# Raport 4

Karol Pustelnik
album 249828

14 czerwca 2020

## Spis treści

# 1 Krótki opis zagadnienia

W raporcie przedstawię różne algorytmy/metody klasteryzacji danych (unsupervised learning) i pare algorytmów supervised learning oraz metodę wektorów nośnych. Do analizy użyję danych spam, które zawieraja informacje o mailach chcianych i niechcianych wraz z ich własnościami. Spróbuję odpowiedzieć na pytania:

- Jak wygląda czas działania algorytmów supervised learning?
- Jakie są różnice pomiędzy wybranymi algorytmami unsupervsied learning?

# 2 Opis eksperymentów

Do analizy algorytmów supervised learning skorzystam z:

- bagging,
- boosting,
- random forest,
- classification trees,

A do analizy algorytmów unsupervised learning posłużę się:

- kmeans,
- PAM,

# 3 Wykorzystane biblioteki

```
library(MASS)
library("datasets")
library(ElemStatLearn)
library("HDclassif")
library(class)
library(ipred)
library(MASS)
library(rpart)
library(rpart.plot)
library(e1071)
library(randomForest)
library("adabag")
library(mlbench)
library(stats)
library(MASS)
library(cluster)
library(factoextra)
library(fpc)
library("devtools")
```

```
## List of 1
##  $ plot.title:List of 11
##   ..$ family       : NULL
##   ..$ face         : chr "bold"
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : NULL
```

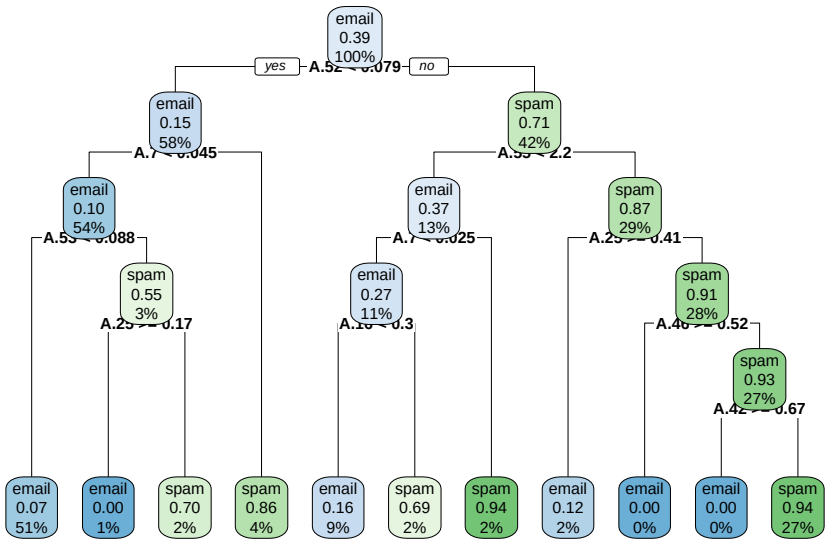
```
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi FALSE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  - attr(*, "class")= chr [1:2] "theme" "gg"
##  - attr(*, "complete")= logi FALSE
##  - attr(*, "validate")= logi TRUE
```

# 4 Rodziny klasyfikatorów - porównanie czasu i efektywnośći algrytmów supervised learning

## 4.1 Drzewo klasyfikacyjne

```
data(spam)
data<-spam
porownanie<-matrix(nrow=4, ncol=4) #tworzę macierz do zapisywania wyników
rownames(porownanie) <- c("Czułość", "Specyficzność",
                          "Szansa na klas.", "Czas")
colnames(porownanie)<-c("Drzewo Klas.", "Bagging",
                        "Lasy losowe",
                        "Boosting")

dt = sample(nrow(data), nrow(data)*0.7) #dzielę dane
train<-data[dt,]
test<-data[-dt,]
start.time <- Sys.time()
tree <- rpart(spam~., data=train)
end.time <- Sys.time()
time.taken <- end.time - start.time
predicted.values<-predict(tree, test, type="class")
results<-table(test$spam,predicted.values)
results

##         predicted.values
##          email spam
##   email   761   55
##   spam     83  482

rpart.plot(tree, main="Drzewo klasyfikacyjne - dane spam", cex=.5)
```

**Drzewo klasyfikacyjne − dane spam**




```
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[1,1]<-sensitivity<-TP/(TP+FN)
porownanie[2,1]<-specificity<-TN/(TN+FP)
porownanie[3,1]<-overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[4,1]<-time.taken
```

