# Wissensentdeckung in Datenbanken SoSe 2018

Übungsblatt 04

*Alexander Kojen, Robin Czarnetzki, Jonas Kauke*

## Aufgabe 1

### a)

```
data(Ionosphere)
set.seed(1273)

#Split data 80% train / 20% test: https://stackoverflow.com/questions/17200114/how-to-split-data-into-t

sample <- sample.int(n = nrow(Ionosphere), size = floor(.8*nrow(Ionosphere)), replace = F)
train <- Ionosphere[sample, ]
test  <- Ionosphere[-sample, ]

task = makeClassifTask(data = train, target = "Class")
```

### b)

```
learner = makeLearner("classif.naiveBayes")
model = train(learner, task)
```

### c)

```
prediction = as.data.frame(predict(model, newdata = test))

#Ausgabe ueber performance Funktion (mit Daten der Klasse Prediction, mmce: mean misclassification erro
performance(predict(model, newdata = test), measures = list(mmce, acc))
```

```
##      mmce       acc
## 0.1408451 0.8591549
```

Interpretation: Mit fast 86% trifft die vorhergesagte Klasse mit der wahren Klasse überein. Daher ist das Ergebniss zufriedenstellend.

jns: Aufgabe 1: 5/5

## Aufgabe 2

### a)

```
# https://www.rdocumentation.org/packages/caret/versions/6.0-78/source

train.mylda <- function(data, target) {
  targetIndex <- grep(target, colnames(data)) #Index der Klassen-Spalte
```

```
  classes <- unique(data[, targetIndex]) #z.B. {"good", "bad"} für Ionosphere
  cleanData <- data[,-targetIndex] #Daten ohne die Target-Spalte
  p <- ncol(cleanData) #Anzahl der Spalten abzueglich der Target-Spalte = Anzahl der Dimensionen.
  n <- c()
  meanList <- c()

  matSum <- matrix(0, ncol=p, nrow=p)

  for (i in 1: length(classes)) {
    classSpecificData <- data[data[,targetIndex] == toString(classes[i]),] #z.B. nur Daten zur Klasse "
    cleanClassSpecificData <- classSpecificData[, -targetIndex]
    n <- c(n, nrow(cleanClassSpecificData)) ###Anzahl der p-dimensionalen Beobachtungen. Muss kein Vekt
    meanVec <- apply(cleanClassSpecificData, 2, mean) #Parameter 2: spaltenweise
    meanList <- c(meanList, meanVec)

    for (j in 1:nrow(cleanClassSpecificData)){ #Pro Beoabchtung wird summiert nach Formel (13), s. 143
      mat <- as.matrix(cleanClassSpecificData[j,] - meanVec)
      matGes <- t(mat) %*% mat
      matSum <- matSum+matGes
    }
  }

  meanMat <- matrix(meanList, ncol=p)
  meanFrame <- data.frame(meanMat, classes)
  colnames(meanFrame) <- colnames(data)

  firstPartCov <- (1 / (sum(n)-length(classes)))
  covMat <- firstPartCov*matSum

  return( list("cov"=covMat, "means"=meanFrame, "apriori"= n/sum(n)) ) ###Apriori-Wahrscheinlichkeit ei
}
```

mj: Bei der Erstellung eurer meanMat beachtet ihr nicht, dass die Werte spaltenweise und nicht zeilenweise eingefügt werden, daher passt die Zuordnung der Variablen/Klassen dort nicht mehr. Das verfälscht eure Egebnisse gravierend -1

mj: 1/2

**b)**

```
predict.mylda <- function(model, newdata) {
  cleanNewdata <- newdata[ , unlist(lapply(newdata, is.numeric))] #Nicht numerische (=Klassen)Spalte en
  invCov <- solve(model$cov)
  predResult <- data.frame(matrix(ncol = 2, nrow = nrow(newdata)))
  colnames(predResult) <- c("truth", "response")

  for (i in 1:nrow(cleanNewdata)){ #Für jede Test-Beobachtung
    predResult[i,1] <- toString(newdata[i, ncol(model$means)]) #(Wahre Klasse wird schon im Voraus eing
    maxH <- -Inf
    for (j in 1:nrow(model$means)){ #Und jede Klasse soll Formel (6), s. 130 berechnet werden
      classSpecificMeanVector <- as.numeric(model$means[j,-(ncol(model$means))])
      first <- t(invCov %*% classSpecificMeanVector) %*% as.numeric(cleanNewdata[i,])
      second <- 0.5 * classSpecificMeanVector %*% invCov %*% classSpecificMeanVector
```

```
      third <- log( model$apriori[j] , base = exp(1))
      ges <- first-second+third
      sumGes <- sum(ges)

      if (sumGes >= maxH) {
        predResult[i,2] <- toString(model$means[j, ncol(model$means)]) #(Vorhergesate Klasse wird ergän
        maxH <- sumGes
      }
    }
  }
  return(predResult)
}
```

mj: 2/2

**c)**

80/20 Aufteilungen des Iris-Datensatzes

```
set.seed(201805)

samples = sample(1:150, 90) # 80/20 Aufteilung, statt 60/40

train = iris[samples,]
test = iris[-samples,]
```

