wyniki. Lasy losowe działają na takiej samej zasadzie, tylko proces budowy drzewa nieco się różni. Jak już wspomniałem wcześniej, celem drzewa decyzyjnego jest na każdym etapie drzewa umiejętne rozdzielanie obserwacji na podstawie wszystkich zmiennych objaśniających tak, aby móc lepiej przywidzieć wartość zmiennej objaśnianej. Lasy losowe na każdym etapie rozdzielania obserwacji rozpatrują tylko podzbiór wszystkich dostępnych zmiennych objaśniających. W ten sposób zamiast wszystkich 20 zmiennych, rozpatrywanych jest tylko na przykład losowo wybranych 5 zmiennych i można powiedzieć, że rywalizują one między sobą, która z nich będzie użyta do rozdzielania obserwacji. W każdym kolejnym etapie rozdzielania obserwacji ponownie jest losowany podzbiór zmiennych objaśniających służących do oddzielania obserwacji. W skrócie na tym polegają metody bagging i lasów losowych. Teraz podczas klasyfikacji nowej obserwacji, będziemy korzystać z wszystkich drzew w celu podjęcia decyzji, co do tego do jakiej klasy przypisać naszą obserwację. Przykładowo, jeśli na podstawie 100 drzew, większość z nich sklasyfikowałoby obserwację do klasy pierwszej, to ostatecznie tę obserwację sklasyfikujemy do klasy pierwszej. Zastosowana jest tutaj, tzw. reguła większości głosów. Do utworzenia modeli baggingu i lasów losowych można posłużyć się funkcją randomForest z pakietu o dokładnie takiej samej nazwie. Utworzymy najpierw model korzystający z baggingu.

```
m.bagging=randomForest(response~.,data=dataset.train,mtry=20)
m.bagging

##
## Call:
##  randomForest(formula = response ~ ., data = dataset.train, mtry = 20)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 20
##
##         OOB estimate of  error rate: 22.71%
## Confusion matrix:
##     0  1 class.error
## 0 444 49  0.09939148
## 1 110 97  0.53140097

dokladnosc1=(1-m.bagging$err.rate[500])*100
```

Na podstawie powyższego wyniku widzimy, że nasz model składa się z 500 drzew i w każdym podziale było rozpatrywanych 20 zmiennych. Nazwa “OOB estimate of error rate” wskazuje nam ile procent obserwacji zostało źle sklasyfikowanych, czyli błąd klasyfikacji wyniosł 0.2271429. Ale uzyskaliśmy ten wynik na jakich obserwacjach? Otóż jak już wspomniałem, losowanie obserwacji było dokonywane ze zwracaniem i te obserwacje, które nie były użyte do budowy danego drzewa, mogą być naszą tzw. obserwacją ze zbioru testowego dla tego drzewa. Dzięki temu możemy wykorzystać obserwacje, które nie służyły do budowy tego drzewa. Czyli uzyskaliśmy dokładność na poziomie 77.2857143%.

Poniżej pojawia się również tabela pomyłek utworzona na podstawie obserwacji ze zbioru uczącego. Jako, że mamy jeszcze zbiór testowy, dokonajmy predykcji nowych obserwacji ze zbioru testowego i zobaczmy jak wygląda dla nich tabela pomyłek z obliczoną dokładnością.

```
pre8=predict(m.bagging,newdata=dataset.test)
tt8=table(predicted=pre8,actual=dataset.test$response)
tt8

##          actual
## predicted   0   1
##         0 183  54
##         1  24  39

(metody.ML$dokladnosc[3]=sum(diag(tt8))/sum(tt8))

## [1] 0.74
```

Otrzymaliśmy dokładność na poziomie 74%, co jest bardzo dobrym wynikiem. Utwórzmy w bardzo podobny sposób teraz model lasu losowego. Tutaj różnica polega tylko na tym, że nie musimy określać liczby zmiennych potrzebnych do rozdzielenia obserwacji. Odpowiada za to właśnie parametr mtry.

```
m.randomforest=randomForest(response~.,data=dataset.train)
m.randomforest

##
## Call:
##  randomForest(formula = response ~ ., data = dataset.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 24.14%
## Confusion matrix:
##     0  1 class.error
## 0 458 35  0.07099391
## 1 134 73  0.64734300

dokladnosc2=(1-m.randomforest$err.rate[500])*100
```