Wyniki:

```
data.frame(porownanie[,1])

##                  porownanie...1.
## Czułość               0.8530973
## Specyficzność         0.9325980
## Szansa na klas.       0.9000724
## Czas                  0.4374051
```

## 4.2 Bagging

```
start.time <- Sys.time()
btree <- bagging(spam~., data=train, nbagg=30, minsplit=1, cp=0)
end.time <- Sys.time()
time.taken <- end.time - start.time
predicted.values<-predict(btree, test, type="class")
results<-predicted.values$confusion
results
```

```
##                Observed Class
## Predicted Class email spam
##           email   772   81
##           spam     44  484
```

```
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[1,2]<-sensitivity<-TP/(TP+FN)
porownanie[2,2]<-specificity<-TN/(TN+FP)
porownanie[3,2]<-overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[4,2]<-time.taken
```

Wyniki:

```
data.frame(porownanie[,2])
```

```
##                  porownanie...2.
## Czułość                0.9166667
## Specyficzność          0.9050410
## Szansa na klas.        0.9094859
## Czas                   1.0685159
```

## 4.3 Lasy losowe

```
n<-ncol(spam)-1
start.time <- Sys.time()
rf.1 <- randomForest(spam~., data=train, ntree=100, mtry=n, importance=TRUE)
end.time <- Sys.time()
time.taken <- end.time - start.time
predicted.values<-predict(rf.1, test, type="class")
results<-table(test$spam,predicted.values)
results
```

```
##        predicted.values
```

```
##         email spam
##   email   784   32
##   spam     58  507

TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[1,3]<-sensitivity<-TP/(TP+FN)
porownanie[2,3]<-specificity<-TN/(TN+FP)
porownanie[3,3]<-overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[4,3]<-time.taken
```

Wyniki:

```
data.frame(porownanie[,3])

##                 porownanie...3.
## Czułość               0.8973451
## Specyficzność         0.9607843
## Szansa na klas.       0.9348298
## Czas                  4.9553549
```

## 4.4 Boosting

```
start.time <- Sys.time()
bst<-boosting(spam~., data=train, mfinal=100)
end.time <- Sys.time()
time.taken <- end.time - start.time
predicted.values<-predict(bst, test, type="class")
results<-predicted.values$confusion
results

##                 Observed Class
## Predicted Class email spam
##           email   787   34
##           spam     29  531

TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
```

```
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[1,4]<-sensitivity<-TP/(TP+FN)
porownanie[2,4]<-specificity<-TN/(TN+FP)
porownanie[3,4]<-overall.correct<-(TP+TN)/(TP+TN+FN+FP)
porownanie[4,4]<-time.taken
```

Wyniki:

```
data.frame(porownanie[,4])

##                 porownanie...4.
## Czułość               0.9482143
## Specyficzność         0.9585871
## Szansa na klas.       0.9543809
## Czas                  1.2675760
```

## 4.5 Podsumowanie

Porównanie w tabelce:

```
porownanie

##                 Drzewo Klas.   Bagging Lasy losowe  Boosting
## Czułość            0.8530973 0.9166667   0.8973451 0.9482143
## Specyficzność      0.9325980 0.9050410   0.9607843 0.9585871
## Szansa na klas.    0.9000724 0.9094859   0.9348298 0.9543809
## Czas               0.4374051 1.0685159   4.9553549 1.2675760
```

Najlepiej wypadł algorytm boosting, a najsłabiej algorytm Bagging (biorąc też pod uwagę specyifczność, która w tym zbiorze danych jest ważnym parametrem). Zdecydowanie najdłużej działał algorytm Lasów losowych, potem algrytm boosting i bagging, a najkróćej algorytm drzewa klasyfikacyjnego. Przeprowadzę teraz podobną analizę, ale dla innych parametrów algorytmów.

## 4.6 Ranking Cech według algorytmu Lasów losowych

Ranking Cech

```
rf.1 <- randomForest(spam~., data=spam, ntree=50, mtry=n, importance=TRUE)
```

W kolejności malejącej nastepujące indeksy cech mają znaczny wpływ na klasyfikacje:

```
order(importance(rf.1)[,4], decreasing=TRUE)

##  [1] 53 52  7 55 25 16 56 57 46 19 24 27  5 21 17 45  8 50 18 42 12 28 10 23 49
## [26] 11 37  6 26 22 36  3 13  1 39 14 33  2  9 44 48 20 54 30  4 43 51 35 29 38
## [51] 31 15 41 32 40 47 34
```

# 5 Metoda wektorów nośnych SVM

## 5.1 Jądro liniowe

Przedstawię działanie metody wektorów nośnych dla różnych jąder i ich parametrów.

