Lab 11: Support Vector Machines and the ROC curve

Johnny Hong

Stat 154, Fall 2017

Introduction

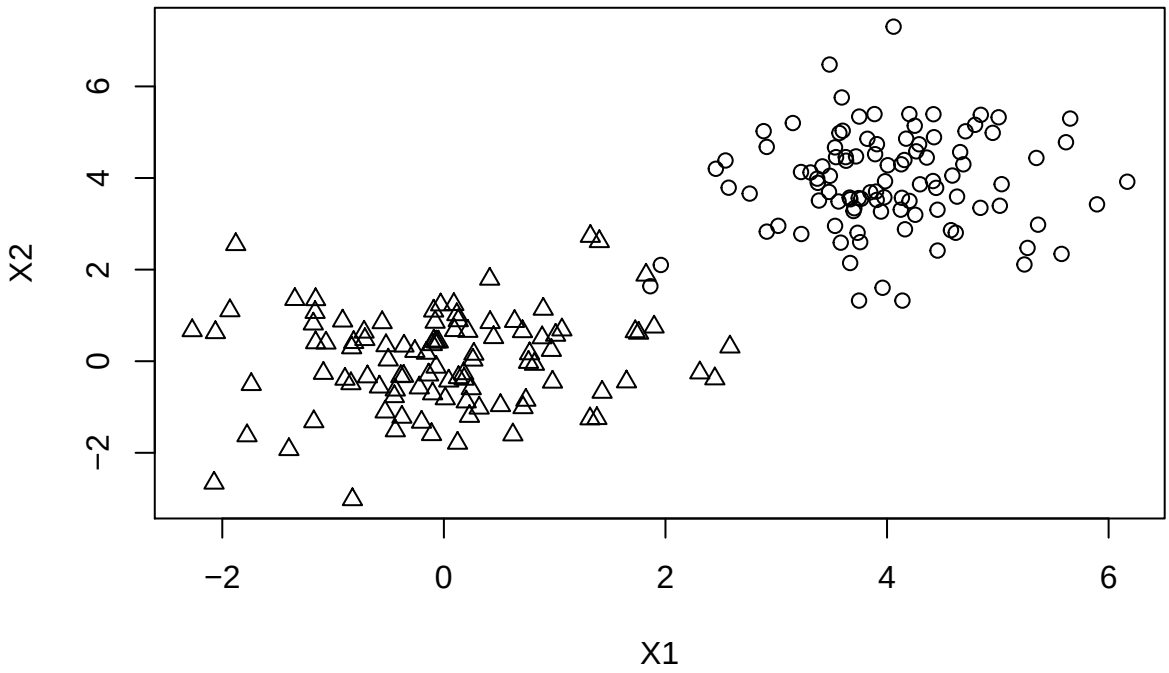
In this lab, we will explore the *support vector machine* (SVM), a popular approach for classification problems. We will use two simulated datasets throughout the lab to illustrate different aspects of SVMs. In each of the datasets, the repsonse is `y` and the predictors are `X1` and `X2`. The response `y` is a categorical variable indicating the class the observation belongs to and `X1` and `X2` are assumed to be continuous.

```
set.seed(100)
X1 <- c(rnorm(100), rnorm(100, mean = 4))
X2 <- c(rnorm(100), rnorm(100, mean = 4))
y <- factor(c(rep(0,100), rep(1,100)))
df1 <- data.frame(X1, X2, y)

set.seed(200)
r <- c(runif(100, 1, 2), runif(100, 5, 6))
theta <- runif(200, 0, 2 * pi)
X1 <- r * cos(theta) + rnorm(200)
X2 <- r * sin(theta) + rnorm(200)
y <- factor(c(rep(0,100), rep(1,100)))
df2 <- data.frame(X1, X2, y)
```

