Projekt Zaliczeniowy nr 2

Mateusz Kapusta

2022-06-16

Przygotowanie techniczne

Ze względu na dużą ilość obliczeń do wykonania skorzystamy z równoległego modelu wykonania obliczeń. W tym celu ustawiamy liczbę dostępnych wątków jako 10 dostępnych wątków oraz ustawiamy jako domyślną metodę fork-a ze standardu POSIX. Skorzystamy z biblioteki doParallel oraz foreach.

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
n<-10
para<-parallel::makeCluster(n,type="FORK")
doParallel::registerDoParallel(cl = para)
foreach::getDoParRegistered()
```

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

```
print(para)
```

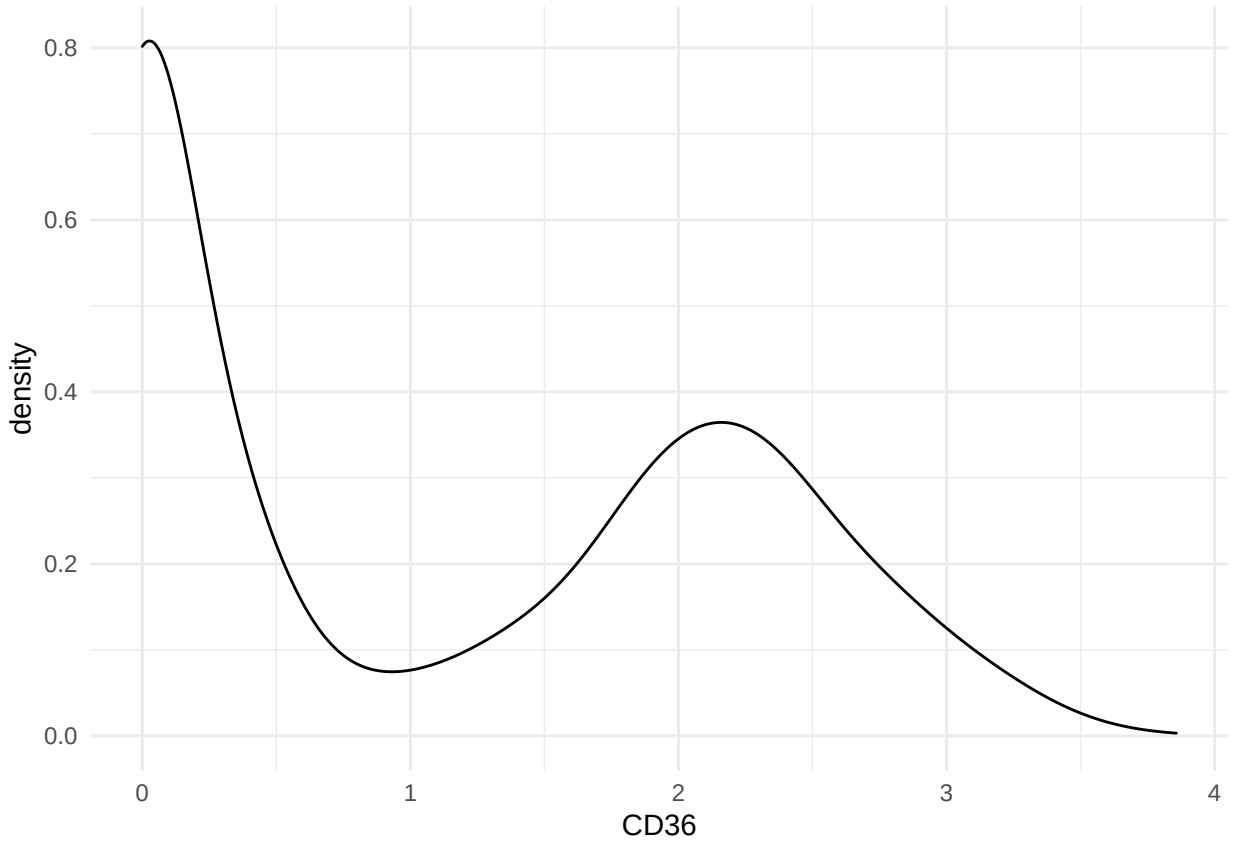
```
## socket cluster with 10 nodes on host 'localhost'
```

Eksploracja danych

```
xtrain<-read.csv("X_train.csv")
ytrain<-read.csv("y_train.csv")
xtest<-read.csv("X_test.csv")
```

W naszych danych mamy 3794 obserwacji z czego każdej obserwacji odpowiada 9000 zmiennych objaśniających. W danych mamy 0 braków w danych. Wszystkie kolumny zawierają dodatnie dane numeryczne. Zwizualizujmy teraz rozkład zmiennej objaśnianej.

