Exercise 4 – Classification II

Introduction to Machine Learning

Hint: Useful libraries

R

```
# you may need the following packages for this exercise sheet:

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

Python

```
# Consider the following libraries for this exercise sheet:

# general
import numpy as np
import pandas as pd
from scipy.stats import norm
# plotting
import matplotlib.pyplot as plt
import seaborn as sns
# sklearn
from sklearn.naive_bayes import CategoricalNB # import Naive Bayes Classifier for categori
from sklearn.naive_bayes import GaussianNB # import Naive Bayes Classifier for normal dist
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
```

Exercise 1: Naive Bayes

> Learning goals
>
> Compute Naive Bayes predictions by hand

You are given the following table with the target variable `Banana`:

ID	Color	Form	Origin	Banana
1	yellow	oblong	imported	yes
2	yellow	round	domestic	no
3	yellow	oblong	imported	no
4	brown	oblong	imported	yes
5	brown	round	domestic	no
6	green	round	imported	yes
7	green	oblong	domestic	no
8	red	round	imported	no

We want to use a Naive Bayes classifier to predict whether a new fruit is a `Banana` or not. Estimate the posterior probability $\hat{\pi}(\mathbf{x}_*)$ for a new observation $\mathbf{x}_* = (\text{yellow}, \text{round}, \text{imported})$. How would you classify the object?

Solution

When using the naive Bayes classifier, the features $\mathbf{x} := (x_{\text{Color}}, x_{\text{Form}}, x_{\text{Origin}})$ are assumed to be conditionally independent of each other, given the category $y = k \in \{\text{yes}, \text{no}\}$, s.t.

$$\begin{aligned}\mathbb{P}(\mathbf{x} \mid y = k) &= \mathbb{P}((x_{\text{Color}}, x_{\text{Form}}, x_{\text{Origin}}) \mid y = k) \\ &= \mathbb{P}(x_{\text{Color}} \mid y = k) \cdot \mathbb{P}(x_{\text{Form}} \mid y = k) \cdot \mathbb{P}(x_{\text{Origin}} \mid y = k).\end{aligned}$$

Recall Bayes' theorem:

$$\pi_k(\mathbf{x}) = \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

As the denominator is constant across all classes, the following holds for the posterior probabilities:

$$\begin{aligned}\pi_k(\mathbf{x}) &\propto \underbrace{\pi_k \cdot \mathbb{P}(x_{\text{Color}} \mid y = k) \cdot \mathbb{P}(x_{\text{Form}} \mid y = k) \cdot \mathbb{P}(x_{\text{Origin}} \mid y = k)}_{=:\alpha_k(x)} \\ &\iff \exists c \in \mathbb{R} : \pi_k(\mathbf{x}) = c \cdot \alpha_k(\mathbf{x})\end{aligned}$$

where $\pi_k = \mathbb{P}(y = k)$ is the prior probability of class k and c is the normalizing constant.

From this and since the posterior probabilities need to sum up to 1, we know that

$$1 = \; c \cdot \alpha_{\text{yes}}(\mathbf{x}) + c \cdot \alpha_{\text{no}}(\mathbf{x}) \iff c = \frac{1}{\alpha_{\text{yes}}(\mathbf{x}) + \alpha_{\text{no}}(\mathbf{x})}.$$

This means that, in order to compute $\pi_{\text{yes}}(\mathbf{x})$, the scores $\alpha_{\text{yes}}(\mathbf{x})$ and $\alpha_{\text{no}}(\mathbf{x})$ are needed.

Now we want to estimate for a new fruit the posterior probability $\hat{\pi}_{yes}((\text{yellow}, \text{round}, \text{imported}))$.

Obviously, we do not know the *true* prior probability and the *true* conditional densities. Here – since the target and the features are categorical – we use a categorical distribution, i.e., the simplest distribution over a g-way event that is fully specified by the individual probabilities for each class (which must of course sum to 1). This is a generalization of the Bernoulli distribution to the multi-class case. We can estimate the distribution parameters via the relative frequencies encountered in the data:

$$\begin{aligned}\hat{\alpha}_{\text{yes}}(\mathbf{x}_*) &= \hat{\pi}_{\text{yes}} \cdot \hat{\mathbb{P}}(x_{\text{Color}} = \text{yellow} \mid y = \text{yes}) \cdot \hat{\mathbb{P}}(x_{\text{Form}} = \text{round} \mid y = \text{yes}) \cdot \hat{\mathbb{P}}(x_{\text{Origin}} = \text{imp} \mid y = \text{yes}) \\ &= \frac{3}{8} \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot 1 = \frac{1}{24} \approx 0.042\end{aligned}$$

$$\begin{aligned}\hat{\alpha}_{\text{no}}(\mathbf{x}_*) &= \hat{\pi}_{\text{no}} \cdot \hat{\mathbb{P}}(x_{\text{Color}} = \text{yellow} \mid y = \text{no}) \cdot \hat{\mathbb{P}}(x_{\text{Form}} = \text{round} \mid y = \text{no}) \cdot \hat{\mathbb{P}}(x_{\text{Origin}} = \text{imp} \mid y = \text{no}) \\ &= \frac{5}{8} \cdot \frac{2}{5} \cdot \frac{3}{5} \cdot \frac{2}{5} = \frac{3}{50} = 0.060\end{aligned}$$

At this stage we can already see that the predicted label is "no", since $\hat{\alpha}_{\text{no}}(\mathbf{x}_*) = 0.060 > \frac{1}{24} = \hat{\alpha}_{\text{yes}}(\mathbf{x}_*)$ – that is, if we threshold at 0.5 for predicting "yes".

With the above we can compute the posterior probability

$$\hat{\pi}_{\text{yes}}(\mathbf{x}_*) = \frac{\hat{\alpha}_{\text{yes}}(\mathbf{x}_*)}{\hat{\alpha}_{\text{yes}}(\mathbf{x}_*) + \hat{\alpha}_{\text{no}}(\mathbf{x}_*)} \approx 0.410 < 0.5$$

and check our calculations against the corresponding results:

R

```r
df_banana <- data.frame(
  color = as.factor(
    c("yellow", "yellow", "yellow", "brown", "brown", "green", "green", "red")),
  form = as.factor(
    c("oblong", "round", "oblong", "oblong", "round", "round", "oblong", "round")),
  origin = as.factor(
    c("imported", "domestic", "imported", "imported", "domestic", "imported",
    "domestic", "imported")),
  banana = as.factor(c("yes", "no", "no", "yes", "no", "yes", "no", "no")))

new_fruit <- data.frame(color = "yellow", form = "round", origin = "imported")