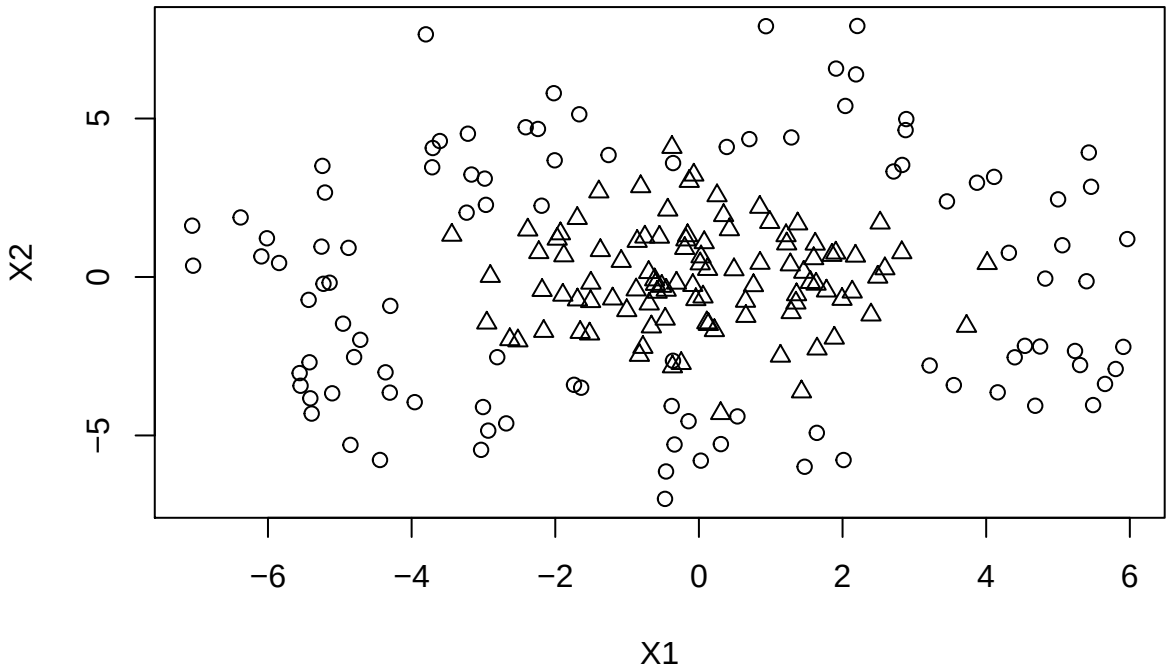
Your turn

- For each dataset, make a scatterplot of `X2` against `X1`. Use a different plotting character for each class.
- Comment on the scatterplots.

```
pchs <- c(2,1)
with(df1, plot(X1, X2, pch = pchs[y]))
```




```
pchs <- c(2,1)
with(df2, plot(X1, X2, pch = pchs[y]))
```



Support Vector Classifier

The maximal margin classifier requires the dataset to be linearly separable, which is typically too restrictive in practice. The support vector classifier introduces slack variables to handle

this issue, and it has been shown empirically that this reduces overfitting and improve the robustness of the classifier. In particular, the support vector classifier is determined by solving the following optimization problem:

$$\max_{\beta_0,...,\beta_p,\epsilon_1,...,\epsilon_n} M$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$
$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

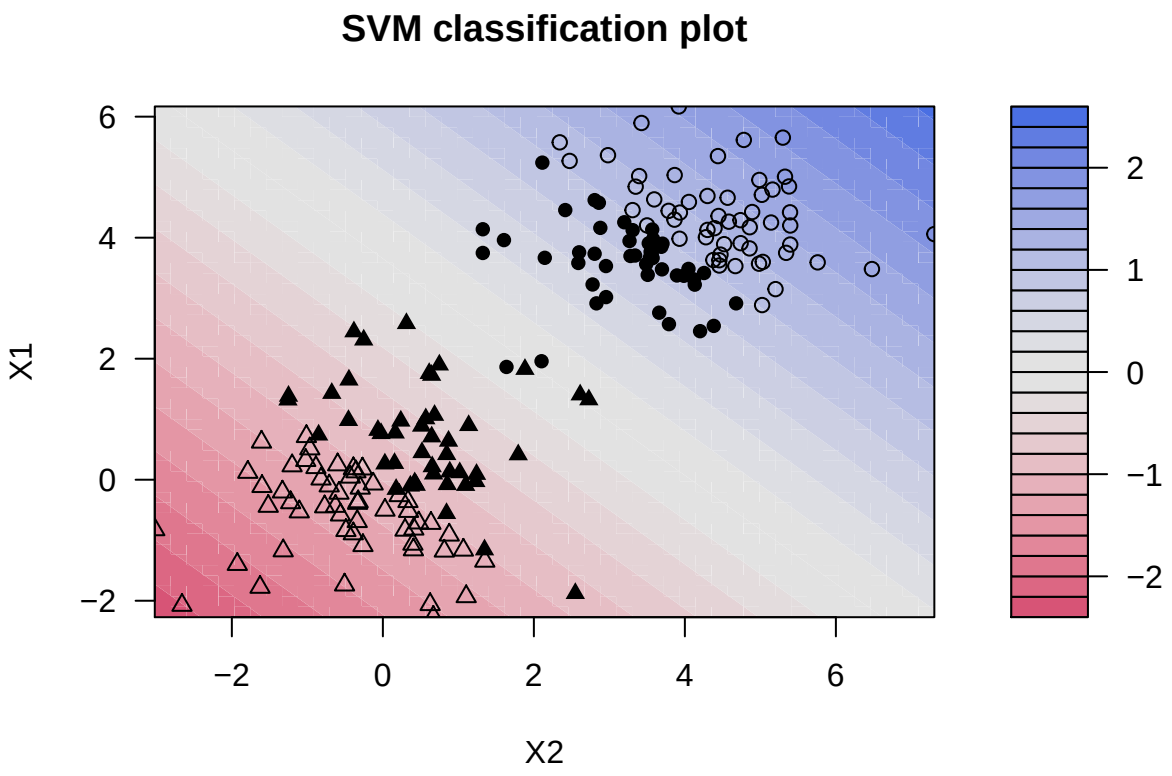
where $\epsilon_1, ..., \epsilon_n$ are slack variables and C is a nonnegative hyperparameter. Let's investigate the impact of C on the resulting support vector classifiers.

