

Raport_finalny

Przykładowy przebieg doboru parametrów algorytmów grupowania

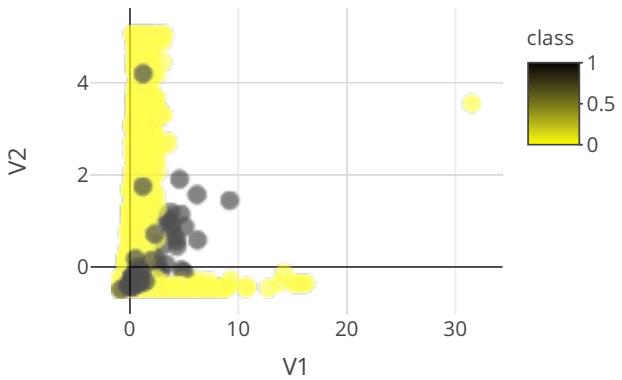
```
mammography_x = read.csv(file = 'mammography_x.csv', header=FALSE)
mammography_y = read.csv(file = 'mammography_y.csv', header=FALSE)

mammography = mammography_x %>% mutate(class = mammography_y$V1)
```

Aby lepiej zrozumieć charakterystykę danych, wyświetlm ją i zaznaczmy znane anomalie zanim przejdziemy do nienadzorowanej detekcji anomalii.

```
plot_ly(mammography,
        x = ~V1,
        y = ~V2,
        color=~class,
        colors=c('#ffff00', '#000000'),
        opacity=0.7,
        size=1,
        type="scatter",
        mode="markers"
    )

## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



W przypadku danych o więcej niż 3 wymiarach, ciężko jest zwizualizować dane tak by widać było anomalie na pierwszy rzut oka.

Dowiedzmy się przynajmniej ile takich anomalii w zbiorze jest, aby poznać skalę problemu.

```
sum(mammography_y)
```

```
## [1] 260
```

Podział zbioru danych na treningowe i testowe z informacją o ilości anomalii w podzbiorach.

```
split_ratio = 0.8
```

```
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_train_x = data_train %>% select(1:6)
data_train_y = data_train %>% select(7)

data_test = subset(mammography, split == FALSE)
data_test_x = data_test %>% select(1:6)
data_test_y = data_test %>% select(7)

sum(data_train_y)
```

```
## [1] 185
```

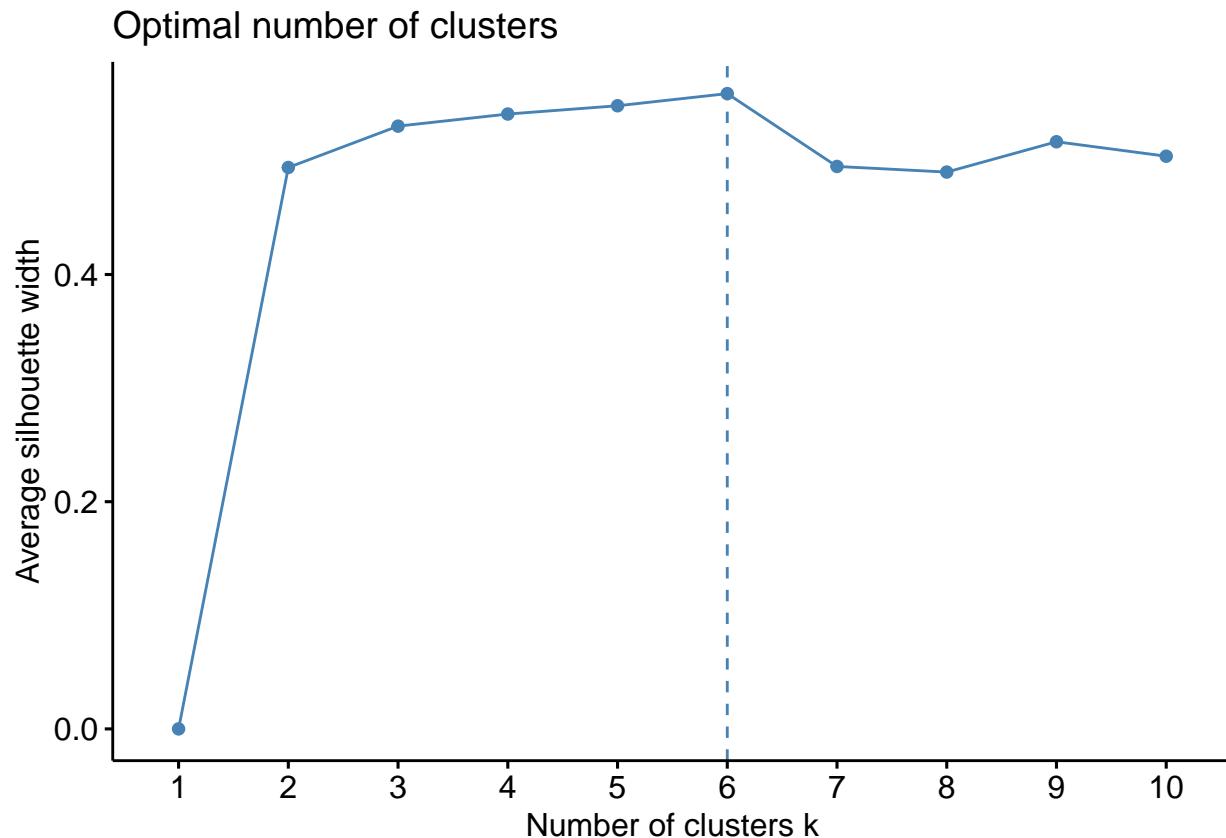
```
sum(data_test_y)
```

```
## [1] 75
```

Grupowanie

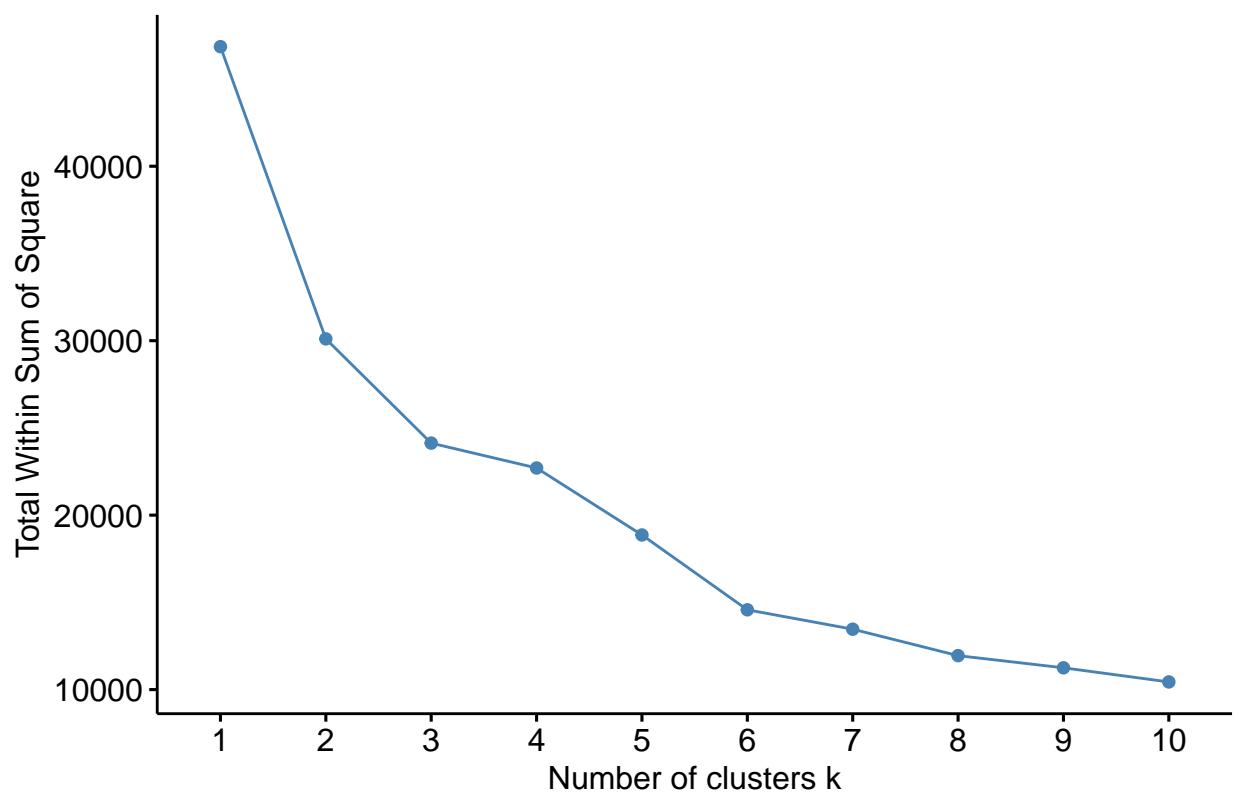
Do pogrupowania danych z użyciem algorytmu centroidów, trzeba wiedzieć ile takich centroidów ma być w danych. Sprawdzimy to z wykorzystaniem funkcji fviz_nbclust z pakietu factoextra. Jako, że można użyć kilku metod do sprawdzenia optymalnej ilości klastrów, sprawdźmy więcej niż jedną.

```
fviz_nbclust(data_train_x, kmeans, method = "silhouette")
```

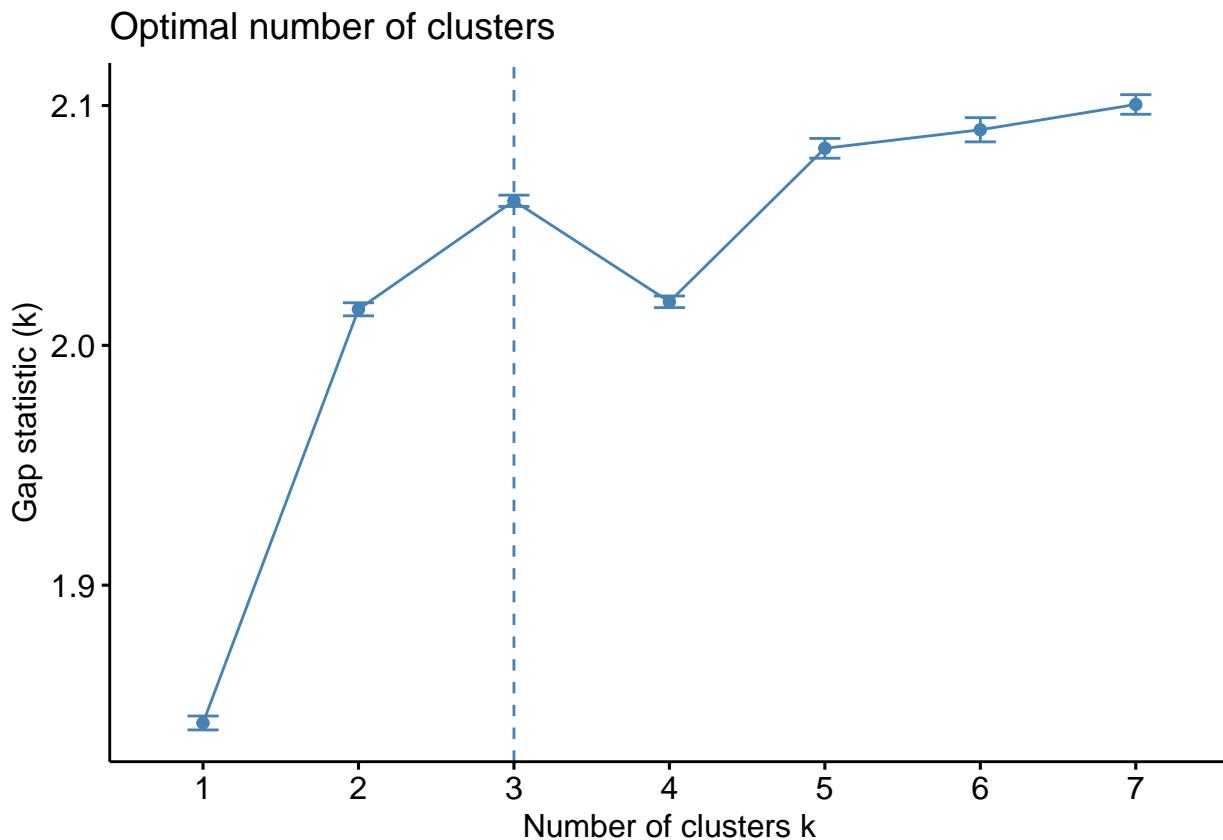


```
fviz_nbclust(data_train_x, kmeans, method = "wss")
```

Optimal number of clusters



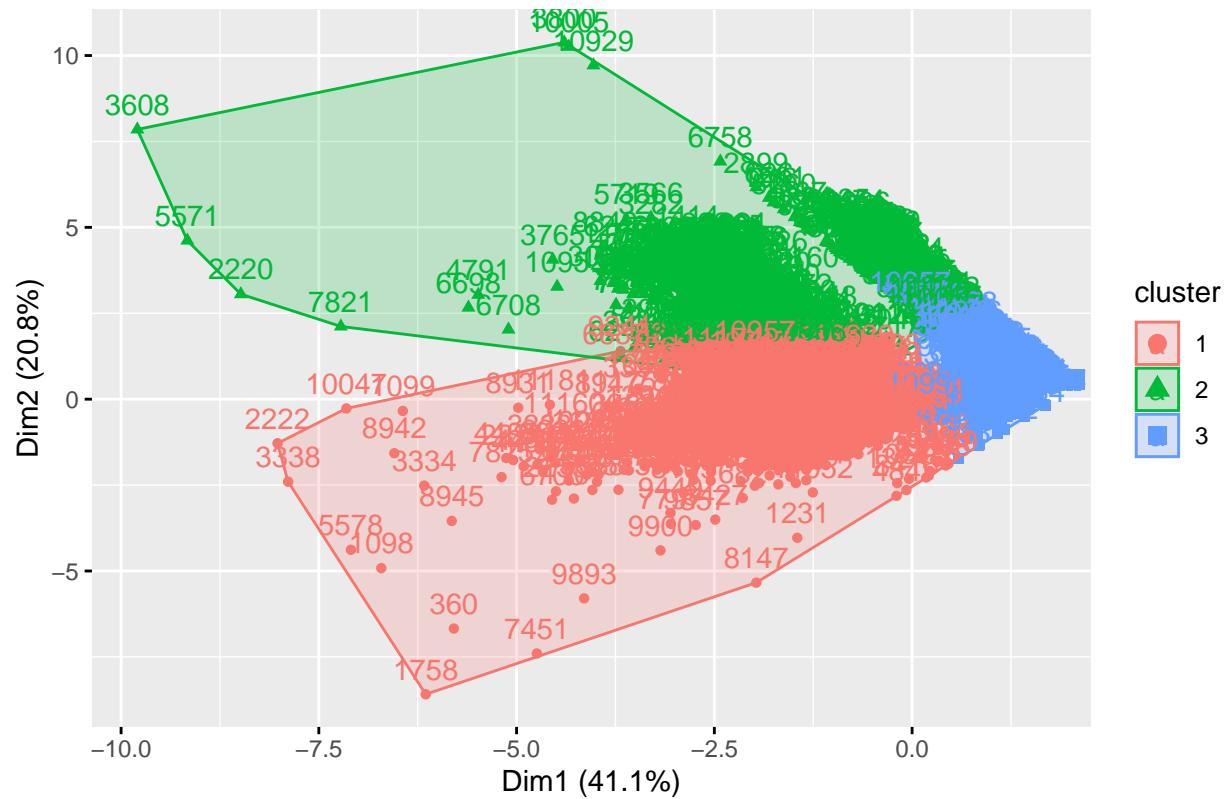
```
fviz_nbclust(data_train_x, kmeans, method = "gap_stat", k.max=7, nboot=30)
```



Zwizualizujmy, jak wygląda grupowanie dla wybranej ilości centroidów.

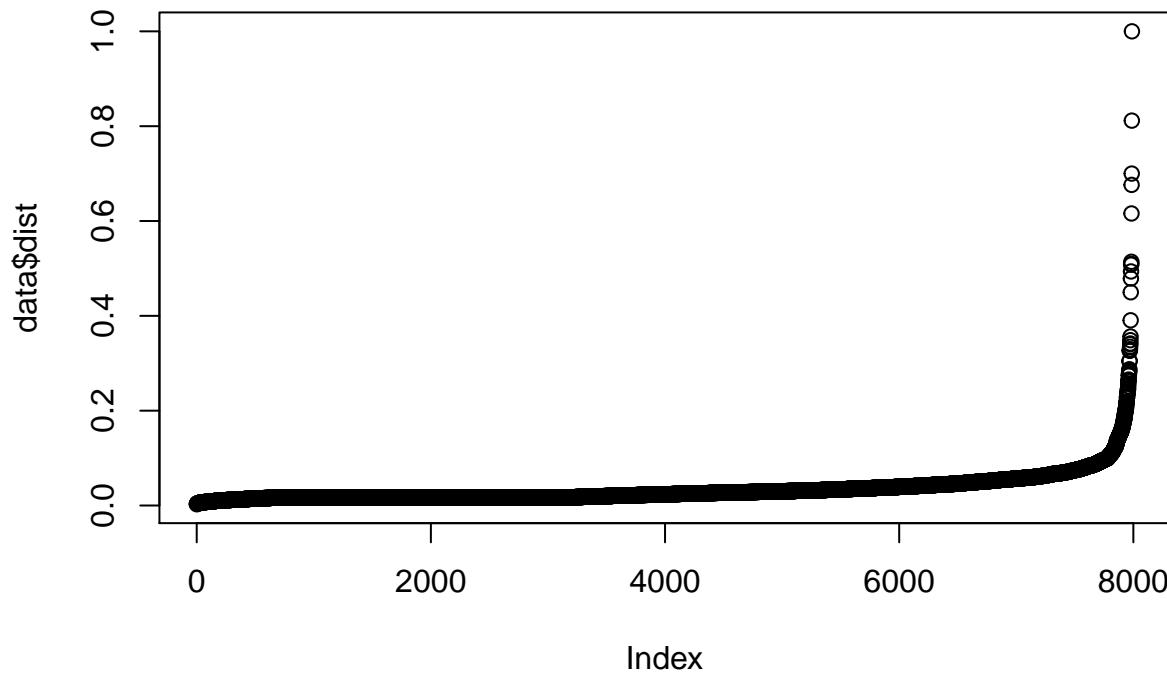
```
no_clusters = 3
cluster_data = kmeans(data_train_x, no_clusters, algorithm="Hartigan-Wong")
fviz_cluster(cluster_data, data = data_train_x)
```

Cluster plot



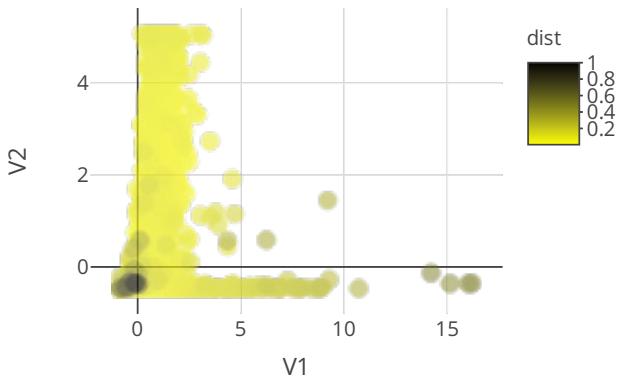
Tutaj, aby wybrać anomalie spośród punktów w grupach należałoby dobrać jakąś miarę niepodobieństwa. Określmy przykładową miarę niepodobieństwa jako odległość danej od środka grupy do której została zakwalifikowana.