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

banana_task <- TaskClassif$new(
  id = "banana",
  backend = df_banana,
  target = "banana")

nb_learner$train(banana_task)
nb_learner$predict_newdata(new_fruit)
```

```
<PredictionClassif> for 1 observations:
 row_ids truth response   prob.no  prob.yes
       1  <NA>       no 0.5901639 0.4098361
```

Python

```
# initializing the NB
classifer = CategoricalNB(alpha=1.0e-10) # alpha = 0 for no smoothing towards uniform dist

# training the model
classifer.fit(X, y)

# testing the model
y_pred = classifer.predict(x_new)
print("Prediction (0 = no, 1 = yes)", y_pred)
# Prediction is "not Banana"

y_prop = classifer.predict_proba(x_new)
print("Probabilities for (no - yes)", y_prop)
```

```
Prediction (0 = no, 1 = yes) [0]
Probabilities for (no - yes) [[0.59016393 0.40983607]]
```

Assume you have an additional feature `Length` that measures the length in cm. Describe in 1-2 sentences how you would handle this numeric feature with Naive Bayes.

Solution

Before, we only had categorical features and could use the empirical frequencies as our parameters in a categorical distribution. For the distribution of a numerical feature, given the the category, we need to define a probability distribution with continuous support. A popular choice is to use Gaussian distributions. For example, for the information x_{Length} we could assume that

$$\mathbb{P}(x_{\text{Length}} \mid y = \text{yes}) \sim \mathcal{N}(\mu_{\text{yes}}, \sigma^2_{\text{yes}})$$

and

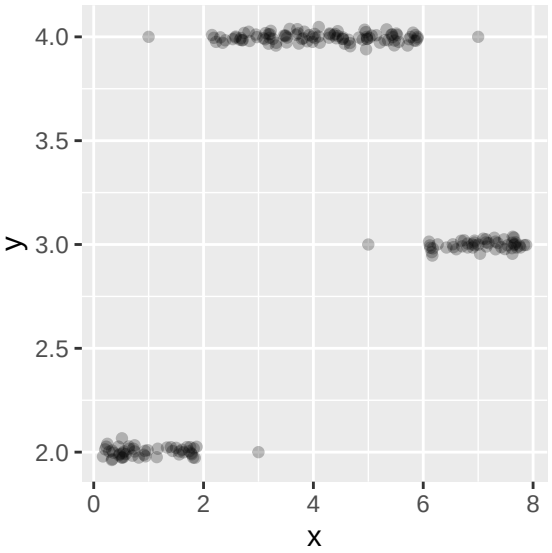
$$\mathbb{P}(x_{\text{Length}} \mid y = \text{no}) \sim \mathcal{N}(\mu_{\text{no}}, \sigma^2_{\text{no}})$$