```
ggplot(ytrain)+geom_density(aes(CD36))+theme_minimal()
```



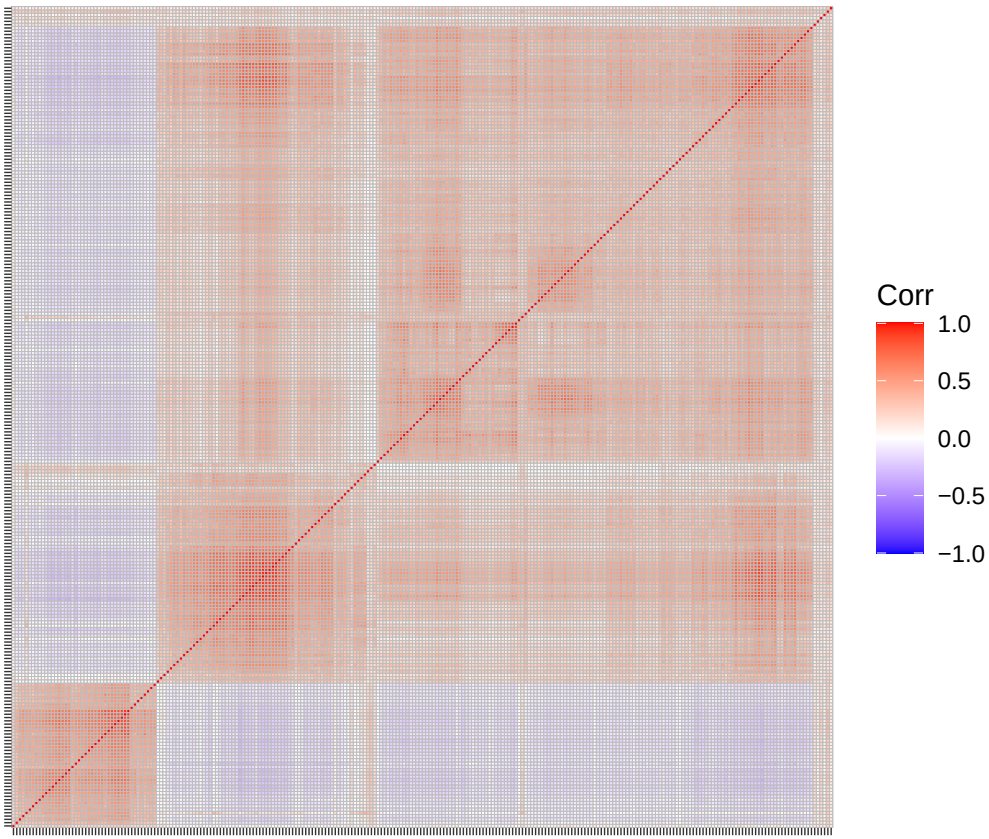
Dokładne statystyki zmiennej objaśnianej uzyskujemy wykorzystując funkcję summary.

```
summary(ytrain)
```

```
##       CD36
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.9147
##  Mean   :1.1454
##  3rd Qu.:2.1666
##  Max.   :3.8572
```

Teraz zbadajmy najbardziej skorelowane zmienne ze zmienną objaśnianą.

```
core<-numeric(ncol(xtrain))
for (i in 1:ncol(xtrain))
{
  core[i]<-cor(ytrain$CD36,xtrain[,i])
}
indexy<-order(core, decreasing=TRUE)[1:250]
mat <- round(cor(xtrain[indexy]),3)
ggcorrplot(mat,"hc.order"=TRUE,
          ggtheme=ggplot2::theme_dark,
          tl.cex=0
          )
```



Na koniec dzielimy dane na zbiór walidacyjny i treningowy.

```
indexy<-sample(1:nrow(xtrain),300,replace=FALSE)
xval<-xtrain[indexy,]
xtrain<-xtrain[-indexy,]
yval<-ytrain[indexy,]
ytrain<-ytrain[-indexy,]
```

Elastic Net

Model elastic net charakteryzowany jest przez dwa parametry λ oraz α. Dla podanych argumentów celem jest minimalizacja

$$RSS + \lambda \left(\sum_i \beta_i^2 \frac{(1-\alpha)}{2} + \alpha \sum_i |\beta_i| \right) \tag{1}$$

Dla $\alpha = 1$ odzyskujemy regresję lasso natomiast dla $\alpha = 0$ odzyskujemy regresję grzbietową. W przeciwnym przypadku otrzymujemy model mieszany. Do przeprwadzenia regresji wykorzystamy funkcję glmnet wykorzystującą domyślnie 100 wartości parametru λ oraz przeprwadzając walidację krzyżową dla 10 liczby próbek. W celu zbadabnia najlepszego zestawy hiperparametrów wartość α zmieniana była od 0 do 1 skacząc co 1/10. Powyższe wartości zostały dobrane tak, aby dość gęsto przeszukać możliwe parametry jednocześnie starając się aby kod można było wykonać w krótkim czasie.