Nasza dokładność wynosi teraz 75.8571429%. Sprawdźmy jak nasz las losowy poradzi sobie na zbiorze testowym.

```
pre9=predict(m.randomforest,newdata=dataset.test)
tt9=table(predicted=pre9,actual=dataset.test$response)
tt9

##          actual
## predicted   0   1
##         0 193  62
##         1  14  31

(metody.ML$dokladnosc[4]=sum(diag(tt9))/sum(tt9))

## [1] 0.7466667
```

Na ogół lasy losowe lepiej radzą sobie z klasyfikowaniem obserwacji niż bagging. Przy głębszej analizie warto też uwzględnić różne wartości dla parametru mtry, aby zobaczyć, który daje najwyższą dokładność.

6.4.Maszyna wektorów nośnych

Opis działania maszyny wektorów nośnych można przedstawić za pomocą matematyki. Jest to jednak trochę zaawansowany temat i nie podejmę się tej czynności. Jednakże można wyobrazić sobie graficznie w jaki sposób ona działa. Załóżmy, że mamy na płaszczyźnie punkty w dwóch kolorach (czerwony i zielony). Celem maszyny wektorów nośnych jest znalezienie takiej prostej, która oddziela punkty w taki sposób, aby po jednej stronie znajdowały się czerwone punkty, a po drugiej stronie zielone punkty. Czasem stosuje się też przekształcenia używanych zmiennych objaśniających tak, aby dało się lepiej odzielić obserwacje należące do dwóch klas. Określa się to za pomocą parametru kernel w funkcji svm() służącej do utworzenia modelu maszyny wektorów nośnych. Najpierw zastosujemy parametr kernel z domyślną opcją "linear".

```
m.svm=svm(response~.,data=dataset.train,kernel="linear")
pre10=predict(m.svm,newdata=dataset.train)
tt10=table(predicted=pre10, actual=dataset.train$response)
tt10
```

```
##          actual
## predicted   0   1
##         0 444  94
##         1  49 113
```

```
sum(diag(tt10))/sum(tt10)
```

```
## [1] 0.7957143
```

Utworzyliśmy model maszyny wektorów nośnych z parametrem kernel="linear" oraz dokonaliśmy predykcji obserwacji ze zbioru uczącego. Widać, że nasza dokładność jest bardzo dobra, na poziomie około 79.5714286%. Sprawdźmy jak poradzi sobie model z obserwacjami ze zbioru testowego.

```
pre11=predict(m.svm,newdata=dataset.test)
tt11=table(predicted=pre11, actual=dataset.test$response)
tt11
```

```
##          actual
## predicted   0   1
##         0 183  57
##         1  24  36
```

```
sum(diag(tt11))/sum(tt11)
```

```
## [1] 0.73
```

Dokładność wynosi 73%, więc odrobinę gorzej niż dla zbioru uczącego. Dla paraemtru kernel="linear" możemy dobrać różne wartości parametru cost oznaczającego na ile jesteśmy w stanie zaakceptować błąd podczas klasyfikacji obserwacji. Domyślnie wartość parametru cost wynosi 1. Im większą wartość ustalimy dla parametru cost, tym model będzie bardziej starał się dopasowywać do danych. Jest to dobra informacja, ale każdy kij ma dwa końce. Jeśli wartość dla parametru cost będzie za duża, to błąd w klasyfikacji obserwacji ze zbioru testowego może być niestety większy. Poniżej wypróbujemy kilka wartości dla parametru cost i wybierzemy ten, który da nam najwyższą dokładność na zbiorze testowym.

```
v_cost=c(1,5,10,20,35,50,75,100)
b_accuracy=c()
for (c in v_cost) {
m.SVM=svm(response~.,data=dataset.train,kernel="linear",cost=c)
preSVM=predict(m.SVM,newdata=dataset.test)
ttSVM=table(predicted=preSVM,actual=dataset.test$response)
b_accuracy=c(b_accuracy,sum(diag(ttSVM))/sum(ttSVM))
}
tab2=round(rbind(v_cost,b_accuracy),3)
tab2
```

```
##            [,1]  [,2]   [,3]   [,4]   [,5]   [,6]   [,7]    [,8]
## v_cost     1.00 5.000 10.000 20.000 35.000 50.000 75.000 100.000
## b_accuracy 0.73 0.737  0.733  0.737  0.733  0.733  0.733   0.733
```

```
cat("\n","Najlepsza wartość dla cost i odpowiadająca jej dokładność:","\n")
```

```
##
##  Najlepsza wartość dla cost i odpowiadająca jej dokładność:
```

```
tab2[,which.max(tab2[2,])]
```

```
##     v_cost b_accuracy
##      5.000      0.737
```

```
metody.ML$dokladnosc[5]=tab2[2,which.max(tab2[2,])]
```