In order to fully specify these normal distributions we need to estimate their parameters $\mu_{\text{yes}}, \mu_{\text{no}}, \sigma^2_{\text{yes}}, \sigma^2_{\text{no}}$ from the data via the usual estimators (empirical mean and empirical variance with bias correction).

Exercise 2: Discriminant analysis

Learning goals

1) Set up discriminant analysis by hand
2) Make predictions with discriminant analysis
3) Discuss difference between LDA and QDA



The above plot shows $\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$, a data set with $n = 200$ observations of a continuous target variable y and a continuous, 1-dimensional feature variable $\mathbf{x}$. In the following, we aim at predicting y with a machine learning model that takes $\mathbf{x}$ as input.

To prepare the data for classification, we categorize the target variable y in 3 classes and call the transformed target variable z, as follows:

$$z^{(i)} = \begin{cases} 1, & y^{(i)} \in (-\infty, 2.5] \\ 2, & y^{(i)} \in (2.5, 3.5] \\ 3, & y^{(i)} \in (3.5, \infty) \end{cases}$$

Now we can apply quadratic discriminant analysis (QDA):

Estimate the class means $\mu_k = \mathbb{E}(\mathbf{x}|z = k)$ for each of the three classes $k \in \{1, 2, 3\}$ visually from the plot. Do not overcomplicate this, a rough estimate is sufficient here.

Solution

As the data seem to be pretty symmetric conditional on the respective class, we estimate the class means to lie roughly in the middle of the data clusters: $\hat{\mu}_1 = 1$, $\hat{\mu}_2 = 7$, $\hat{\mu}_3 = 4$.

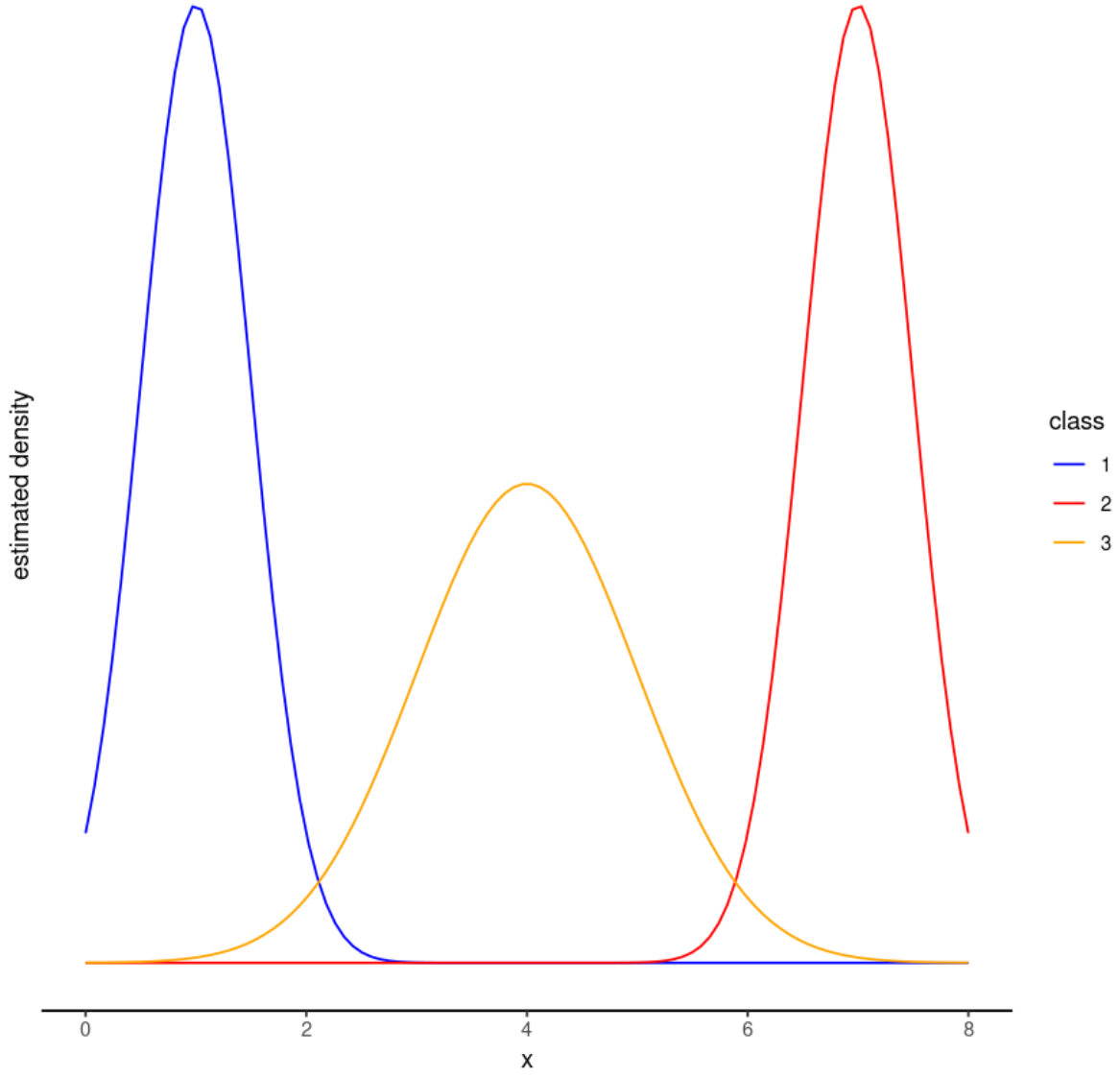
Make a plot that visualizes the different estimated densities per class.

Solution

We see that the variances in classes 1 and 2 are similar and also much smaller than in class 3. Therefore, the densities could look roughly like this:

R

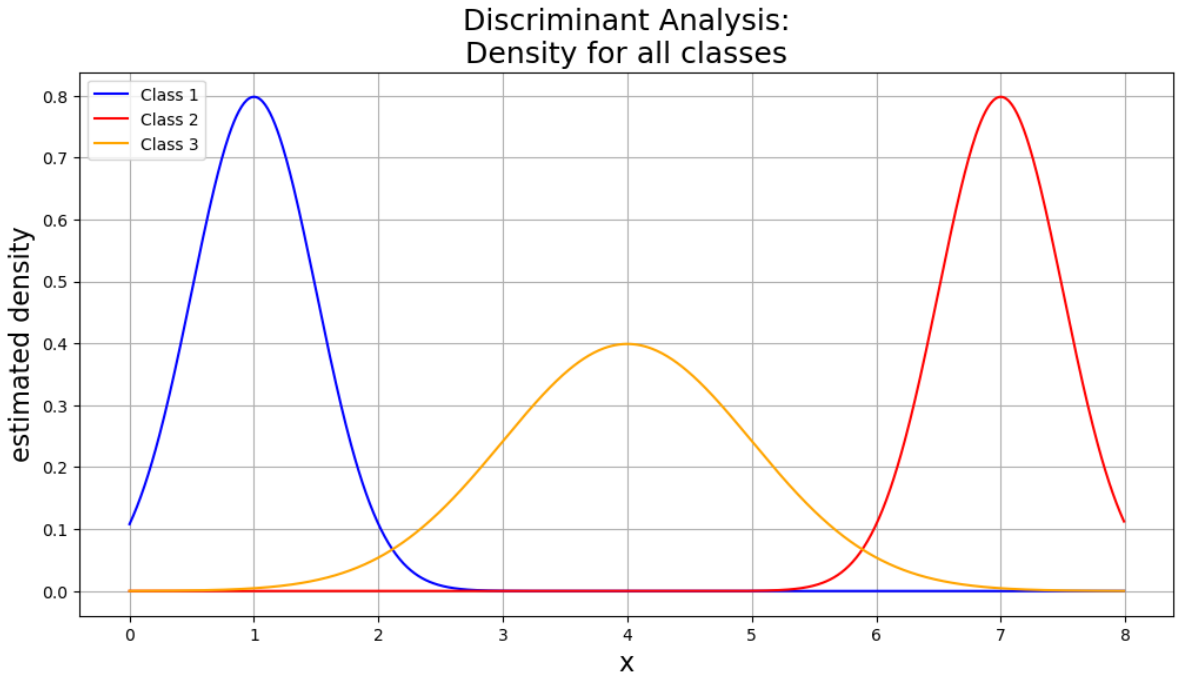
```r
library(ggplot2)