Algorytm SVM dla jądra liniowego i C=0.5

```
svm.linear.C0.5<- svm(spam~., data=train, kernel="linear", cost=.5)
predicted.values <- predict(svm.linear.C0.5, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki<-cbind(sensitivity, specificity,overall.correct)
wyniki

##      sensitivity specificity overall.correct
## [1,]   0.8973451   0.9534314       0.9304852
```

Dla C=5:

```
svm.linear.C5<- svm(spam~., data=train, kernel="linear", cost=5)
predicted.values <- predict(svm.linear.C5, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki<-cbind(sensitivity, specificity,overall.correct)
wyniki

##      sensitivity specificity overall.correct
## [1,]   0.8973451   0.9509804       0.9290369
```

Dla C=20

```
svm.linear.C20<- svm(spam~., data=train, kernel="linear", cost=20)
predicted.values <- predict(svm.linear.C5, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
```

```
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki<-cbind(sensitivity, specificity,overall.correct)
wyniki

##      sensitivity specificity overall.correct
## [1,]   0.8973451   0.9509804       0.9290369
```

Dla większego C, dostajemy troszkę lepsze wyniki.

## 5.2 Jądro wielomianowe

Algorytm SVM dla funkcji jądrowej wielomianowej.

```
svm.poly4 <- svm(spam~., data=train, kernel="polynomial", degree = 4)

predicted.values<-predict(svm.poly4, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki<-cbind(sensitivity, specificity,overall.correct)
wyniki

##      sensitivity specificity overall.correct
## [1,]   0.3097345   0.9803922       0.7060101
```

Wyniki dla jądra wielomianowego są bardzo słabe.

## 5.3 Jądro radialne

Sprawdźmy jak poradzi sobie jądro radialne.

```
svm.radial.gamma1 <- svm(spam~., data=train, kernel="radial", gamma=1)

predicted.values<-predict(svm.radial.gamma1, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
```

```
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki1<-cbind(sensitivity, specificity,overall.correct)
wyniki1

##      sensitivity specificity overall.correct
## [1,]   0.4938053   0.9987745       0.7921796
```

Znowu wyniki są niezadawalające, ale odrobinę lepsze niż dla jądra wielomianowego.

Przeprowadzę teraz optymalizacje dla jądra radialnego, ale na podzbiorze zbioru treningowego, bo w przeciwnym razie trwało by to bardzo długo.

```
dt2 = sample(nrow(train), nrow(train)*0.1)
train.r<-train[dt2,]
C.range <- 2^((-4):4)
gamma.range <- 2^((-8):4)
radial.tune <- tune(svm, spam~.,
                    data=train.r,
                    kernel="radial",
                    ranges=list(cost=C.range, gamma=gamma.range), scale=FALSE)
print(radial.tune)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost      gamma
##     8 0.00390625
##
## - best performance: 0.2732955

C.best <- radial.tune$best.parameters[["cost"]]
gamma.best <- radial.tune$best.parameters[["gamma"]]
svm.radial.best <- svm(spam~., data=train, kernel="radial",
                       gamma=gamma.best, cost=C.best)
predicted.values<-predict(svm.radial.best, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki1<-cbind(sensitivity, specificity,overall.correct)
svm.default <- svm(spam~., data=train)
```

```
predicted.values<-predict(svm.default, test, type="class")
results<-table(test$spam,predicted.values)
TP<-results[2,2]
TN<-results[1,1]
FP<-results[1,2]
FN<-results[2,1]
sensitivity<-TP/(TP+FN)
specificity<-TN/(TN+FP)
overall.correct<-(TP+TN)/(TP+TN+FN+FP)
wyniki2<-cbind(sensitivity, specificity,overall.correct)
porownanie<-rbind(wyniki1, wyniki2)
rownames(porownanie)<-c("Dostrojone parametry", "Domyślne parametry")
porownanie

##                       sensitivity specificity overall.correct
## Dostrojone parametry   0.8973451   0.9558824       0.9319334
## Domyślne parametry     0.8831858   0.9607843       0.9290369
```

Dla dostrojonych parametrów dostaliśmy praktycznie takie same wyniki.