Your turn

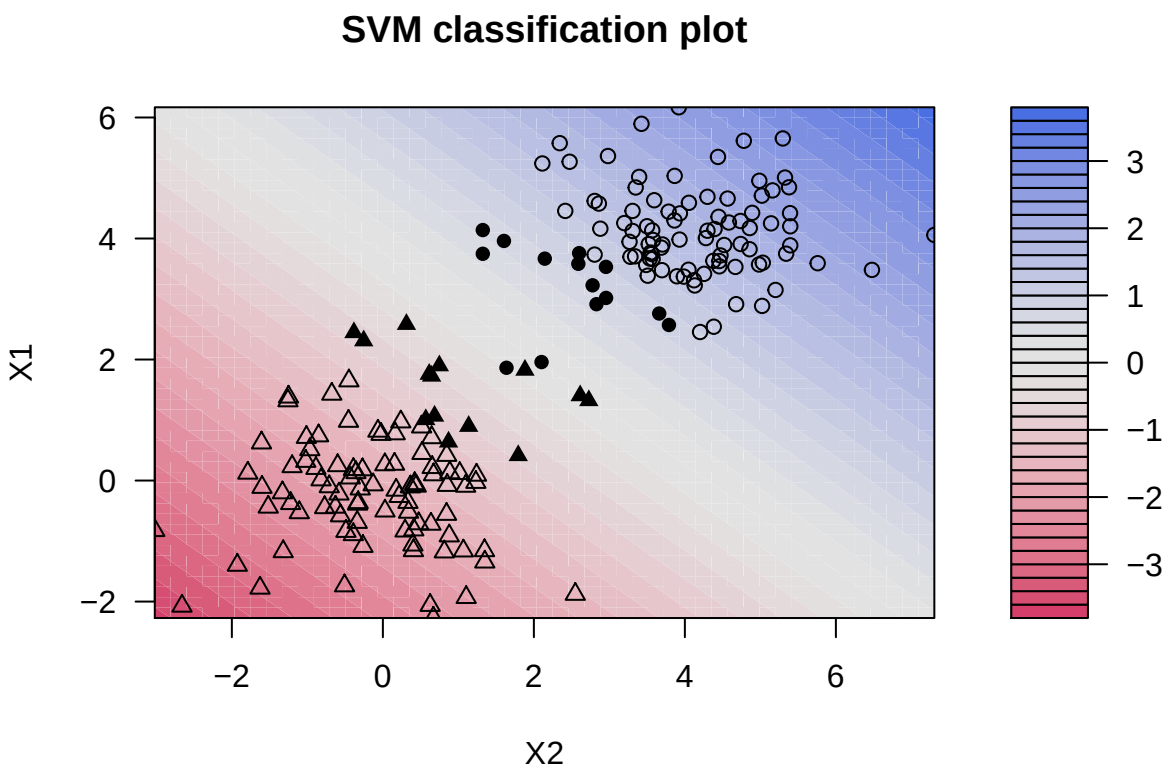
- For each dataset,
  1. fit a support vector classifier using `ksvm()` with `kernel="vanilladot"` and `C=0.01` and save the `ksvm` object. The `ksvm()` function is in the R package `kernlab`.
  2. use `plot()` with the `ksvm()` object and the argument `data=dfx`, where `dfx` is the dataset. Describe what you see in the plot.
  3. repeat 1. and 2. with $C \in \{0.1, 1, 10, 100, 1000, 10000\}$.
- As C increases, how does the number of support vectors change?

```
C_vector <- c(0.01, 0.1, 1, 10, 100, 1000, 10000)
dataset_list <- list(df1, df2)

for (df in dataset_list) {
  for (C in C_vector) {
    fit <- ksvm(y~X1+X2, data = df, kernel = "vanilladot", C=C)
    plot(fit, data=df)
  }
}
```