```
cluster_centers = cluster_data$centers
data = data_train
data %<-% mutate(group = cluster_data$cluster) %>% mutate(dist = sqrt((V1-cluster_centers[group, 1])^2 +
data$dist = data$dist / max(data$dist)
plot(data$dist)
```



Miara niepodobieństwa mapuje nam dane wielowymiarowe na jeden wymiar. Zwracają one liczby w zakresie $[0, +\infty)$. Ustalenie wartości odcięcia na takim zakresie jest niewygodne, postanowiliśmy ją więc znormalizować. Użyliśmy normalizacji metodą min-max, jako że minimum tak czy inaczej mieliśmy w zerze, to po prostu dzielimy wyniki przez ich maksimum.

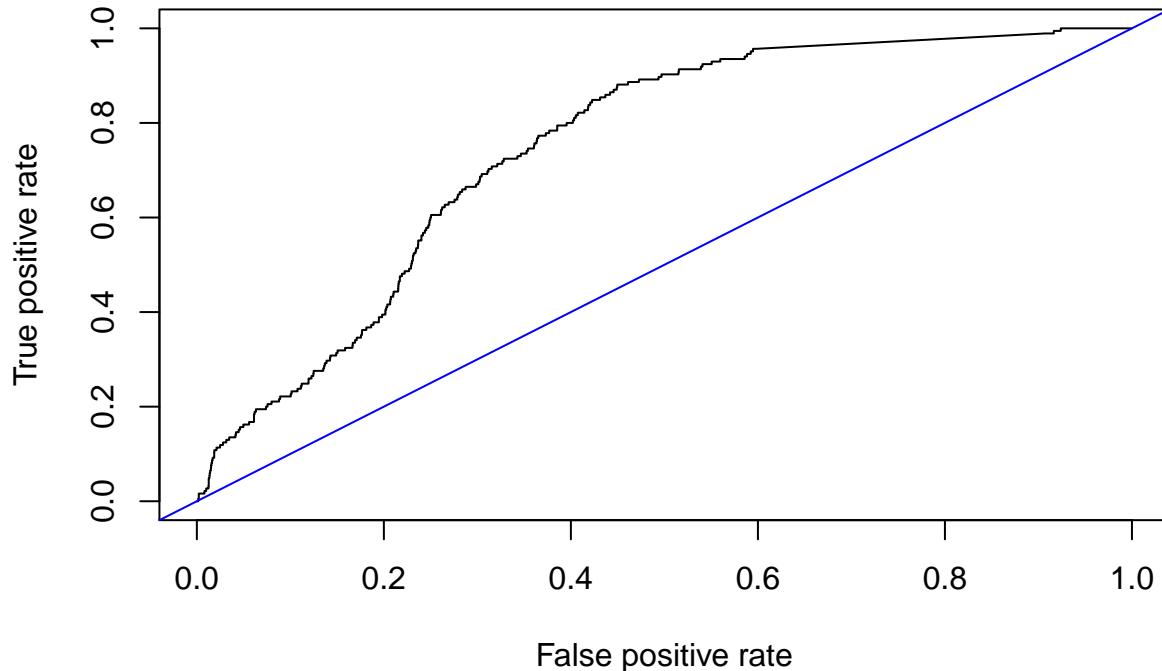
```
plot_ly(data,
        x = ~V1,
        y = ~V2,
        color=~dist,
        colors=c('#ffff00', '#000000'),
        opacity=0.7,
        size=1,
        type="scatter",
        mode="markers"
    )
```



Jednowymiarowe dane jesteśmy w stanie porównać, dzięki czemu możemy określić jakiś próg, powyżej którego będziemy dane traktowali jako anomalie. Po zakwalifikowaniu anomalii, możemy porównać wyniki z faktycznie wykrytymi anomaliami podanymi w danych wejściowych. W ten sposób dowiemy się, jaka jest jakość modelu a-priori. Na tym etapie wybieramy jakimi miarami jakości chcielibyśmy się posłużyć. W naszym przypadku wybieramy analizę ROC oraz miarę F1.

```
outlier_threshold = 0.35
data = data %>% mutate(pred_int = as.integer(dist > outlier_threshold))

pred = prediction(predictions=data$dist, labels=data$class)
perf = performance(pred, "tpr", "fpr")
plot(perf)
abline(0,1,col="#0000ff")
```



Sprawdźmy jeszcze miarę F1. Zwraca wysokie wyniki tylko, gdy jednocześnie oba współczynniki, precyzji i odzysku, dają wysokie wyniki.

```
data = data %>% mutate(tp = class & pred_int)
data = data %>% mutate(fp = !class & pred_int)
data = data %>% mutate(tn = !class & !pred_int)
data = data %>% mutate(fn = class & !pred_int)
recall = tally(data, tp) / (tally(data, tp) + tally(data, fn))
precision = tally(data, tp) / (tally(data, tp) + tally(data, fp))
f_value = 2 * recall * precision / (recall + precision)
f_value

##          n
## 1 0.01010101
```

Na tym etapie mamy wyniki jakie metody grupowania mogłyby dać dla tego zbioru danych. Są to niestety miary które niewiele mówią o jakości predykcji dla nowych danych. Są one w zasadzie tylko poglądowe, aby upewnić się że można wytrenować model w sposób nienadzorowany metodami grupowania, tak aby wyniki detekcji anomalii na podstawie tego modelu były niesłosowe i relatywnie poprawne.

W tym miejscu warto zaznaczyć, że wynik NaN oznacza, że i precyzja i odzysk osiągnęły wartość 0, czyli nie było poprawnie zakwalifikowanej nawet jednej anomalii. W pozostałych przypadkach, określona wartość będzie płaściwa się w przedziale [0,1], gdzie 0 będzie oznaczało jakość niską, a 1 wysoką.

Natomiast, aby określić jakość modelu, musielibyśmy zastosować go dla nowych danych i sprawdzić czy predykcje okazałyby się identyczne z faktycznym wynikiem kwalifikacji anomalii.

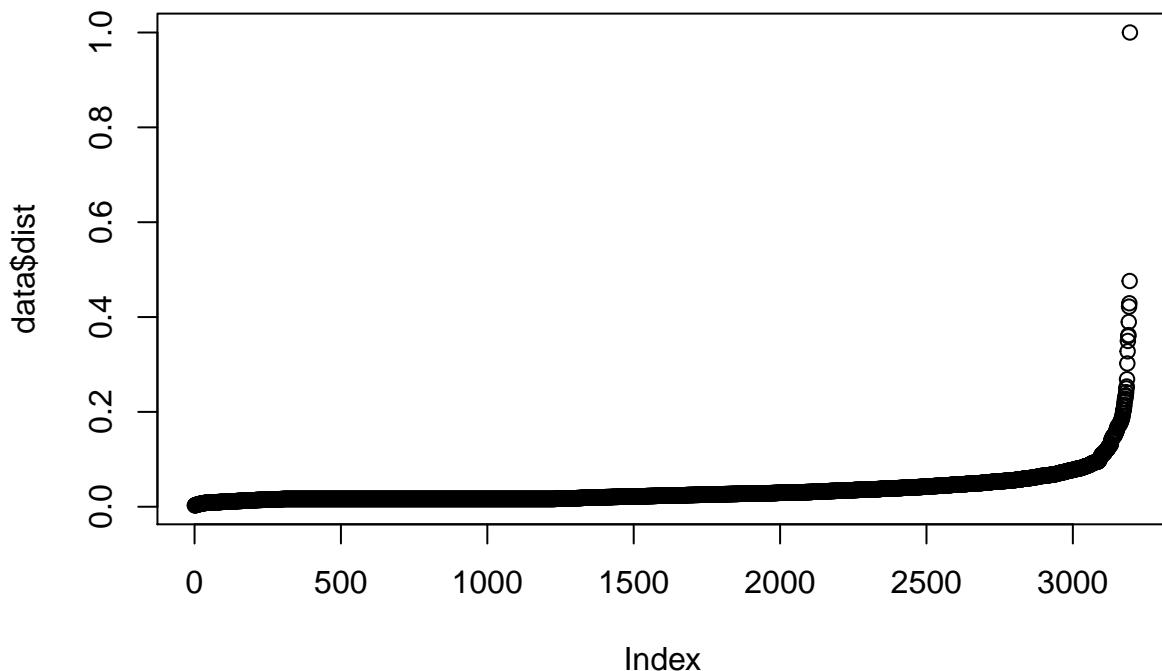
Aby określić jak model radzi sobie z nowymi danymi, będziemy musieli go zastosować dla zbioru danych który nie został wzięty pod uwagę przy tworzeniu modelu.

```

outlier_threshold = 0.4
predict.kmeans <- function(object, newdata){
  centers <- object$centers
  n_centers <- nrow(centers)
  dist_mat <- as.matrix(dist(rbind(centers, newdata)))
  dist_mat <- dist_mat[-seq(n_centers), seq(n_centers)]
  max.col(-dist_mat)
}

groups = predict(cluster_data, data_test_x)
data = data_test %>% mutate(group = groups)
data %>% mutate(dist = sqrt((V1-cluster_centers[group, 1])^2 + (V2-cluster_centers[group, 2])^2 + (V3-cluster_centers[group, 3])^2))
data$dist = data$dist / max(data$dist)
data = data %>% mutate(pred_int = as.integer(dist > outlier_threshold))
plot(data$dist)

```



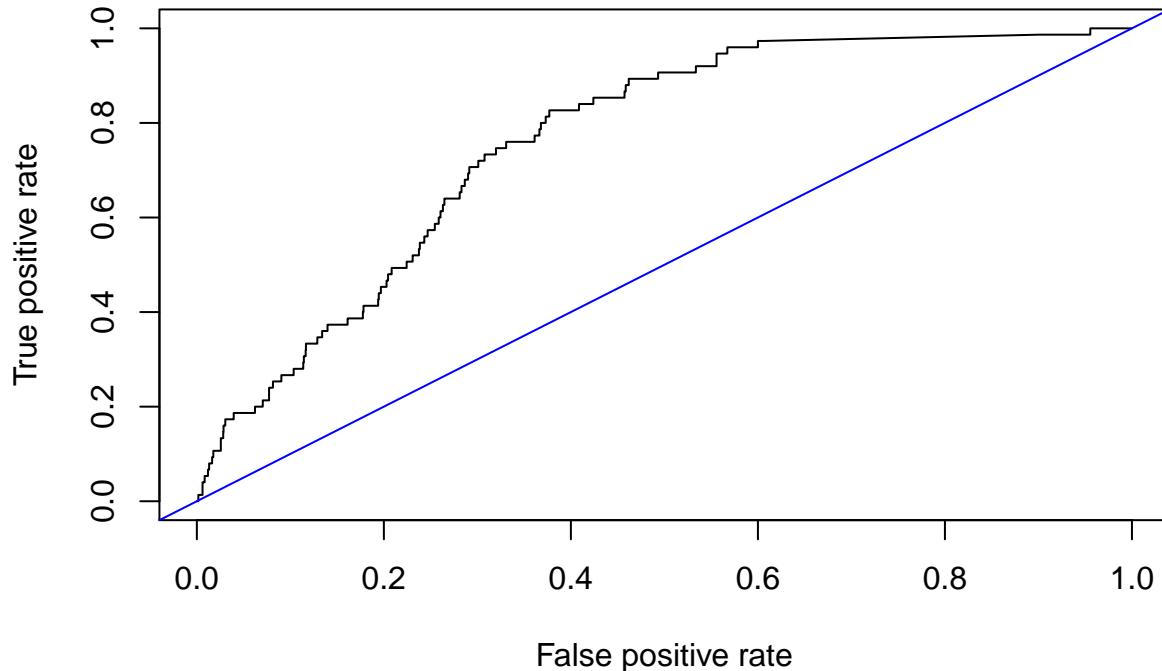
Zrobiliśmy predykcje grup na podstawie modelu, zaaplikowaliśmy miarę niepodobieństwa na wyniki danych testowych. Rozkład posortowanych miar niepodobieństwa dla danych testowych prezentuje się jak wyżej.

Aby określić jakość predykcji, zastosujemy analizę roc oraz wyliczymy miarę F1.

```

pred = prediction(predictions=data$dist, labels=data$class)
perf = performance(pred, "tpr", "fpr")
plot(perf)
abline(0,1,col="#0000ff")

```



```

data = data %>% mutate(tp = class & pred_int)
data = data %>% mutate(fp = !class & pred_int)
data = data %>% mutate(tn = !class & !pred_int)
data = data %>% mutate(fn = class & !pred_int)
recall = tally(data, tp) / (tally(data, tp) + tally(data, fn))
precision = tally(data, tp) / (tally(data, tp) + tally(data, fp))
f_value = 2 * recall * precision / (recall + precision)
f_value

##      n
## 1  NaN

```

Miary jakości powyżej oznaczają, że detekcja anomalii z użyciem takich parametrów, miary niepodobieństwa w postaci odległości od najbliższego klastra i z zastosowaniem grupowania centroidami nie przynosi znaczącego uzysku ponad losowe wybieranie anomalii spośród danych. Może być to spowodowane, że algorytm centroidów jest słabo dostosowany do charakterystyki danych tego zbioru, jak również słabym dopasowaniem miary niepodobieństwa.

Możliwe, że w przypadku tego zbioru danych zastosowanie innych algorytmów grupowania oraz innych miar niepodobieństwa do grup okazałoby się jakościowo lepszym rozwiązaniem.

Klasyfikacja

Na tych samych zbiorach danych uruchomiono nadzorowane algorytmy klasyfikacji w celu porównania ich skuteczności z algorytmami grupującymi uczonymi bez nadzoru. W tym celu wykorzystano wbudowane metody klasyfikacji: * **svm** (Support Vector Machine) z funkcją jądrową wielomianową * **naiveBayes** (Naiwny klasyfikator Bayesowski) * **ctree** (Conditional Inference Tree)

Do każdej z metod podano wartości ze zbioru treningowego wraz z etykietą przynależności do klasy anomalii, a następnie dokonano predykcji klasy dla punktów ze zbioru testowego. Zbiory były takie same jak podczas nienadzorowanego grupowania.

Dla każdej z metod dokonano pomiaru wskaźnika F na podstawie macierzy błędów oraz wyznaczono kształt krzywej ROC.

Do określania miary F1 użyliśmy definicji funkcji:

```
f_value <- function (cm) {  
  fp = cm[1,2]  
  fn = cm[2,1]  
  tp = cm[2,2]  
  recall = tp / (tp+fn)  
  precision = tp / (tp+fp)  
  f = (2 * recall * precision) / (recall + precision)  
  return(f)  
}
```

Algorytmy

Support Vector Machines (SVM) Metoda ma za zadanie wyznaczyć hiperpłaszczyznę oddzielającą przykłady należące do dwóch klas w taki sposób aby uzyskać między nimi największy margines. Funkcja jądrowa ma zadanie przekształcić dane wejściowe do takiej przestrzeni, aby możliwe było wyznaczenie płaszczyzny która oddziela najwięcej punktów.

```
svm.model = svm(as.factor(class) ~ ., data = data_train, type = 'C-classification', kernel = 'polynomial')  
svm.pred = predict(svm.model, newdata = data_test_x)  
svm.cm = table(data_test_y$class, svm.pred)  
f_value(svm.cm)  
  
## [1] 0.6086957
```

Naiwny klasyfikator Bayesowski Metoda zakłada niezależność zmiennych decyzyjnych, i wyznacza na ich podstawie prawdopodobieństwo przynależności przynależności punktu wejściowego do danej klasy, tworząc modele rozkładów ze na podstawie przykładów ze zbioru uczącego. Ponieważ jedną z dwu klas stanowią anomalie, modelowanie ich rozkładu z definicji powinno okazać się nieskuteczne.