```
a<-seq(0,1,1/10)
lambda<-seq(0,1,1/10)
grid_net<-expand.grid(alpha = a,lambda = lambda)
star<-Sys.time()
```

```
inde<-createFolds(ytrain,10,1)
con<-trainControl(method="cv",number=10,index=inde)
model_net<-train(x=as.matrix(xtrain),
             y=ytrain,
             method="glmnet",
             trControl = con,
             tuneGrid=grid_net)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
pred<-predict(model_net,xval)
errors_test<-sqrt(mean((pred-yval)^2))
```

Najlepszy model uzyskano dla parametrów α oraz λ odpowiednio 0.1 oraz 0.4 a średni błąd kwadratowy to 0.4023786. Błąd na zbiorze walidacyjnym to 0.35972

Random forest

Rozważmy model lasów losowy. W celu zbadania skuteczności sprawdzimy modele, wykorzystując jako hiperparametry liczbę zmiennych na dzrzewo, zasadę rodziału oraz minimalnją głębokość drzewa. Aby zapobiec przeuczeniu oraz przyśpieszyć obliczenia przycinamy drzewo tak, aby nie było głębsze niż na 15 rozdziałów. Paranetry wybieramy tak aby zrealizować możliwie dużo wartości jednocześnie minimalizując czas wykonywania kodu.

```
grid_tree <- expand.grid(
  .mtry =c(500,1000,2000),
  .splitrule = c("variance","extratrees"),
  .min.node.size = c(1,5,8)
)

  model_tree<-train(x=xtrain,
                    y=ytrain,
                    method="ranger",
                    trControl=con,
                    tuneGrid=grid_tree,
                    max.depth=15
                    )
```