# 6 Analiza skupisk - algorytmy unsupervised learning

Teraz zajmiemy się algorytmami unsupervised learning, które polegają na grupowaniu danych poprzez wykryte zależności, bez poprzedniej wiedzy o rzeczywistych etykietach.

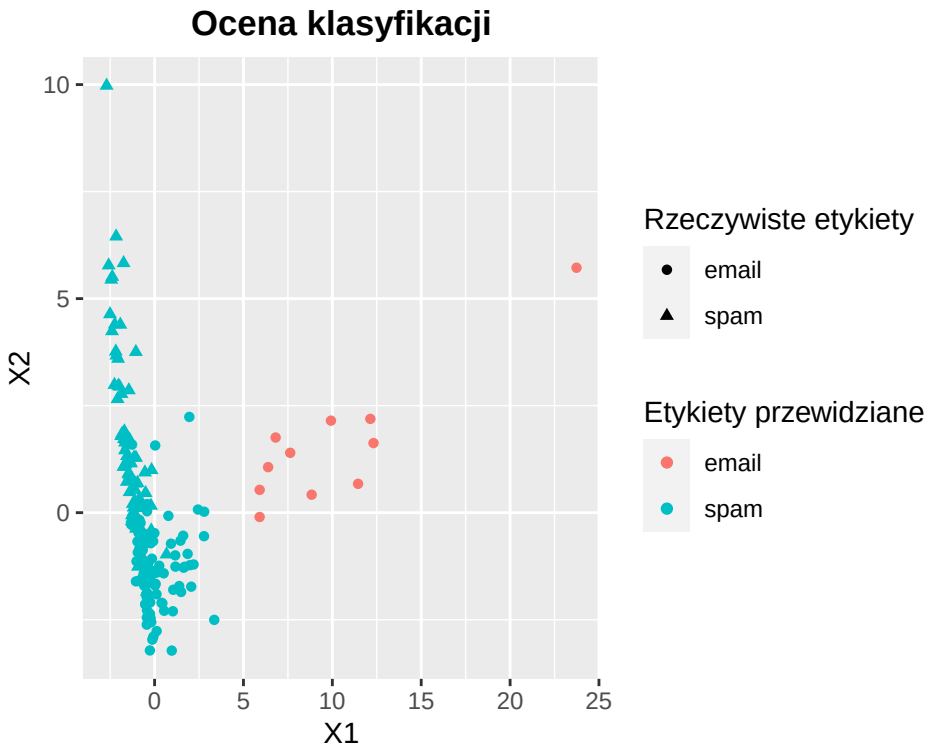
## 6.1 Algorytm k-means

Ten algorytm dzieli dane na ilość grup z góry zadanych przez użytkownika. Próbuje znaleźć "naturalny" podział obiektów na grupy. Nasz zbiór danych warto standaryzować. Ogólnie, prawie zawsze warto standaryzować dane, tym bardziej, że później będe korzystał z PCA.

```
set.seed(123)
data(spam)
spam.200<-spam[sample(nrow(spam), 200), ]
spam.col<-spam.200$spam
spam.200<-spam.200[,1:57]
spam.200<-scale(spam.200)
spam.dissimilarity <- daisy(spam.200) # macierz niepod.
cluster.results <- eclust(spam.200,FUNcluster = "kmeans", k=2, graph=FALSE)
cluster.predictions<-cluster.results$cluster
cluster.predictions[cluster.predictions==1]<-"spam"
cluster.predictions[cluster.predictions==2]<-"email"
```

Wizualiacja wyników z użyciem PCA
Wybrane komponenty objaśniają dane w ok. 22 procent co jest dosyć kiepskim wynikiem.

```
ggplot(dane.plot, aes(x=X1, y=X2,), group=factor(cluster.predictions))+
  geom_point(aes(shape=factor(spam.col),
                 color=factor(cluster.predictions)))+
  labs(shape="Rzeczywiste etykiety", color="Etykiety przewidziane",
       title="Ocena klasyfikacji")+
  guides(size=FALSE)
```