```
nvb.model = naiveBayes(as.factor(class) ~ ., data = data_train)  
nvb.pred = predict(nvb.model, newdata = data_test_x)  
nvb.cm = table(data_test_y$class, nvb.pred)  
f_value(nvb.cm)  
  
## [1] 0.3700787
```

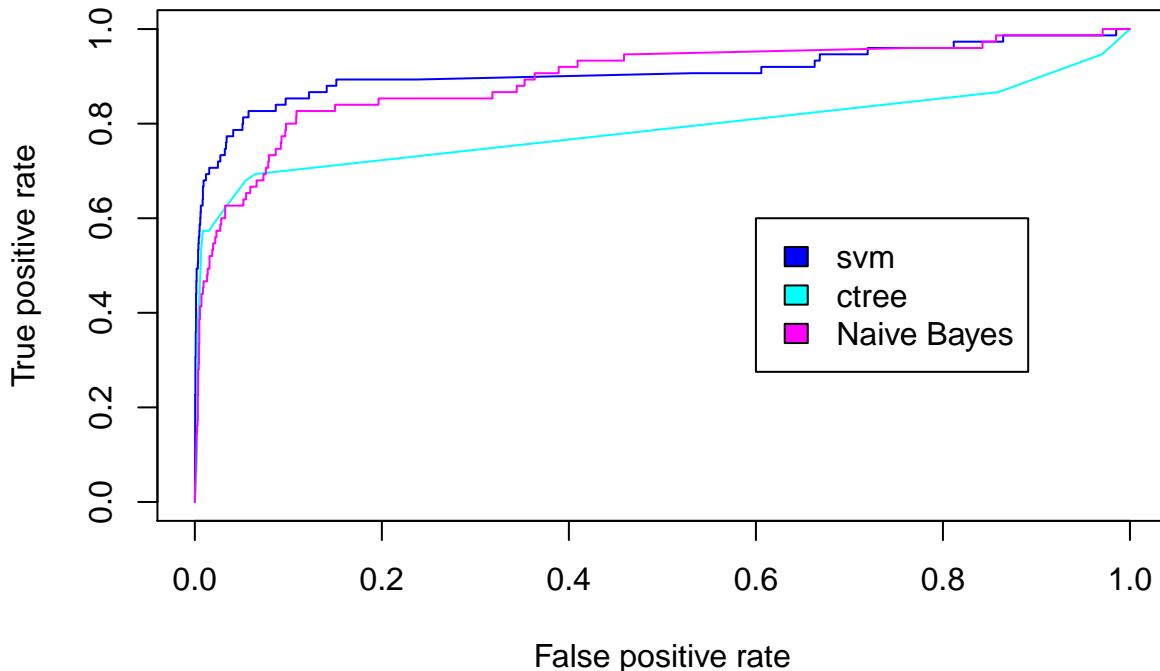
Conditional Inference Tree (ctree) Drzewo wnioskowania warunkowego jest drzewem decyzyjnym konstrującym kolejne poziomy podziału na podstawie statystycznego testu niezależności zmiennych decyzyjnych od klasy. Dla każdego poziomu decyzji wybierana jest ta zmienna decyzyjna dla której test niezależności klasy od zmiennej ma najwyższą istotność (zmienna jest silnie związana z klasą). Tak skonstuuowane drzewo nie wymaga dalszego przycinania.

```
ct.model = ctree(as.factor(class) ~ ., data = data_train)  
ct.pred = predict(ct.model, newdata = data_test_x)  
ct.cm = table(data_test_y$class, ct.pred)  
f_value(ct.cm)  
  
## [1] 0.5511811
```

Jakość modeli Analiza ROC pozwala nam zobaczyć jakość predykcji danego algorytmu. Im większe jest pole pod wykresem, tym wyższa jakość modelu.

```
require(ROCR)
svm.prob = predict(svm.model, type="prob", newdata= data_test_x, probability = TRUE)
svm.rocr = prediction(attr(svm.prob, "probabilities")[,2], data_test_y$class)
svm.perf = performance(svm.rocr, "tpr","fpr")
nvb.prob = predict(nvb.model, newdata=data_test_x, type="raw")
nvb.rocr = prediction(nvb.prob[,2], data_test_y$class)
nvb.perf = performance(nvb.rocr, "tpr","fpr")
ct.prob = predict(ct.model, newdata=data_test_x, type="prob")
ct.rocr = prediction(ct.prob[,2], data_test_y$class)
ct.perf = performance(ct.rocr, "tpr","fpr")
plot(svm.perf, col=4, main="ROC curves of different machine learning classifier")
legend(0.6, 0.6, c('svm', 'ctree', 'Naive Bayes'), 4:6)
plot(ct.perf, col=5, add=TRUE)
plot(nvb.perf, col=6, add=TRUE)
```

ROC curves of different machine learning classifier



Przebieg badania

Porównanie algorytmów grupowania cmeans, kmeans dla miar niepodobieństwa CBLOF, uCBLOF i LDCOF względem klasyfikacji algorytmami SVM, Naive-Bayes i CTree

Założenia W tym dokumencie chcemy się skupić na porównaniu doboru miary niepodobieństwa i algorytmu grupowania, a wynikami wybranych algorytmów klasyfikacji. W związku z tym, porównania będziemy wykonywać już na dobranych parametrach. Nie zapewniamy, że parametry zostały dobrane idealnie do podanych zbiorów danych i z użyciem odpowiednich narzędzi. Dobór parametrów został wykonany dla

każdego zbioru danych oddzielnie. Sposób doboru parametrów został opisany w oddzielnym pliku.

Przebieg badania Zaczynamy najpierw dane dwóch zbiorów danych, oba mają dane 6-wymiarowe. Zbiór mammography składa się z 11183 rekordów, z czego 260 to anomalie. Zbiór annthyroid składa się z 7200 rekordów, z czego 534 to anomalie

```
rm(list = ls())
mammography_x = read.csv(file = 'mammography_x.csv', header=FALSE)
mammography_y = read.csv(file = 'mammography_y.csv', header=FALSE)
mammography = mammography_x %>% mutate(class = mammography_y$V1)

annthyroid_x = read.csv(file = 'annthyroid_x.csv', header=FALSE)
annthyroid_y = read.csv(file = 'annthyroid_y.csv', header=FALSE)
annthyroid = annthyroid_x %>% mutate(class = annthyroid_y$V1)
```

Wyniki poszczególnych bloków kodu będą przedstawiały odpowiednio: - wykres wstępnie wizualizujący efekty grupowania, - wykres przedstawiający rozkład wyników pomiaru niepodobieństwa do dopasowanych grup dla poszczególnych punktów ze zbioru danych, - wynik pomiaru współczynnika F dla zadanej wartości odcięcia, - wykres ROC

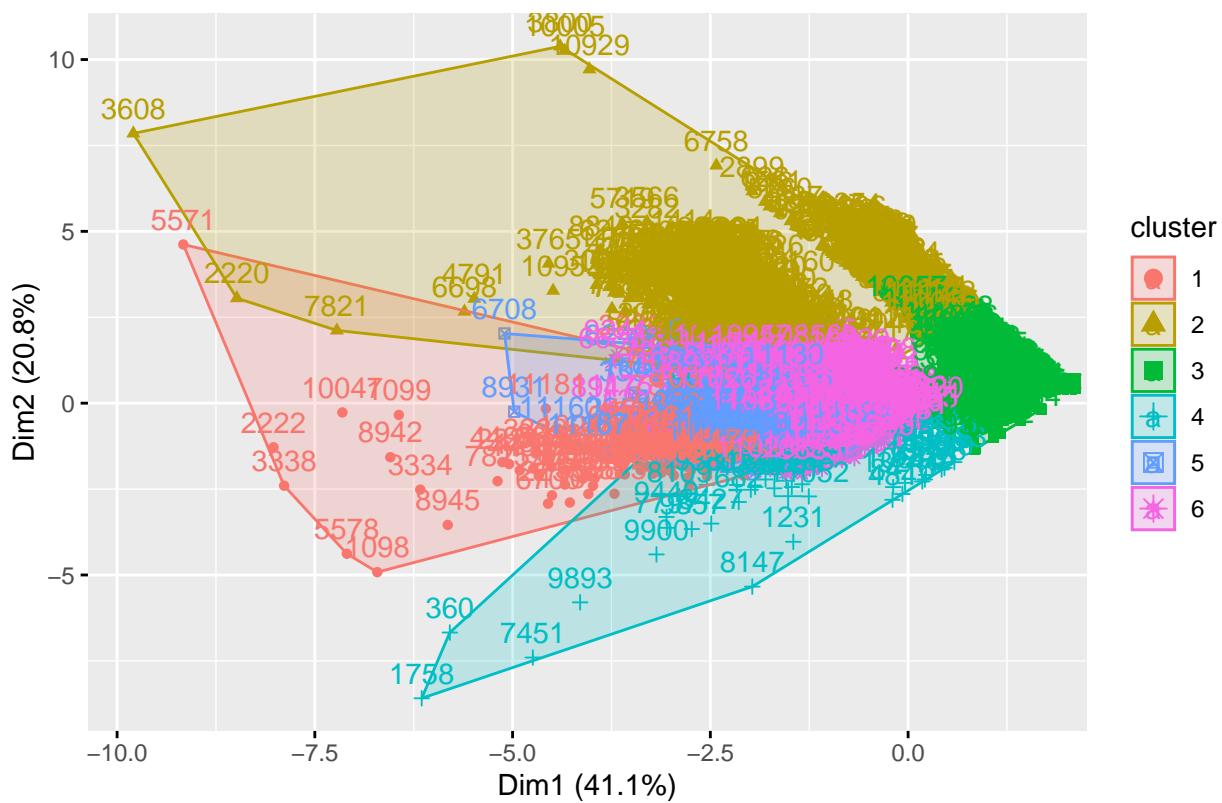
Wewnątrz bloków kodu będziemy ustalać podział dla danych treningowych i testowych, określić z którego algorytmu będziemy korzystać, podawać hiperparametry niezbędne do jego działania oraz dodatkowo określić poziom odcięcia miary niepodobieństwa, od którego będziemy uznawać punkty jako anomalie.

Porównanie wpływu miary niepodobieństwa Sprawdźmy najpierw w ramach pierwszego zbioru danych, jaki wpływ na wynik ma dobór miary niepodobieństwa.

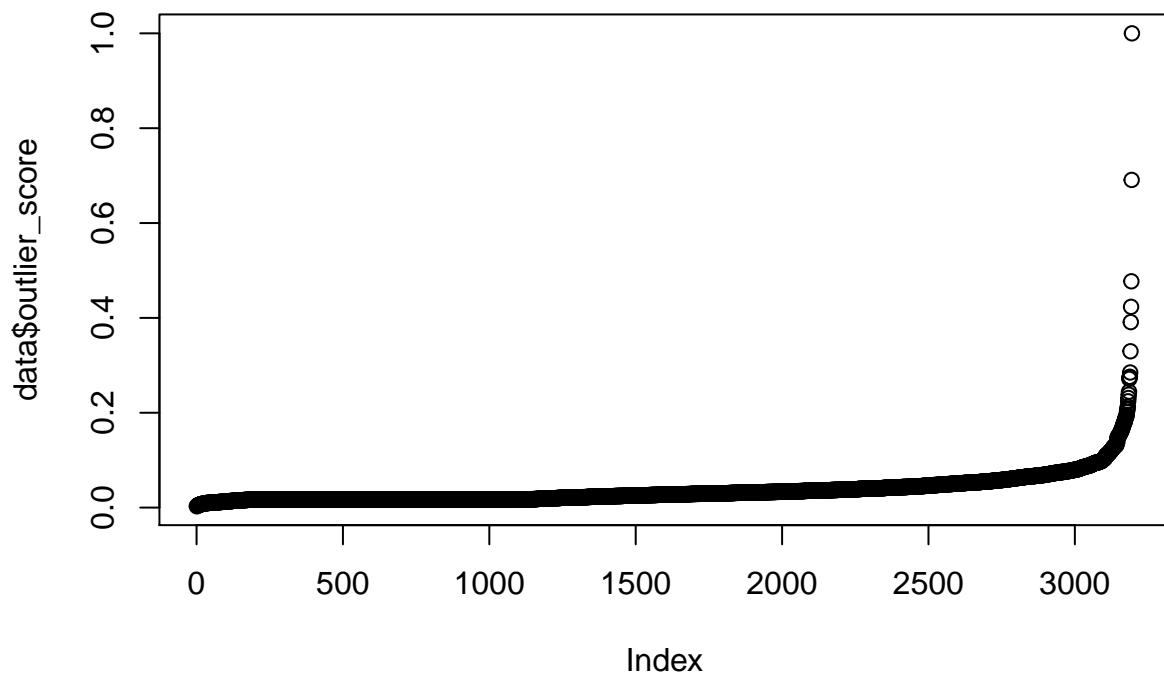
```
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

metric = "euclidean"
method = "kmeans"
dissimilarity_measure = "uCBLOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)
```

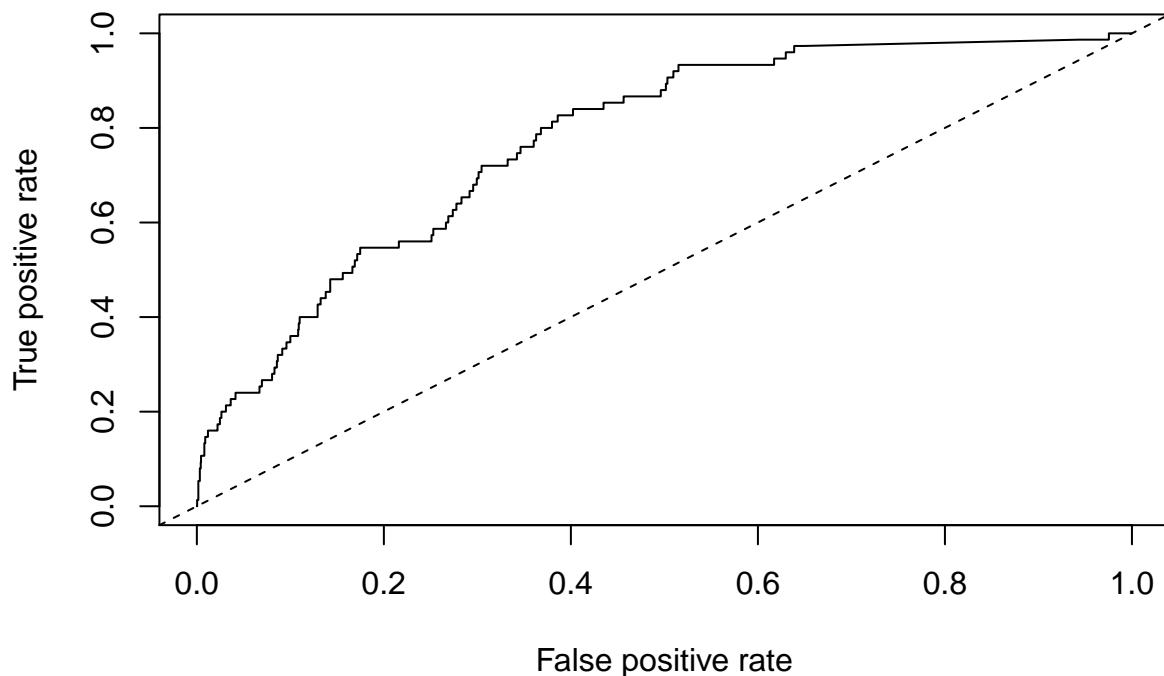
Cluster plot



dissimilarity distribution



ROC curve

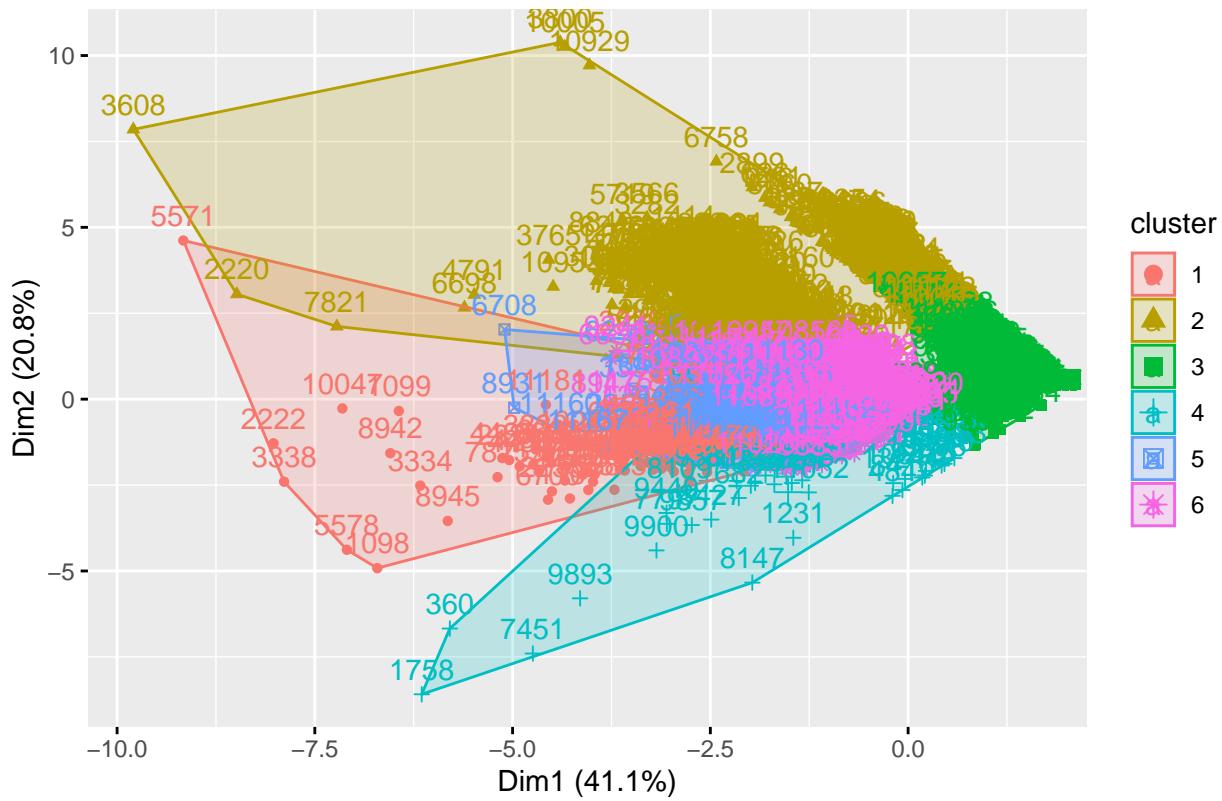


```
## [1] "F-value: 0.0246913580246914"
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