Zatem najlepszą dokładność otrzymujemy dla cost=5 i wynosi ona 73.7% na zbiorze testowym. Poniżej zrobimy to samo dla kernel = “radial” oraz “polynomial”. Najpierw użyjemy kernel=“polynomial”, w którym mamy dodatkowy parametr degree określający stopień wielomianu funkcji, która jest używana do oddzielenia obserwacji.

```
v_cost2=c(1,5,10,25,50)
v_degree=c(1,2,3)
accuracy2=c()
tab3=c()
for (c in v_cost2) {
  for (d in v_degree) {
    m.SVM2=svm(response~.,data=dataset.train,kernel="polynomial",cost=c,degree=d)
    preSVM2=predict(m.SVM2,newdata=dataset.test)
    ttSVM2=table(predicted=preSVM2,actual=dataset.test$response)
    accuracy2=sum(diag(ttSVM2))/sum(ttSVM2)
    tab3=cbind(tab3,round(c(c,d,accuracy2),3))
  }
}
rownames(tab3)=c("cost","degree","accuracy")
tab3
```

```
##           [,1]  [,2] [,3]  [,4]  [,5]  [,6]  [,7]  [,8] [,9]  [,10]  [,11]
## cost     1.000 1.000 1.00 5.000 5.000 5.000 10.00 10.00 10.0 25.000 25.000
## degree   1.000 2.000 3.00 1.000 2.000 3.000  1.00  2.00  3.0  1.000  2.000
## accuracy 0.727 0.693 0.69 0.727 0.737 0.697  0.72  0.74  0.7  0.727  0.723
##           [,12] [,13]  [,14]  [,15]
## cost     25.000 50.00 50.000 50.000
## degree    3.000  1.00  2.000  3.000
## accuracy  0.727  0.73  0.723  0.737
```

```
cat("\n","Najlepsza wartość dla cost i degree oraz odpowiadająca im dokładność:","\n")
```

```
##  Najlepsza wartość dla cost i degree oraz odpowiadająca im dokładność:
```

```
tab3[,which.max(tab3[3,])]
```

```
##     cost   degree accuracy
##    10.00     2.00     0.74
```

```
metody.ML$dokladnosc[6]=tab3[3,which.max(tab3[3,])]
```