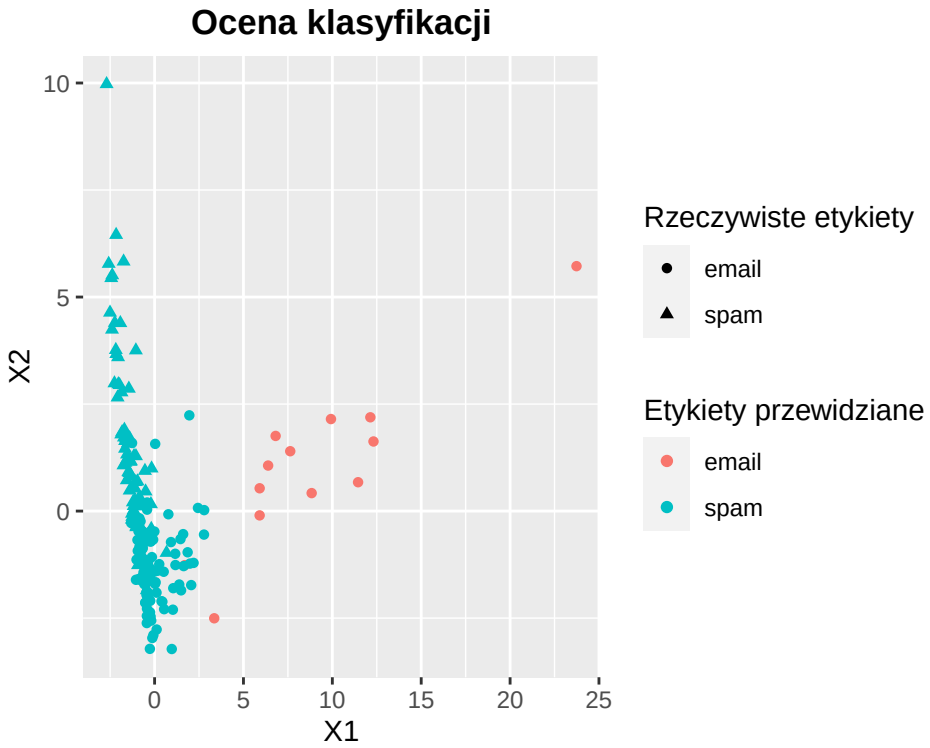



Algorytm stworzył wyraźną separację klas. W obrębie grup elementy są jednak dosyć mocno rozsiane (moim zdaniem), co oznacza słabą zwartość. Występują elementy odstające. Niestety algorytm nie poradził sobie z dopasowaniem do rzeczywistych etykiet.

## 6.2 Algorytm PAM

Algorytm k-medoidów pozwoli nam wykorzystywać inne miary odległości (niż euklidesowa jak w przypadku kmeans) za cenę wyższej złożoności obliczeniowej.

```
pam.k <- eclust(spam.200,FUNcluster = "pam", k=2, graph = FALSE)
cluster.predictions<-pam.k$cluster
cluster.predictions[cluster.predictions==1]<-"spam"
cluster.predictions[cluster.predictions==2]<-"email"
ggplot(dane.plot, aes(x=X1, y=X2,),
       group=factor(cluster.predictions))+
  geom_point(aes(shape=factor(spam.col),
                 color=factor(cluster.predictions)))+
  labs(shape="Rzeczywiste etykiety",
       color="Etykiety przewidziane",
       title="Ocena klasyfikacji")+
  guides(size=FALSE)
```




Wyniki są podobne jak dla poprzedniego algorytmu.

## 6.3 Optymalna liczba klastrów

Dla algorytmu kmeans:

```
partition.agreement <- numeric(10)
for (K in 2:10)
{
  kmeans.k <- eclust(spam.200,FUNcluster = "kmeans",
                     k=K, graph = FALSE)$cluster
  print(paste('Odsetek dobrze sklasyfikowanych klas dla: K=',K))
  results<-table(as.numeric(spam.col),kmeans.k)
  TP<-results[2,2]
  TN<-results[1,1]
  FP<-results[1,2]
  FN<-results[2,1]
  print(overall.correct<-(TP+TN)/(TP+TN+FN+FP))
}

## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 2"
## [1] 0.535
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 3"
## [1] 0.05194805
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 4"
## [1] 0.2307692
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 5"
## [1] 0.07042254
```

```
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 6"
## [1] 0.08219178
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 7"
## [1] 0.2352941
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 8"
## [1] 0.2162162
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 9"
## [1] 0.2272727
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 10"
## [1] 0.2794118
```