Modell, Vorhersage und Fehlerrate

```
model <- train.mylda(train, "Species")
prediction <- predict.mylda(model, test)

prediction_eval <- function (pred) {
  n <- nrow(pred)
  goodness <- 0
  predEval <- data.frame(matrix(ncol = 2, nrow = 1))
  colnames(predEval) <- c("mmce", "acc")
  for (i in 1:n) {
    if (pred[i,1] == pred[i,2]) {
      goodness <- goodness + 1
    }
  }
  predEval[1,1] <- (n - goodness) / n
  predEval[1,2] <- goodness / n
  return(predEval)
}

print(model)
```

```
## $cov
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length   0.31200057  0.10038467   0.20844942  0.04800409
## Sepal.Width    0.10038467  0.11147721   0.05992722  0.03271330
## Petal.Length   0.20844942  0.05992722   0.20943117  0.04846209
## Petal.Width    0.04800409  0.03271330   0.04846209  0.04490750
##
```

```
## $means
##   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1     5.951515    1.339394     5.555172   3.4642857 versicolor
## 2     2.745455    6.586207     2.017241   1.4500000  virginica
## 3     4.239394    2.948276     5.021429   0.2392857     setosa
##
## $apriori
## [1] 0.3666667 0.3222222 0.3111111
```

```
print(prediction)
```

```
##         truth   response
## 1      setosa  virginica
## 2      setosa     setosa
## 3      setosa     setosa
## 4      setosa     setosa
## 5      setosa     setosa
## 6      setosa  virginica
## 7      setosa  virginica
## 8      setosa     setosa
## 9      setosa  virginica
## 10     setosa     setosa
## 11     setosa  virginica
## 12     setosa     setosa
## 13     setosa     setosa
## 14     setosa     setosa
## 15     setosa     setosa
## 16     setosa     setosa
## 17     setosa     setosa
## 18     setosa     setosa
## 19     setosa     setosa
## 20     setosa     setosa
## 21     setosa     setosa
## 22     setosa     setosa
## 23 versicolor     setosa
## 24 versicolor     setosa
## 25 versicolor     setosa
## 26 versicolor     setosa
## 27 versicolor     setosa
## 28 versicolor     setosa
## 29 versicolor     setosa
## 30 versicolor     setosa
## 31 versicolor     setosa
## 32 versicolor     setosa
## 33 versicolor     setosa
## 34 versicolor versicolor
## 35 versicolor     setosa
## 36 versicolor     setosa
## 37 versicolor     setosa
## 38 versicolor     setosa
## 39 versicolor     setosa
## 40  virginica versicolor
## 41  virginica versicolor
## 42  virginica versicolor
## 43  virginica     setosa
```

```
## 44  virginica     setosa
## 45  virginica versicolor
## 46  virginica versicolor
## 47  virginica versicolor
## 48  virginica     setosa
## 49  virginica versicolor
## 50  virginica     setosa
## 51  virginica versicolor
## 52  virginica versicolor
## 53  virginica     setosa
## 54  virginica versicolor
## 55  virginica     setosa
## 56  virginica versicolor
## 57  virginica versicolor
## 58  virginica versicolor
## 59  virginica versicolor
## 60  virginica versicolor
```

```
print("Prediction evaluation:")
```

```
## [1] "Prediction evaluation:"
```

```
eval <- prediction_eval(prediction)
eval
```

```
##   mmce acc
## 1  0.7 0.3
```

```
paste(c("Mean misclassification error (Fehlerklassifikationsrate): ", round(eval[1,1]*100,digits= 2), "
```

```
## [1] "Mean misclassification error (Fehlerklassifikationsrate): 70%"
```

mj: Solch eine Fehlerrate ist extrem schlecht und sollte euch dazu bringen, die Funktion noch einmal zu überprüfen

mj: 1/1

mj: Aufgabe 2 4/5

## Aufgabe 3

### a)

```
task = makeClassifTask(data = train, target = "Species")
learner = makeLearner("classif.lda")
model = train(learner, task)

prediction = predict(model, newdata = test)
performance(prediction, measures = list(mmce, acc))
```

```
##       mmce        acc
## 0.03333333 0.96666667
```

```
# Vergleichen Sie die Fehlklassifikationsrate mit Ihrer eigenen Implementierung. Hätte man sich diesen
```

Interpretation: Die Fehlklassifikationsrate ist mit 3,33% deutlich niedriger als die unserer eigenen Implementierung (70%). Daher scheint es, als wäre die Fischer'sche LDA der Kanonischen LDA überlegen.

mj: Das liegt natürlich an eurem Fehler, den eigentlichen Grund, warum die LDAs hier nicht zwingend gleich sein müssen, nennt ihr leider nicht -0.5

mj: 1/1.5

**b)**

```
task = makeClassifTask(data = train, target = "Species")
learner = makeLearner("classif.qda")
model = train(learner, task)

prediction = as.data.frame(predict(model, newdata = test))
performance(predict(model, newdata = test), measures = list(mmce, acc))
```

```
##       mmce        acc
## 0.01666667 0.98333333
```

Interpretation: Die Fehlklassifikationsrate hat sich halbiert. Nur in einem von 60 Fällen entspricht die Vorhersage nicht der Wahrheit.

mj: 0.5/0.5

**c)**

```
task = makeClassifTask(data = train, target = "Species")

rda_param_eval <- function (gam, lam) {
  if (length(gam) != length(lam)) {
    return("Vectors of lamda and gamma params need to be equally long")
  }
  results = matrix(0, nrow = 36, ncol = 3)
  colnames(results) = c("Gamma", "Lambda", "Fehlklassifikationsrate")
  n <- length(gam)
  tabIndx <- 0
  for (i in 1:n) {
    for (j in 1:n) {
      tabIndx <- tabIndx + 1
      learner = makeLearner("classif.rda", crossval = FALSE, gamma = gam[i], lambda = lam[j])
      model <- train(learner, task)
      pred <- predict(model, newdata = test)
      results[tabIndx,1] <- gam[i]
      results[tabIndx,2] <- lam[j]
      results[tabIndx,3] <- performance(pred, measures = mmce)
    }
  }
  results <- as.data.frame(results)
  results <- results[order(-results$Fehlklassifikationsrate),] # sortieren desc
  return(results)
}

gamma <- c(0, 0.2, 0.4, 0.6, 0.8, 1)
lambda <- c(0, 0.2, 0.4, 0.6, 0.8, 1)
```

```r
param_results <- rda_param_eval(gamma, lambda)
print(param_results)
```

```
##    Gamma Lambda Fehlklassifikationsrate
## 13   0.4    0.0              0.06666667
## 19   0.6    0.0              0.06666667
## 20   0.6    0.2              0.06666667
## 7    0.2    0.0              0.05000000
## 14   0.4    0.2              0.05000000
## 25   0.8    0.0              0.05000000
## 26   0.8    0.2              0.05000000
## 27   0.8    0.4              0.05000000
## 31   1.0    0.0              0.05000000
## 32   1.0    0.2              0.05000000
## 33   1.0    0.4              0.05000000
## 34   1.0    0.6              0.05000000
## 35   1.0    0.8              0.05000000
## 36   1.0    1.0              0.05000000
## 2    0.0    0.2              0.03333333
## 3    0.0    0.4              0.03333333
## 4    0.0    0.6              0.03333333
## 5    0.0    0.8              0.03333333
## 6    0.0    1.0              0.03333333
## 8    0.2    0.2              0.03333333
## 9    0.2    0.4              0.03333333
## 15   0.4    0.4              0.03333333
## 16   0.4    0.6              0.03333333
## 21   0.6    0.4              0.03333333
## 22   0.6    0.6              0.03333333
## 23   0.6    0.8              0.03333333
## 24   0.6    1.0              0.03333333
## 28   0.8    0.6              0.03333333
## 29   0.8    0.8              0.03333333
## 30   0.8    1.0              0.03333333
## 1    0.0    0.0              0.01666667
## 10   0.2    0.6              0.01666667
## 11   0.2    0.8              0.01666667
## 12   0.2    1.0              0.01666667
## 17   0.4    0.8              0.01666667
## 18   0.4    1.0              0.01666667
```