colors <- c("1" = "blue", "2" = "red", "3" = "orange")

ggplot(data.frame(x = c(0, 8)), aes(x = x)) +
  stat_function(fun = dnorm, n = 100, args = list(mean = 1, sd = 0.5), aes(col = "1")) +
  stat_function(fun = dnorm, n = 100, args = list(mean = 7, sd = 0.5), aes(col = "2")) +
  stat_function(fun = dnorm, n = 100, args = list(mean = 4, sd = 1), aes(col = "3")) +
  scale_color_manual("class", values = colors) +
  theme_classic() +
  ylab("estimated density") +
  scale_y_continuous(breaks = NULL)
```



Python

```
# Plot differences

#x-axis ranges from -3 and 3 with .001 steps
x = np.arange(0, 8, 0.01)
```

```
#plot normal distribution with mean 0 and standard deviation 1
plt.figure(figsize=(12, 6))
plt.plot(x, norm.pdf(x, 1, 0.5), color = 'blue', label = 'Class 1')
plt.plot(x, norm.pdf(x, 7, 0.5), color = 'red', label = 'Class 2')
plt.plot(x, norm.pdf(x, 4, 1), color = 'orange', label = 'Class 3')

# title & label axes
plt.grid(True)
plt.title('Discriminant Analysis:\nDensity for all classes', size=18)
plt.xlabel('x', size=16)
plt.ylabel('estimated density', size=16)
plt.legend(loc='upper left', prop={'size': 10})
plt.show()
```



How would your plot from ii) change if we used linear discriminant analysis (LDA) instead of QDA? Explain your answer.

Solution

Since LDA assumes constant variances across all classes (also if this does not reflect the data situation), all densities would have the same shape and only differ in location.

Why is QDA preferable over LDA for this data?

Solution

As we have already noted, the assumption of equal class variances is not justified here, but LDA is confined to equivariant distributions. Therefore, the more flexible QDA is preferable in this case. Note, however, that the Gaussian distributions both variants of discriminant analysis use might not be perfectly appropriate, as the data seems to be more uniformly distributed (conditional on the classes).

Given are two new observations $\mathbf{x}_{*1} = -10$ and $\mathbf{x}_{*2} = 7$. Assuming roughly equal class sizes, state the prediction for QDA and explain how you arrive there.

Solution

Assuming equal class sizes means it's sufficient to compare the class-wise densities at the two points of interest. The prediction for $\mathbf{x}_{*1}$ will probably be $\hat{z}_{*1} = 3$ because the density of class 3 has much larger variance and will therefore overshoot the density of class 1. For $\mathbf{x}_{*2}$ the case is clear and we have $\hat{z}_{*2} = 2$.

Exercise 3: Decision boundaries for classification learners

> Learning goals
>
> Get a feeling for decision boundaries produced by LDA/QDA/NB

We will now visualize how well different learners classify the three-class `mlbench::mlbench.cassini` data set.

- Generate 1000 points from `cassini` using R or import `cassini_data.csv` in Python.
- Then, perturb the `x.2` dimension with Gaussian noise (mean 0, standard deviation 0.5), and consider the classifiers already introduced in the lecture:
  - LDA (Linear Discriminant Analysis),
  - QDA (Quadratic Discriminant Analysis), and
  - Naive Bayes.

Plot the learners' decision boundaries. Can you spot differences in separation ability?

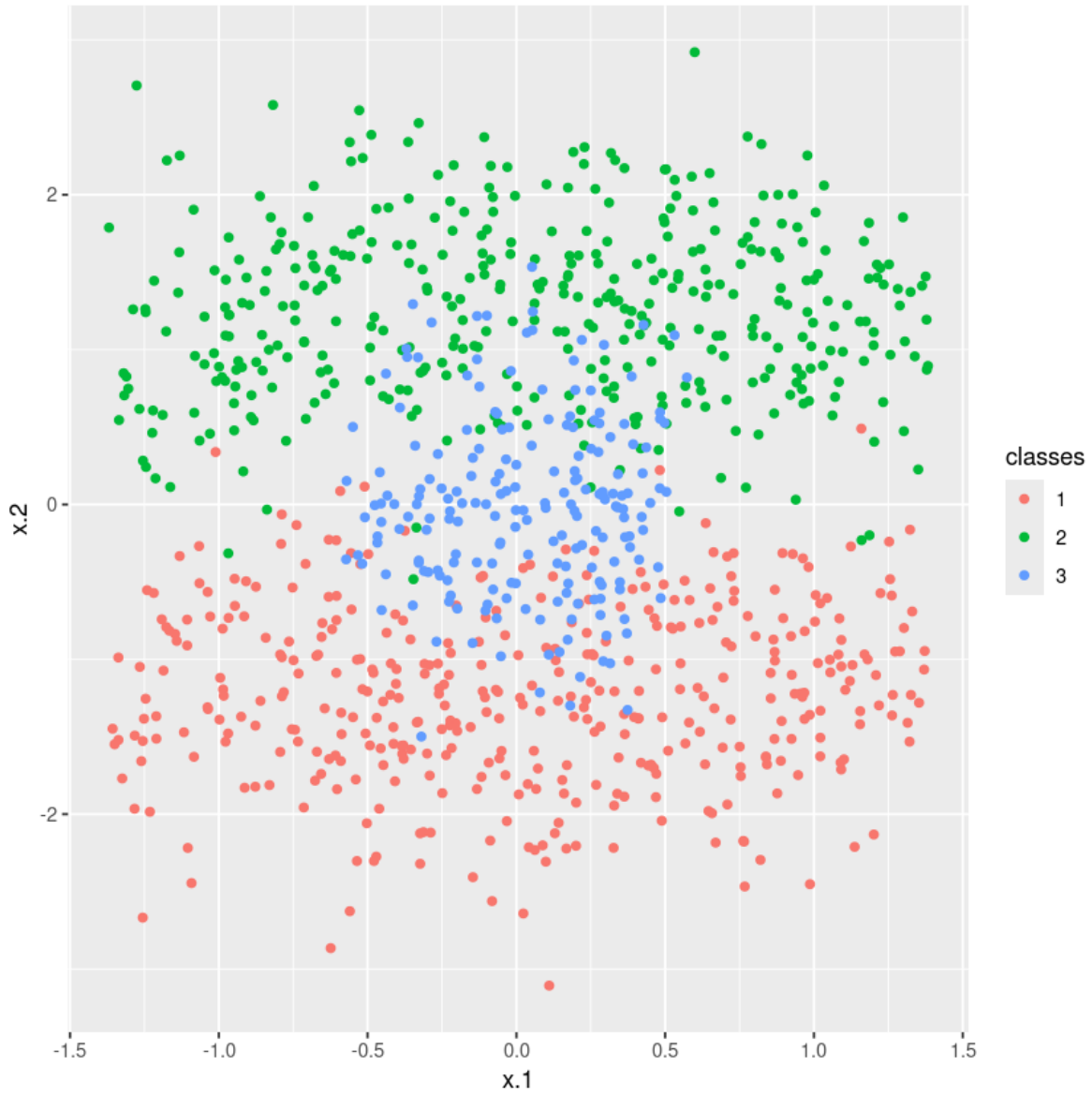
(Note that logistic regression cannot handle more than two classes and is therefore not listed here.)

Solution

First, visualize the simulated `cassini` data:

R

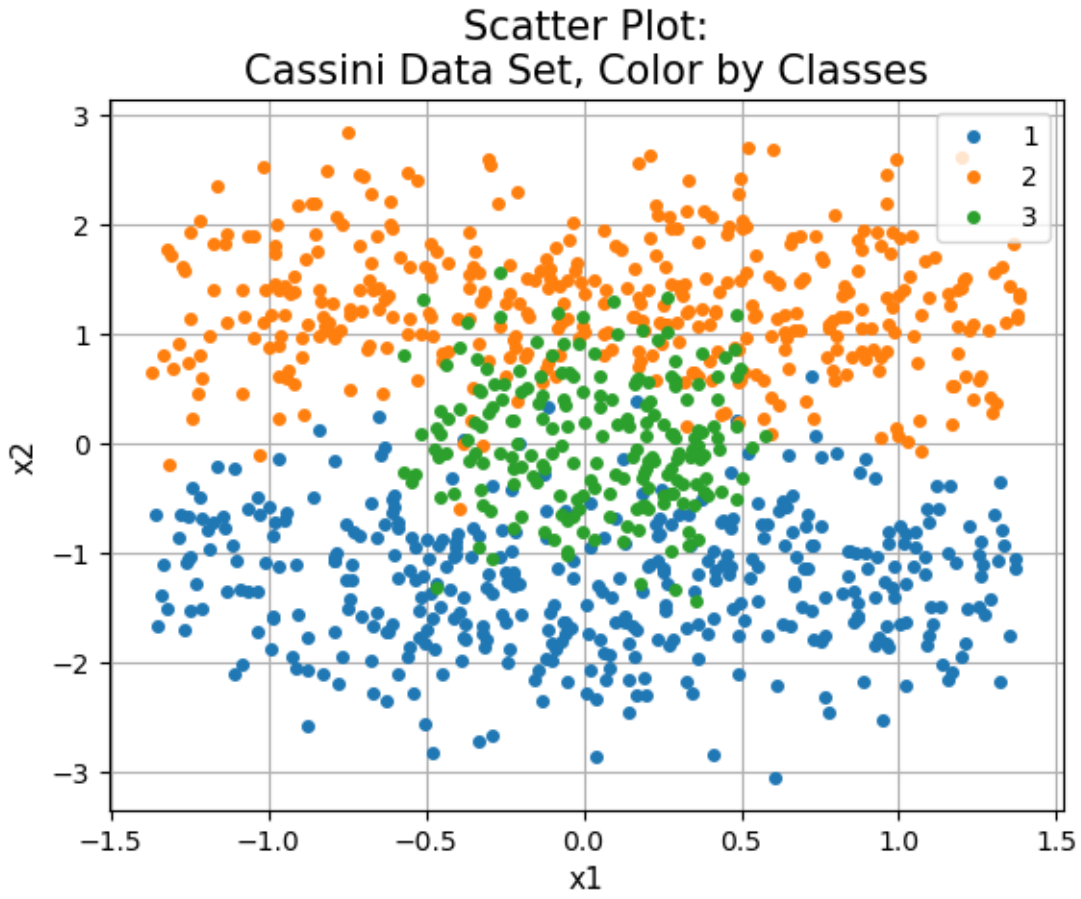
```
# Make sure all necessary packages are loaded
# and visualize the simulated cassini data:

# simulate data
set.seed(123L)
data_raw <- mlbench::mlbench.cassini(n = 1000)
df <- as.data.frame(data_raw)
df$x.2 <- df$x.2 + rnorm(nrow(df), sd = 0.5)

# visualize
ggplot2::ggplot(df, aes(x = x.1, y = x.2, col = classes)) +
  geom_point()
```



Python

```
# Plot data by group
groups = cassini.groupby('classes')
for name, group in groups:
    plt.plot(group["x.1"], group["x.2"], marker='o', linestyle='', markersize=4, label=nam
```

```
plt.legend()
plt.grid(True)
plt.title('Scatter Plot:\nCassini Data Set, Color by Classes', size=15)
plt.xlabel('x1', size=11)
plt.ylabel('x2', size=11)
plt.show()
```



Then, define the learners (and the `task` for `R`) and train the models:

R

```r
# Create the task and train all 3 Learners:

# create task
task <- mlr3::TaskClassif$new(
  id = "spirals_task",
  backend = df,
  target = "classes")

# define learners
learners <- list(
  mlr3::lrn("classif.lda"),
  mlr3::lrn("classif.qda"),
  mlr3::lrn("classif.naive_bayes"))
```

Python

```python
# Train all 3 models
X_cass = cassini.iloc[:, 0:2].values
y_cass = cassini.iloc[:, 2].values