```
## Growing trees.. Progress: 65%. Estimated remaining time: 16 seconds.
```

```
pred<-predict(model_tree,xval)
errors_test<-sqrt(mean((pred-yval)^2))
```

Błąd treningowy naszego klasyfikatora to 0.3772701 natomiast błąd testowy to 0.319444. Wybrane przy pomocy walidacji krzyżowej parametry z siatki to kolejno 2000, extratrees oraz 8. Teraz pora na porównanie naszych modeli dla najlepszych parametrów

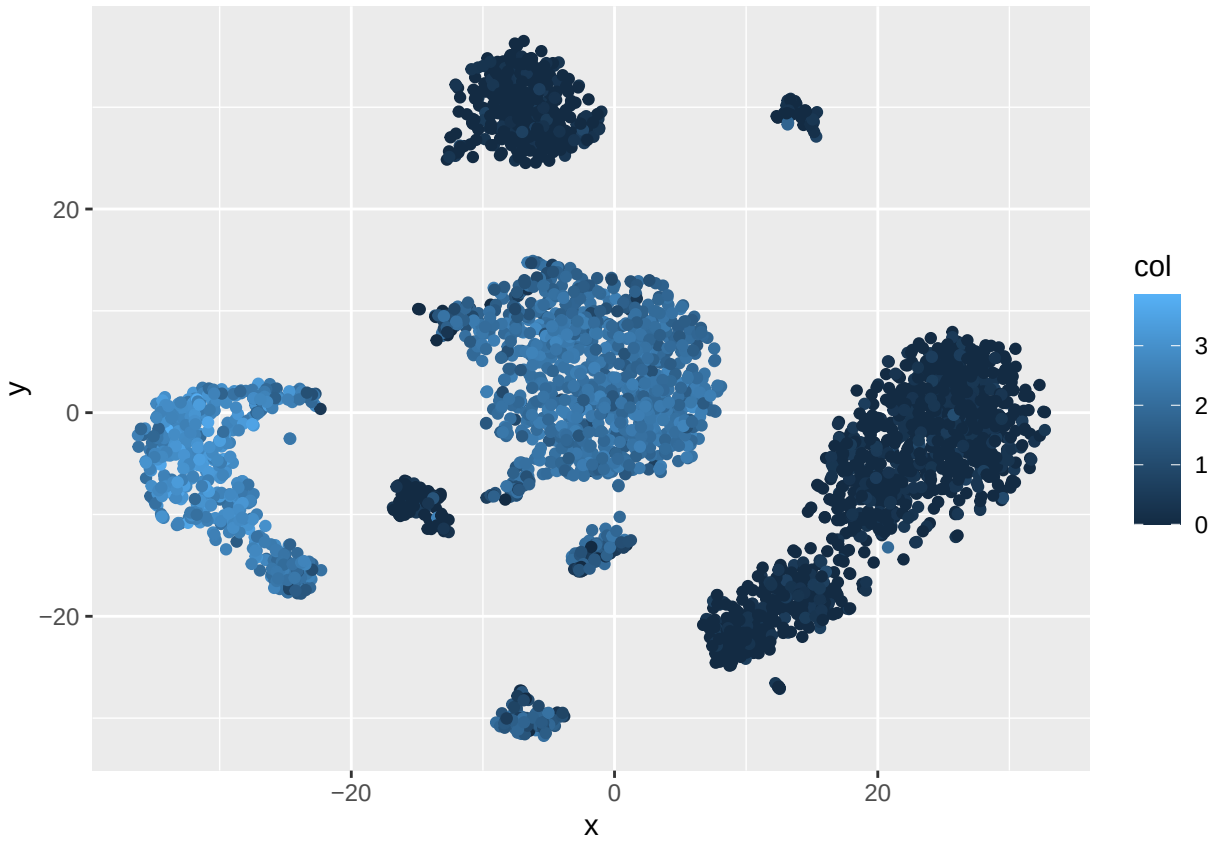
Nr foldu	$RMSE$ dla sieci	$RMSE$ dla lasu losowego	model średniej
1	0.4084161	0.3741903	1.0991993
2	0.3982367	0.3779316	1.103919
3	0.4067037	0.3791555	1.1185764
4	0.3993643	0.3721689	1.1106621
5	0.4001819	0.378062	1.1263838
6	0.3974018	0.3794875	1.0865672
7	0.4167606	0.3953833	1.089172
8	0.4077469	0.3675639	1.1223609
9	0.3952569	0.3697348	1.0993427
10	0.3937171	0.3779316	1.0994916

Widzimy więc, że drzewo losowe swoimi osiagami przebija zarówno model elastic net jak i model referencyjny i to dla prawie każdego foldu.

Zadanie nr 4

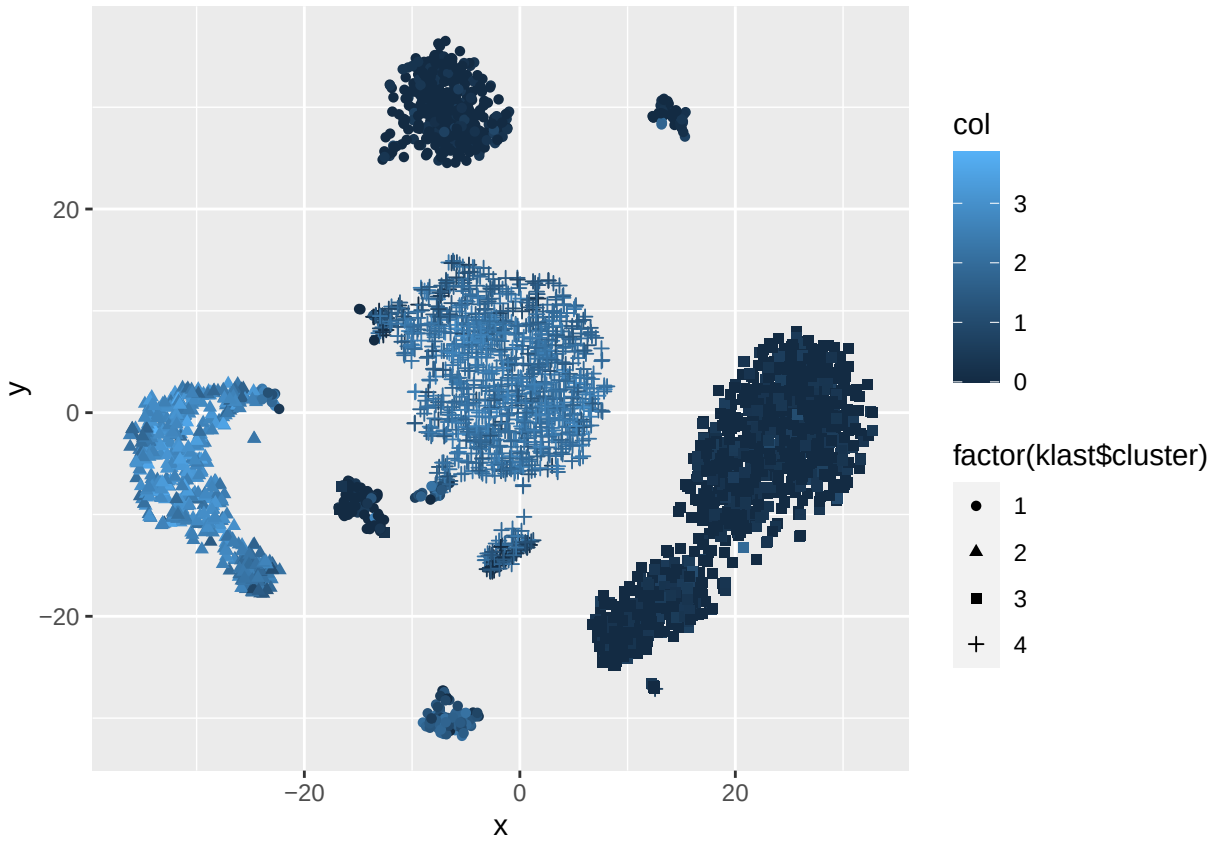
Zanim przejdziemy do predykcji sprawdźmy jak wyglądają dane przy pomocy wyzualizacji T-SNE.