Anwedung der RDA mit optimalen Parametern

```r
task = makeClassifTask(data = train, target = "Species")
learner = makeLearner("classif.rda", crossval = FALSE, gamma = 0, lambda = 0)
model = train(learner, task)

prediction = as.data.frame(predict(model, newdata = test))
performance(predict(model, newdata = test), measures = list(mmce, acc))
```

```
##       mmce        acc
## 0.01666667 0.98333333
```

Interpretation:

Die RDA erreicht die geringste Fehlklassifikationsrate von 1,6% für mehere Parameterkombinationen, u.A.

auch fuer Lambda = 0, Gamma = 0. Dieses Ergebnis wird auch von der QDA erreicht. Daher lohnt sich der Aufwand fuer RDA nicht.

mj: 3/3

mj: Aufgabe 3 4.5/5

## Aufgabe 4

### a)

Learner anlegen

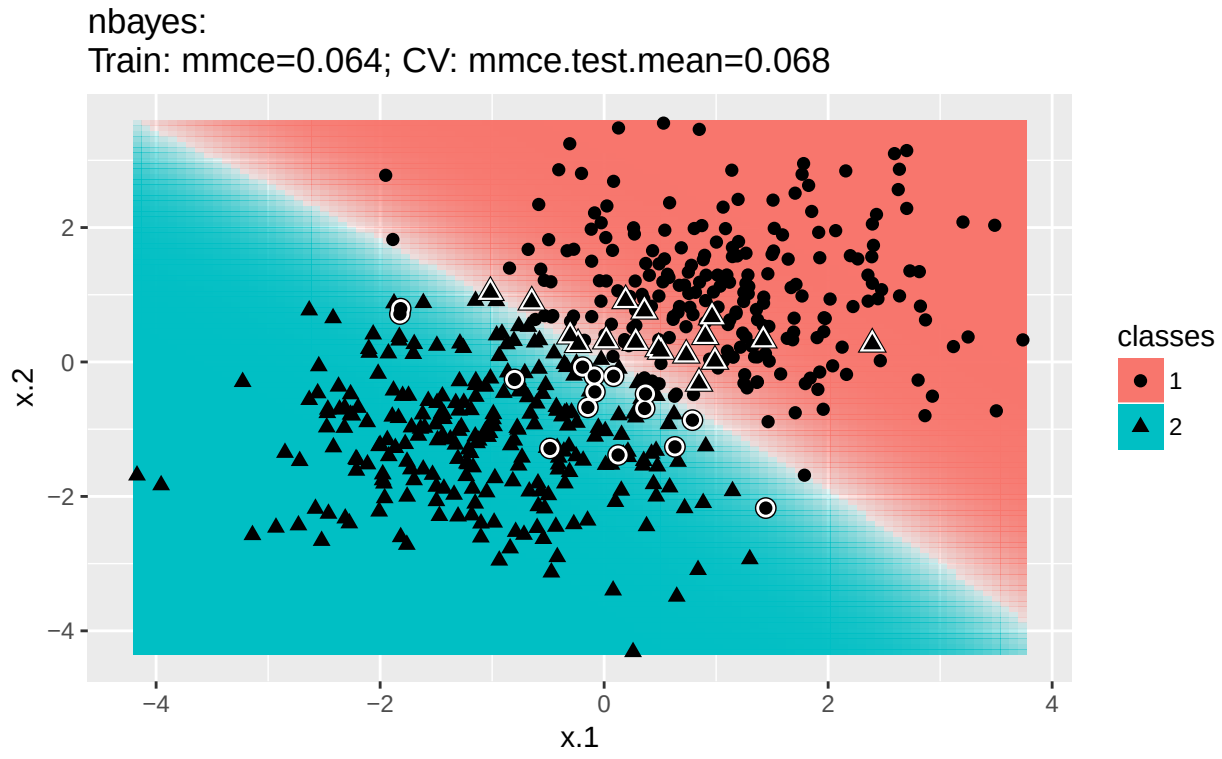
```
learnerNBay = makeLearner("classif.naiveBayes")
learnerLDA = makeLearner("classif.lda")
learnerQDA = makeLearner("classif.qda")
learnerRDA = makeLearner("classif.rda", crossval = FALSE, gamma = 0, lambda = 0)
#To do:  Setzen Sie moegliche Parameter der Verfahren sinnvoll. Ggf. durch rda_param_eval laufen lassen

learnerList <- list(learnerNBay, learnerLDA, learnerQDA, learnerRDA)
```