Zatem dla parametru cost=10 oraz degree=2 otrzymujemy dokładność równą 74% na zbiorze testowym. Teraz użyjemy kernel="radial", w którym mamy dodatkowy parametr gamma określający wpływ każdej obserwacji na granicę decyzyjną, który tworzy model przy klasyfikowaniu obserwacji. Im większa wartość dla parametru gamma, tym model bardziej dopasowuje się do danych ze zbioru uczącego. Dokonamy analogicznie to samo co wcześniej, ale teraz dla kilku wartości dla parametrów cost i gamma.

```
v_cost3=c(1,5,10,25,50)
v_gamma=c(0.05,0.1,0.25,0.5,1,3)
accuracy3=c()
tab4=c()
for (c in v_cost3) {
  for (g in v_gamma) {
    m.SVM3=svm(response~.,data=dataset.train,kernel="radial",cost=c,gamma=g)
    preSVM3=predict(m.SVM3,newdata=dataset.test)
    ttSVM3=table(predicted=preSVM3,actual=dataset.test$response)
    accuracy3=sum(diag(ttSVM3))/sum(ttSVM3)
    tab4=cbind(tab4,round(c(c,g,accuracy3),3))
  }
}
rownames(tab4)=c("cost","gamma","accuracy")
tab4
```

```
##           [,1]  [,2]  [,3]  [,4] [,5] [,6] [,7] [,8]  [,9] [,10] [,11] [,12]
## cost     1.000 1.000 1.000 1.000 1.00 1.00 5.00 5.00 5.000   5.0 5.000  5.00
## gamma    0.050 0.100 0.250 0.500 1.00 3.00 0.05 0.10 0.250   0.5 1.000  3.00
## accuracy 0.737 0.723 0.717 0.697 0.69 0.69 0.74 0.73 0.733   0.7 0.687  0.69
##           [,13] [,14]  [,15] [,16]  [,17] [,18] [,19]  [,20]  [,21] [,22]
## cost     10.000 10.00 10.000  10.0 10.000 10.00 25.00 25.000 25.000  25.0
## gamma     0.050  0.10  0.250   0.5  1.000  3.00  0.05  0.100  0.250   0.5
## accuracy  0.717  0.73  0.737   0.7  0.687  0.69  0.73  0.723  0.737   0.7
##           [,23] [,24] [,25]  [,26]  [,27] [,28]  [,29] [,30]
## cost     25.000 25.00 50.00 50.000 50.000  50.0 50.000 50.00
## gamma     1.000  3.00  0.05  0.100  0.250   0.5  1.000  3.00
## accuracy  0.687  0.69  0.73  0.723  0.737   0.7  0.687  0.69
```

```
cat("\n","Najlepsza wartość dla cost i gamma oraz odpowiadająca im dokładność:","\n")
```

```
##
##  Najlepsza wartość dla cost i gamma oraz odpowiadająca im dokładność:
```

```
tab4[,which.max(tab4[3,])]
```

```
##     cost    gamma accuracy
##     5.00     0.05     0.74
```

```
metody.ML$dokladnosc[7]=tab4[3,which.max(tab4[3,])]
```

Tak więc dla paraemtrów cost=5 oraz gamma=0.05 otrzymaliśmy dokładność 74% klasyfikacji obserwacji ze zbioru testowego.

6.5.Naiwny klasyfikator bayesowski

Metoda naiwnego klasyfikatora bayesowskiego opiera się w zasadzie na twierdzeniu matematycznym, a dokładniej na twierdzeniu Bayesa, które przedstawia wzór na obliczenie prawdopodobieństwa zajścia zdarzenia A pod warunkiem wystąpienia zdarzenia B. Naiwny klasyfikator bayesowski zakłada, iż zmienne objaśniające w modelu są niezależne od siebie, co w praktyce rzadko kiedy jest spełnione. Mimo niespełnienia tego założenia, metoda ta radzi sobie całkiem dobrze w klasyfikacji obserwacji. I choć właśnie założenia są przeważnie niespełnione, to stosuję się naiwny klasyfikator bayesowski. Stąd pojawia się wytłumaczenie, dlaczego ta metoda jest nazywana naiwną. Dodatkowo zakłada się też, że zmienne ciągłe użyte w modelu pochodzą z rozkładu normalnego, co też jest naiwnym założeniem. Tak więc model naiwnego klasyfikatora bayesowskiego możemy utworzyć używając funkcji naiveBayes().

```
m.NaiveBayes=naiveBayes(response~.,data=dataset.train)
m.NaiveBayes$tables$chk_acct

##    chk_acct
## Y          A11        A12        A13        A14
##   0 0.19878296 0.25557809 0.05070994 0.49492901
##   1 0.44927536 0.34299517 0.04347826 0.16425121

m.NaiveBayes$tables$duration

##    duration
## Y       [,1]     [,2]
##   0 19.21704 11.34544
##   1 24.55556 13.40177
```

Powyżej wyświetlono tylko część wyników. Co one oznaczają? W dużym uproszczeniu unikając wzorów, te wyniki są używane do wyznaczania prawdopodobieństw w twierdzeniu Bayesa, a co za tym idzie do klasyfikacji obserwacji. Podobnie jak w regresji logistycznej, otrzymujemy tutaj prawdopodobieństwo. Wyznaczane jest prawdopodobieństwo, że zmienna objaśniana response przyjmuje wartość 1 na podstawie wartości zmiennych objaśniających. Jeśli to prawdopodobieństwo jest większe od 0.5, to ustalamy, że dany klient powinien miec pozytywną zdolność kredytową. Zatem jak widać nie jest to skomplikowane. Zobaczmy teraz jak nasz naiwny klasyfikator bayesowski radzi sobie z klasyfikowaniem obserwacji ze zbioru uczącego oraz co ważniejsze ze zbioru testowego.

```
pre12=predict(m.NaiveBayes,newdata=dataset.train,type="class")
tt12=table(predicted=pre12, actual=dataset.train$response)
tt12

##          actual
## predicted   0   1
##         0 421  88
##         1  72 119

sum(diag(tt12))/sum(tt12)

## [1] 0.7714286
```

Zatem na zbiorze uczącym otrzymaliśmy dokładność na poziomie 77.1428571%, co jest nawet niezłym wynikiem.