```
library(Rtsne)
proc<-preProcess(xtrain,method=c("scale","center","nzv"))
xtrainp<-predict(proc,xtrain)
xtestp<-predict(proc,xtest)
xvalp<-predict(proc,xval)
a<-Rtsne(xtrainp,dims=2, perplexity=30)
a<-a[2]
a<-a$Y
dat<-data.frame("x"=a[,1],"y"=a[,2],"col"=ytrain)
ggplot(dat)+geom_point(aes(x=x,y=y,colour=col))
```



Widzimy, że dane wykazują pewną klastryzację, która ponadto wykazuje silne skorelowanie z wartością produckji białka. Widzimy więc dlaczego las losowy był skuteczniejszy od regresji liniowej. Lasy pozwalają na większe wyłapanie klastryzacji i dzięki temu regresja ma większą jakość. Poklastrujmy dane metodą k-średnich szukając 4 klastrów. Żądamy, aby jeden z klastrów miał mniejszą średnią niż $0,13$ i więcej niż 500 punktów. Powód jest prozaiczny, chcemy jak najbardziej pomóc danym aby zapewnić, że dane zostaną poklastrowane w sposób sprzyjajacy regresji co nie zawsze zachodzi z wykorzystaniem algorytmu k-średnich.

```
t<-TRUE
while(t)
{
 klast<-kmeans(xtrainp,4)
 for (i in 1:4)
 {
   if (mean(ytrain[klast$cluster==i])<0.13 & sum(klast$cluster==i)>500)
   {
     t<-FALSE
   }
 }
}
ggplot(dat)+geom_point(aes(x=x,y=y,colour=col,shape=factor(klast$cluster)))
```

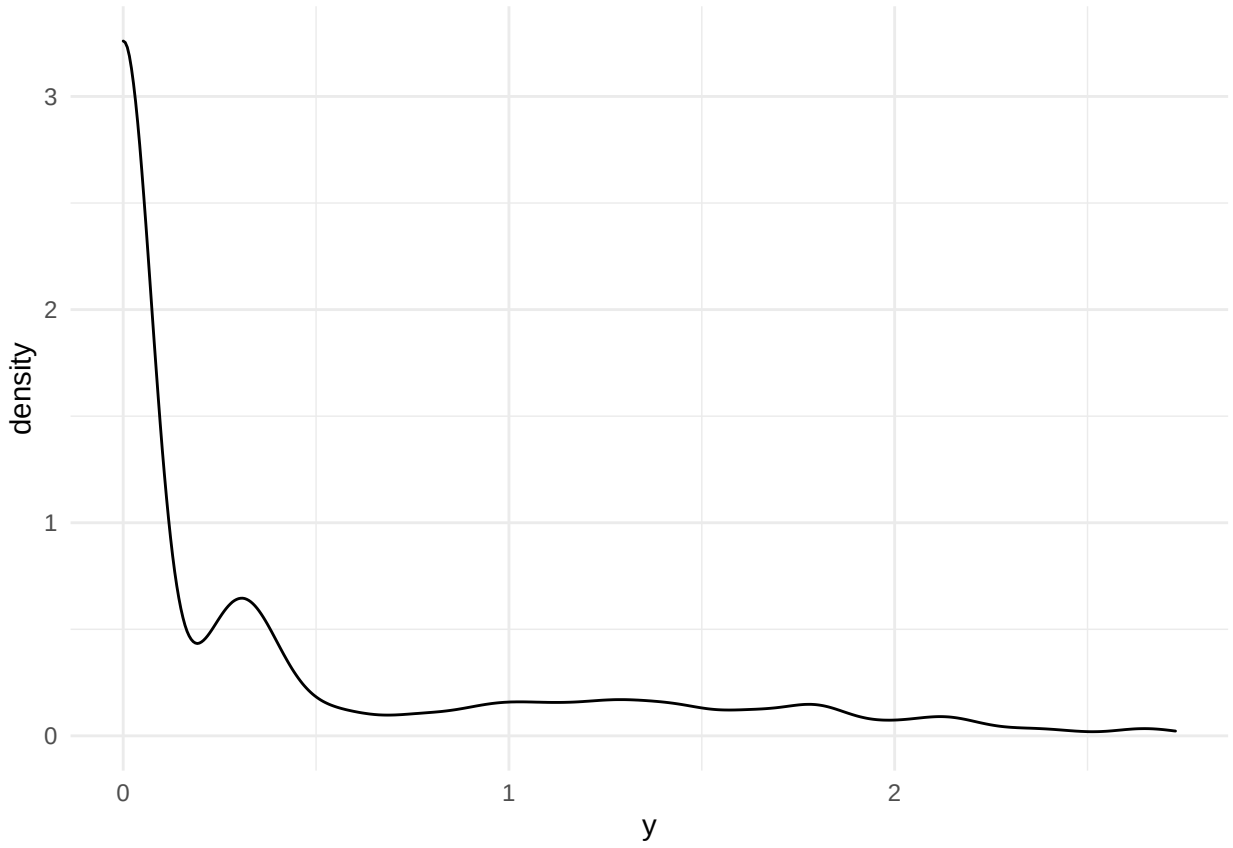


Widzimy, że klastryzacja spisała się i zgadza się z wizualizacją T-SNE. Zbadajmy także podstawowe własności poklastrowanych danych wraz z profilami gęstości w każdym z klastrów.

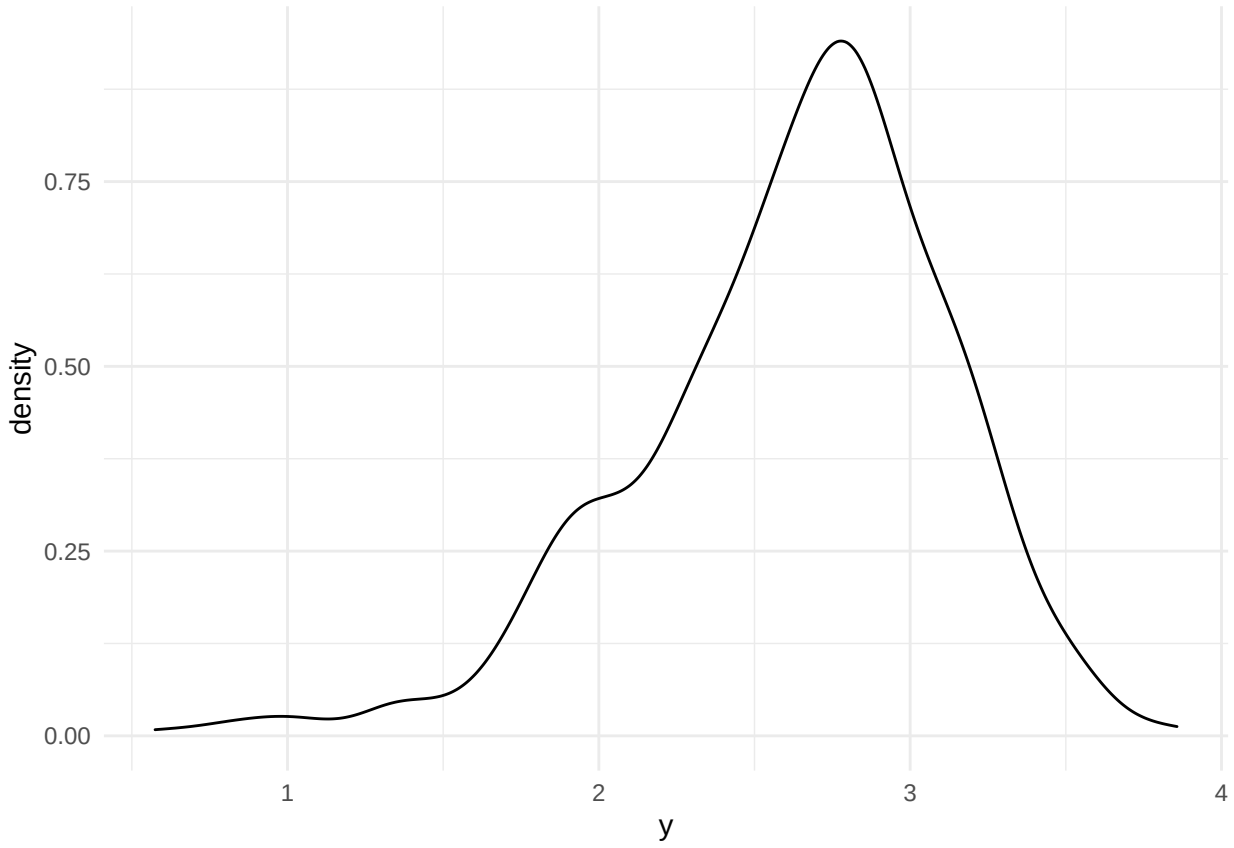
```
y<-list("1"=ytrain[klast$cluster==1],
              "2"=ytrain[klast$cluster==2],
              "3"=ytrain[klast$cluster==3],
              "4"=ytrain[klast$cluster==4]
             )
for (a in y)
    {
    print(c(mean(a),sd(a),length(a)))
}
```

