Introduction to Machine Learning

Working Group "Computational Statistics" – Bernd Bischl et al.

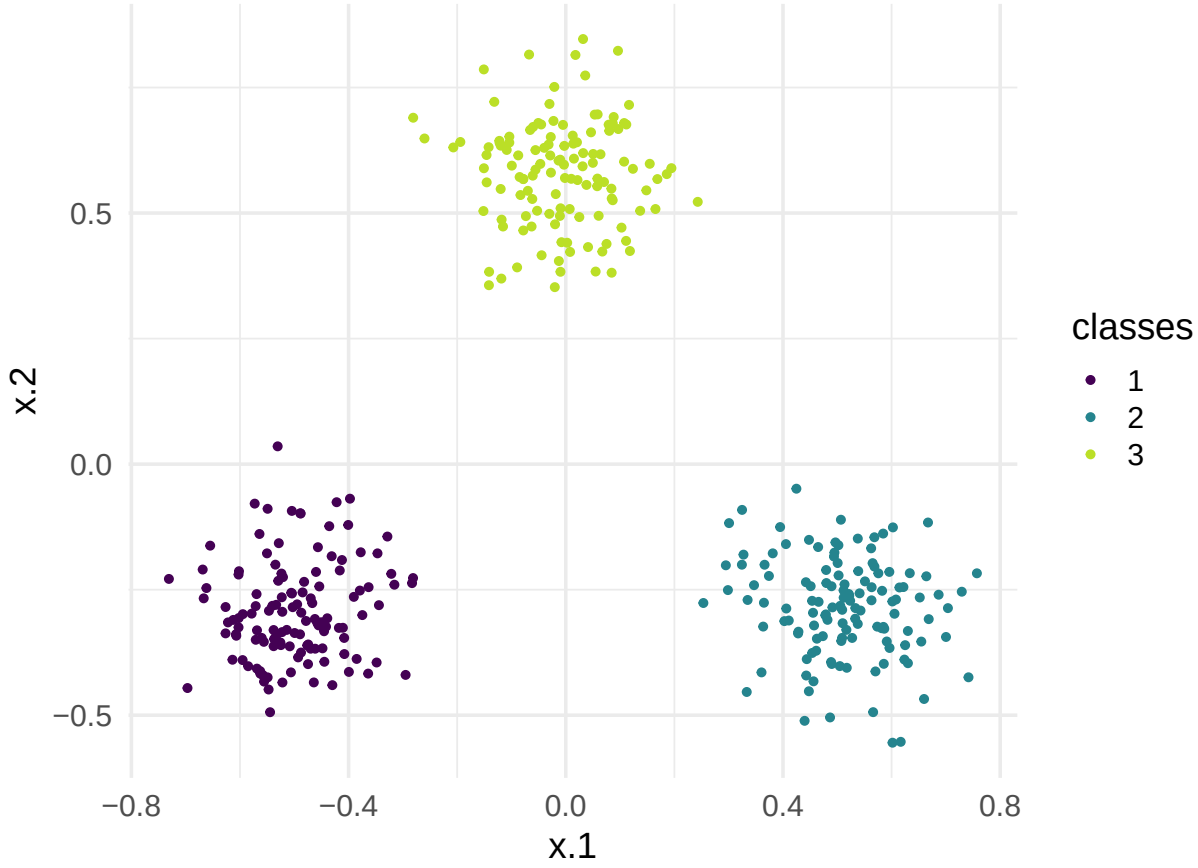
```
##   |                                                                      |
```

In this code demo we want to get a general impression how the choice of one of the classifier we learnt about influences the classification results. Every classification method we encountered in the lecture can be used for the case of more than two target class labels except the logistic regression. However, *softmax regression* is a direct generalization of logistic regression to that setting, so we use that.
For the Naive Bayes classifier, we use Gaussian likelihoods in the following.