Optymalna licba klas to oczywiście 2.
Dla algorytmu pam:

```
partition.agreement <- numeric(10)
for (K in 2:10)
{
  pam.k <- eclust(spam.200,FUNcluster = "pam", k=K, graph = FALSE)$cluster
  print(paste('Odsetek dobrze sklasyfikowanych klas dla: K=',K))
  results<-table(as.numeric(spam.col),pam.k)
  TP<-results[2,2]
  TN<-results[1,1]
  FP<-results[1,2]
  FN<-results[2,1]
  print(overall.correct<-(TP+TN)/(TP+TN+FN+FP))
}
```

```
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 2"
## [1] 0.525
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 3"
## [1] 0.2887701
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 4"
## [1] 0.4
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 5"
## [1] 0.4313725
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 6"
## [1] 0.4343434
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 7"
## [1] 0.4343434
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 8"
## [1] 0.4343434
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 9"
## [1] 0.4343434
## [1] "Odsetek dobrze sklasyfikowanych klas dla: K= 10"
## [1] 0.4343434
```

Tutaj oczywiście również optymalna liczba klas to 2. Ten algorytm poradził sobie odrobinę lepiej, ale nie ma jakichś istotnych różnić. Ogólnie szansa na dobre sklasyfikowanie maila dla

obu algorytmów wynosi ok. 50 procent co jest mało w porównaniu z wynikami na poziomie 90 procent wśród algorytmów supervised learning.

Ważnym wskaźnikiem poprawności metody skupień jest średni silhouette. Im bliżej 1 tym lepiej dane zostały podzielone na klastry. Wartości bliżej -1 sugerują, że obserwacje mogą być przyporządkowane złej klasie. Wartości sillhouette, średnice, separacje skupisk i rozmiary wyznaczę tylko dla najlepszej ilości klastrów, ponieważ dla każdego K od 2 do 10 byłoby to bardzo uciążliwe i ciężkie do interpretacji.

Dla algorytmu kmeans:

```
wyniki.k<-data.frame(matrix(ncol=2, nrow=4))
rownames(wyniki.k)<-c("avg silwidth", "diameter","separation","cluster size")
colnames(wyniki.k)<-c("Klaster 1", "Klaster 2")
kmeans.k <- eclust(spam.200,FUNcluster = "kmeans", k=2, graph = FALSE)

stats<-cluster.stats(spam.dissimilarity,kmeans.k$cluster)
wyniki.k[1,]<-stats$avg.silwidth
wyniki.k[2,]<-stats$diameter
wyniki.k[3,]<-stats$separation
wyniki.k[4,]<-stats$cluster.size
wyniki.k

##                Klaster 1  Klaster 2
## avg silwidth   0.3572754  0.3572754
## diameter      25.6946908 22.3972693
## separation     5.1680518  5.1680518
## cluster size 189.0000000 11.0000000
```

Dla algorytmu PAM:

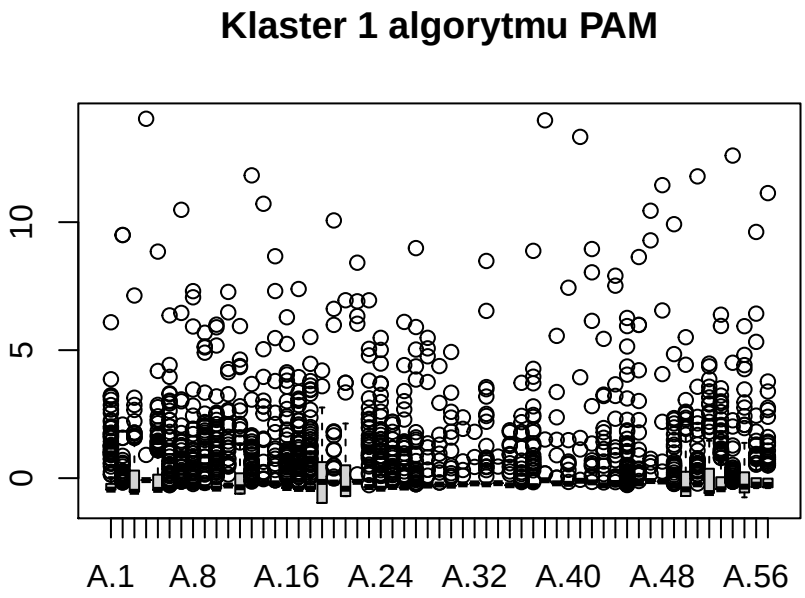
```
wyniki.pam<-data.frame(matrix(ncol=2, nrow=4))
rownames(wyniki.pam)<-c("avg silwidth", "diameter","separation","cluster size")
colnames(wyniki.pam)<-c("Klaster 1", "Klaster 2")
pam.k <- eclust(spam.200,FUNcluster = "pam", k=2, graph = FALSE)

stats<-cluster.stats(spam.dissimilarity,pam.k$cluster)
wyniki.pam[1,]<-stats$avg.silwidth
wyniki.pam[2,]<-stats$diameter
wyniki.pam[3,]<-stats$separation
wyniki.pam[4,]<-stats$cluster.size
wyniki.pam

##                Klaster 1  Klaster 2
## avg silwidth   0.3506319  0.3506319
## diameter      25.6946908 29.3388238
## separation     5.1680518  5.1680518
## cluster size 187.0000000 13.0000000
```