```
## [1]   0.3798079   0.6332535 703.0000000
## [1]   2.6249308   0.4974895 542.0000000
## [1]    0.1074103    0.1965945 1124.0000000
## [1]    1.9276897    0.5192029 1125.0000000
```

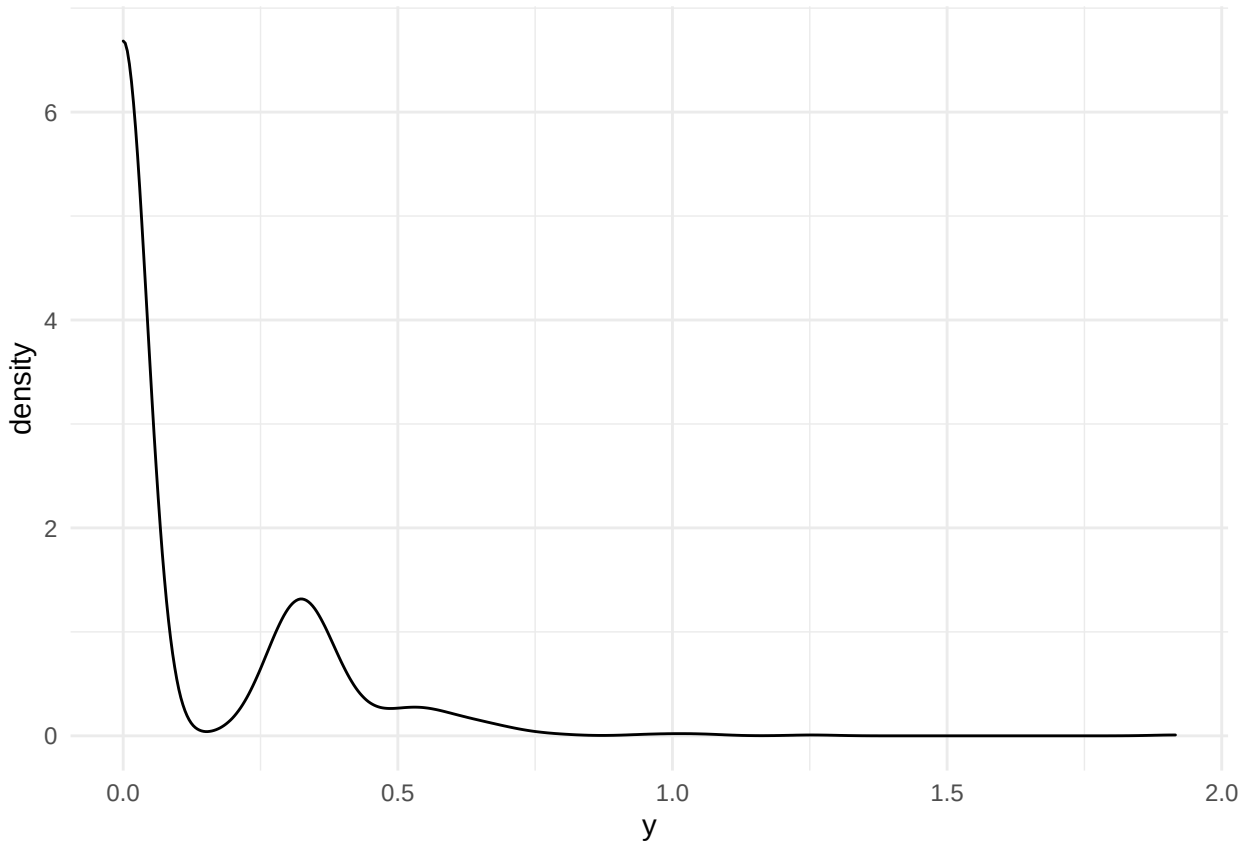
```
ggplot(data.frame("y"=y[[1]]))+geom_density(aes(y))+theme_minimal()
```



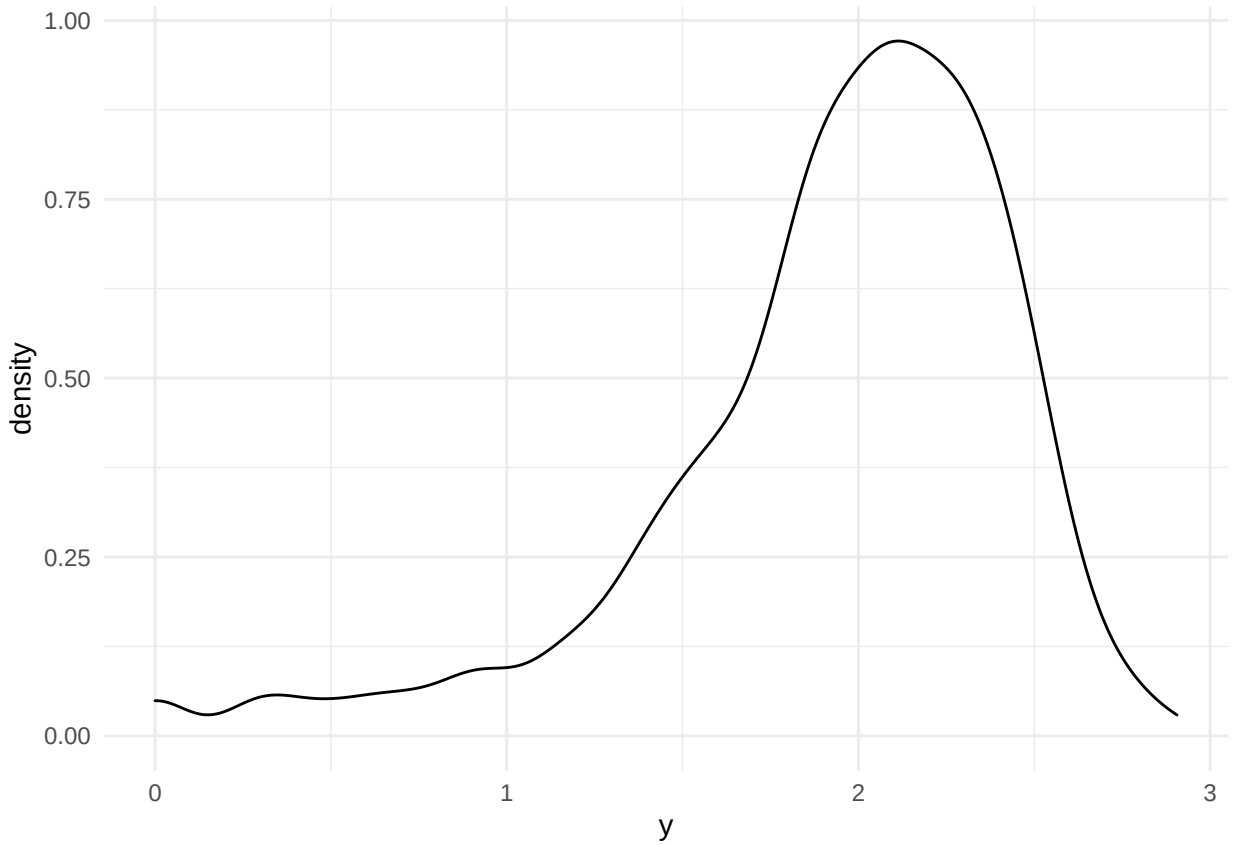