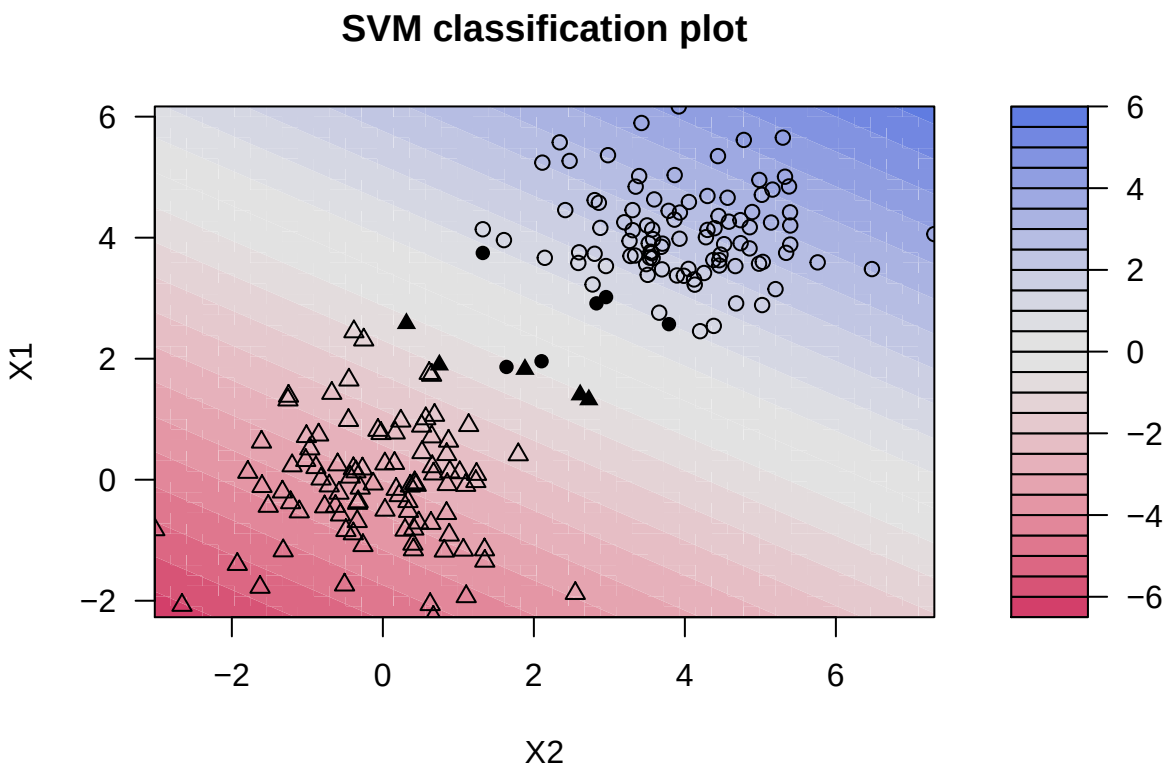
```
##  Setting default kernel parameters
```