Zarówna dla algorytmu kmeans jak i pam, średni wskaźnik silhouette jest bliski 0.36. Jest to raczej umiarkowany wynik. Pozostałe parametry takie jak średnica, separacja i wielkość klastrów są podobne. Ogólnie nie ma znacznej różnicy w jakości obu modeli.

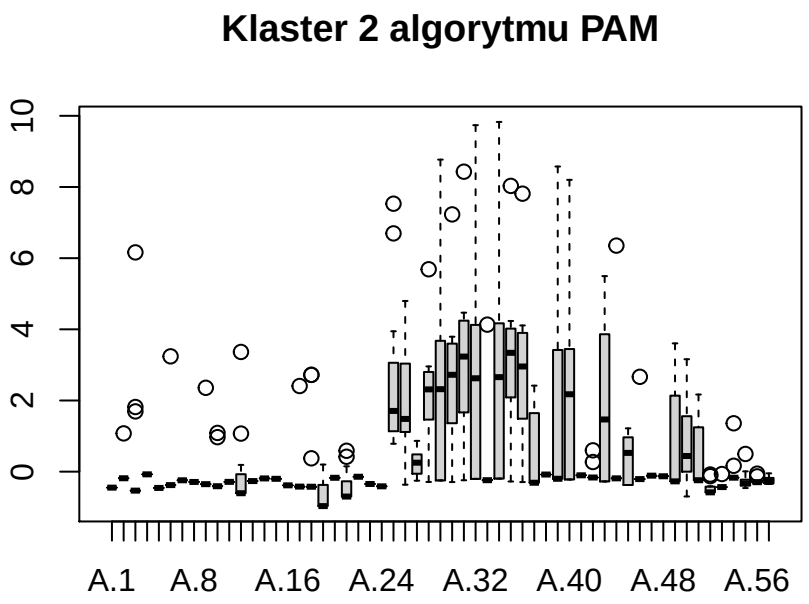
## 6.4 Analiza poszczególnych klastrów

Analiza poszczególnych klastrów dla obu algorytmów (PAN i kmeans)

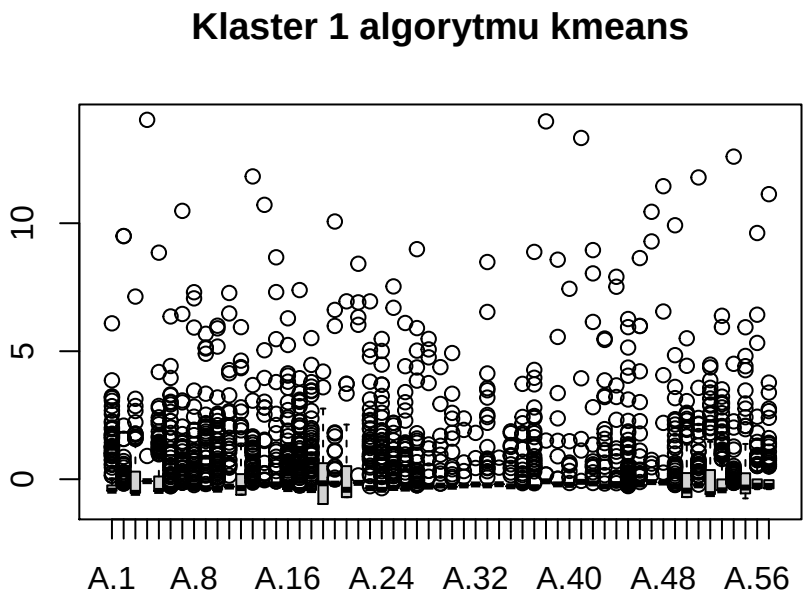
```
results.p<-pam.k$cluster
results.k<-kmeans.k$cluster
klaster1.p<-spam.200[results.p==1,]
klaster2.p<-spam.200[results.p==2,]
klaster1.k<-spam.200[results.k==1,]
klaster2.k<-spam.200[results.k==2,]
boxplot(klaster1.p, main= "Klaster 1 algorytmu PAM")
```