# LDA
lda = LDA()
lda.fit(X_cass, y_cass)

# QDA
qda = QDA()
qda.fit(X_cass, y_cass)

# Naive Bayes
# Use Gaussian Naive Bayes
gnb = GaussianNB(var_smoothing=0) # no smoothing wanted here
gnb.fit(X_cass, y_cass)
```

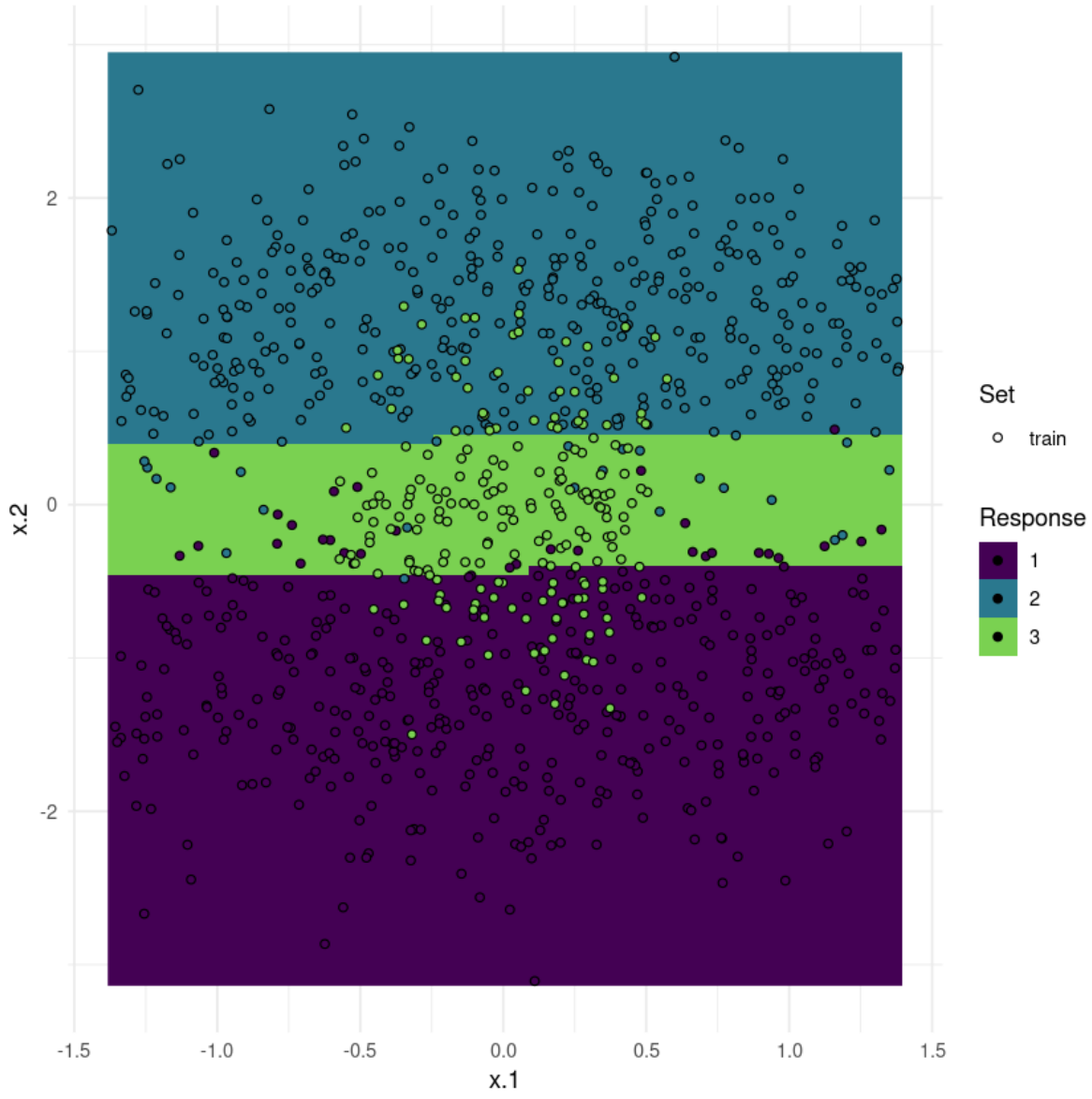
```
GaussianNB(var_smoothing=0)
```

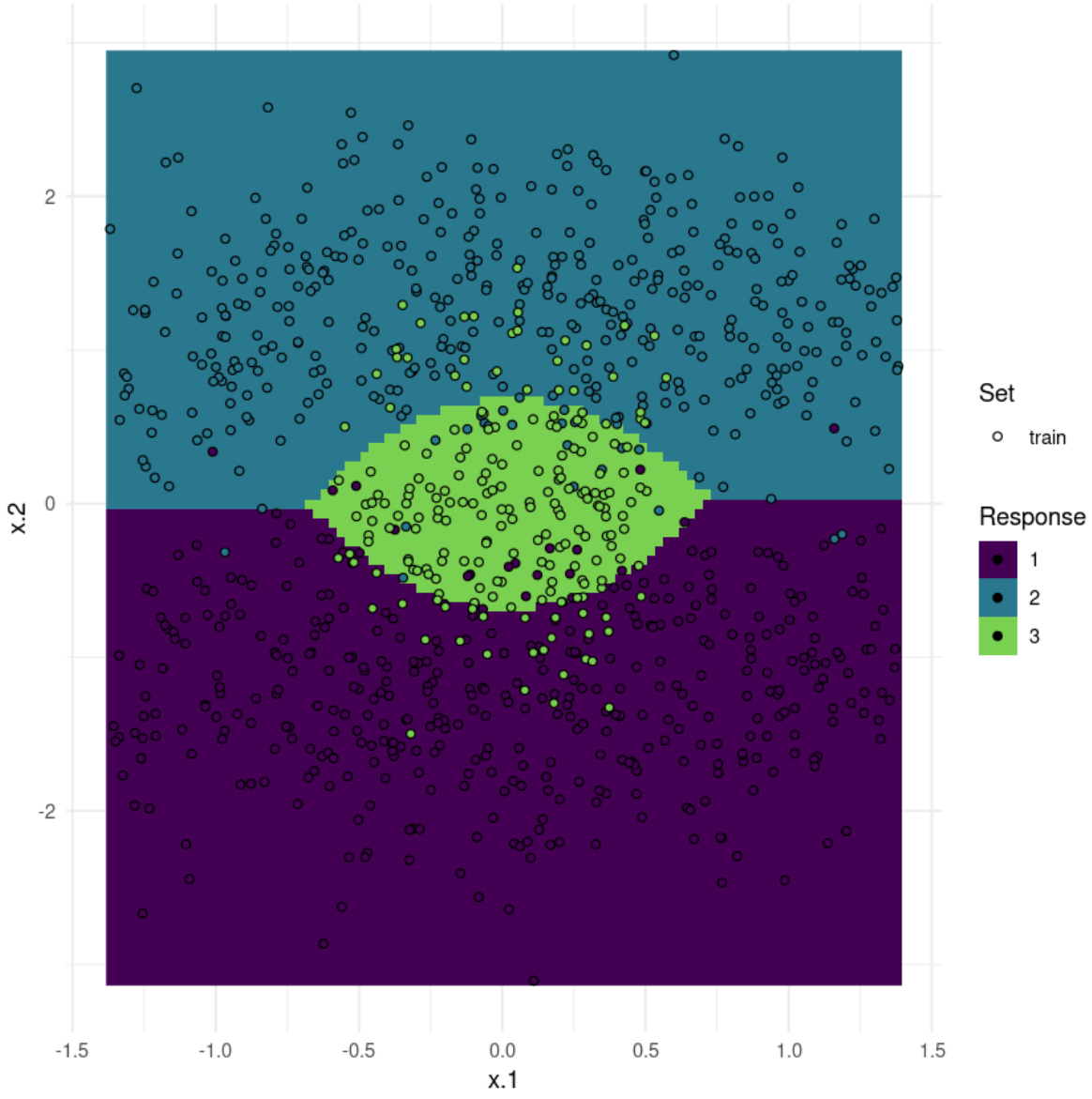
Lastly, plot their decision boundaries:

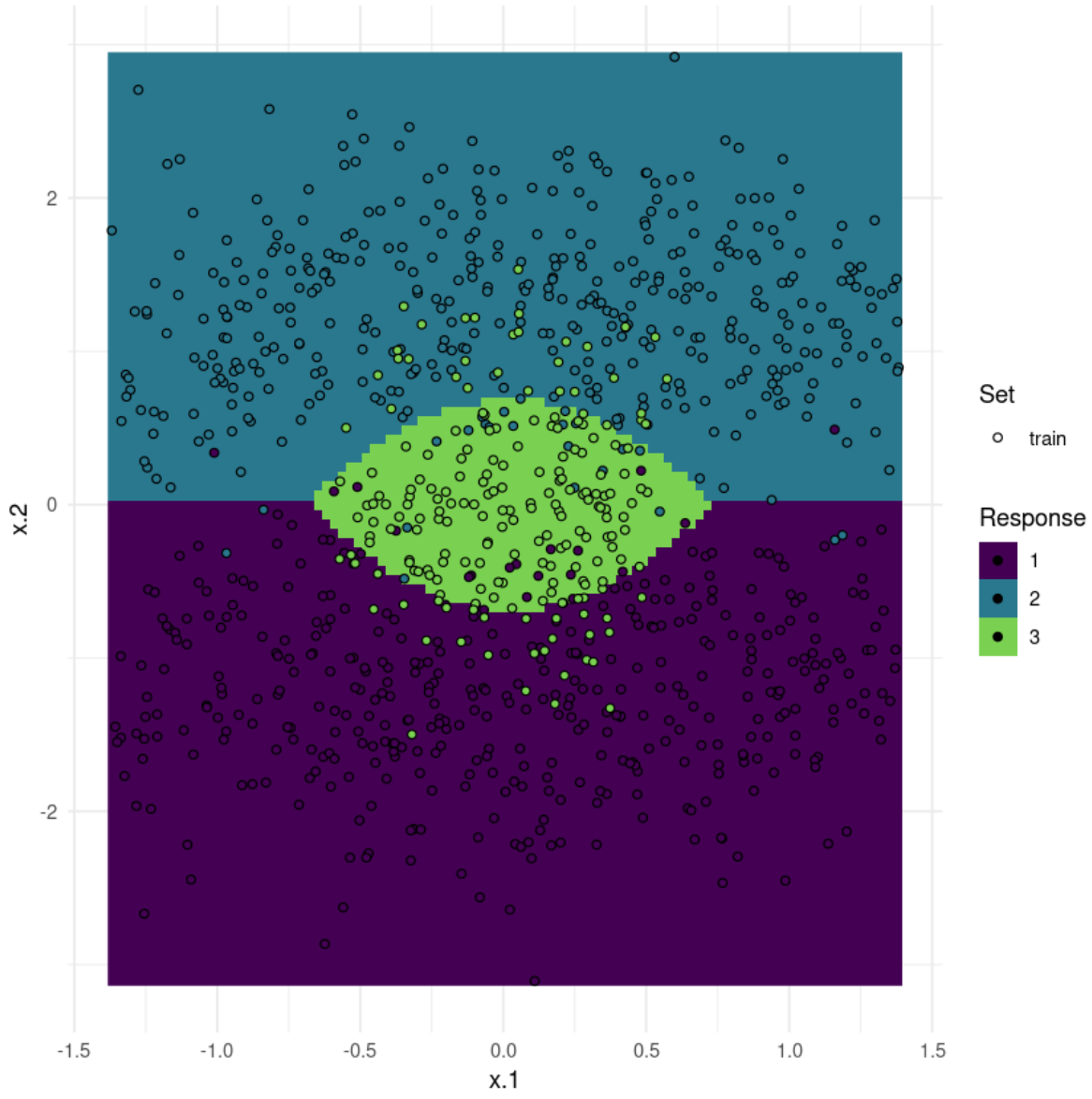
R