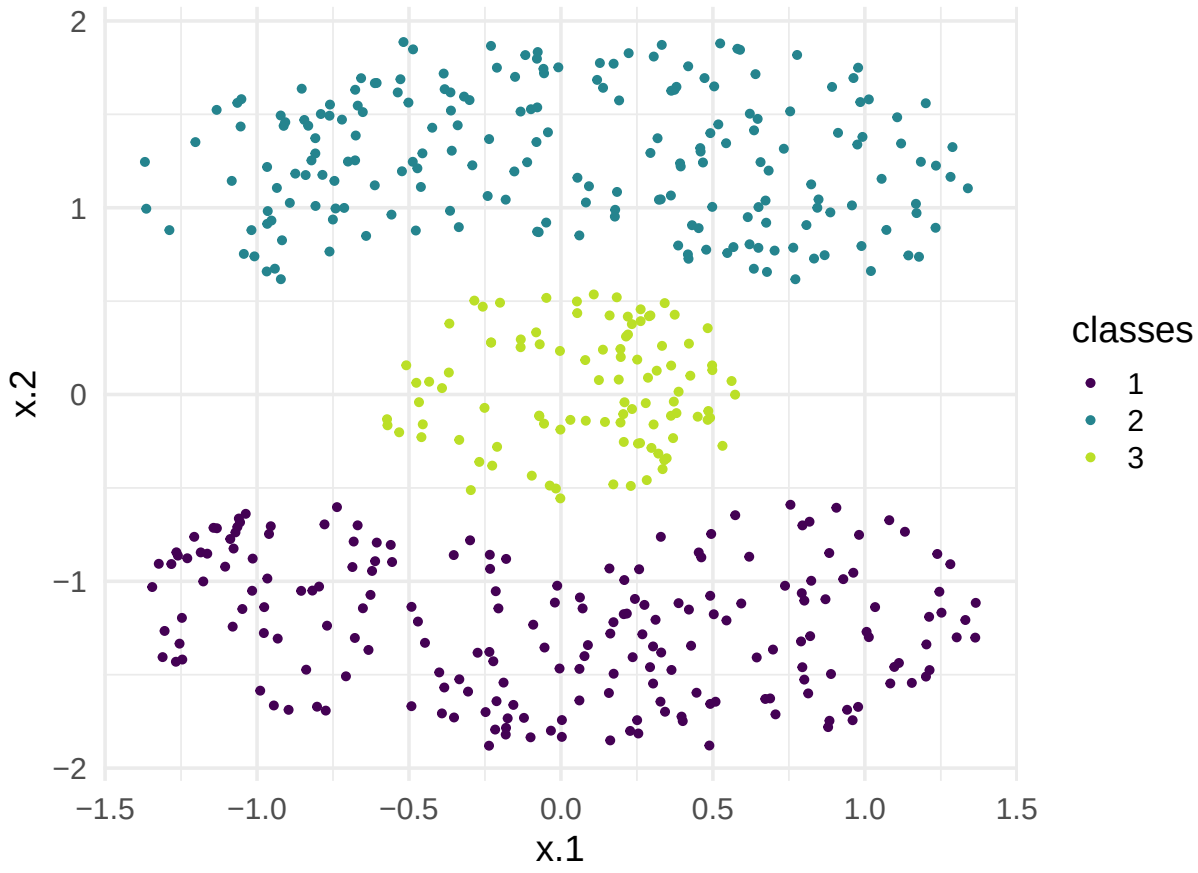
Data

We create some artificial benchmark data sets with the *mlbench* package:

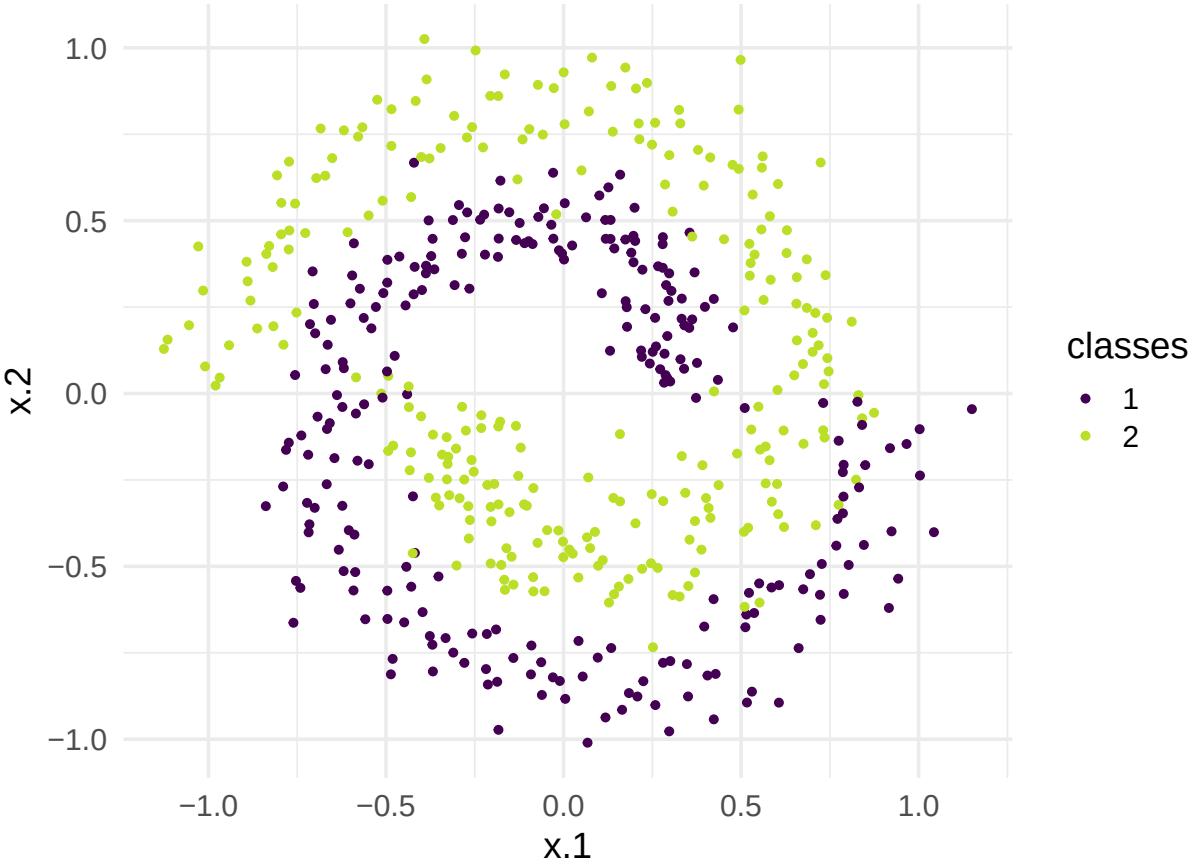
```
library(mlbench)
library(mlr3)
library(mlr3learners)
library(mlr3viz)
library(ggplot2)

set.seed(123L)

simplex <- data.frame(mlbench.simplex(n = 500, d = 2))
ggplot(simplex, aes(x.1, x.2, color = classes)) + geom_point()
```




```
cassini <- data.frame(mlbench.cassini(500))
ggplot(cassini, aes(x.1, x.2, color = classes)) + geom_point()
```




```
spirals <- data.frame(mlbench.spirals(n = 500, sd = 0.1))
ggplot(spirals, aes(x.1, x.2, color = classes)) + geom_point()
```



For the classification, we use the *mlr3* package, which means we need to create classification tasks:

```
spirals_task <- TaskClassif$new(id = "spirals", backend = spirals, target = "classes")
cassini_task <- TaskClassif$new(id = "cassini", backend = cassini, target = "classes")
simplex_task <- TaskClassif$new(id = "simplex", backend = simplex, target = "classes")
```

The methods we use for classification are represented by learner objects:

```
learners <- list(
  # Softmax regression
  softmax_learner = lrn("classif.multinom",
    trace = FALSE,
    predict_type = "prob"
  ),
  # K-nearest neighbors
  knn_learner = lrn("classif.kknn",
    k = 3,
    predict_type = "prob"
  ),
  # Linear discriminant analysis
  lda_learner = lrn("classif.lda",
    predict_type = "prob"
  ),
  # Quadratic discriminant analysis
  qda_learner = lrn("classif.qda",
```

```
    predict_type = "prob"
  ),
  nb_learner = lrn("classif.naive_bayes",
    predict_type = "prob"
  )
)
```