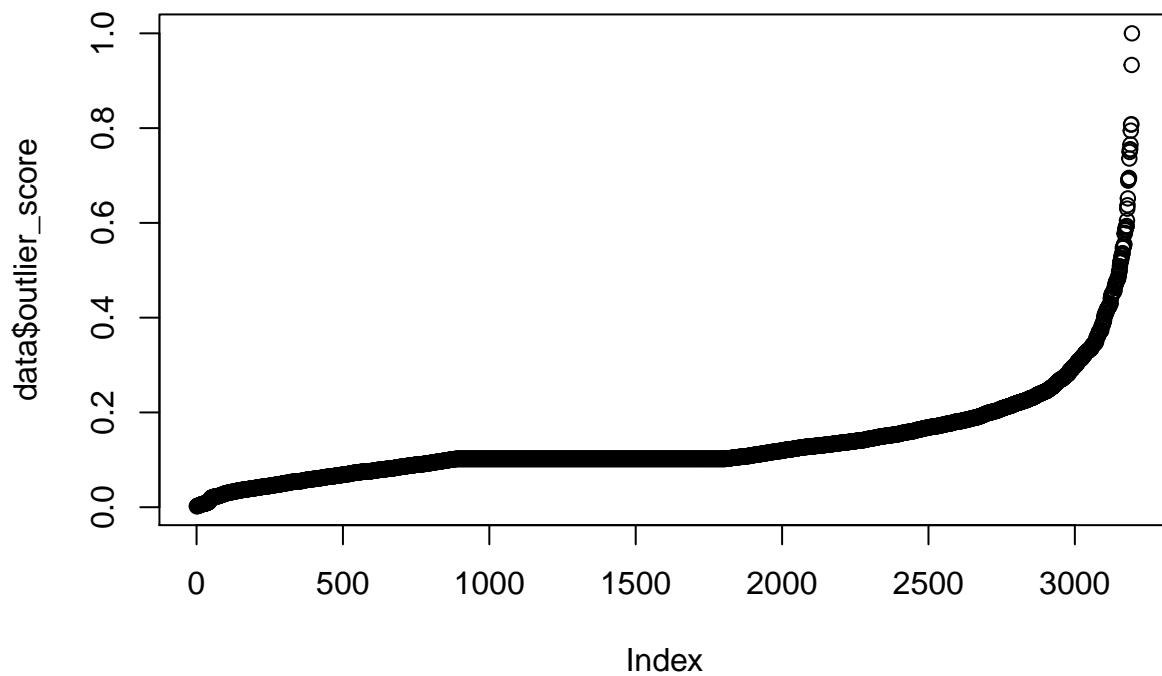
metric = "euclidean"
method = "kmeans"
dissimilarity_measure = "CBLOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

## `summarise()` ungrouping output (override with `groups` argument)
```

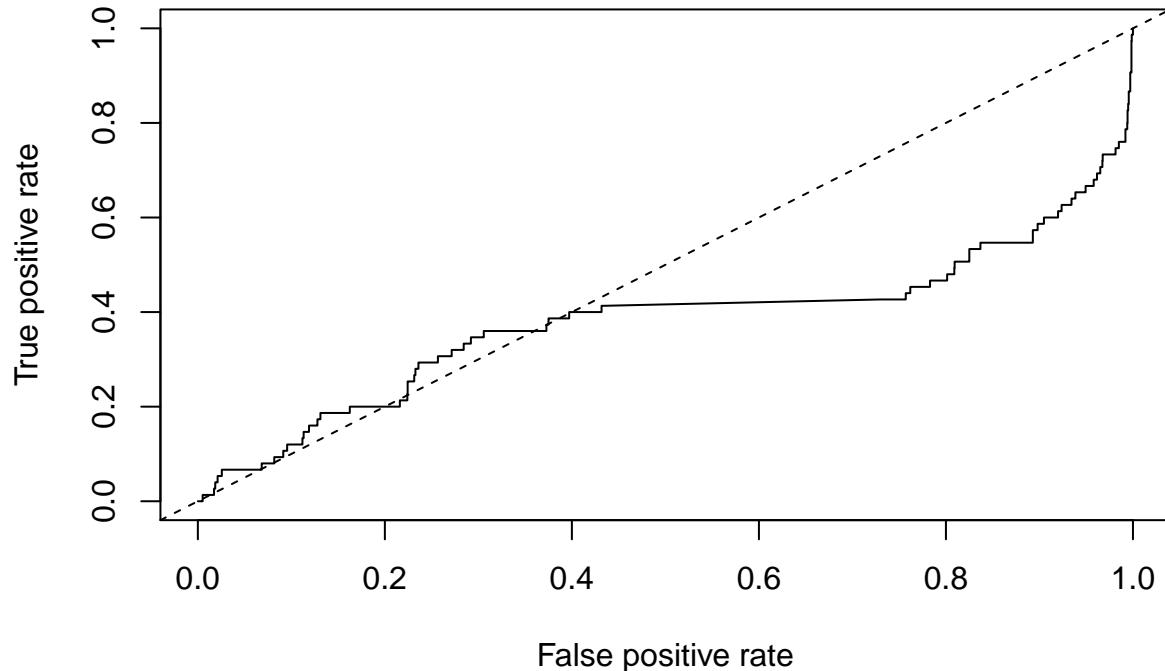
Cluster plot



dissimilarity distribution



ROC curve

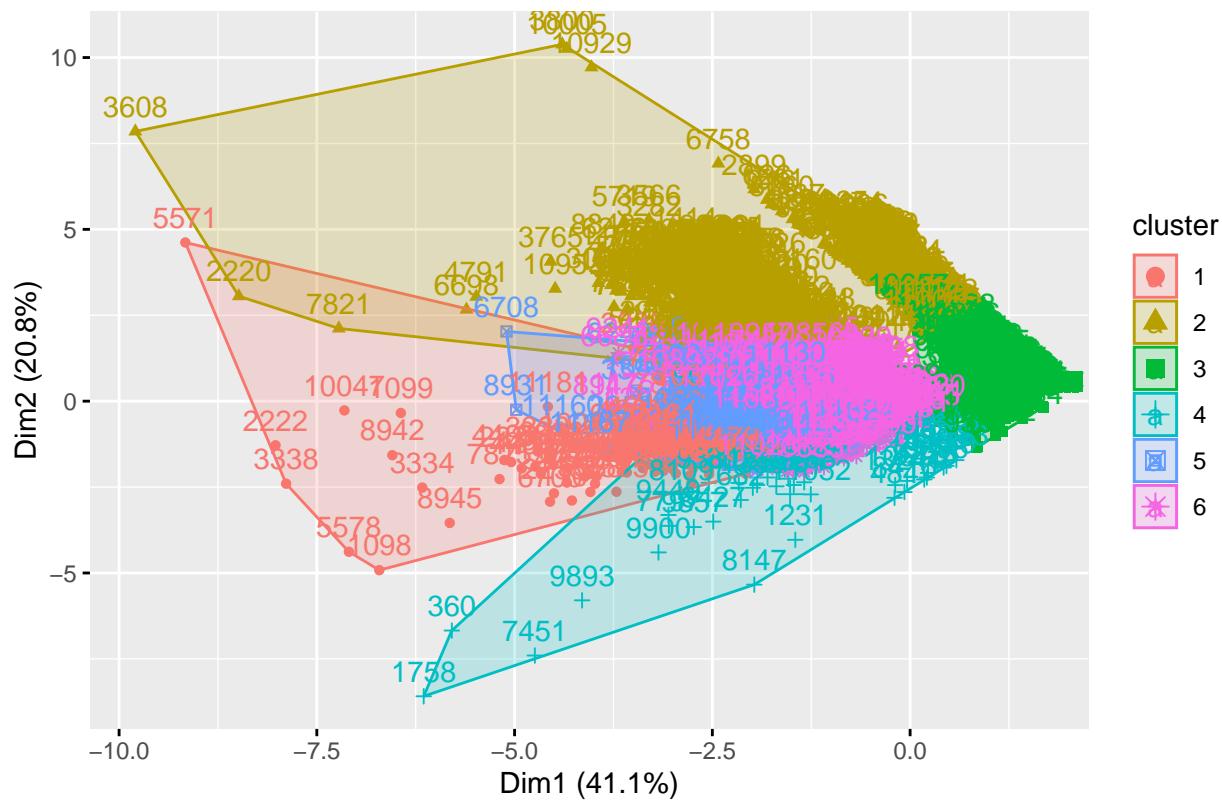


```
## [1] "F-value: 0.0374531835205993"
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

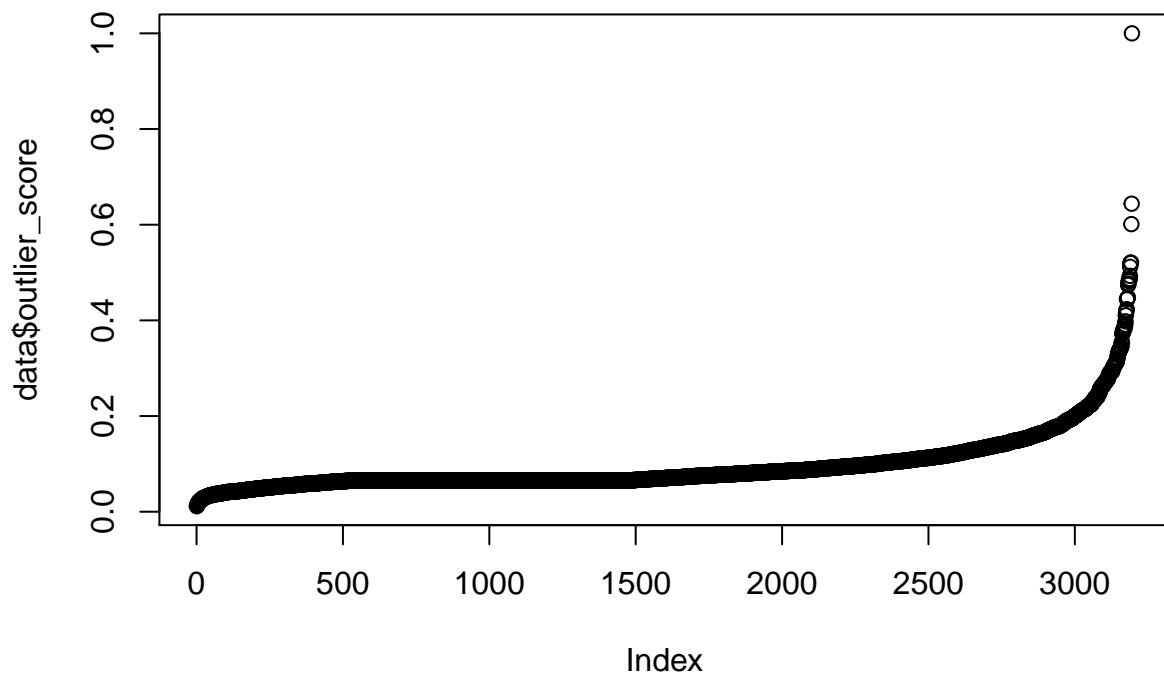
metric = "euclidean"
method = "kmeans"
dissimilarity_measure = "LDCOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.5
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

## `summarise()` ungrouping output (override with `.`groups` argument)
```

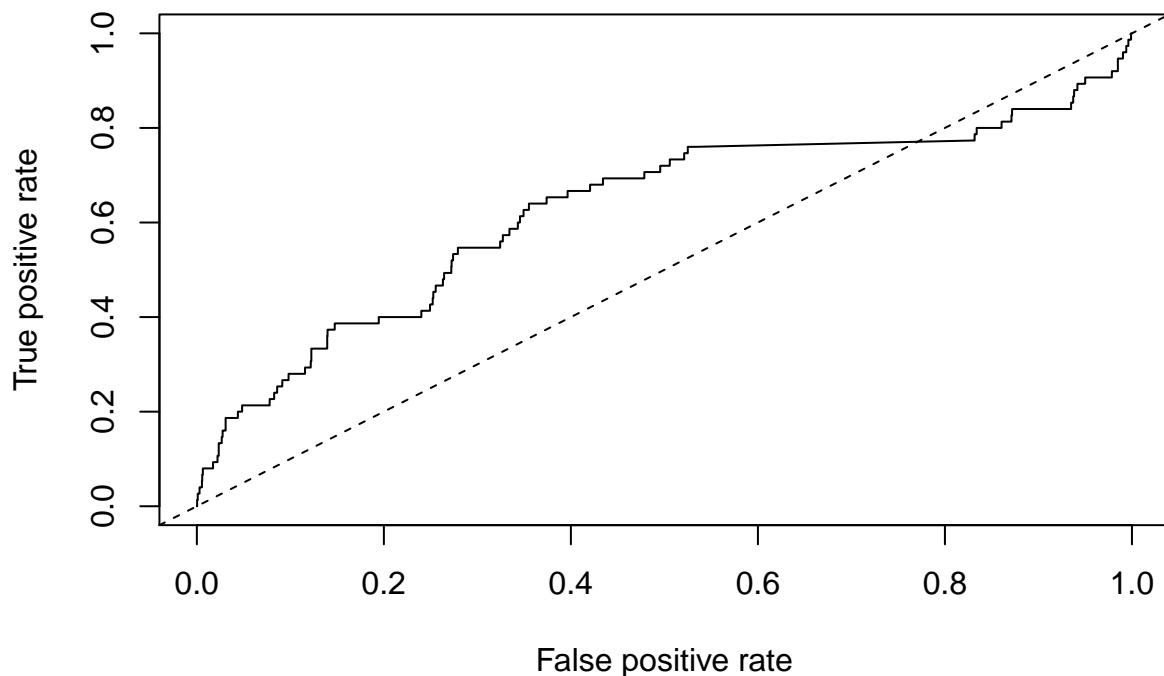
Cluster plot



dissimilarity distribution



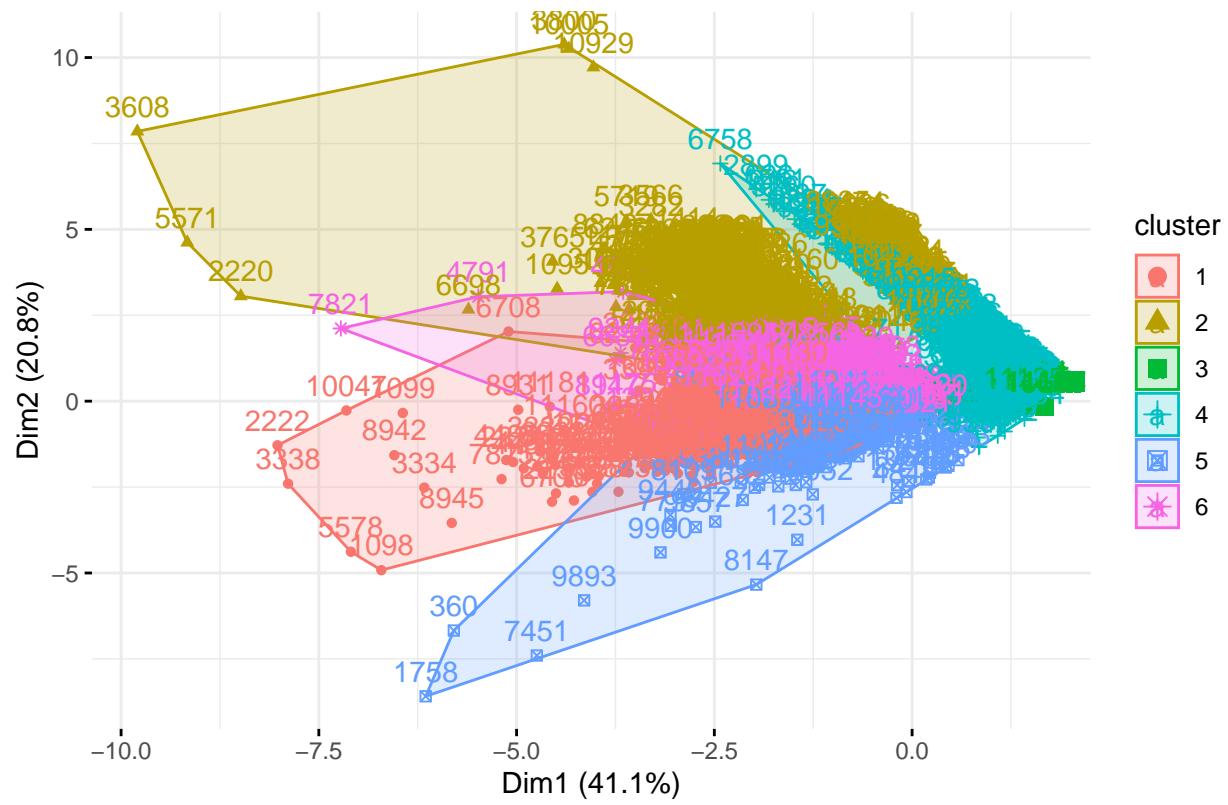
ROC curve



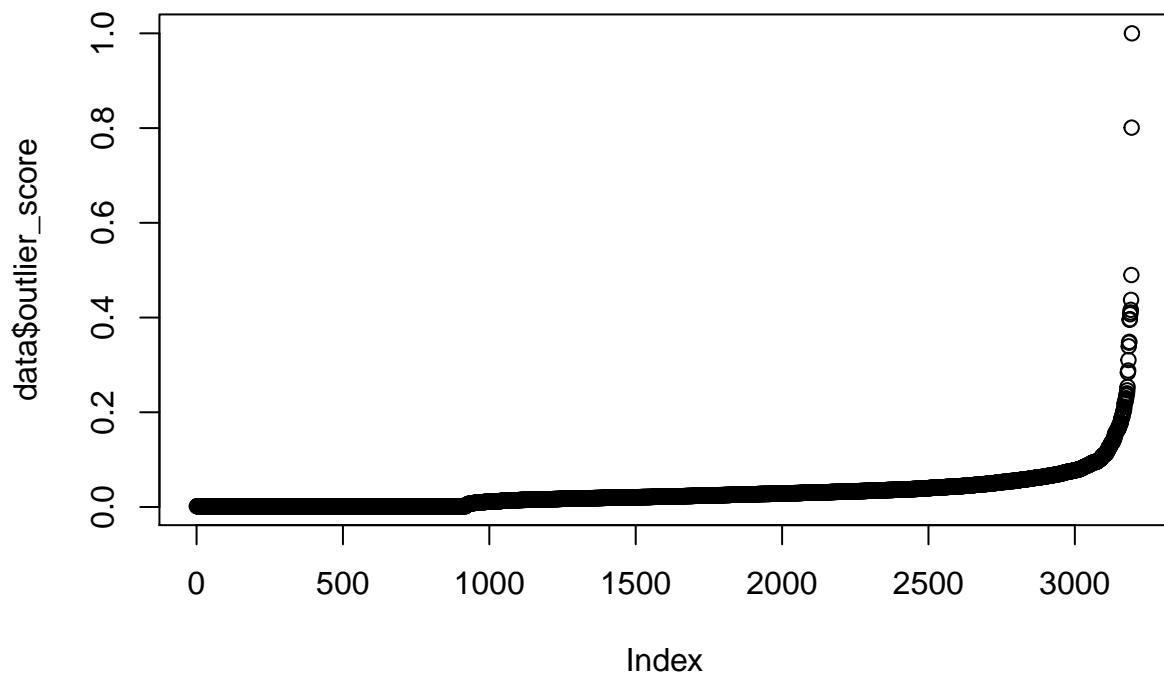
```
## [1] "F-value: 0.0487804878048781"
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "uCBLOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.8
source("Interface.R", local=knitr::knit_global(), echo=FALSE)
```

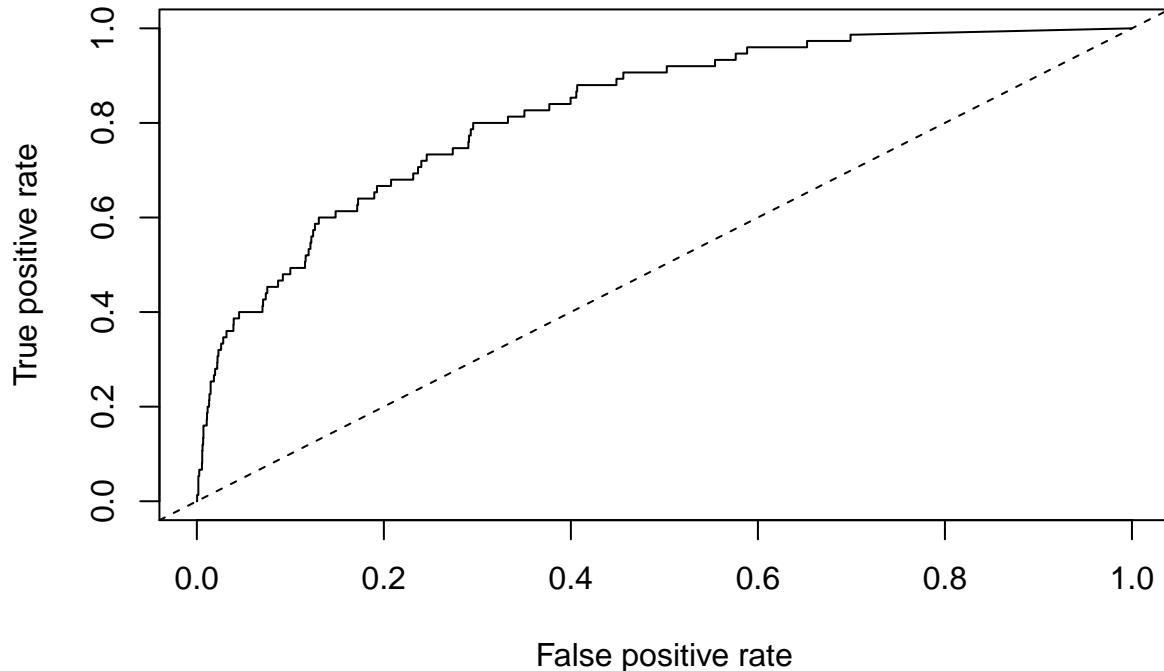
Cluster plot



dissimilarity distribution



ROC curve

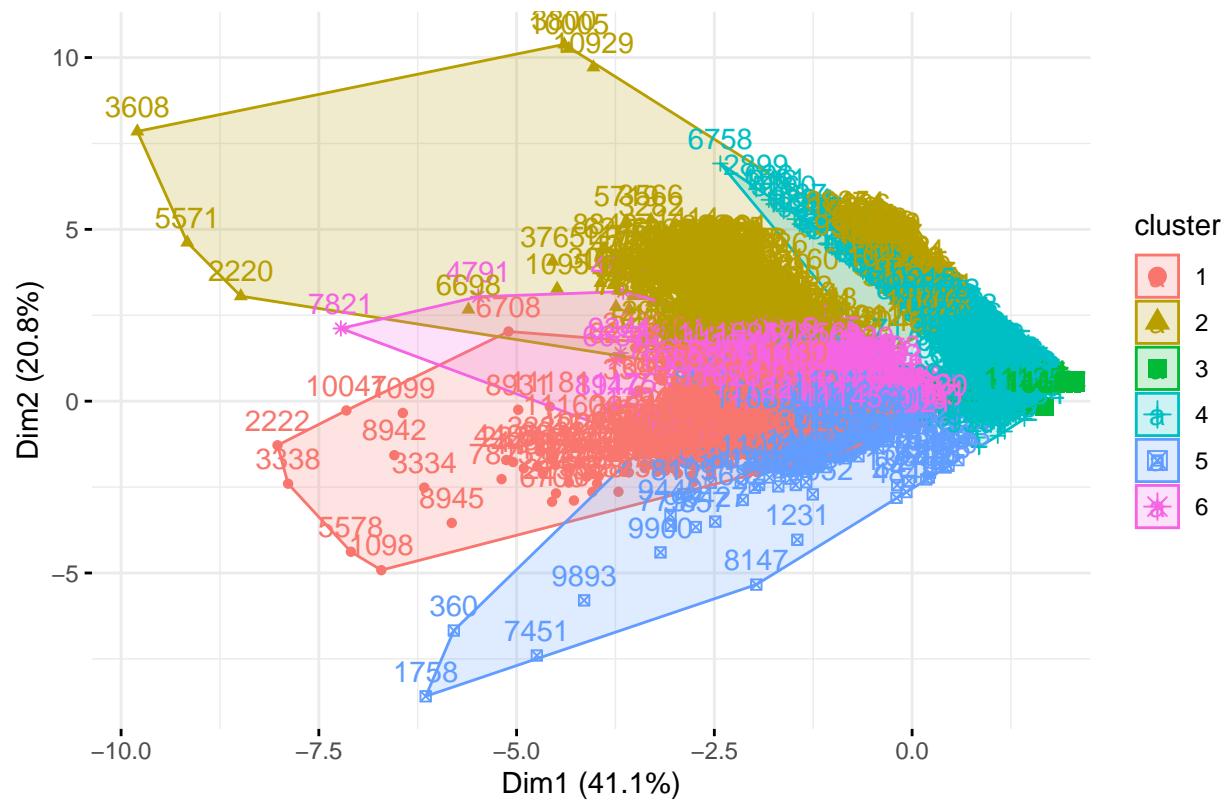