Tasks anlegen

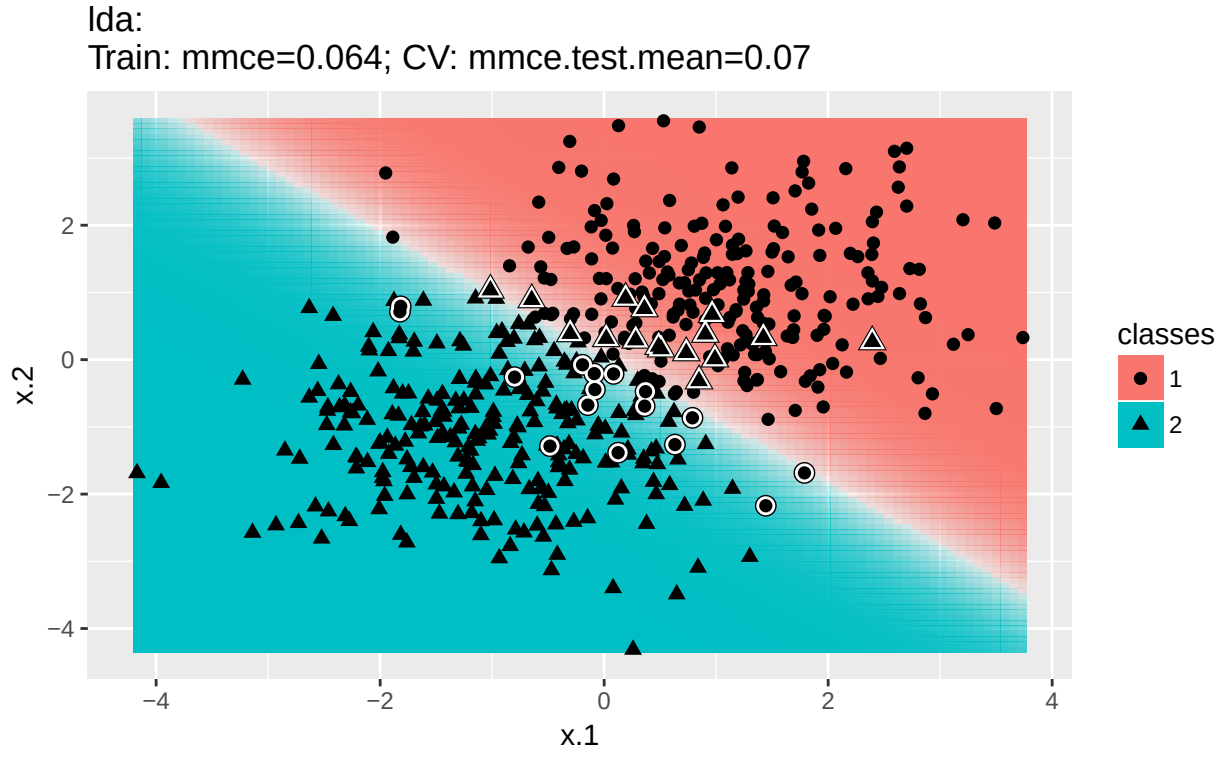
```
# ftp://cran.r-project.org/pub/R/web/packages/mlbench/mlbench.pdf
data2dnormals <- as.data.frame(mlbench.2dnormals(500,2))
dataSmiley <- as.data.frame(mlbench.smiley(500, 0.1, 0.05))
dataCassini <- as.data.frame(mlbench.cassini(5000))

task2dnormals = makeClassifTask(data = data2dnormals, target = "classes")
taskSmiley = makeClassifTask(data = dataSmiley, target = "classes")
taskCassini = makeClassifTask(data = dataCassini, target = "classes")

taskList <- list(task2dnormals, taskSmiley, taskCassini)
```

Grafiken erzeugen

```
plotLearnerPrediction(learner = learnerNBay, task = task2dnormals)
```

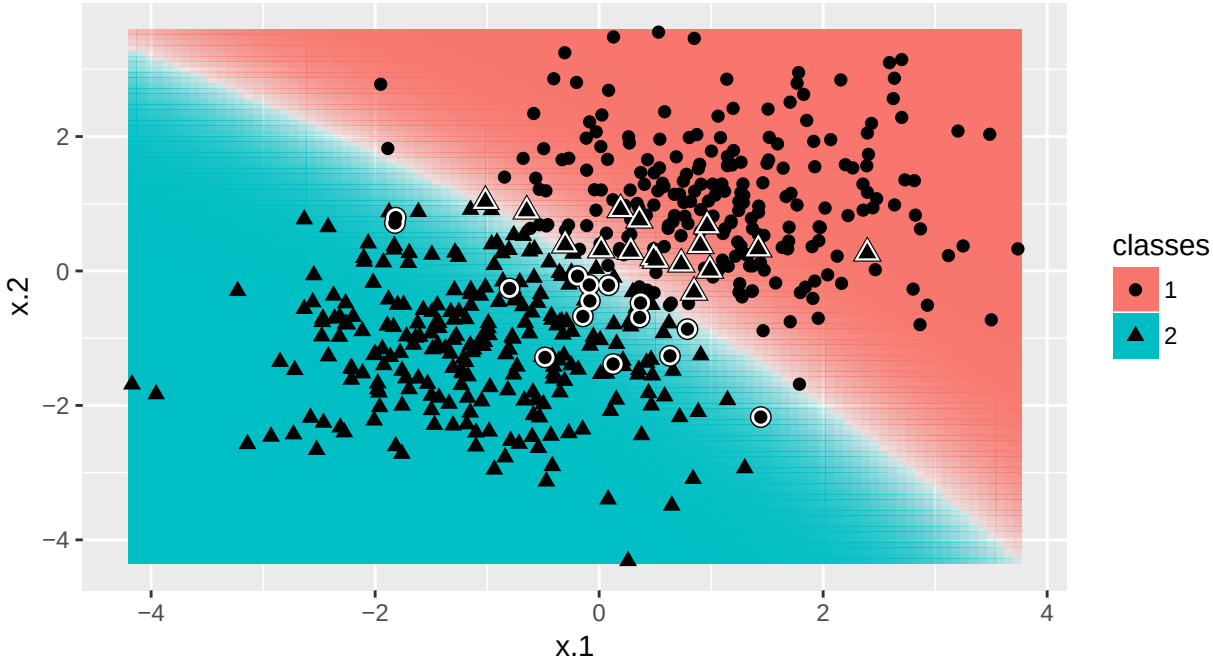



```
plotLearnerPrediction(learner = learnerLDA, task = task2dnormals)
```



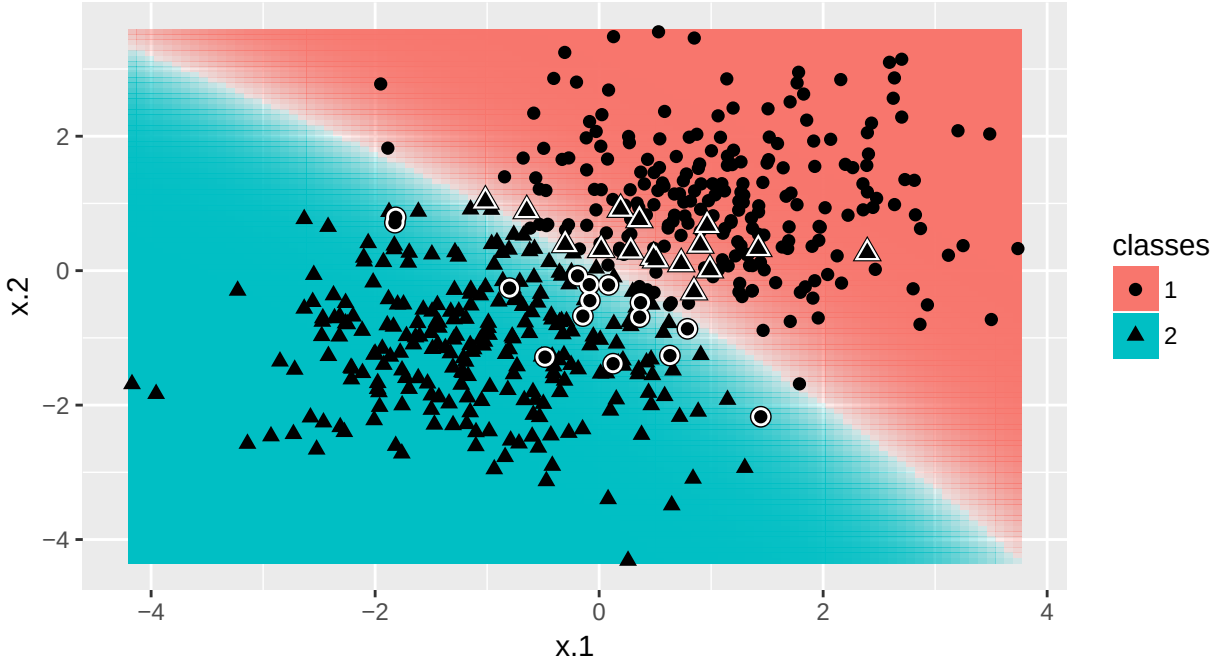

```
plotLearnerPrediction(learner = learnerQDA, task = task2dnormals)
```