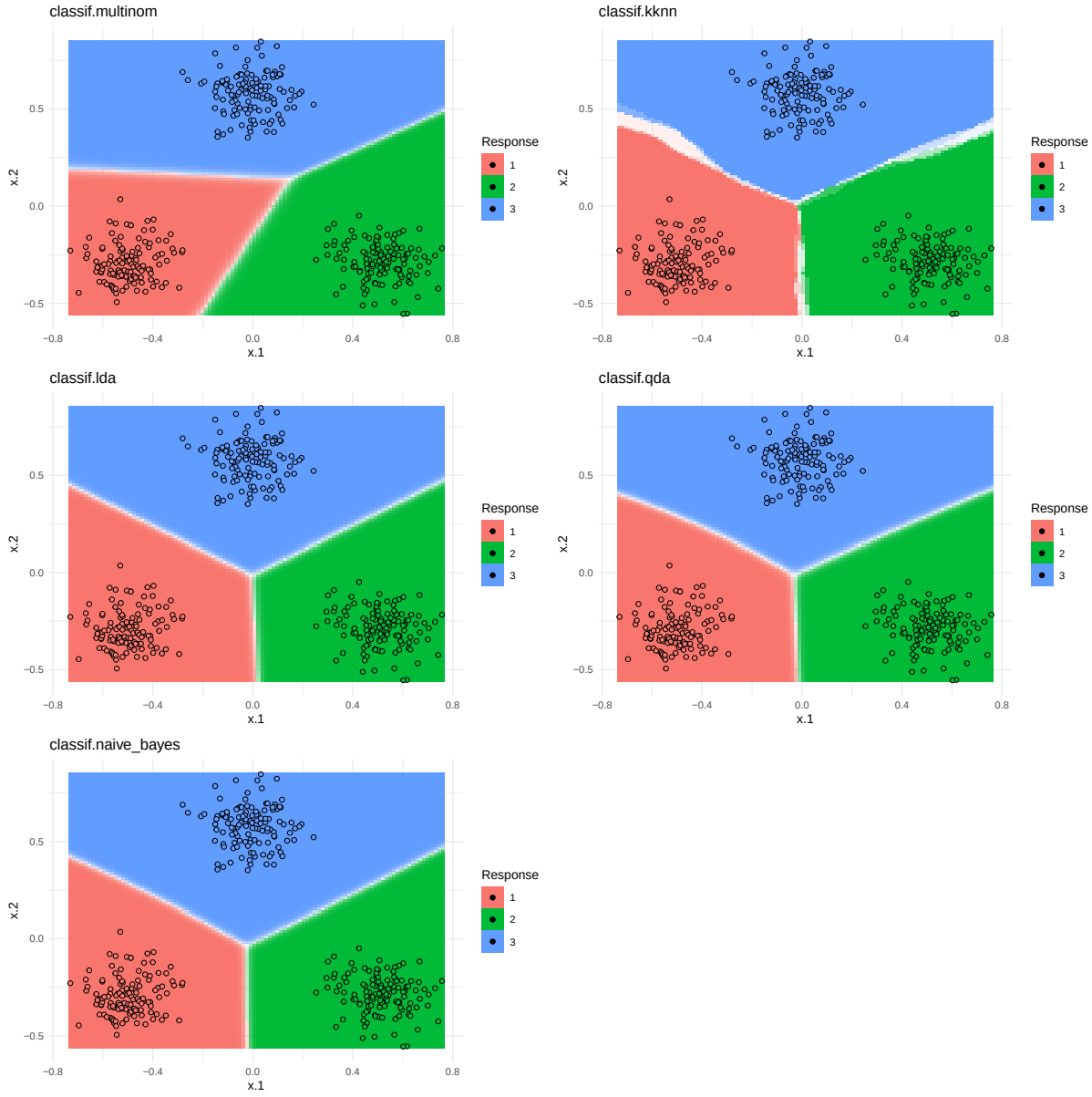
Comparison

We start by looking at the `simplex` data set which is linearly separable:

```
ggplot_list <- lapply(
  learners,
  function(learner) plot_learner_prediction(learner, simplex_task) +
      theme_minimal(base_size = 10) + guides(alpha = "none", shape = "none") +
      ggtitle(learner$id)
)
```

```
## INFO  [10:39:36.661] Applying learner 'classif.multinom' on task 'simplex' (iter 1/1)
## INFO  [10:39:37.125] Applying learner 'classif.kknn' on task 'simplex' (iter 1/1)
## INFO  [10:39:38.906] Applying learner 'classif.lda' on task 'simplex' (iter 1/1)
## INFO  [10:39:39.307] Applying learner 'classif.qda' on task 'simplex' (iter 1/1)
## INFO  [10:39:39.545] Applying learner 'classif.naive_bayes' on task 'simplex' (iter 1/1)
```

```
do.call(grid.arrange, ggplot_list)
```



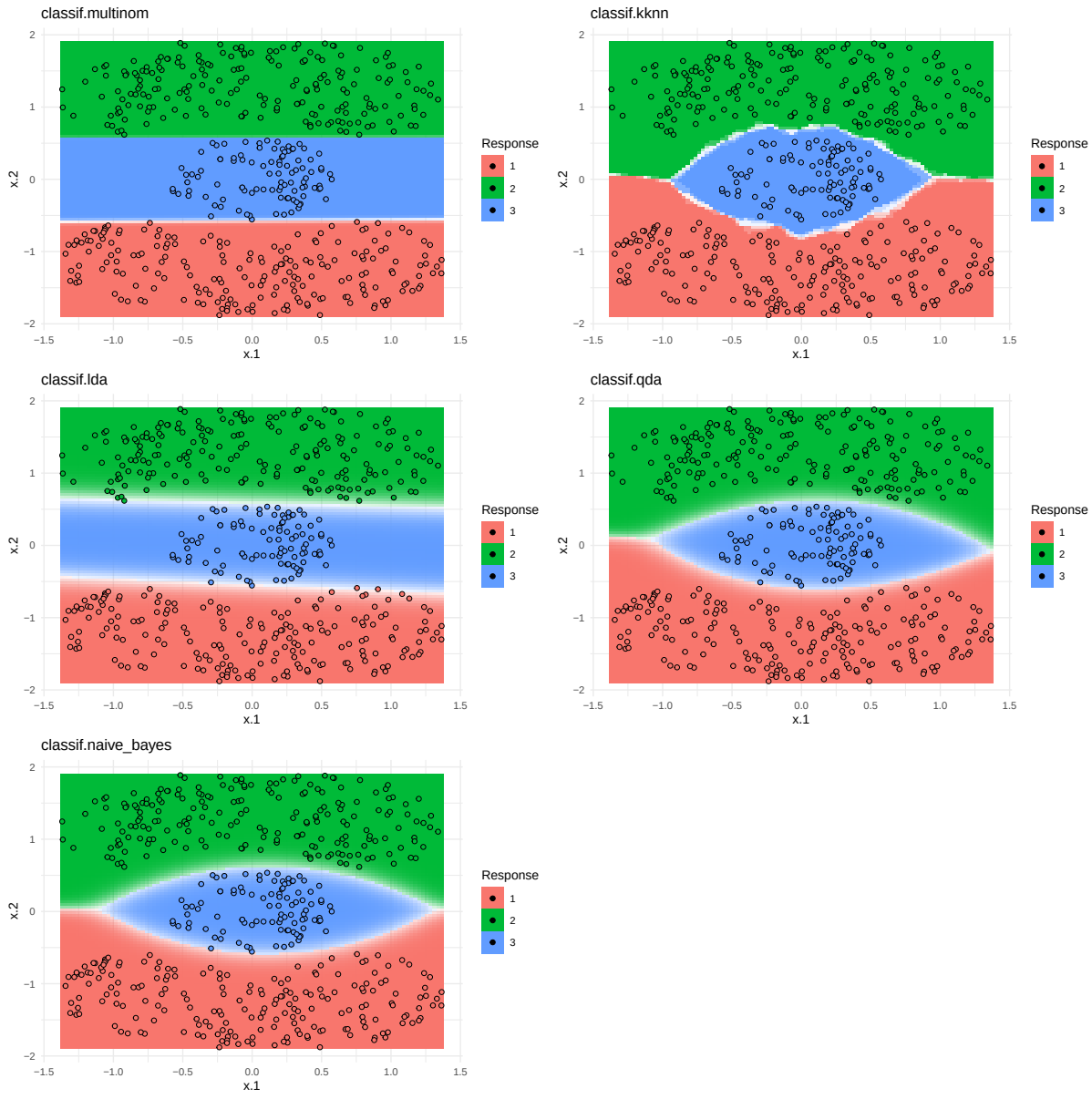
We see that we are able to classify the data correctly with every method. Since both LDA and softmax regression are linear classifiers, their decision boundaries are linear as expected. We also observe that only softmax regression has clearly asymmetric boundaries.

```
ggplot_list <- lapply(
  learners,
  function(learner) plot_learner_prediction(learner, cassini_task) +
      theme_minimal(base_size = 10) + guides(alpha = "none", shape = "none") +
      ggtitle(learner$id)
)
```

```
## INFO  [10:39:45.701] Applying learner 'classif.multinom' on task 'cassini' (iter 1/1)
## INFO  [10:39:45.872] Applying learner 'classif.kknn' on task 'cassini' (iter 1/1)
## INFO  [10:39:46.466] Applying learner 'classif.lda' on task 'cassini' (iter 1/1)
## INFO  [10:39:46.691] Applying learner 'classif.qda' on task 'cassini' (iter 1/1)
```

```
## INFO  [10:39:46.912] Applying learner 'classif.naive_bayes' on task 'cassini' (iter 1/1)
```

```
do.call(grid.arrange, ggplot_list)
```