```
## [1] "F-value: 0.025974025974026"
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

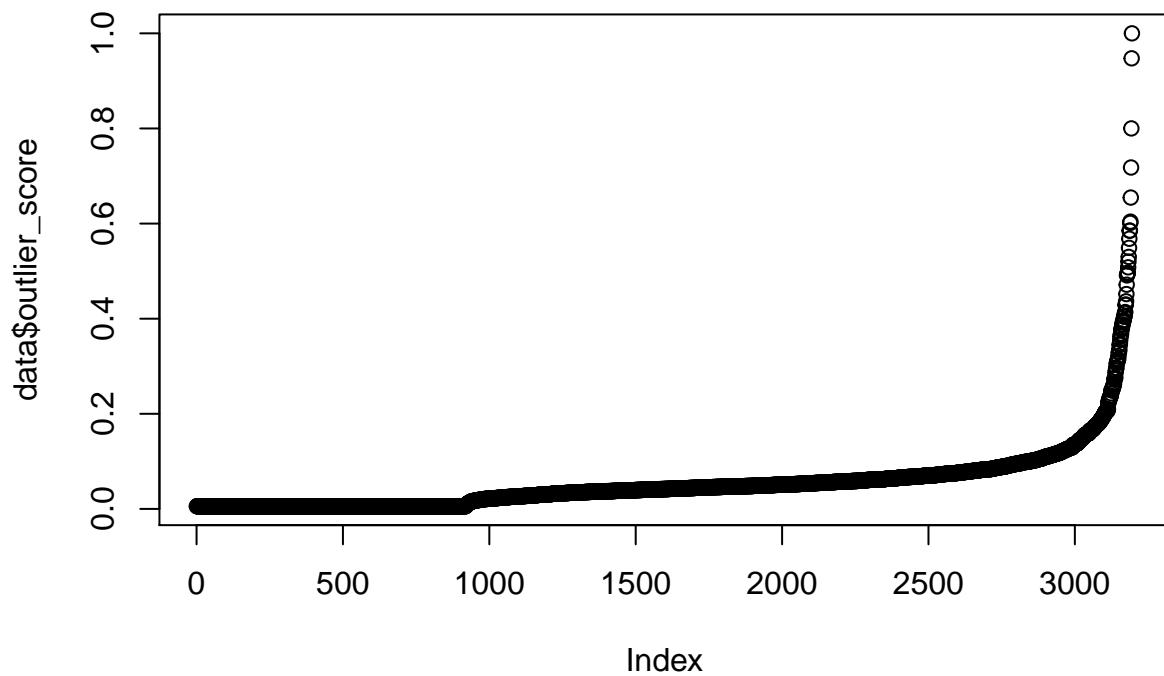
metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "CBLOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.8
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

## `summarise()` ungrouping output (override with `groups` argument)
```

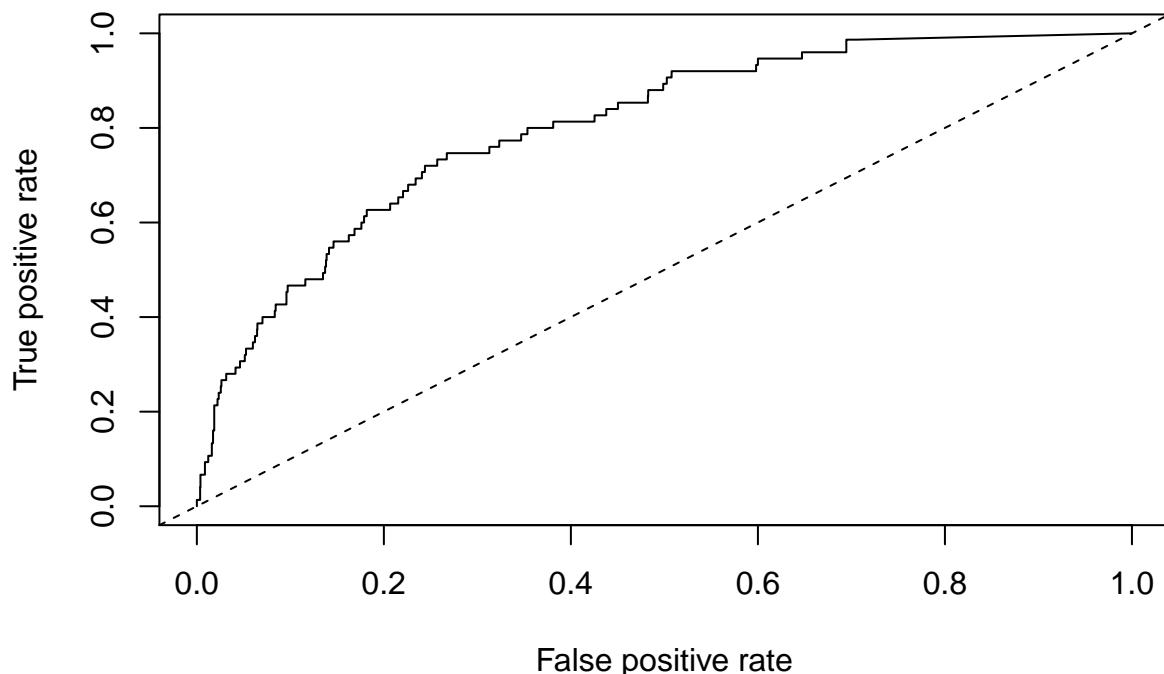
Cluster plot



dissimilarity distribution



ROC curve

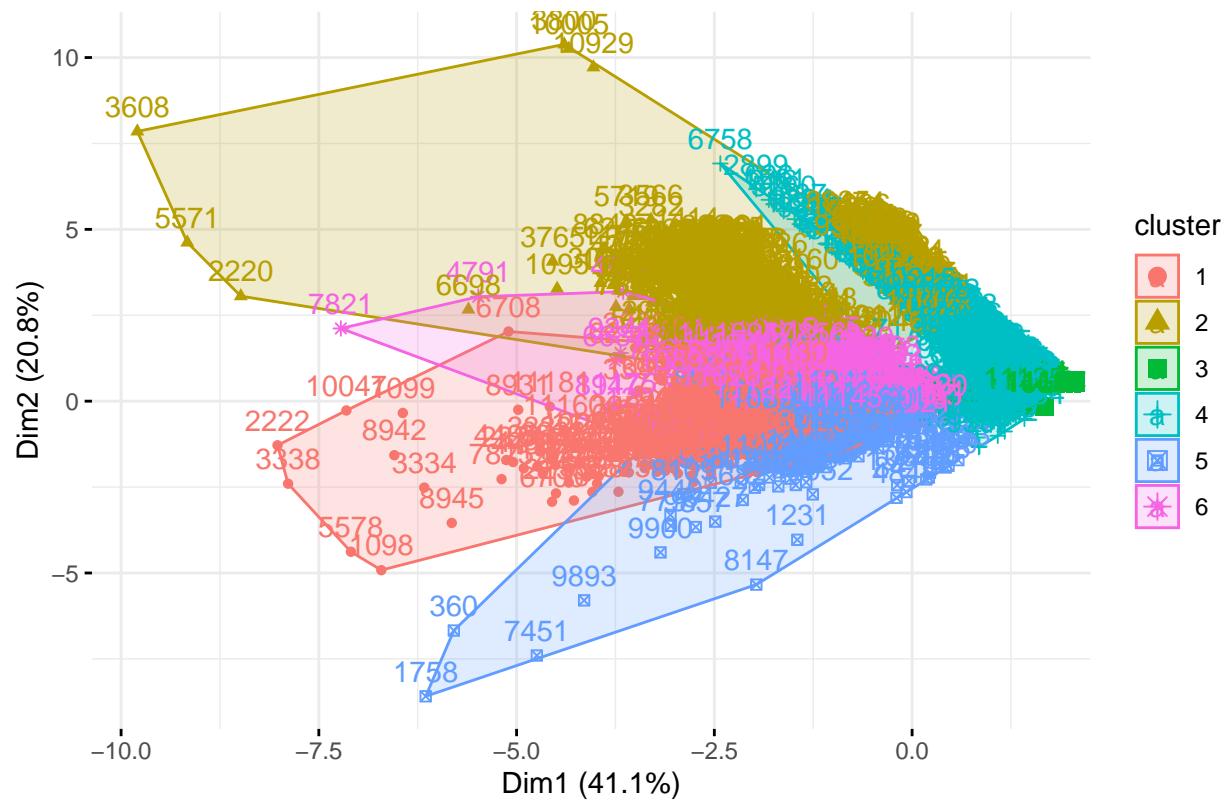


```
## [1] "F-value: 0.0256410256410256"
split_ratio = 0.8
set.seed(123)
split = sample.split(mammography, SplitRatio = split_ratio)
data_train = subset(mammography, split == TRUE)
data_test = subset(mammography, split == FALSE)

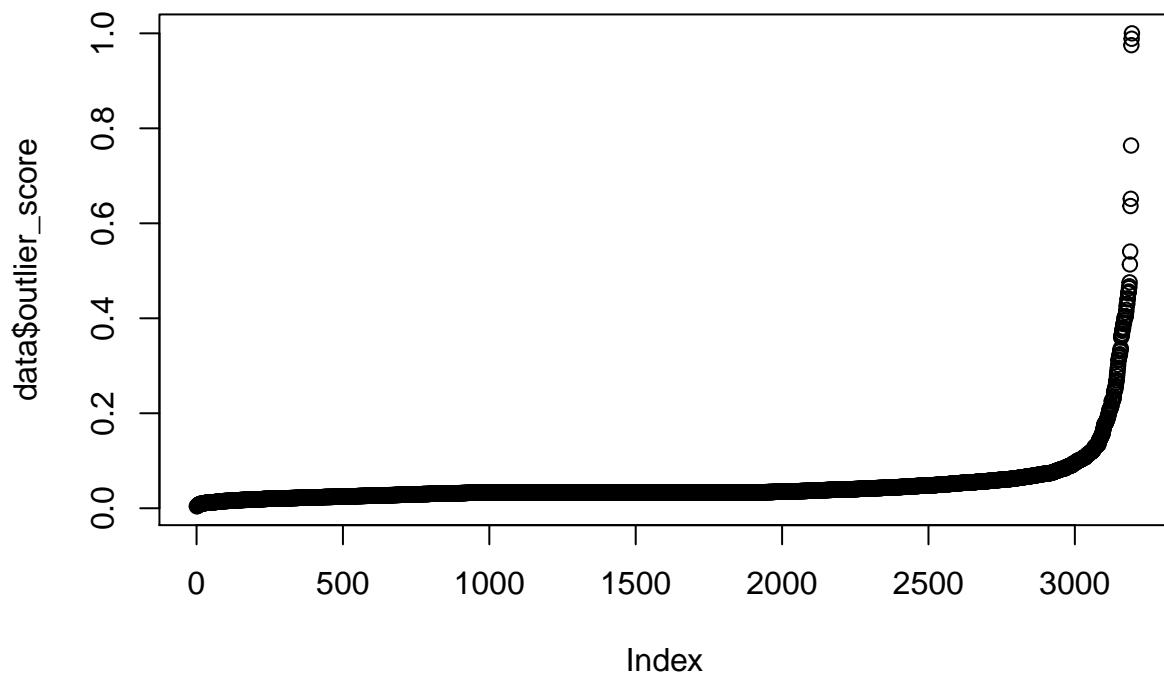
metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "LDCOF"
no_clusters = 6
algorithm = "Hartigan-Wong"
outlier_threshold = 0.8
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