qda:
Train: mmce=0.062; CV: mmce.test.mean=0.076

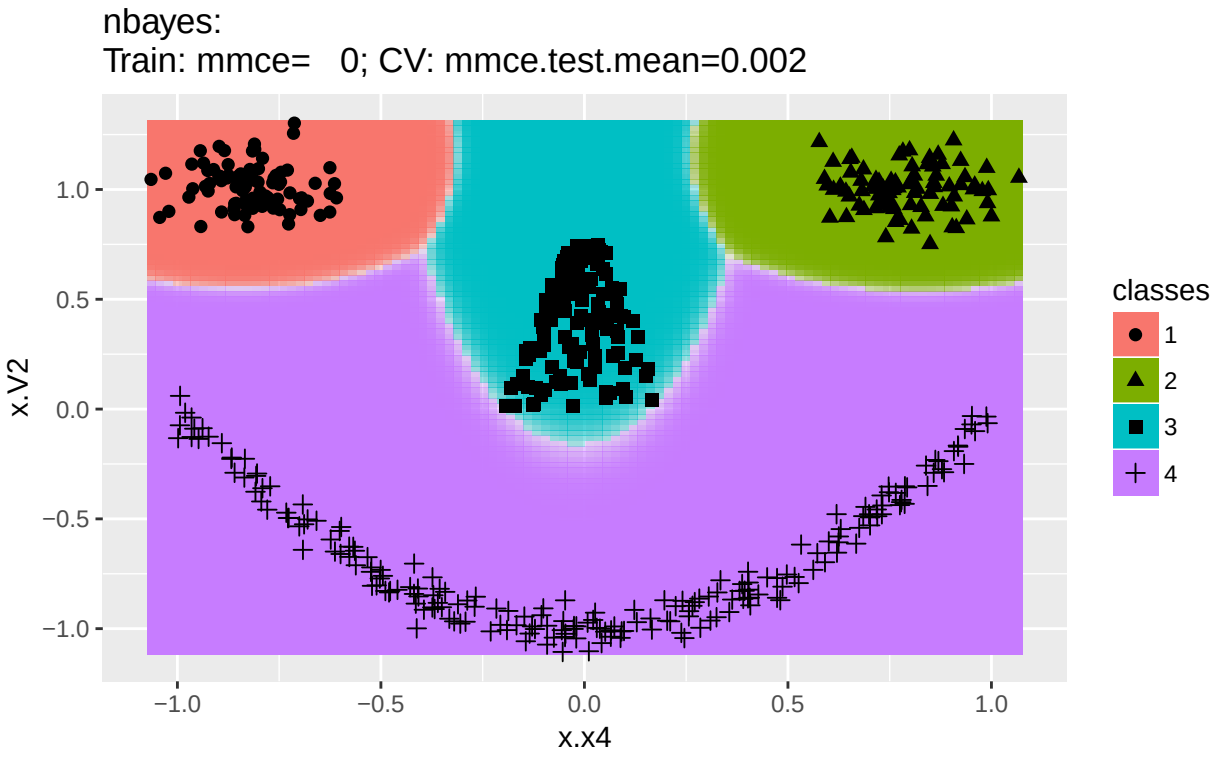



```
plotLearnerPrediction(learner = learnerRDA, task = task2dnormals)
```

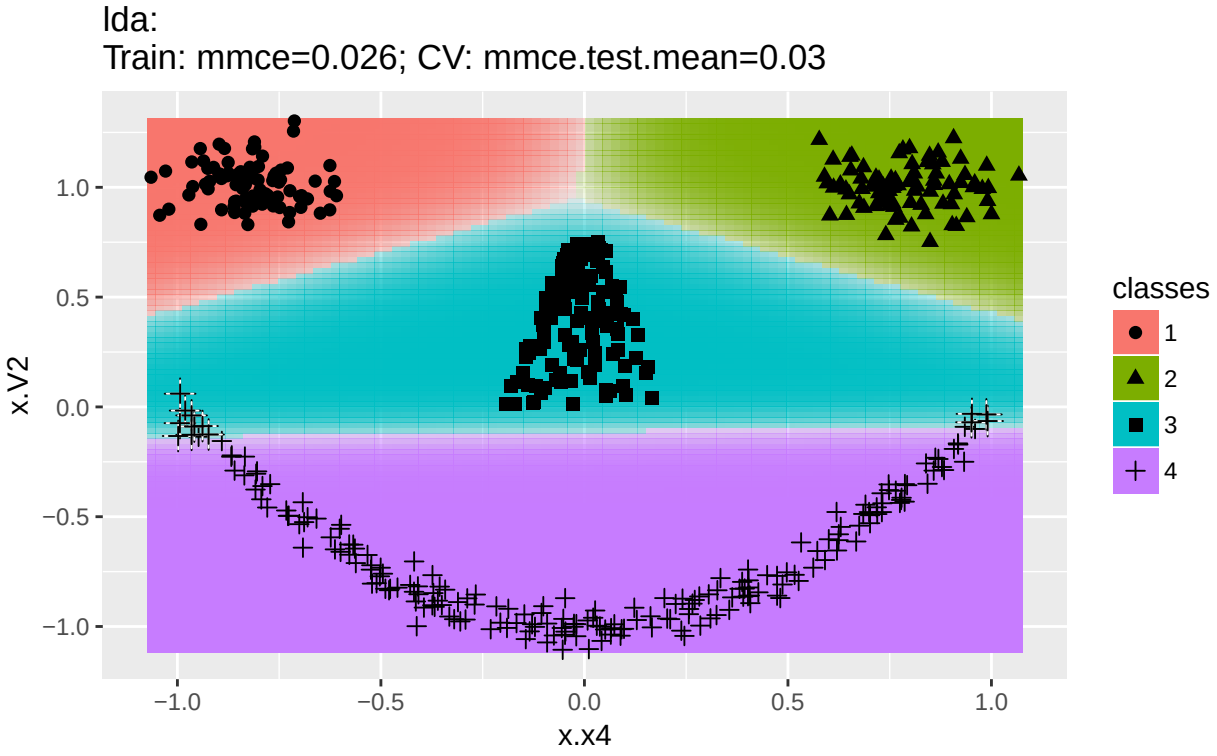
rda: estimate.error=FALSE; crossval=FALSE; gamma=0; lambda=0
Train: mmce=0.062; CV: mmce.test.mean=0.064




```
plotLearnerPrediction(learner = learnerNBay, task = taskSmiley)
```