Zobaczmy jak poradzi sobie model z obserwacjami ze zbioru testowego.

```
pre13=predict(m.NaiveBayes,newdata=dataset.test,type="class")
tt13=table(predicted=pre13, actual=dataset.test$response)
tt13
```

```
##          actual
## predicted   0   1
##         0 182  53
##         1  25  40
```

```
(metody.ML$dokladnosc[8]=sum(diag(tt13))/sum(tt13))
```

```
## [1] 0.74
```

Dokładność wynosząca 74% jest trochę gorszym wynikiem. Na wynik mogło, a nawet na pewno złożyła się kwestia braku spełnionych założeń, czyli niezależność zmiennych oraz rozkład normalny dla zmiennych ciągłych. To pierwsze można spróbować częściowo wyeliminować dokonując analizy korelacji, aby sprawdzić liniową zależność pomiędzy zmiennymi. Natomiast sprawienie, aby zmienne ciągłe pochodziły bardziej z rozkładu normalnego można próbować uzyskać poprzez przekształcenia tych zmiennych. Jednak, nie zdecyduję się na te opisane kroki.

6.6.Wzmacniane drzewa decyzyjne

Na tym etapie projektu omówię już 3 ostatnie metody uczenia maszynowego jakim są wzmacniane drzewa decyzyjne. Nazywać się one będą: adaboost, gradient boosting oraz extreme gradient boosting. W każdej z tych metod tworzymy kilka/kilkanaśnie drzew decyzyjnych. Jednak tym razem, każde kolejne drzewo "uczy się danych" na podstawie błędów klasyfikacji popełnionych przez poprzednie drzewo. Okazuje się, że wzmacniane drzewa decyzyjne uzyskują zwykle lepsze wyniki niż pozostałe metody uczenia maszynowego. Niestety odbywa się to kosztem łatwości interpretacji i szybkości dokonywanych obliczeń. W metodzie adaboost każde kolejne drzewo przypisuje większą wagę do obserwacji, które zostały wcześniej niepoprawnie sklasyfikowane i w ten sposób kolejne drzewo koryguje błąd swojego poprzednika. Natomiast w metodach gradient boosting i extreme gradient boosting, zadaniem każdego kolejnego drzewa jest zmniejszanie błędu klasyfikacji obserwacji z poprzedniego drzewa. Chodzi o to, aby dla każdej obserwacji przewidywana wartość była coraz bliżej faktycznej wartości, co przekłada się bezpośrednio na lepsze klasyfikowanie obserwacji. Najpierw utworzymy model adaboost z użyciem funkcji boosting(). Ustawimy wartość parametru mfinal=8 wskazując, że chcemy użyć 8 drzew do budowy ostatecznego modelu.

```
model.adaboost=adabag::boosting(response~.,data=dataset.train,boos=TRUE,mfinal=8)
model.adaboost$importance#im większa wartość, tym zmienna jest ważniejsza w modelu
```

```
##              age           amount         chk_acct       credit_his
##        9.8729152       14.3623492        9.9947369        6.7370699
##         duration          foreign          housing installment_rate
##        9.8732679        0.0000000        0.6358003        5.9949458
##              job         n_credits         n_people     other_debtor
##        1.1454261        0.7215849        1.0557879        1.4669623
##    other_install     present_emp    present_resid         property
##        2.9068385        5.2814432        2.2660781        5.4342816
##          purpose      saving_acct              sex        telephone
##       12.2436411        6.6757195        2.8322474        0.4989042
```