## `summarise()` ungrouping output (override with `groups` argument)
```

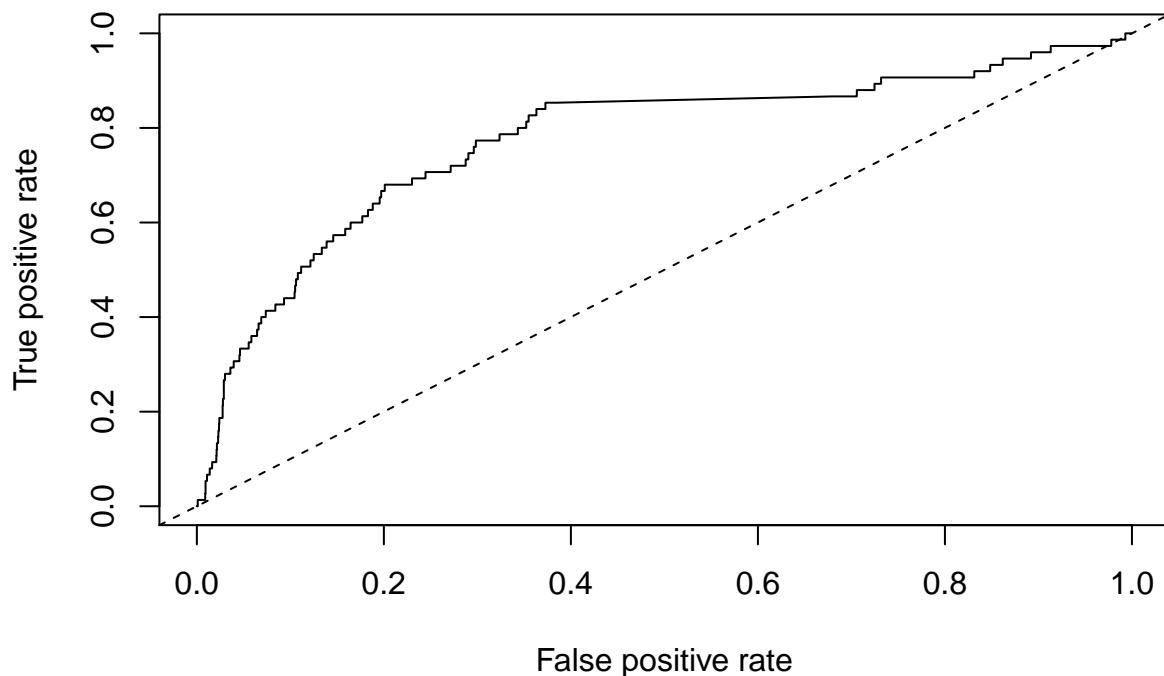
Cluster plot



dissimilarity distribution



ROC curve

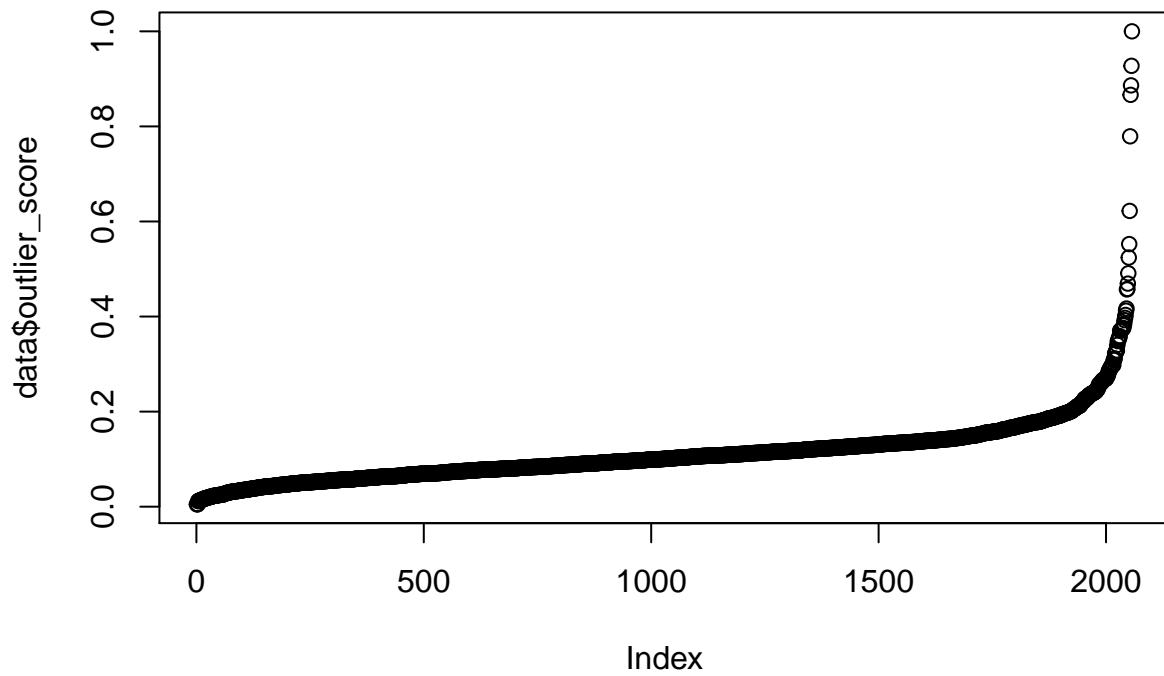


```
## [1] "F-value: NaN"  
split_ratio = 0.8  
set.seed(123)  
split = sample.split(annthyroid, SplitRatio = split_ratio)  
data_train = subset(annthyroid, split == TRUE)  
data_test = subset(annthyroid, split == FALSE)  
  
metric = "euclidean"  
method = "kmeans"  
dissimilarity_measure = "uCBLOF"  
no_clusters = 5  
algorithm = "Hartigan-Wong"  
outlier_threshold = 0.3  
source("Interface.R", local=knitr::knit_global(), echo=FALSE)
```

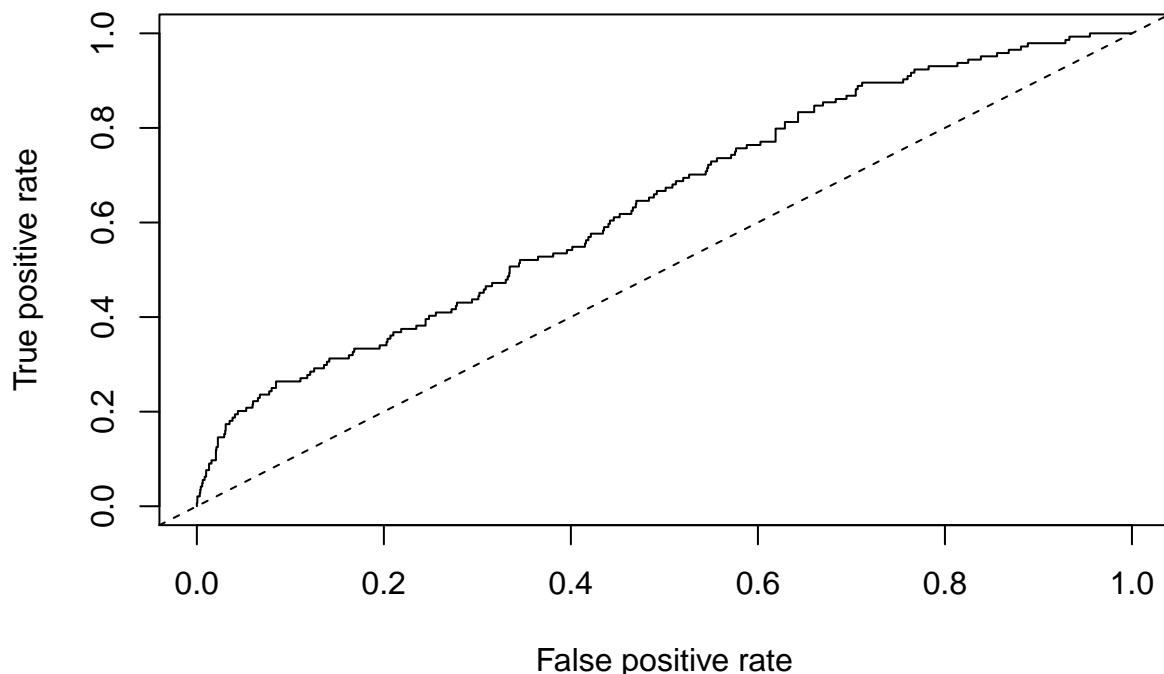
Cluster plot



dissimilarity distribution



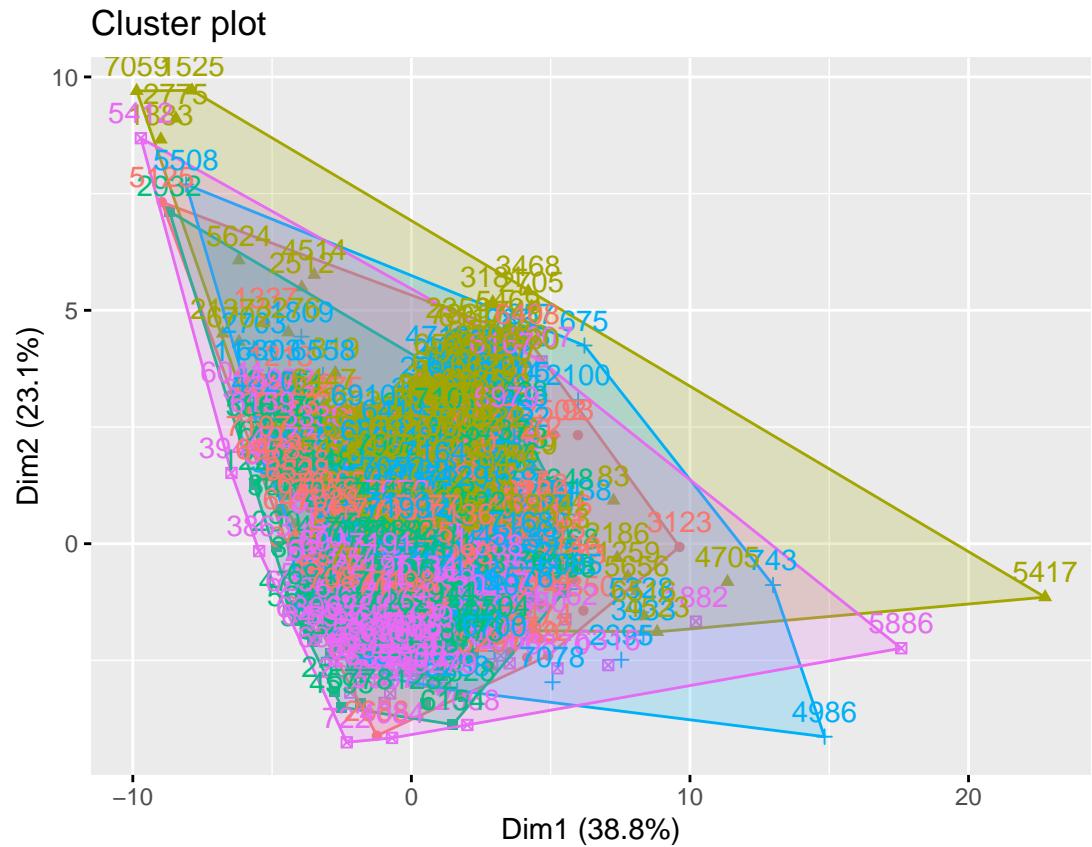
ROC curve



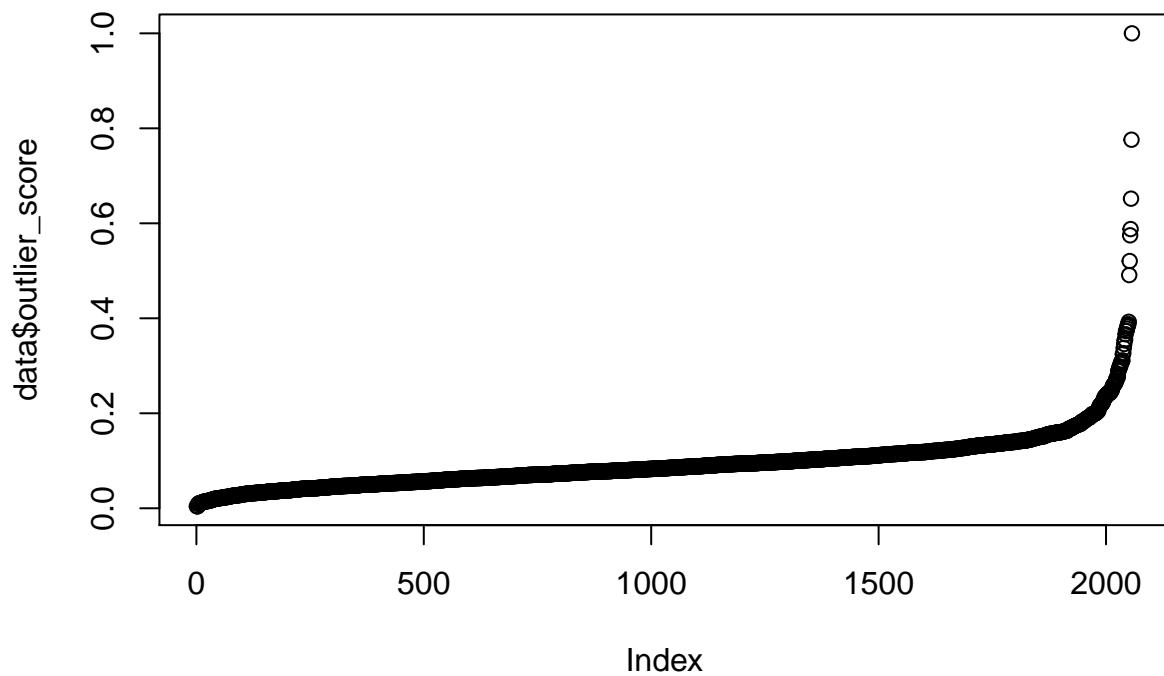
```
## [1] "F-value: 0.140540540540541"
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

metric = "euclidean"
method = "kmeans"
dissimilarity_measure = "CBLOF"
no_clusters = 5
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

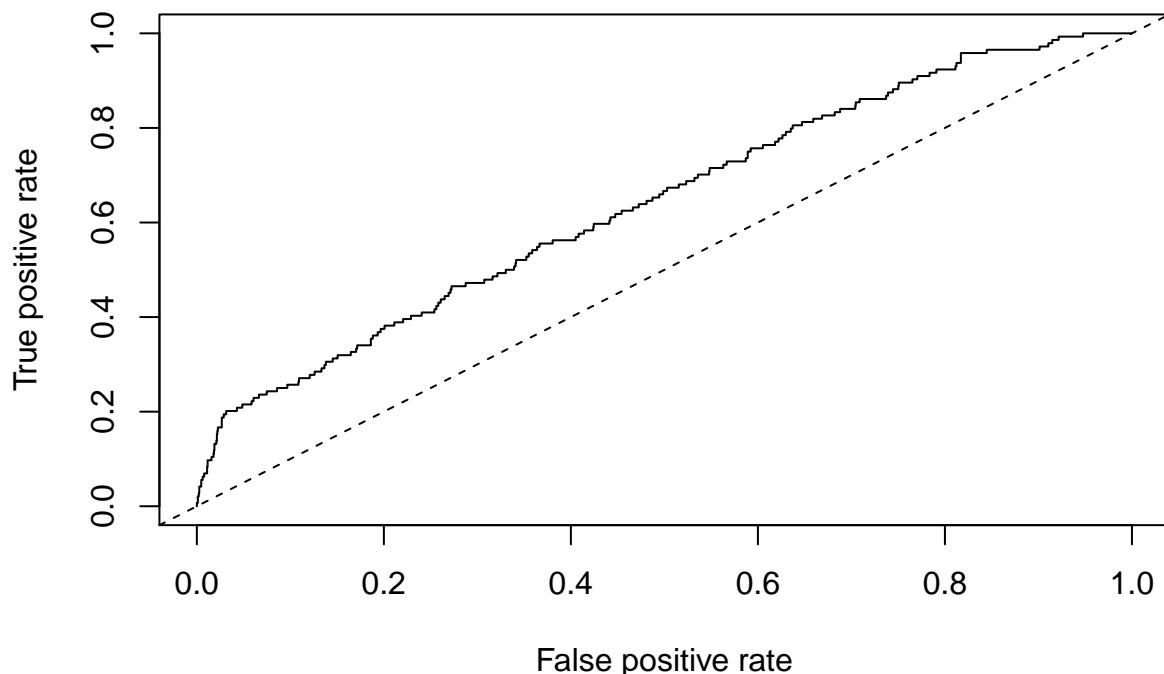
## `summarise()` ungrouping output (override with ` `.groups` argument)
```



dissimilarity distribution



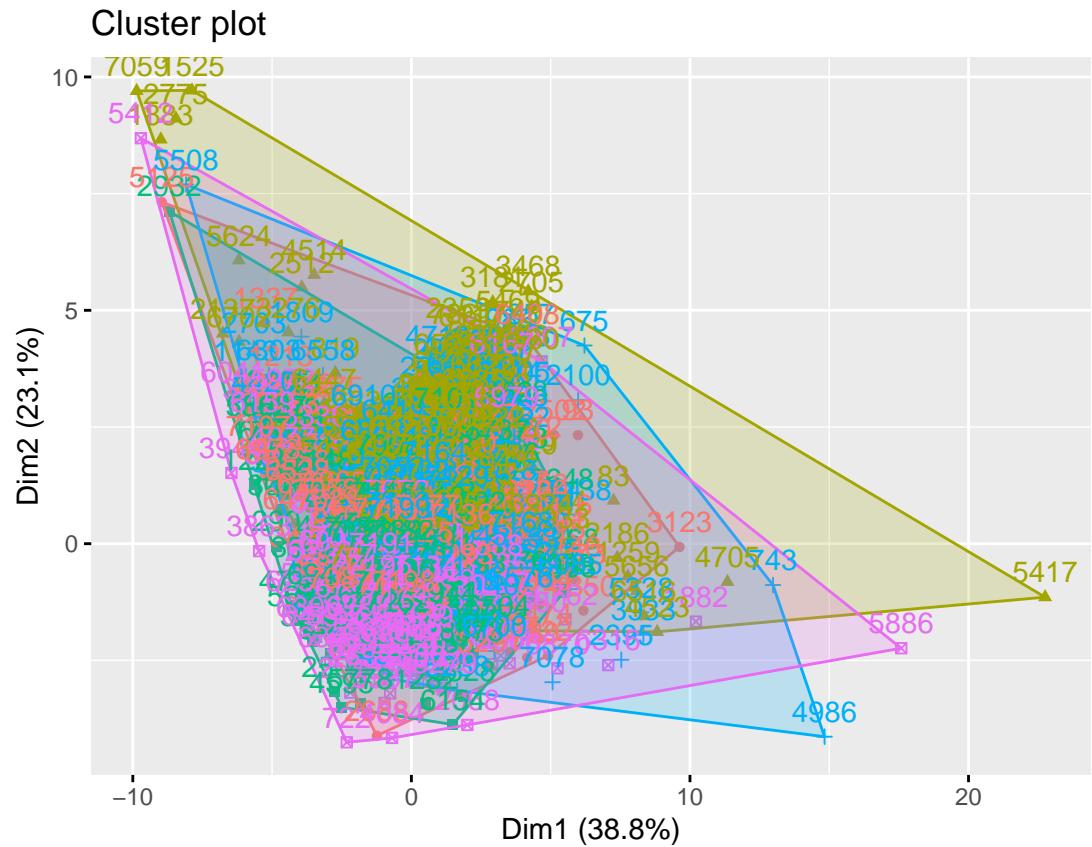
ROC curve



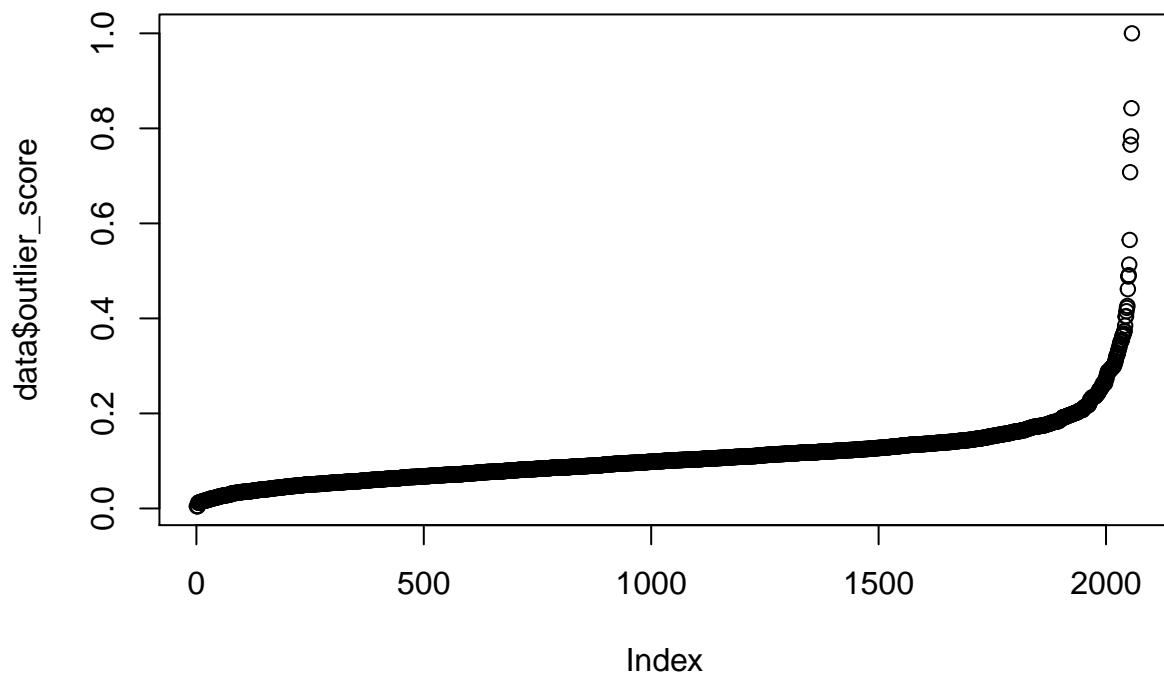
```
## [1] "F-value: 0.117647058823529"
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