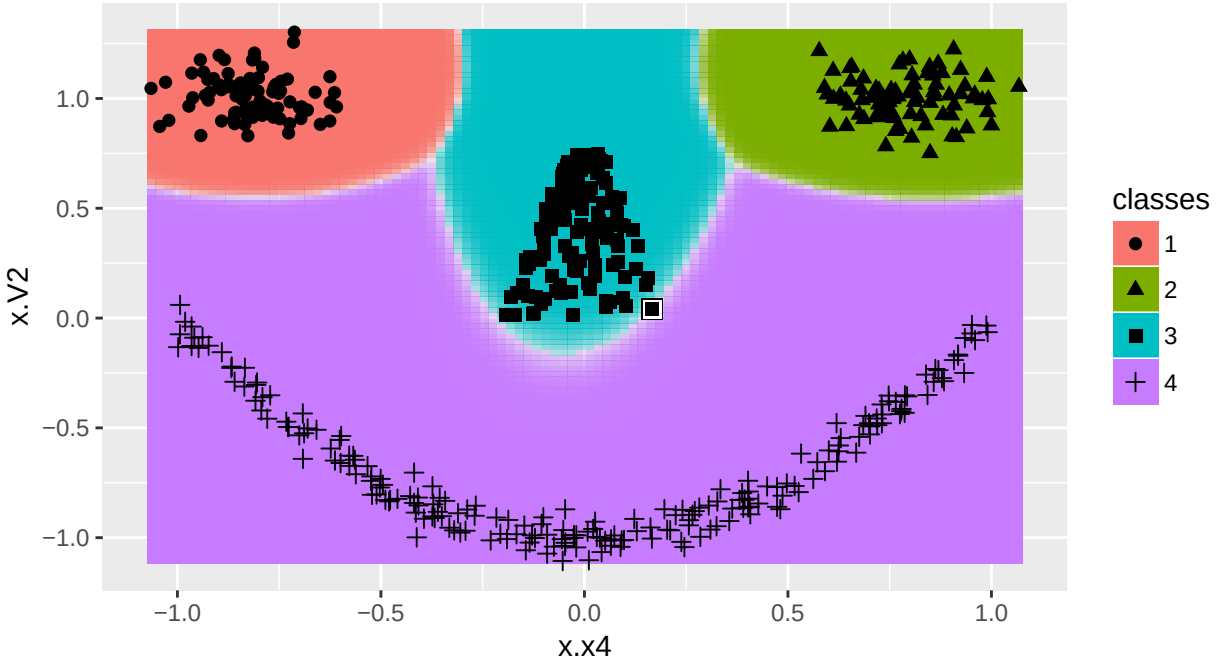



```
plotLearnerPrediction(learner = learnerLDA, task = taskSmiley)
```



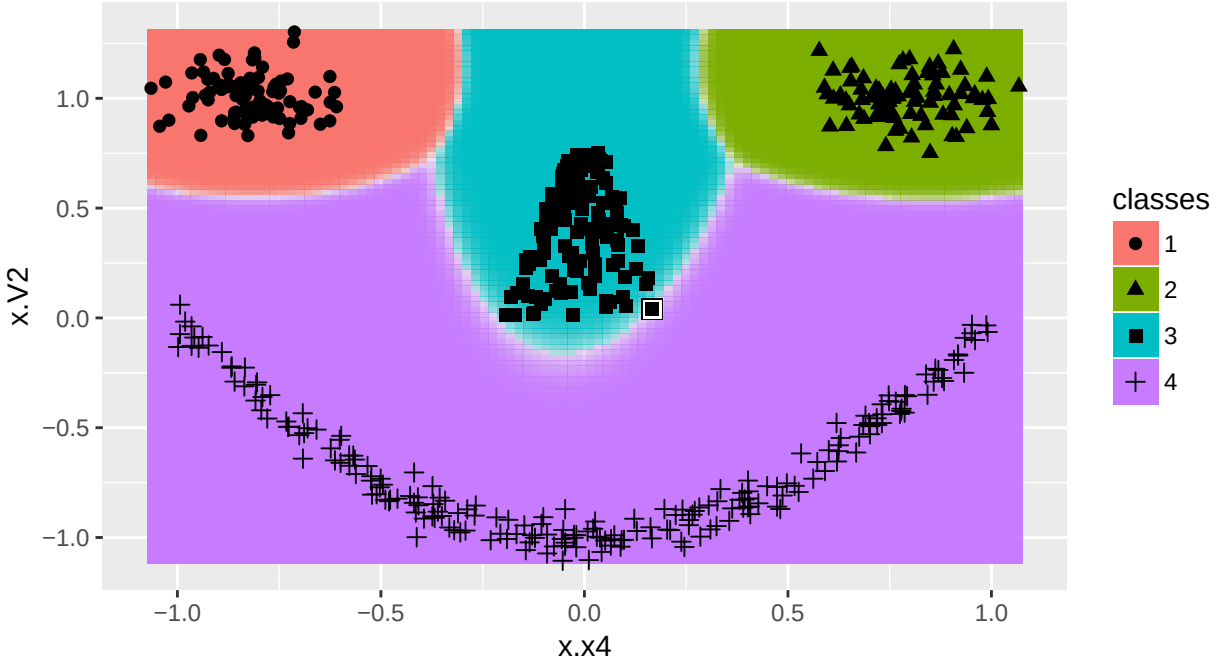

```
plotLearnerPrediction(learner = learnerQDA, task = taskSmiley)
```