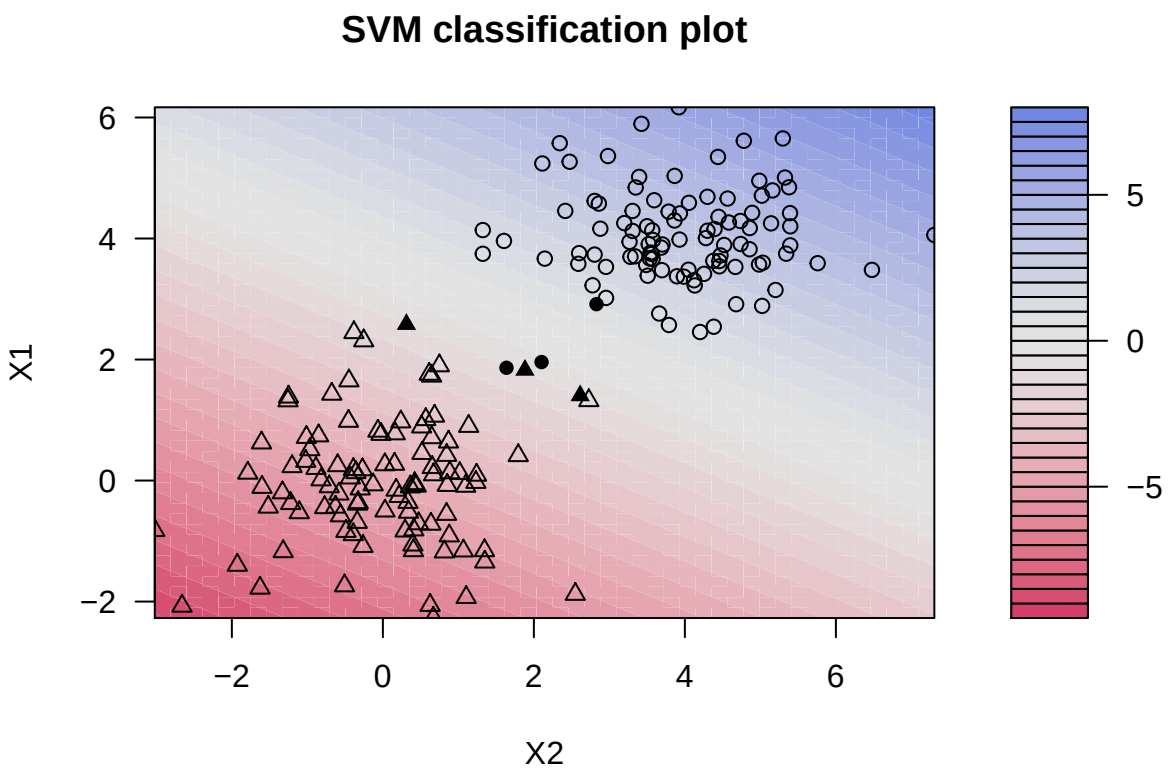

```
##  Setting default kernel parameters
```



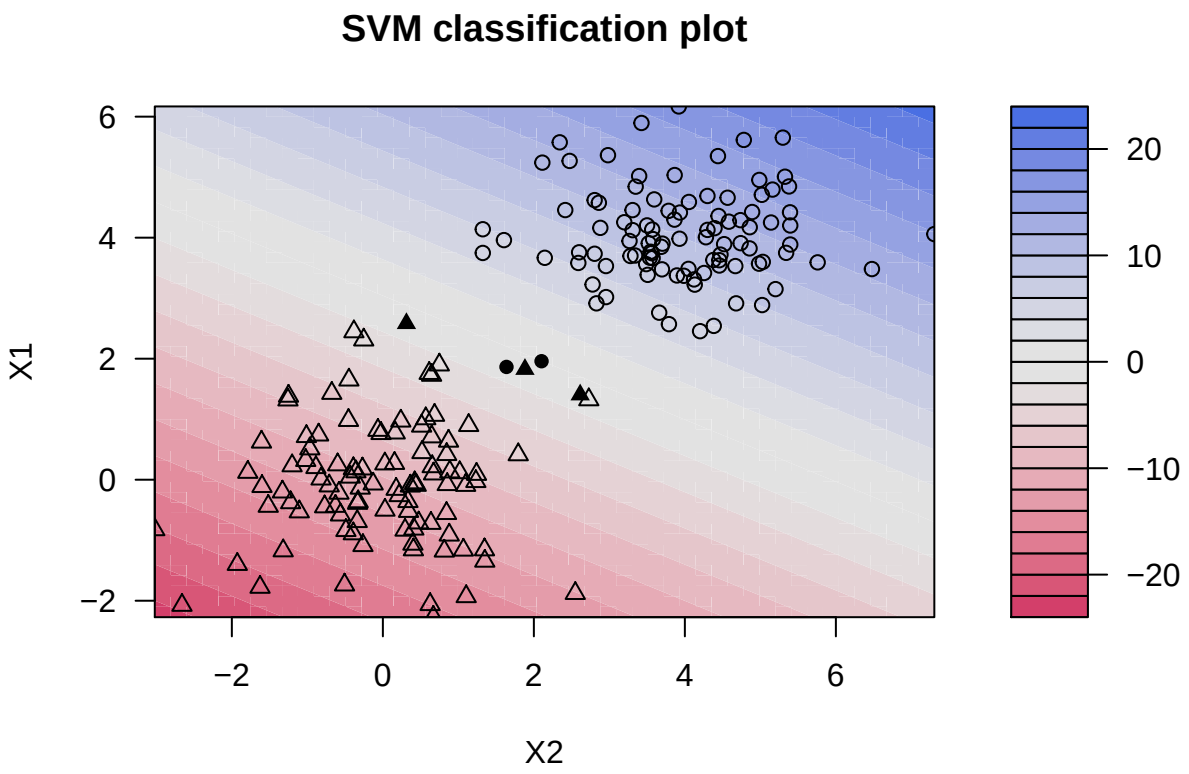

```
##  Setting default kernel parameters
```