Widzimy zatem, że w modelu adaboost najważniejszą zmienną jest zmienna amount. Teraz zastosujemy predykcję na zbiorze uczącym i zobaczmy dla niego tabelę pomyłek z dokładnością.

```
pre14=predict(model.adaboost,newdata=dataset.train,type="class")
pre14=as.factor(pre14$class)
tt14=table(predicted=pre14,actual=dataset.train$response)
tt14
```

```
##          actual
## predicted   0   1
##         0 483  37
##         1  10 170
```

```
sum(diag(tt14))/sum(tt14)
```

```
## [1] 0.9328571
```

Otrzymana dokładność jest bardzo wysoka, ale to tylko predykcja na zbiorze treningowym. Zobaczmy jak model poradzi sobie ze zbiorem testowym.

```
pre15=predict(model.adaboost,newdata=dataset.test,type="class")
pre15=as.factor(pre15$class)
tt15=table(predicted=pre15,actual=dataset.test$response)
tt15
```

```
##          actual
## predicted   0   1
##         0 181  53
##         1  26  40
```

```
sum(diag(tt15))/sum(tt15)
```

```
## [1] 0.7366667
```

Dokładność na zbiorze testowym jest wyraźnie mniejsza. Wypróbujmy teraz kilka wartości dla parametru mfinal i wybierzmy ten, który daje najwyższą dokładność na zbiorze testowym.

```
v_final=c(5,8,10,14,18,25)
v_acc=c()
j=0
for (u in v_final) {
  j=j+1
  modelADA=adabag::boosting(response~.,data=dataset.train,boos=TRUE,mfinal=u)
  preADA=predict(modelADA,newdata=dataset.test,type="class")
  preADA=as.factor(preADA$class)
  ttADA=table(predicted=preADA,actual=dataset.test$response)
  v_acc[j]=sum(diag(ttADA))/sum(ttADA)
}
tab5=round(rbind(v_final,v_acc),4)
tab5
```

```
##           [,1]   [,2]    [,3]    [,4]  [,5]  [,6]
## v_final 5.0000 8.0000 10.0000 14.0000 18.00 25.00
## v_acc   0.7167 0.7267  0.7567  0.7333  0.74  0.75
```

```
cat("\n","Najlepsza wartość dla mfinal i odpowiadająca jej dokładność:","\n")
```

```
##
##  Najlepsza wartość dla mfinal i odpowiadająca jej dokładność:
```

```
tab5[,which.max(tab5[2,])]
```

```
## v_final   v_acc
## 10.0000  0.7567
```

```
metody.ML$dokladnosc[9]=tab5[2,which.max(tab5[2,])]
```

Zatem najwyższą dokładność otrzymujemy dla mfinal=10. Wynosi ona około 75.67%. Można próbować rozważyć też inne parametry, aby uzyskać wyższą dokładność, ale na tym już zakończę adaboost.

Przejdźmy teraz do metody zwanej gradient boosting.

```
dataset.trainDummy=NULL;dataset.testDummy=NULL
dataset.trainDummy=mltools::one_hot(as.data.table(dataset.train[,-21]))
dataset.trainDummy$response=dataset.train$response
dataset.testDummy=mltools::one_hot(as.data.table(dataset.test[,-21]))
dataset.testDummy$response=dataset.test$response
dataset.trainDummy$response=as.integer(dataset.trainDummy$response)-1
dataset.testDummy$response=as.integer(dataset.testDummy$response)-1
model.gbm=gbm::gbm(response~.,data=dataset.trainDummy,
                   distribution="bernoulli",verbose=FALSE,
                   n.trees=300,shrinkage=0.25,interaction.depth=1)
model.gbm

## gbm::gbm(formula = response ~ ., distribution = "bernoulli",
##     data = dataset.trainDummy, n.trees = 300, interaction.depth = 1,
##     shrinkage = 0.25, verbose = FALSE)
## A gradient boosted model with bernoulli loss function.
## 300 iterations were performed.
## There were 61 predictors of which 51 had non-zero influence.
```