```
ggplot(data.frame("y"=y[[2]]))+geom_density(aes(y))+theme_minimal()
```




```
ggplot(data.frame("y"=y[[3]]))+geom_density(aes(y))+theme_minimal()
```




```
ggplot(data.frame("y"=y[[4]]))+geom_density(aes(y))+theme_minimal()
```



Chcemy rozpoznać które dane należą do którego klastra wykorzystując do tego celu metodę SVM z jądrem radialnym.

```
class<-train(xtrain,as.factor(klast$cluster),
             method="svmRadial",
             trControl = con,
             preProcess="nzv")
print(class)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 3494 samples
## 9000 predictors
##    4 classes: '1', '2', '3', '4'
##
## Pre-processing: remove (6265)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 349, 349, 349, 349, 351, 349, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7980345  0.7131480
##   0.50  0.9124853  0.8778920
##   1.00  0.9918273  0.9887696
##
## Tuning parameter 'sigma' was held constant at a value of 0.0002011492
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.0002011492 and C = 1.
```

W celu preprocesowania danych usuwamy te kolumny o prawie zerowej wariancji. Skuteczność naszej klasyfikacji to 0.9918273 więc widzimy że dane udaje się dobrze klasyfikować. Dane każdego klastra będziem trenowali oddzielnie z wykorzystaniem algorytmu XGBoost boostowania drzew. Hiperparametry wykorzystane w predykcji zostały ustalone badając rozkład błędu walidacyjnego oraz błędu treingowego dla różnych wartości hiperparametrów.

```
val<-trainControl(method="cv",number=10)
ret_model<-function(i)
    {
    x<-xtrain[klast$cluster==i,]
    y<-ytrain[klast$cluster==i]
    grid=expand.grid(max_depth=8,nrounds=300,
                     eta=0.1,gamma=0.3,colsample_bytree=0.5,
                     subsample=1, min_child_weight=3)
    model<-train(x=x,
              y=y,
              method="xgbTree",
             trControl=val,
             tuneGrid=grid,
             preProcess="nzv"
             )
    return(list("model"=model,"RMSE"=min(model$results$RMSE)))
}
k1<-ret_model(1)
k2<-ret_model(2)
k3<-ret_model(3)
k4<-ret_model(4)
err_tr<-sqrt(((k1$RMSE)^2*sum(klast$cluster==1)+
                (k2$RMSE)^2*sum(klast$cluster==2)+
                (k3$RMSE)^2*sum(klast$cluster==3)+
                (k4$RMSE)^2*sum(klast$cluster==4))/nrow(xtrain))
```

Błąd treningowy jaki popełniamy to 0.3220542. Sprawdzamy teraz błąd walidacyjny:

```
calidacja_k<-predict(class,xval)
wynp<-foreach(i=1:nrow(xval)) %dopar%
    {
    k<-calidacja_k[i]
    if (k==1)
        {
        predict(k1$model,xval[i,])
    }
    else if (k==2)
        {
        predict(k2$model,xval[i,])
    }
    else if (k==3)
        {
        predict(k3$model,xval[i,])
    }
    else if (k==4)
        {
        predict(k4$model,xval[i,])
    }
    else
```

```
        {
        0
    }
}
wynp<-unlist(wynp)
er<-sqrt(mean((wynp-yval)^2))
```

który w naszym przypadku wynosi 0.3239194. Teraz robimy predykcję dla danych.

```
test_k<-predict(class,xtest)
wynk<-foreach(i=1:nrow(xtest)) %dopar%
    {
    k<-test_k[i]
    if (k==1)
        {
        predict(k1$model,xtest[i,])
    }
    else if (k==2)
        {
        predict(k2$model,xtest[i,])
    }
    else if (k==3)
        {
        predict(k3$model,xtest[i,])
    }
    else if (k==4)
        {
        predict(k4$model,xtest[i,])
    }
    else
        {
        0
    }
    }
wynk<-unlist(wynk)
odp<-data.frame(ID=0:(nrow(xtest)-1),Expected=wynk)
write.csv(odp,"odp.csv",row.names=FALSE)
```