```
# train and plot decision boundaries
plots <- lapply(learners, function(i) mlr3viz::plot_learner_prediction(i, task))
for (i in plots) print(i)
```

```
INFO  [15:46:17.685] [mlr3] Applying learner 'classif.lda' on task 'spirals_task' (iter 1/1)
INFO  [15:46:17.838] [mlr3] Applying learner 'classif.qda' on task 'spirals_task' (iter 1/1)
INFO  [15:46:17.947] [mlr3] Applying learner 'classif.naive_bayes' on task 'spirals_task' (it
```


2
0
-2
x.2
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
x.1
Set
train
Response
1
2
3

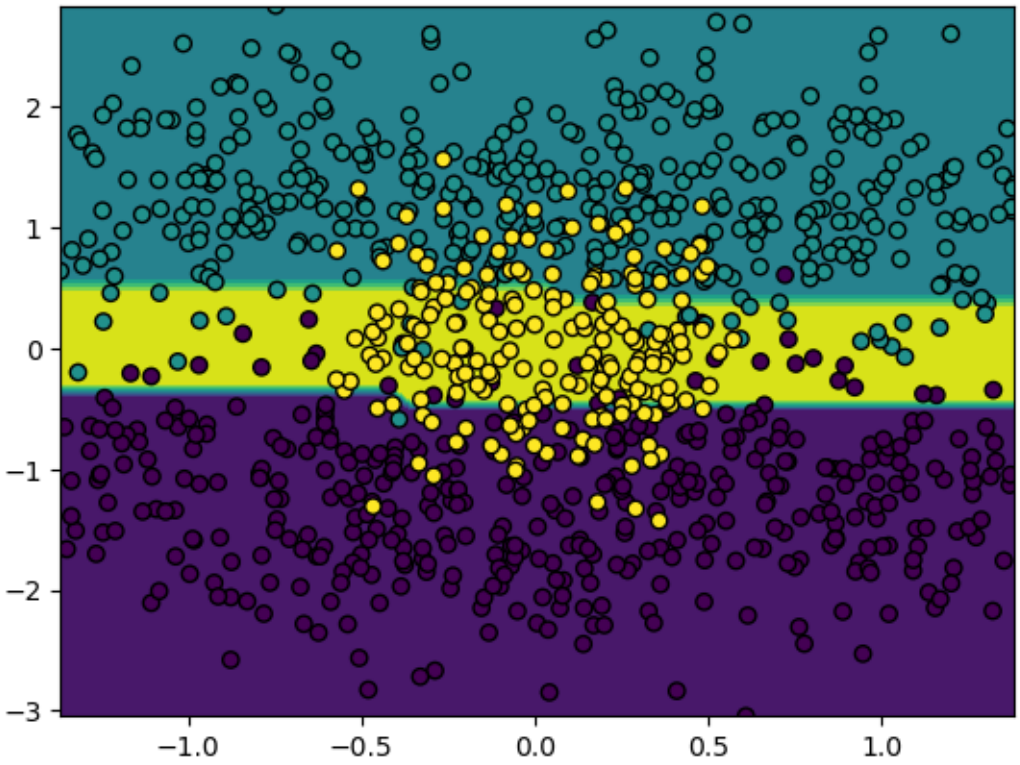

2
0
-2
x.2
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
x.1
Set
train
Response
1
2
3

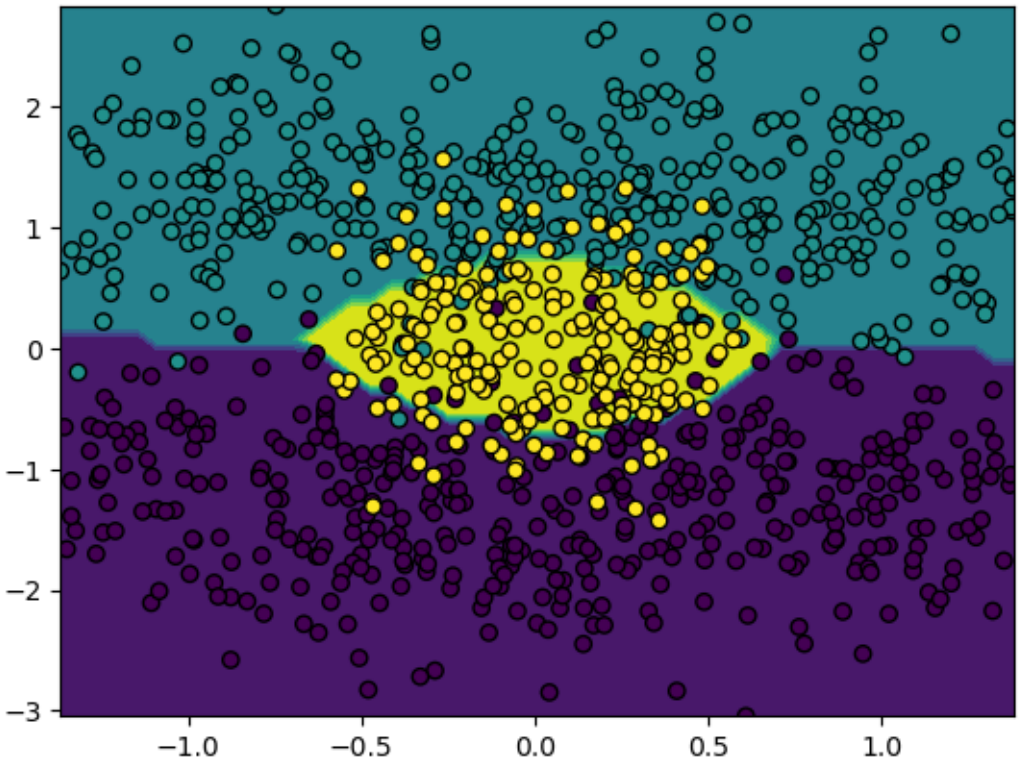


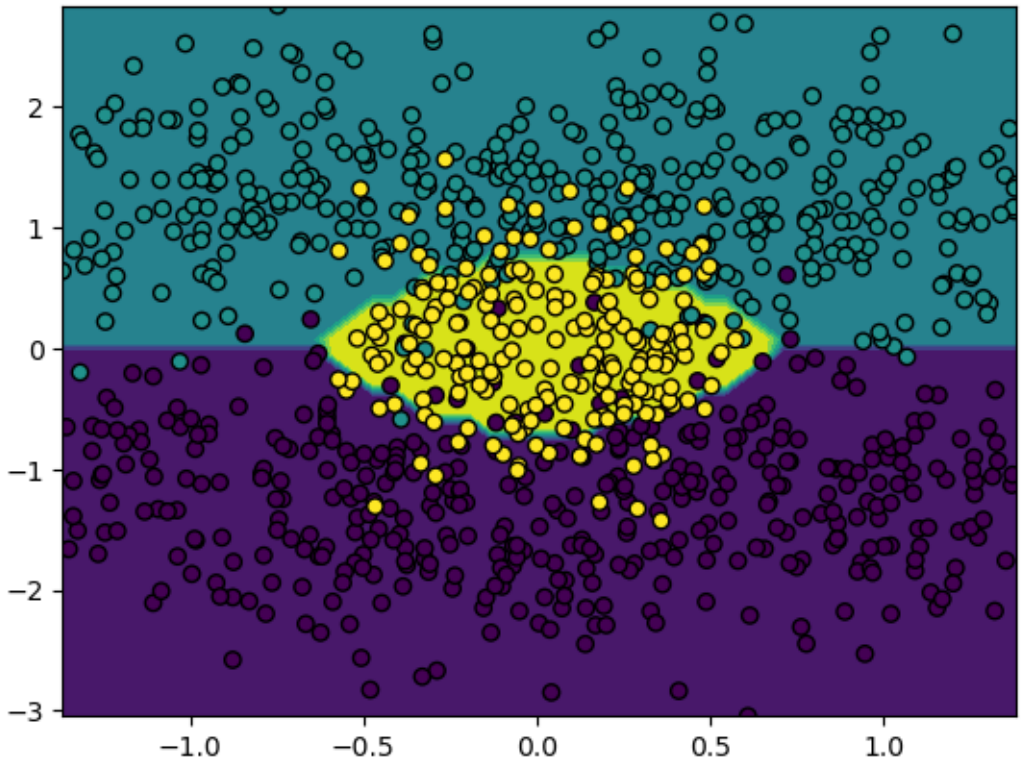
Python

```
# Plot Decision Boundaries
def plot_dec_bound(obj):
    """
    Method to produce Decision Boundary Plots
    Input: trained classifier object
```

```
    Output: -
    """
    feature_1, feature_2 = np.meshgrid(
        np.linspace(X_cass[:, 0].min(), X_cass[:, 0].max()),
        np.linspace(X_cass[:, 1].min(), X_cass[:, 1].max()))
    grid = np.vstack([feature_1.ravel(), feature_2.ravel()]).T

    y_pred = np.reshape(obj.predict(grid), feature_1.shape)
    display = DecisionBoundaryDisplay(xx0=feature_1, xx1=feature_2, response=y_pred)
    display.plot()

    display.ax_.scatter(X_cass[:, 0], X_cass[:, 1], c=y_cass, edgecolor="black")

    plt.show()


class_list = [lda, qda, gnb]

for obj in class_list:
    plot_dec_bound(obj)
```


2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0


2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0



We see how LDA, with its confinement to linear decision boundaries, is not able to classify the data very well. QDA and NB, on the other hand, get the shape of the boundaries right. It also becomes obvious that NB is a quadratic classifier just like QDA - their decision surfaces look pretty much alike.