qda:
Train: mmce=0.002; CV: mmce.test.mean=0.002

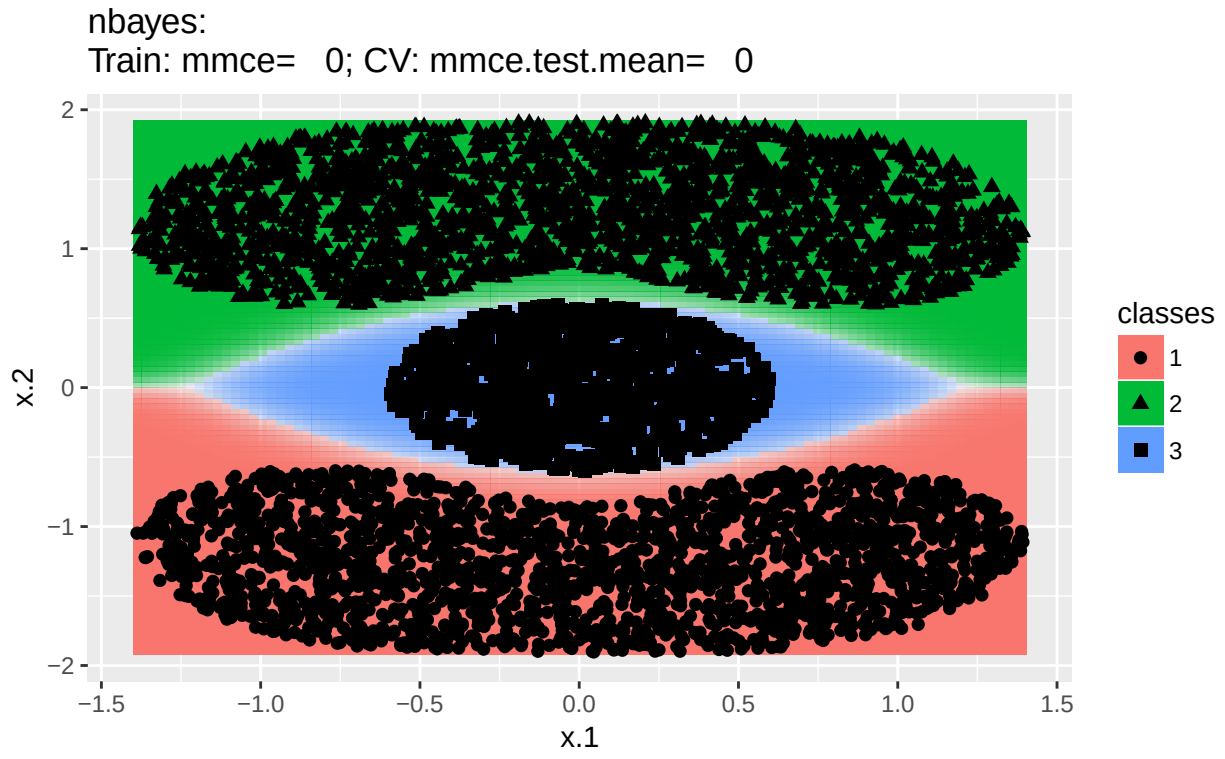



```
plotLearnerPrediction(learner = learnerRDA, task = taskSmiley)
```

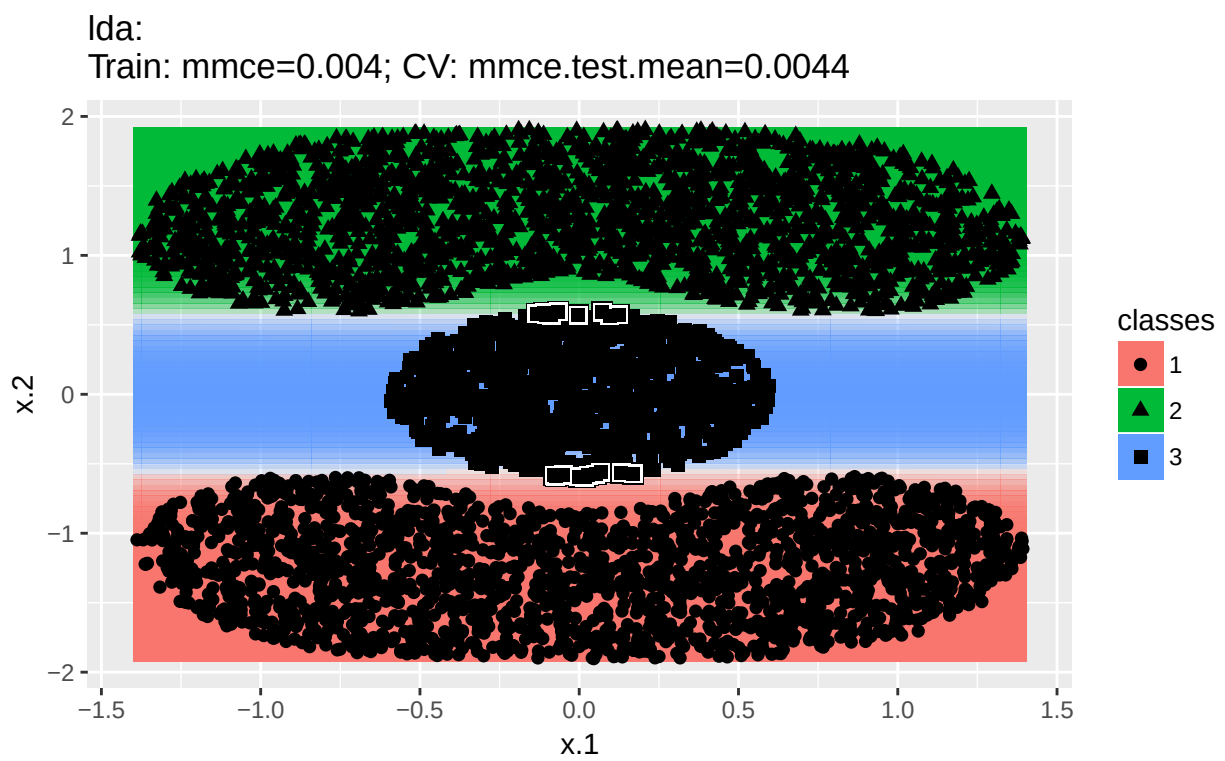
rda: estimate.error=FALSE; crossval=FALSE; gamma=0; lambda=0
Train: mmce=0.002; CV: mmce.test.mean=0.002