Przed utworzeniem modelu gradient boosting musiałem przekształcić wszystkie zmienne jakościowe na numeryczne. Niestety funkcja gbm() wymaga użycia jedynie zmiennych numerycznych. Widać, że przy budowaniu modelu podałem kilka parametrów. To jak dobry model otrzymamy zależy bardzo dużo od wybranych parametrów. Jak wybrać najlepsze parametry? Prostego sposobu na to nie ma. Trzeba utworzyć kilka/kilkanaście modeli z różnymi opcjami dla parametrów i sprawdzić jak dobrze dokonywana jest klasyfikacja obserwacji. Na tym etapie to wszystko staje się już mniej intuicyjne i interpretowalne. Takie skomplikowane modele w uczeniu maszynowym noszą nazwę “black box”. Chodzi o to, że wrzucamy zwyczajnie do modelu dane. Ale w jaki sposób to konkretnie działa i dlaczego? Niestety staje się już to trudne do wytłumaczenia. Poniżej sprawdzimy dokładność klasyfikacji na zbiorze trenigowym.

```
pre17=predict(model.gbm,newdata=dataset.trainDummy,type="response",n.trees=300)
pre17=ifelse(pre17>0.5,1,0)
tt17=table(predicted=pre17,actual=dataset.trainDummy$response)
tt17

##          actual
## predicted   0   1
##         0 454  78
##         1  39 129

sum(diag(tt17))/sum(tt17)

## [1] 0.8328571
```

Uzyskaliśmy dokładność na poziomie 83.2857143%. Teraz sprawdzamy zbiór testowy.

```
pre18=predict(model.gbm,newdata=dataset.testDummy,type="response",n.trees=300)
pre18=ifelse(pre18>0.5,1,0)
tt18=table(predicted=pre18,actual=dataset.testDummy$response)
tt18

##          actual
## predicted   0   1
##         0 186  54
##         1  21  39

(metody.ML$dokladnosc[10]=sum(diag(tt18))/sum(tt18))

## [1] 0.75
```

Miara dokładności dla zbioru testowego jest dużo niższa. Bardzo dużo zależy od doboru parametrów modelu gbm. Zwykle tworzy się modele dla różnych wartości parametrów, a potem się sprawdza, który poradził sobie najlepiej z klasyfikowaniem obserwacji. Zauważmy też, że nasz zbiór danych dataset.train zawiera bardzo dużo zmiennych jakościowych. Funkcja gbm() wymaga, aby zmienne miały typ numeryczny. Zatem zostało utworzonych bardzo dużo dodatkowych zmiennych podczas przekształcania zmiennych jakościowych na numeryczne. To mogło mieć spory wpływ na końcowe wyniki.

Przejdźmy teraz do metody extreme gradient boosting.

```
xgbTrain=xgb.DMatrix(data=as.matrix(dataset.trainDummy[,-62]),label=dataset.trainDummy$response)
xgbTest=xgb.DMatrix(data=as.matrix(dataset.testDummy[,-62]),label=dataset.testDummy$response)
model.xgboost=xgboost::xgboost(data=xgbTrain,nrounds=500,verbose=FALSE)
```

Przed utworzeniem modelu musiałem przekształcić dane na odpowiedni format. Niestety funkcja xgboost tego wymaga i trzeba się do tego zastosować. Metoda (czy też algorytm) extreme gradient boosting został wymyślony, aby dać szybszą obliczeniowo alternatywę dla gradient boosting. Dlatego też słowo extreme nie pojawiło się przypadkiem. Poniżej obliczamy dokładność klasyfikacji na zbiorze treningowym.

```
pre19=predict(model.xgboost,newdata=xgbTrain,type="response")
pre19=ifelse(pre19>0.5,1,0)
tt19=table(predicted=pre19,actual=dataset.trainDummy$response)
tt19

##          actual
## predicted   0   1
##         0 493   0
##         1   0 207

sum(diag(tt19))/sum(tt19)

## [1] 1
```

Wynik jest niesamowicie wysoki. Tak jak wcześniej, bardzo dużo zależy od podanych parametrów. Zobaczmy teraz dokładność dla zbioru testowego.

```
pre20=predict(model.xgboost,newdata=xgbTest,type="response")
pre20=ifelse(pre20>0.5,1,0)
tt20=table(predicted=pre20,actual=dataset.testDummy$response)
tt20

##          actual
## predicted   0   1
##         0 177  48
##         1  30  45

(metody.ML$dokladnosc[11]=sum(diag(tt20))/sum(tt20))

## [1] 0.74
```

Wynik jest zdecydowanie niższy. Bardzo istotne jest to, aby model nie był nadmiernie dopasowany do danych ze zbioru uczącego, bo celem ostatecznym jest to, aby otrzymać jak najlepszą klasyfikację na zbiorze testowym. Tak jak w metodzie gradient boosting bardzo dużo zależy od wybranych parametrów. Bardziej dokładna analiza obejmuje sprawdzanie wielu parametrów i wybranie tych, które dają najwyższą dokładność klasyfikacji. Jednak nie będę tego czynił, gdyż wzmacniane drzewa decyzyjne nie są mi wystarczająco dobrze znane, aby mógł coś więcej z nich uzyskać.