metric = "euclidean"
method = "kmeans"
dissimilarity_measure = "LDCOF"
no_clusters = 5
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

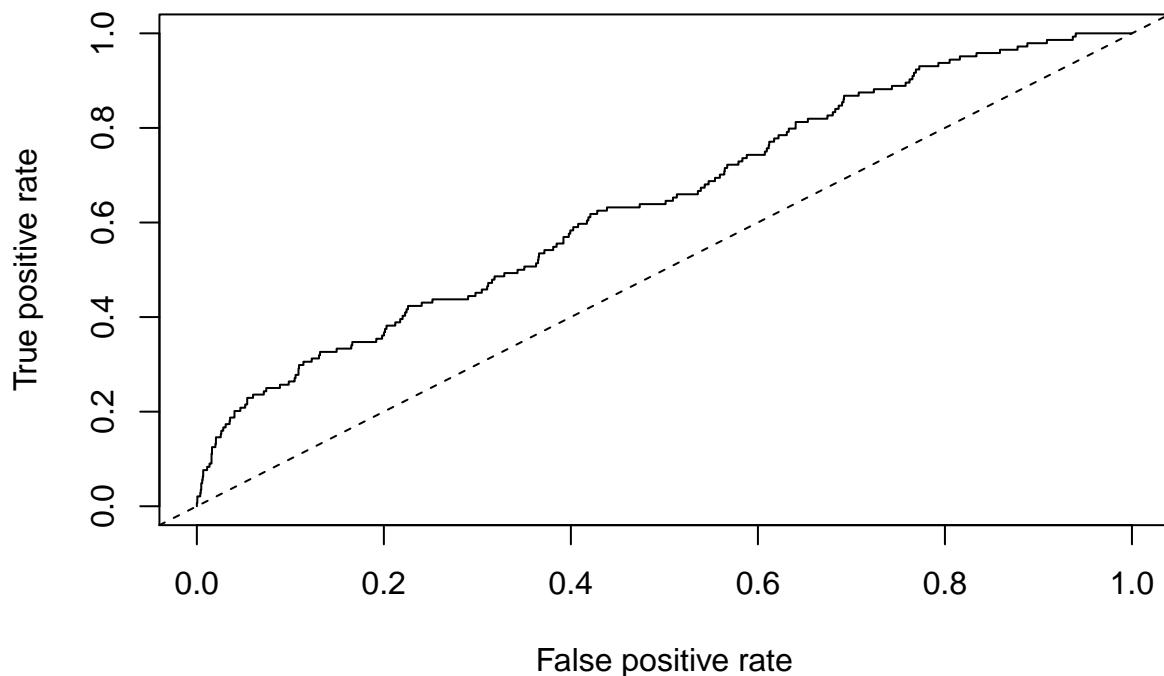
## `summarise()` ungrouping output (override with `groups` argument)
```



dissimilarity distribution

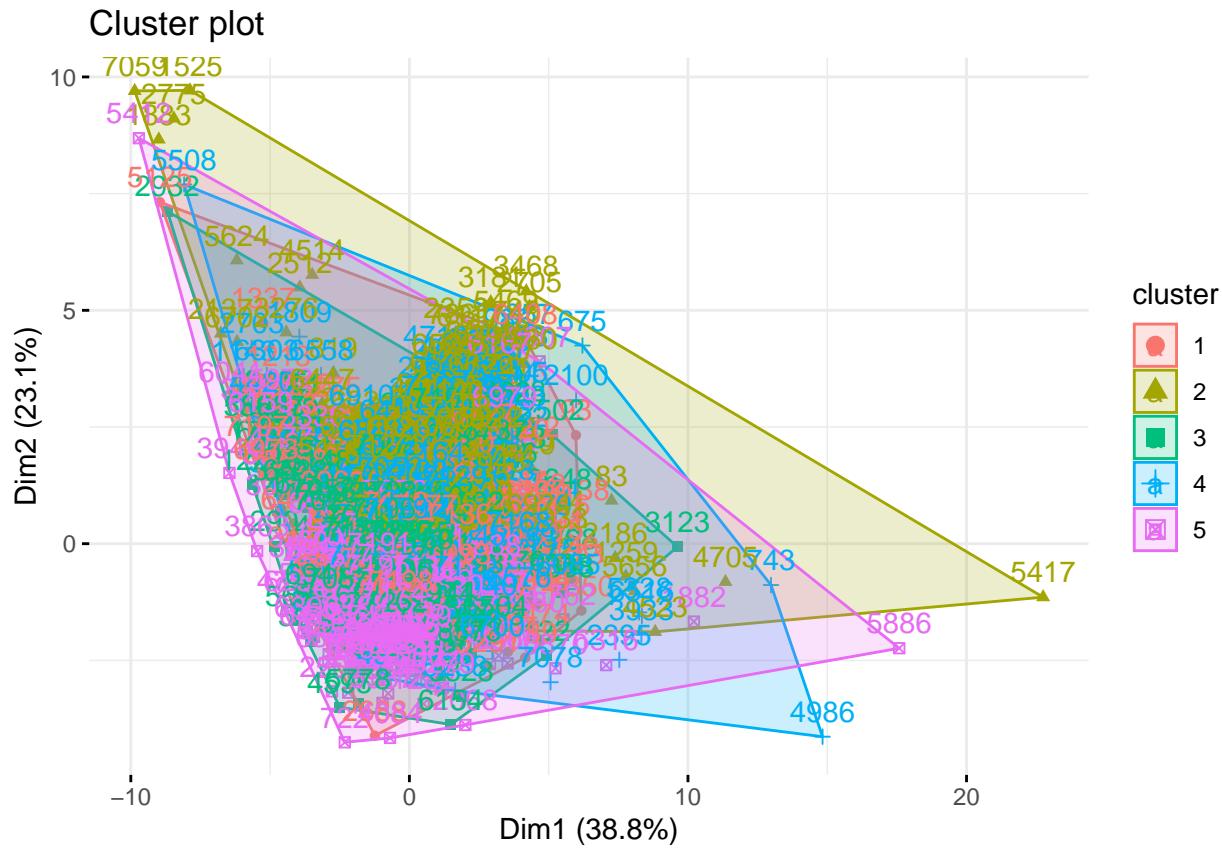


ROC curve

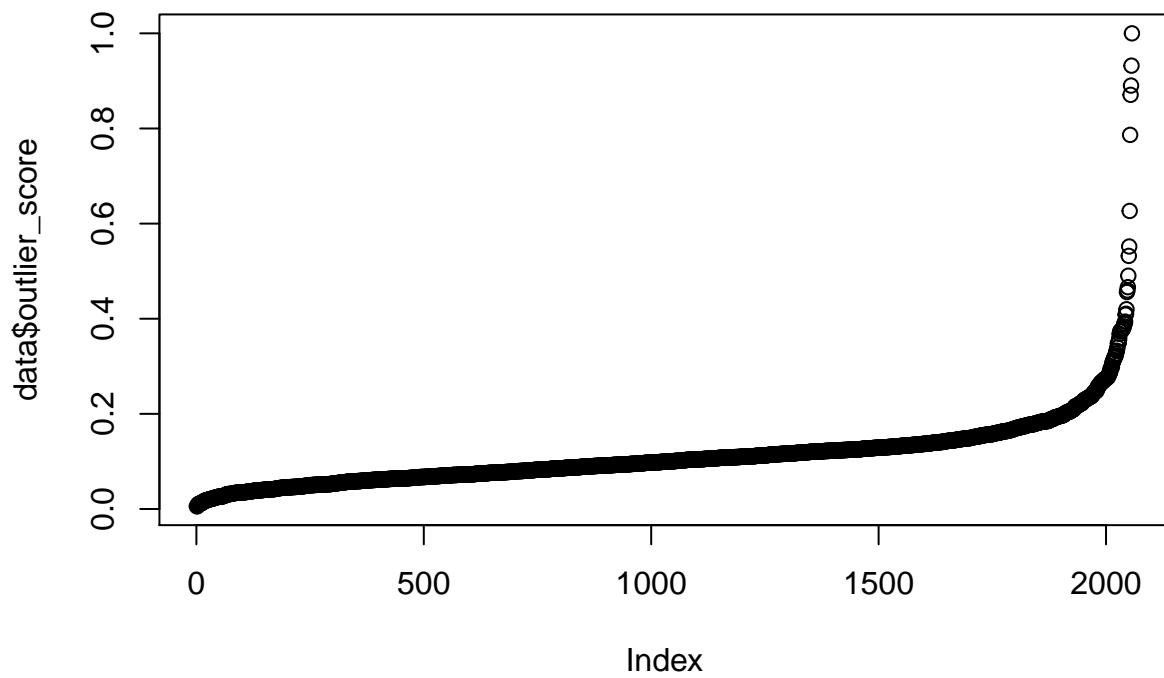


```
## [1] "F-value: 0.140540540540541"
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

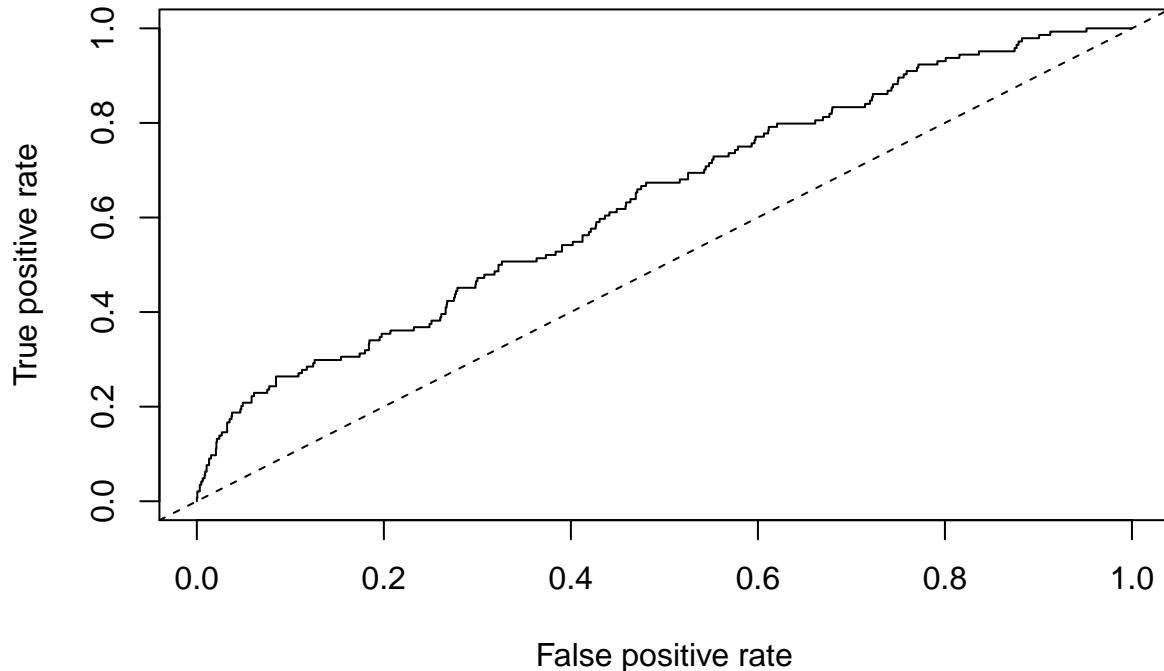
metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "uCBLOF"
no_clusters = 5
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)
```



dissimilarity distribution



ROC curve



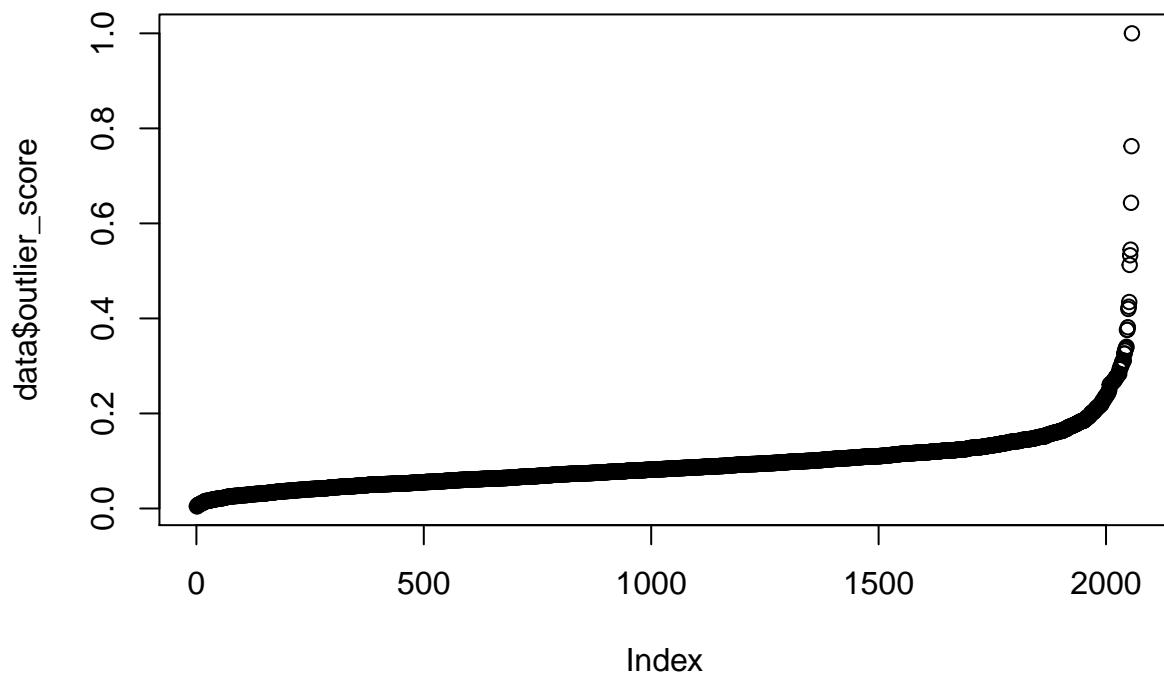
```
## [1] "F-value: 0.148936170212766"
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "CBLOF"
no_clusters = 5
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