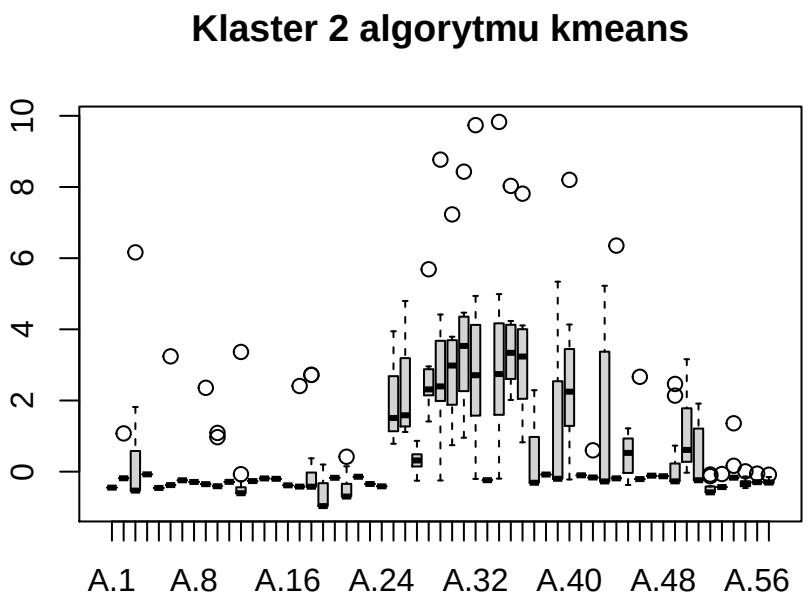

```
boxplot(klaster2.p, main="Klaster 2 algorytmu PAM")
```




```
boxplot(klaster1.k, main= "Klaster 1 algorytmu kmeans")
```




```
boxplot(klaster2.k, main="Klaster 2 algorytmu kmeans")
```




Boxploty pokazują, że w klastrze 1 (dla obu algorytmów) występuje wiele elemntów odstających co potwierdza też wczesniejszy wykres. Klastr 2 (dla obu algorytmów) jest bardziej jednorodny, wyraźnie widać, że charakteryzuje go tylko część zmiennych objaśniających.

```
summary(pam.k$medoids[1,])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6311 -0.3421 -0.2410 -0.2250 -0.1639  0.3963

summary(pam.k$medoids[2,])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6059 -0.3465 -0.1639  0.4741  1.3175  3.9454
```

Klaster 1 reprezentuja te zmienne objaśniające, których wartość jest małą (ujemna). Natomiast klaster 2 raczej reprezentują dodatnie zmienne objaśniające. Wnioski te mogą być nieprecyzyjne ze względu na dużą ilość zmiennych. Ponadto ciężko interpretować ten zbiór danych, ponieważ nie ma zamieszczonych informacji o tym co oznaczają poszczególne zmienne.

# 7 Podsumowanie

Algorytmy supervised learning lepiej radzą sobie z przewidywaniem etykiet od algorytmów unsupervised learning. Ogólnie metody uczenia nienadzorowanego mają raczej inne zastosowania. Przydają sie wtedy kiedy chcemy bardziej poznać zbiór danych i zapytać algorytm o rzeczy, na które nie znamy odpowiedzi. Pozwalają one ujawnić ukryte zależności w zbiorze danych, których nie widać na pierwszy rzut oka.

# Literatura

[1] Stackoverflow, https://stackoverflow.com

[2] Strona kursu Eksploracja danych, prowadzonego przez dr. Adama Zagdańskiego, http://prac.im.pwr.wroc.pl/~zagdan/polish_ver/ED2020/index.html

[3] Wykady na wydziale MIM Uniwersytetu Warszawskiego, , https://www.mimuw.edu.pl/~son/datamining/DM2008/W09-clustering.pdf