6.7.Najlepszy model, wnioski i co można zrobić lepiej

Poniżej mamy umieszczoną tabelkę z metodami uczenia maszynowego i ich wyznaczonymi miarami dokładności na zbiorze testowym.

```
best.model=metody.ML[which.max(metody.ML$dokladnosc),1]
best.accuracy=metody.ML[which.max(metody.ML$dokladnosc),2]
metody.ML
```

```
##                                  nazwa.metody dokladnosc
## 1                        Regresja logistyczna  0.7266667
## 2                            Drzewo decyzyjne  0.7300000
## 3                                     Bagging  0.7400000
## 4                                  Las losowy  0.7466667
## 5      Maszyna wektorów nośnych,kernel=linear  0.7370000
## 6  Maszyna wektorów nośnych,kernel=polynomial  0.7400000
## 7      Maszyna wektorów nośnych,kernel=radial  0.7400000
## 8               Naiwny klasyfikator bayesowski  0.7400000
## 9                                    Adaboost  0.7567000
## 10                         Gradient Boosting  0.7500000
## 11                  Extreme Gradient Boosting  0.7400000
```

Na podstawie uzyskanych wyników najlepszy okazał się model **Adaboost**, który uzyskał na zbiorze testowym dokładność na poziomie **75.67%**. Oczywiście cała analiza była sporo uproszczona, bo niektórych aspektów nie opisano (na przykład krzywej ROC). W uzyskanych wynikach spore znaczenie miała funkcja sample() losująca, które wartości trafiły do zbioru uczącego i zbioru testowego. Lepszym rozwiązaniem w takiej sytuacji jest użycie walidacji krzyżowej, która radzi sobie z tym problemem losowości. Nie została jednak ona użyta i opisana ze względu na prostotę podziału zbioru danych na uczący i testowy. Celem projektu było przede wszystkim zaprezentowanie i użycie wielu metod uczenia maszynowego w celu klasyfikacji nowych klientów na tych, którym powinniśmy udzielić pozytywnej lub negatywnej zdolności kredytowej. A co można byłoby jeszcze zrobić? Z pewnością użyć metody sieci neuronowych, które uzyskują bardzo wysokie wyniki i są coraz bardziej popularne w analizie danych. Dobrym pomysłem byłoby też bliższe przyjrzenie się danym i zrozumienie co one dokładniej opisują i jakie mają znaczenie w ustalaniu typu zdolności kredytowej. Tak więc, wiedza z zakresu bankowości, a dokładnie z udzielania kredytów mogłaby okazać się bardzo pomocna i nieoceniona. Być może też sam raport nie jest zbyt wygodny do czytania i nie ma odpowiednich podrozdziałów. Mógłby się przydać jeszcze bardziej dokładny opis czego dotyczą dane i skąd pochodzą.

Także jeśli chodzi o raport, to chyba już wszystko. Byłbym bardzo wdzięczny o komentarz oraz wszelkie uwagi (być może było coś niezrozumiałe?) [mój e-mail: robertokonek47@gmail.com].

7.Informacje dodatkowe

Cały raport został utworzony w języku programowania R w programie RStudio jako plik RMarkdown (szczególnie używany do tworzenia raportów). Raport wyeksportowałem do programu Word (Opcja: Knit to Word), aby go w paru miejscach poprawić (zmiana koloru czcionek, wielkości wykresów, itp.) i zapisać ostatecznie jako plik PDF. Dodatkowo raport został również wyeksportowany jako plik html (Opcja: Knit to HTML), aby można go sobie przejrzeć wygodnie w przeglądarce.

7.1.Źródła/Przydatne materiały

1. Strona internetowa z używanym zbiorem danych: https://archive.ics.uci.edu/ml/index.php
2. Strona internetowa: http://www.sthda.com/english/ -bardzo przydatna do rysowania wykresów z wykorzystaniem pakietu ggplot2.