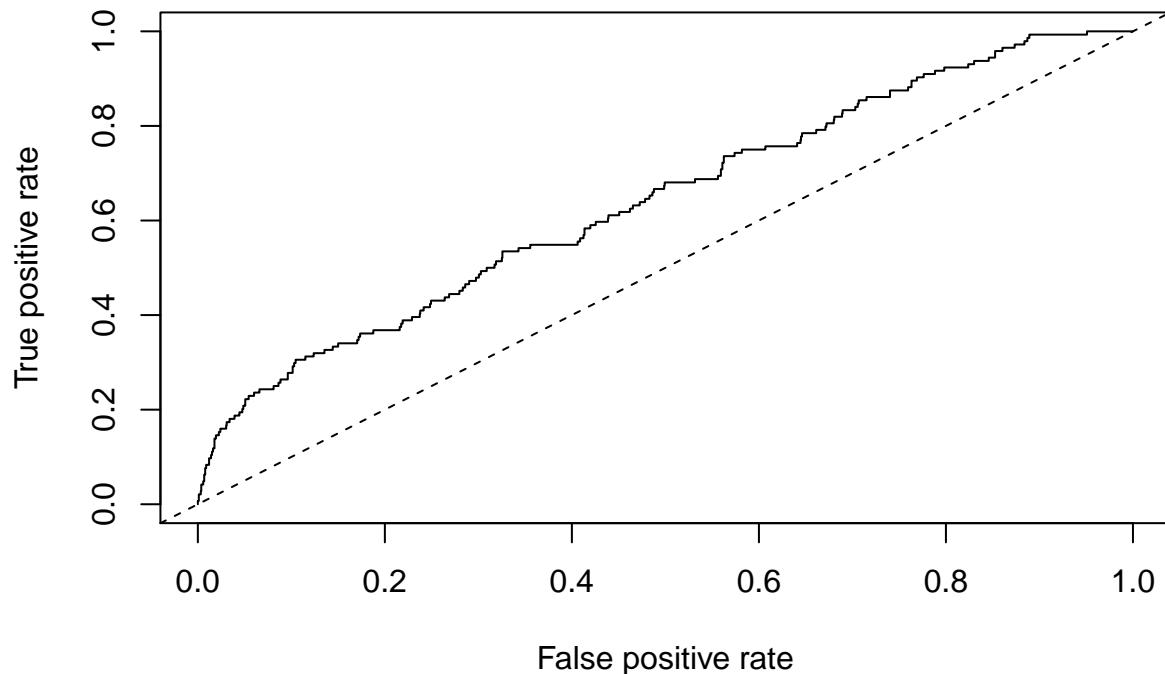
## `summarise()` ungrouping output (override with ` `.groups` argument)
```



dissimilarity distribution



ROC curve



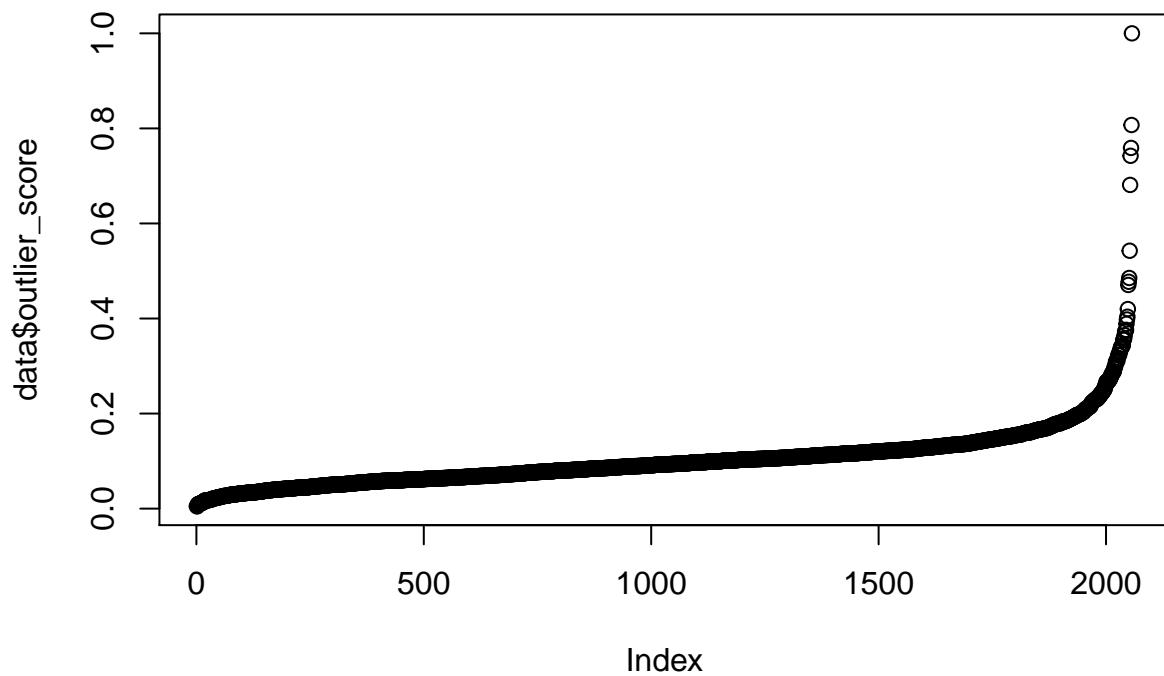
```
## [1] "F-value: 0.107142857142857"
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

metric = "euclidean"
method = "cmeans"
dissimilarity_measure = "LDCOF"
no_clusters = 5
algorithm = "Hartigan-Wong"
outlier_threshold = 0.3
source("Interface.R", local=knitr::knit_global(), echo=FALSE)

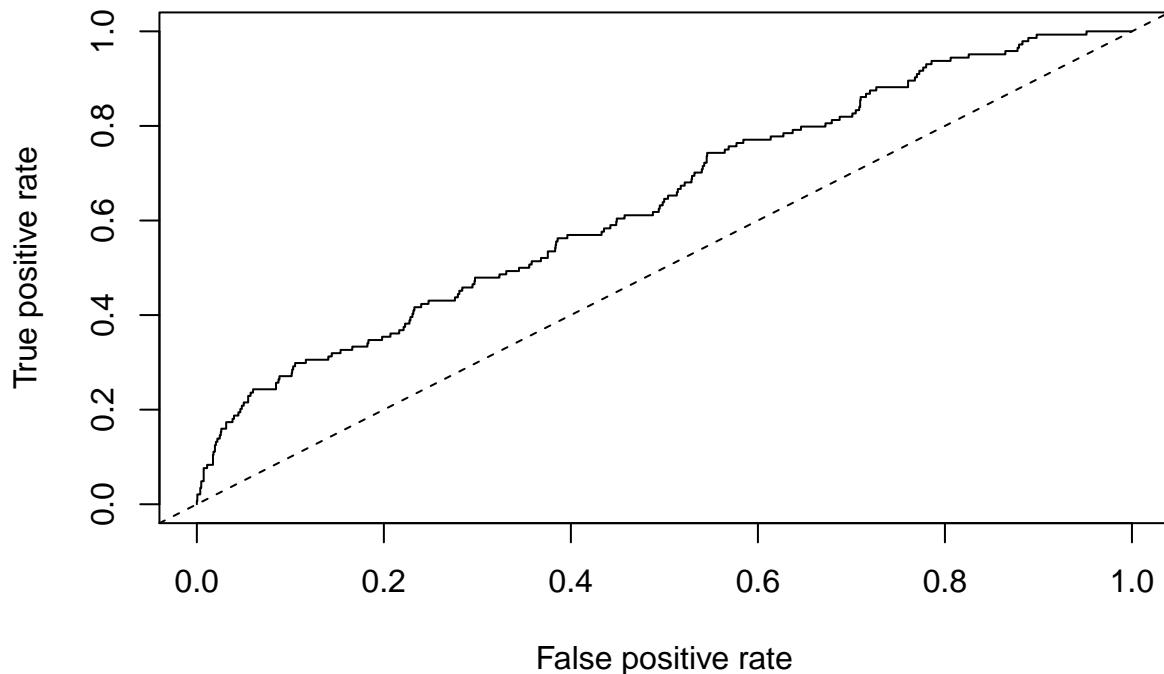
## `summarise()` ungrouping output (override with ` `.groups` argument)
```



dissimilarity distribution



ROC curve



```
## [1] "F-value: 0.132596685082873"
```

Wyniki detekcji anomalii metodami klasyfikacji Wynik klasyfikacji metodami svm, ctree i naiwnego klasyfikatora bayesowskiego.

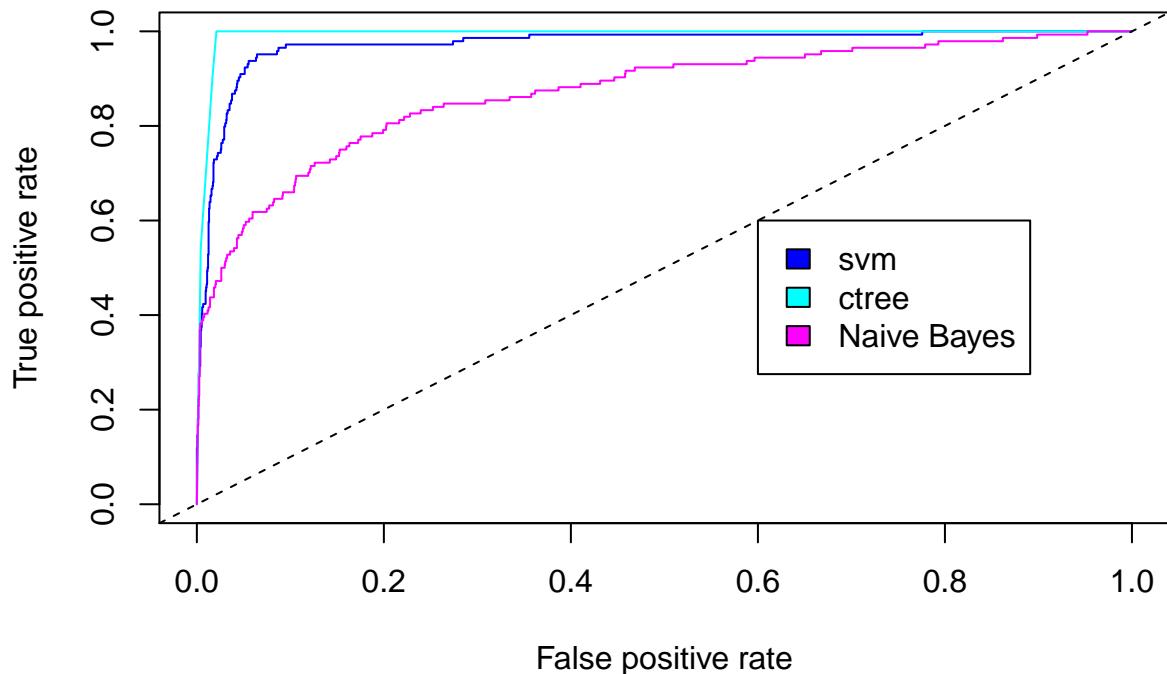
Wyniki są przedstawione w formie tabeli i wykresu. Tabela przedstawia wyniki pomiaru wskaźnika F dla poszczególnych algorytmów klasyfikacji. Wykres przedstawia wynik analizy ROC wybranych algorytmów. Miara F została określona na podstawie wyników klasyfikacji klas anomalii dla danych testowych.

```
split_ratio = 0.8
set.seed(123)
split = sample.split(annthyroid, SplitRatio = split_ratio)
data_train = subset(annthyroid, split == TRUE)
data_test = subset(annthyroid, split == FALSE)

source("Classification.R", local=knitr::knit_global(), echo=FALSE)
```

```
## # A tibble: 1 x 3
##       svm     nvb      ct
##   <dbl> <dbl> <dbl>
## 1 0.443  0.533  0.878
```

ROC curves of different machine learning classifier



Wnioski Niestety, wyniki detekcji anomalii z użyciem algorytmów grupowania na tych zbiorach okazał się niezbyt owocny. Widzimy niewielką zmianę jakości modelu przy zmianie algorytmów grupowania, jak i różnych miarach niepodobieństwa. Metody klasyfikacji okazały się dużo skuteczniejsze w przypadku tych zbiorów danych. W przebiegu badania, zorientowaliśmy się że przy tak przyjętej normalizacji zawsze jeden wynik będzie określony jako anomalia. Być może przyjęcie innego typu normalizacji dawałoby bardziej miarodajne wyniki. Zauważaliśmy również, że przyjęty sposób przedstawienia danych też nie był najlepszym wyborem, ponieważ można było zebrać więcej danych na jednym wykresie. Porównując miary F dla grupowania i klasyfikacji, widzimy znaczną przewagę algorytmów klasyfikacji, chociaż niestety jest ona określona dla danej wartości odcięcia, a tej nie dobieraliśmy na podstawie optymalizacji F-miary, więc może być ona lekko zniżona.

Użyte skrypty

```
data_train_x = data_train %>% select(1:6)
data_train_y = data_train %>% select(7)
data_test_x = data_test %>% select(1:6)
data_test_y = data_test %>% select(7)

svm.model = svm(as.factor(class) ~ ., data = data_train, type = 'C-classification', kernel = 'polynomial')
svm.pred = predict(svm.model, newdata = data_test_x)
svm.cm = table(data_test_y$class, svm.pred)

nvb.model = naiveBayes(as.factor(class) ~ ., data = data_train)
nvb.pred = predict(nvb.model, newdata = data_test_x)
```

```

nvb.cm = table(data_test_y$class, nvb.pred)

ct.model = ctree(as.factor(class) ~ ., data = data_train)
ct.pred = predict(ct.model, newdata = data_test_x)
ct.cm = table(data_test_y$class, ct.pred)

print(tibble(svm = f_value(svm.cm), nvb = f_value(nvb.cm), ct = f_value(ct.cm)))

svm.prob = predict(svm.model, type="prob", newdata= data_test_x, probability = TRUE)
svm.rocr = prediction(attr(svm.prob, "probabilities")[,2], data_test_y$class)
svm.perf = performance(svm.rocr, "tpr","fpr")
nvb.prob = predict(nvb.model, newdata=data_test_x, type="raw")
nvb.rocr = prediction(nvb.prob[,2], data_test_y$class)
nvb.perf = performance(nvb.rocr, "tpr","fpr")
ct.prob = predict(ct.model, newdata=data_test_x, type="prob")
ct.rocr = prediction(ct.prob[,2], data_test_y$class)
ct.perf = performance(ct.rocr, "tpr","fpr")
plot(svm.perf, col=4, main="ROC curves of different machine learning classifier")
legend(0.6, 0.6, c('svm', 'ctree', 'Naive Bayes'), 4:6)
plot(ct.perf, col=5, add=TRUE)
plot(nvb.perf, col=6, add=TRUE)
abline(0,1,col="#000000", lty=2)

```

Skrypt klasyfikacji

```

require(tidyverse)
library(rhdf5)
library(factoextra)
library(NbClust)
library(e1071)
library(dbSCAN)
library(mclust)
library(DDoutlier)
require(ROCR)

#data_test & data_train - subsets of data, where data_test + data_train = data && data_test inner join data_train
#data_train = subset of data
#data_test = subset of data

#Example parameters

#metric = "euclidean"
#method = "kmeans"
#dissimilarity_measure = "CBLOF"
#outlier_threshold = 0.8
#algorithm = "Hartigan-Wong"
#no_clusters = 5

#Dissimilarity measures:
#CBLOF : (odległość od najbliższego centroidu) * (rozmiar grupy tego centroidu)
#uCBLOF: (odległość od najbliższego centroidu)
#LDCOF: (odległość od najbliższego centroidu) / (średnie odległości od tego centroidu)

```

```

# dissimilarity measures should later include LOF, CBLOF, uCBLOF, LDCOF
# allowed metrics depends on used method

#args for kmeans: 1 - number_of_clusters, 2 - kmeans_algorithm
#args for dbscan: 1 - scan_distance, 2 - fraction_of_outliers

f_value <- function (cm) {
  fp = cm[1,2]
  fn = cm[2,1]
  tp = cm[2,2]
  recall = tp / (tp+fn)
  precision = tp / (tp+fp)
  f = (2 * recall * precision) / (recall + precision)
  return(f)
}

predict.kmeans <- function(object, newdata, metric){
  centers <- object$centers
  n_centers <- nrow(centers)
  dist_mat <- as.matrix(dist(rbind(centers, newdata), method=metric))
  dist_mat <- dist_mat[-seq(n_centers), seq(n_centers)]
  max.col(~dist_mat)
}

normalize_outlier_score <- function (data) {
  data$outlier_score / max(data$outlier_score)
}

data_train_x = data_train %>% select(1:6)
data_train_y = data_train %>% select(7)
data_test_x = data_test %>% select(1:6)
data_test_y = data_test %>% select(7)

show_stats <- function(data){
  #prepare data
  data$outlier_score = normalize_outlier_score(data)
  data = data %>% mutate(pred = outlier_score)
  data = data %>% mutate(pred_int = as.integer(outlier_score > outlier_threshold))
  data = data %>% arrange(outlier_score)

  #dissimilarity measure distribution
  plot(data$outlier_score, main="dissimilarity distribution")

  #roc curve
  pred = prediction(predictions=data$pred, labels=data$class)
  perf = performance(pred, "tpr", "fpr")
  plot(perf, main="ROC curve")
  abline(0,1,col="#000000", lty=2)

  #f1-measure
  data = data %>% mutate(tp = class & pred_int)
}

```

```

data = data %>% mutate(fp = !class & pred_int)
data = data %>% mutate(tn = !class & !pred_int)
data = data %>% mutate(fn = class & !pred_int)
recall = tally(data, tp) / (tally(data, tp) + tally(data, fn))
precision = tally(data, tp) / (tally(data, tp) + tally(data, fp))
f_value = 2 * recall * precision / (recall + precision)
print(paste("F-value:", f_value))
}

if(method == "kmeans") {

  cluster_data = kmeans(data_train_x, no_clusters, algorithm=algorithm)
  plot(fviz_cluster(cluster_data, data = data_train_x))
  groups = predict.kmeans(cluster_data, data_test_x, metric)
  data = data_test %>% mutate(group = groups)

} else if(method == "cmeans") {

  cluster_data = cmeans(data_train_x, no_clusters, dist=metric)
  plot(fviz_cluster(list(data = data_train_x, cluster=cluster_data$cluster), ggtheme = theme_minimal()))
  data = data_test
  centers = cluster_data$centers
  data$group = cmeans(data_test_x, centers, dist=metric)$cluster
}

cluster_centers = cluster_data$centers

if(dissimilarity_measure == "uCBLOF") {

  data = data %>% mutate(outlier_score = sqrt((V1-cluster_centers[group, 1])^2 + (V2-cluster_centers[group, 2])^2))

} else if(dissimilarity_measure == "LDCOF") {

  data = data %>% mutate(dist = sqrt((V1-cluster_centers[group, 1])^2 + (V2-cluster_centers[group, 2])^2))
  average_dist = data %>% group_by(group) %>% summarise(avg_dist = mean(dist))
  data = data %>% mutate(outlier_score = dist / average_dist[group,2]$avg_dist)

} else if(dissimilarity_measure == "CBLOF") {

  data = data %>% mutate(dist = sqrt((V1-cluster_centers[group, 1])^2 + (V2-cluster_centers[group, 2])^2))
  group_count = data %>% group_by(group) %>% summarise(count = n())
  data = data %>% mutate(outlier_score = dist * group_count[group,2]$count)

}
show_stats(data)

```

Skrypt grupowania