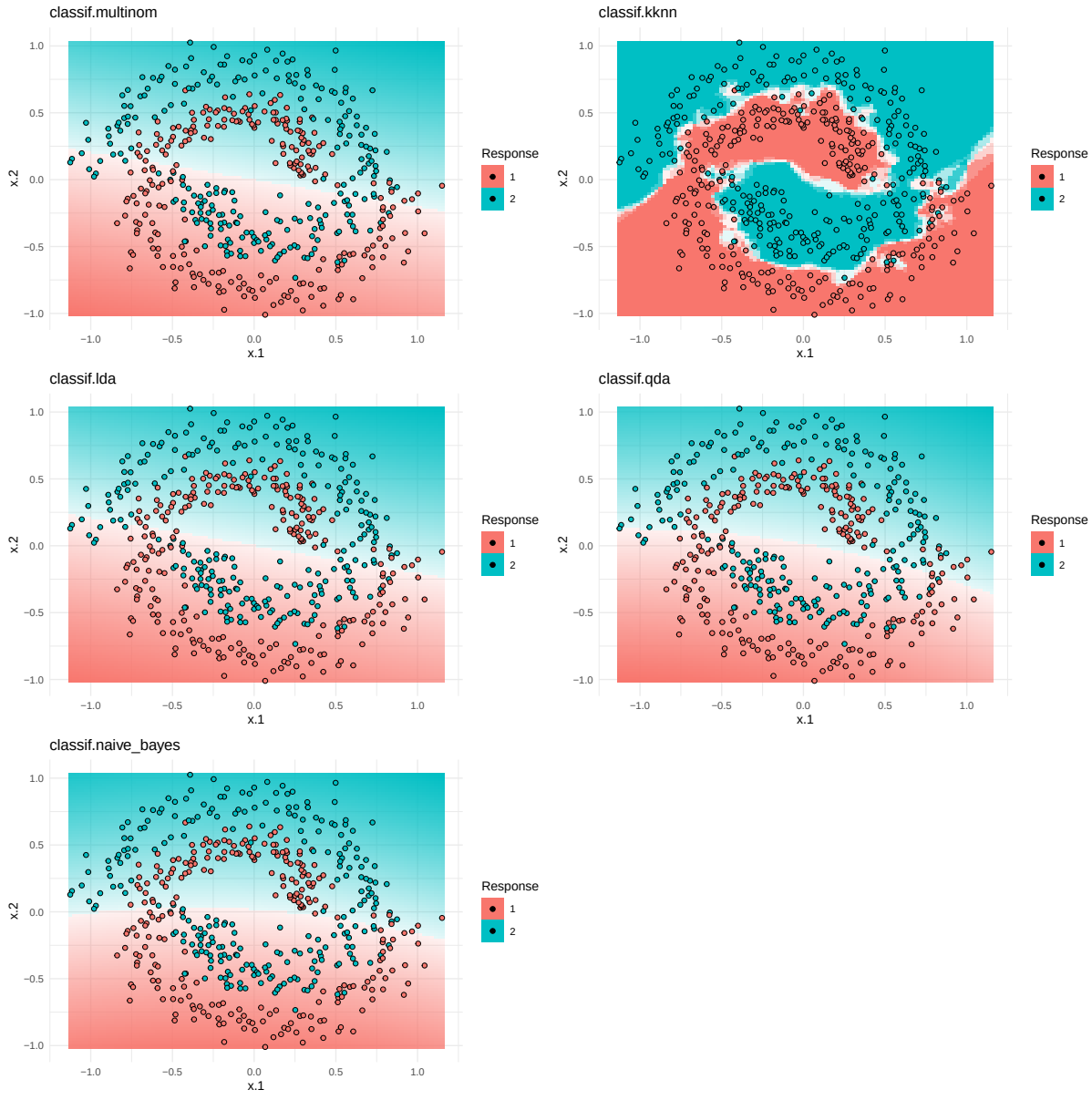


The softmax result (`classif.multinom`) shows that the Cassini data set is also linear separable, but some observations are quite close to its decision boundaries. LDA is the only method which doesn't separate the data perfectly here. The four other methods identify a similar shape of the boundaries.

```
ggplot_list <- lapply(
  learners,
  function(learner) plot_learner_prediction(learner, spirals_task) +
      theme_minimal(base_size = 10) + guides(alpha = "none", shape = "none") +
      ggtitle(learner$id)
)
```

```
## INFO  [10:39:52.859] Applying learner 'classif.multinom' on task 'spirals' (iter 1/1)
## INFO  [10:39:52.982] Applying learner 'classif.kknn' on task 'spirals' (iter 1/1)
## INFO  [10:39:53.577] Applying learner 'classif.lda' on task 'spirals' (iter 1/1)
## INFO  [10:39:53.848] Applying learner 'classif.qda' on task 'spirals' (iter 1/1)
## INFO  [10:39:54.392] Applying learner 'classif.naive_bayes' on task 'spirals' (iter 1/1)
```

```
do.call(grid.arrange, ggplot_list)
```