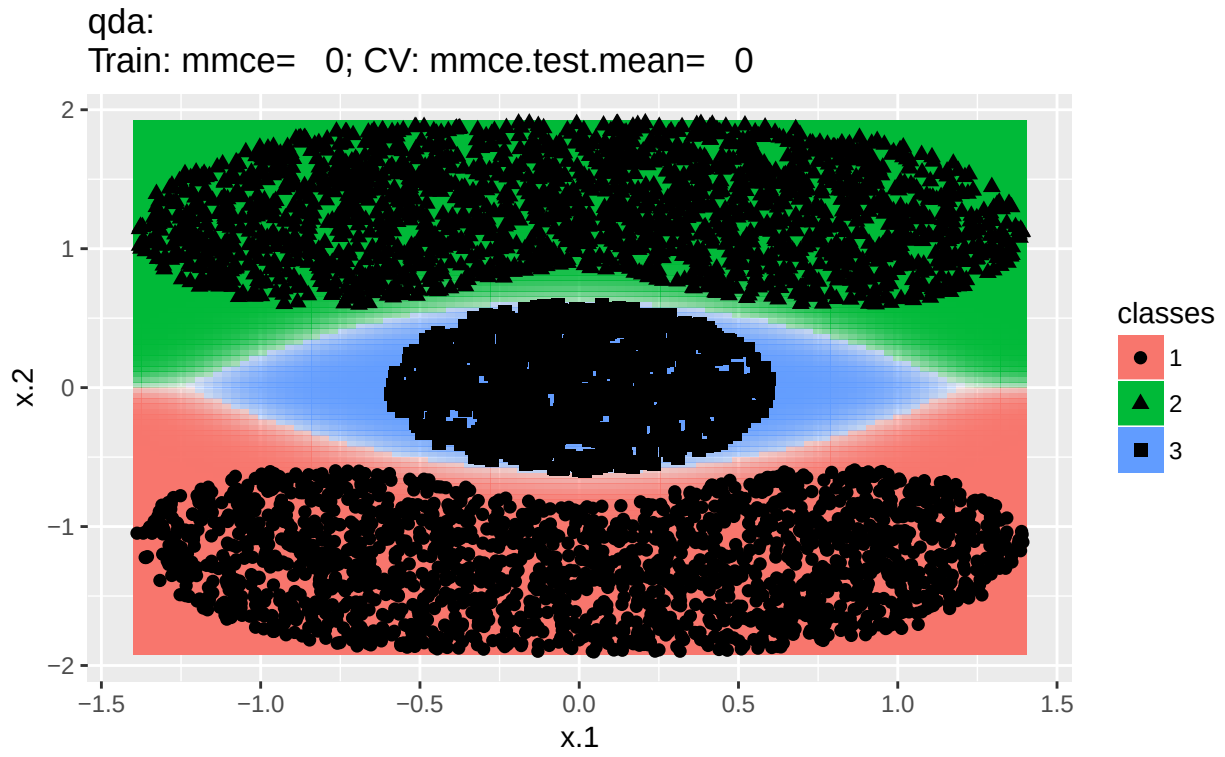

```
plotLearnerPrediction(learner = learnerNBay, task = taskCassini)
```



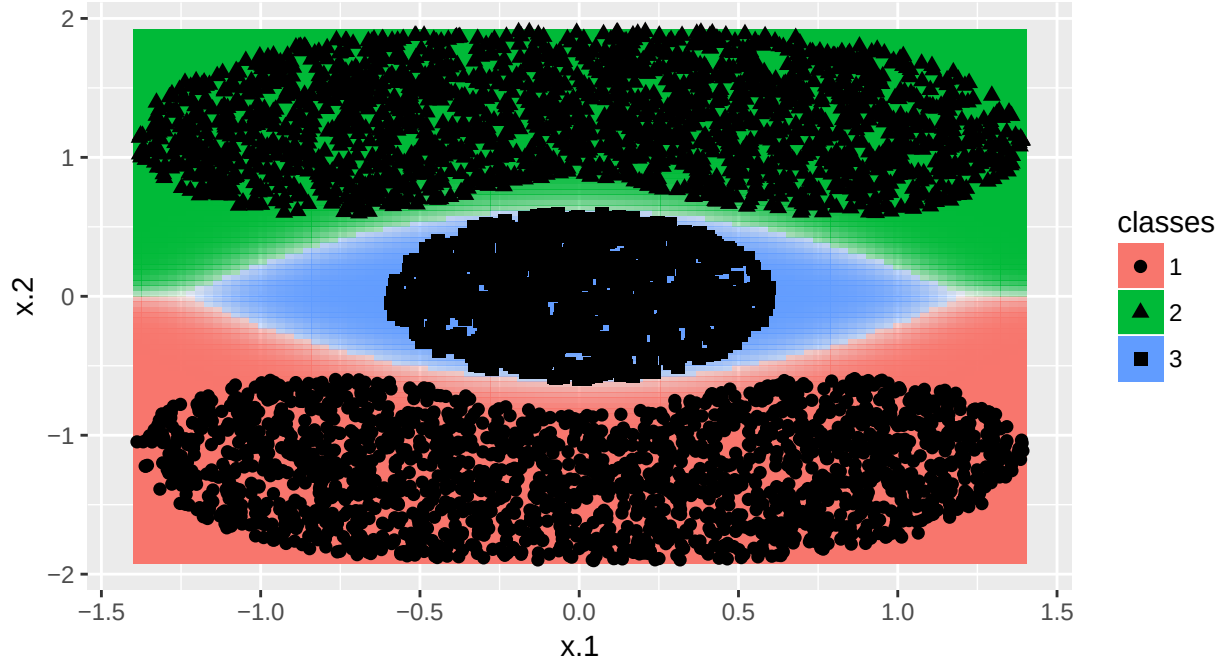

```
plotLearnerPrediction(learner = learnerLDA, task = taskCassini)
```




```
plotLearnerPrediction(learner = learnerQDA, task = taskCassini)
```




```
plotLearnerPrediction(learner = learnerRDA, task = taskCassini)
```




jns: 3/3

**b)**

Insgesamt schein die lineare Diskriminanzanalyse lineare Trennungsverläufe der Entscheidungsgrenzen aufzuweisen.

jns: Genau das ist ja das Besondere an der LDA.

Die Unterschiede zwischen Naive Bayes und der quadratischen Diskriminanzanalyse sind nur marginal, während sie zwischen der QDA und RDA, aufgrund der Wahl der Gamma- und Lambda-Parameter, nicht vorhanden sind.

jns: 2/2

jns: Aufgabe 4: 5/5

mj: Gesamt 18.5/20