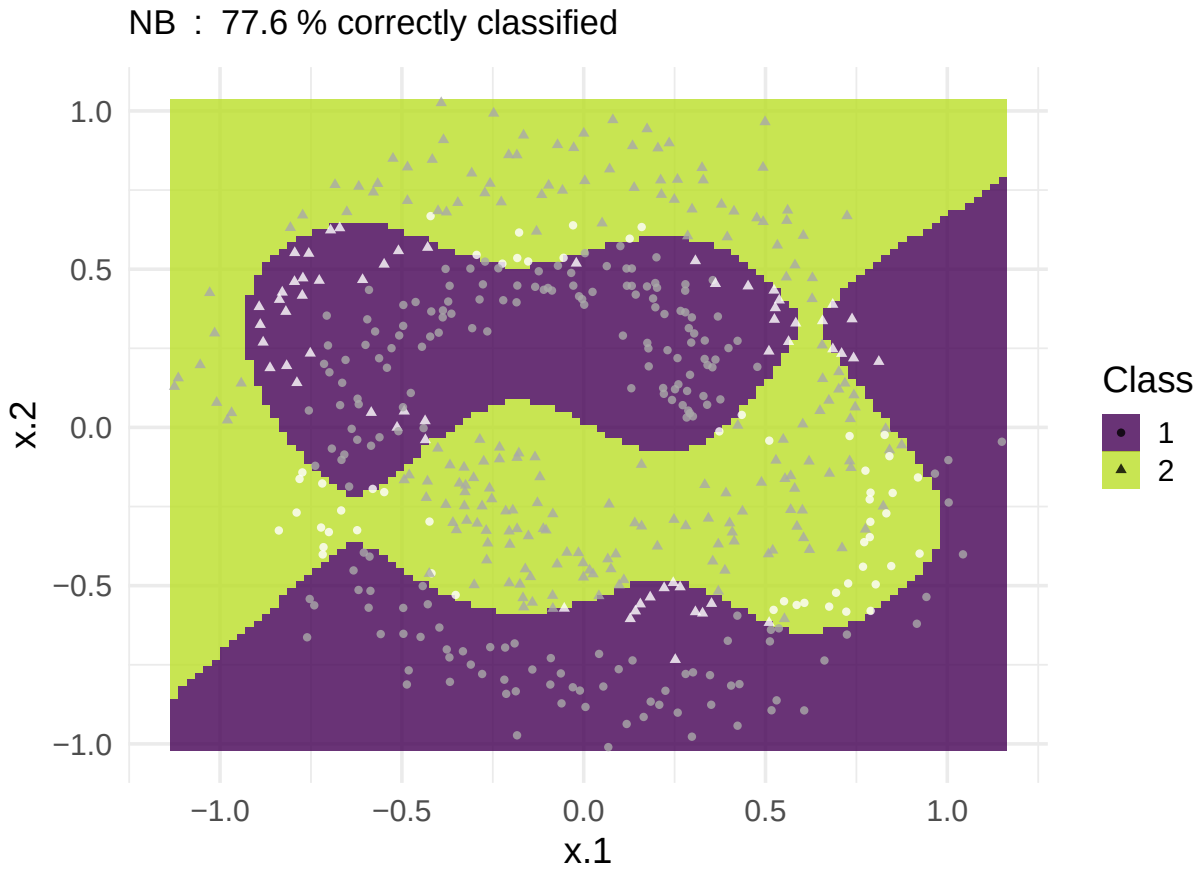


The final data set shows that many methods we talked about so far encounter problems with data sets in which the class boundaries are highly nonlinear, such as these `spirals`. Only the k-NN classifier is able to reconstruct the appropriate spiral shape of the decision boundaries. The other methods yield mean missclassification rates of around 50 percent.

But can't we do better with the methods we learnt? Yes, we can do the following:

- We can use the splines idea from splines code demo to transform our features into higher dimensional features and try to find better decision hyperplanes in this higher dimensional space, which often works. (In fact, if it is high-dimensional "enough", we can *always* find linear decision boundaries in that larger space that separate the classes perfectly. For many methods, we do not even need to explicitly compute these higher dimensional feature vectors in order to estimate the decision boundaries, this is called the kernel trick.)
- We can use kernel density estimations for the Bayes classifier, which we learnt in the generative classifier code demo:

```
library("kdensity")
library("EQL")

features <- c("x.1", "x.2")
target <- "classes"
X <- spirals[, features]
liks_kde <- get_naivebayes_likelihoods(
  spirals, target,
  sapply(features, function(feature)
    list(
      args = list(),
      features = feature,
      type = "kde"
    ),
  simplify = FALSE,
  USE.NAMES = TRUE
  )
)


priors <- get_priors(spirals, target)

plot_2D_naivebayes(priors, liks_kde, "x.1", "x.2",
  spirals$classes, X,
  title = '"NB" ~'
)
```



We *are* able to improve the classification results on the training data with this more flexible model, but the decision boundaries still don't look very convincing.. Can you figure out what could be the limiting problem for the Naive Bayes classifier here?
Hint: Think about its assumptions.