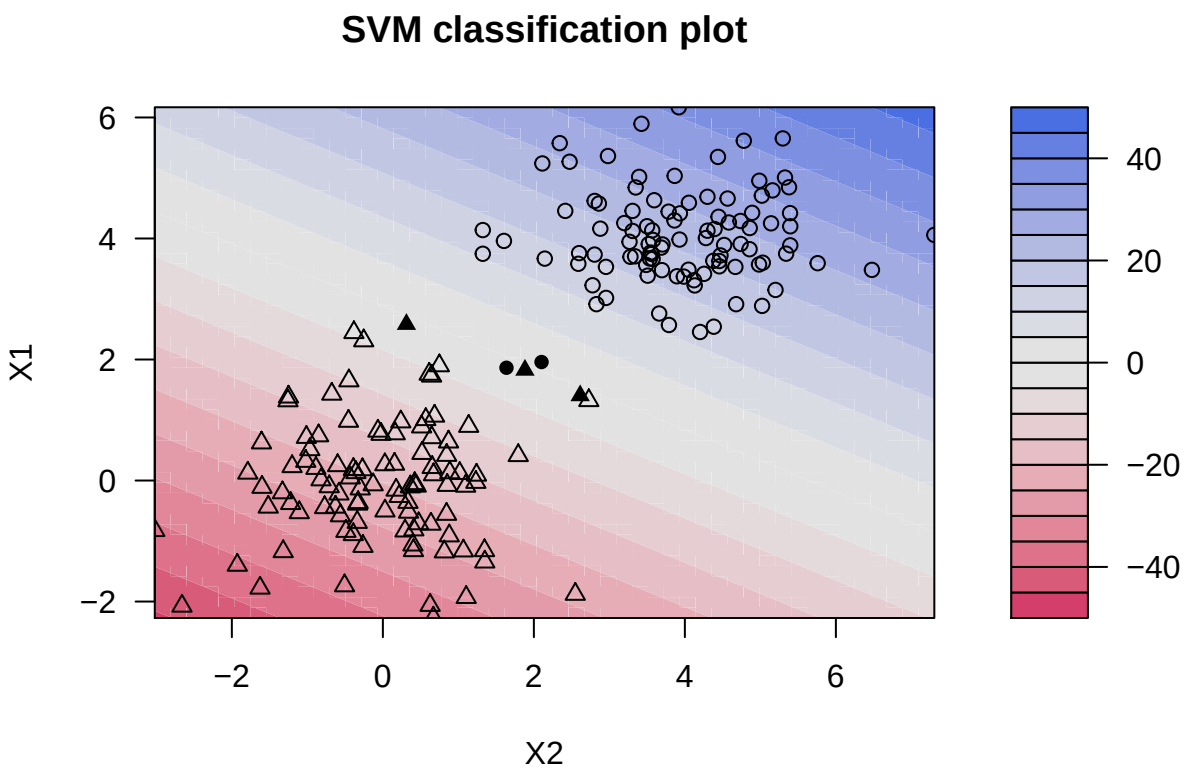

```
##  Setting default kernel parameters
```



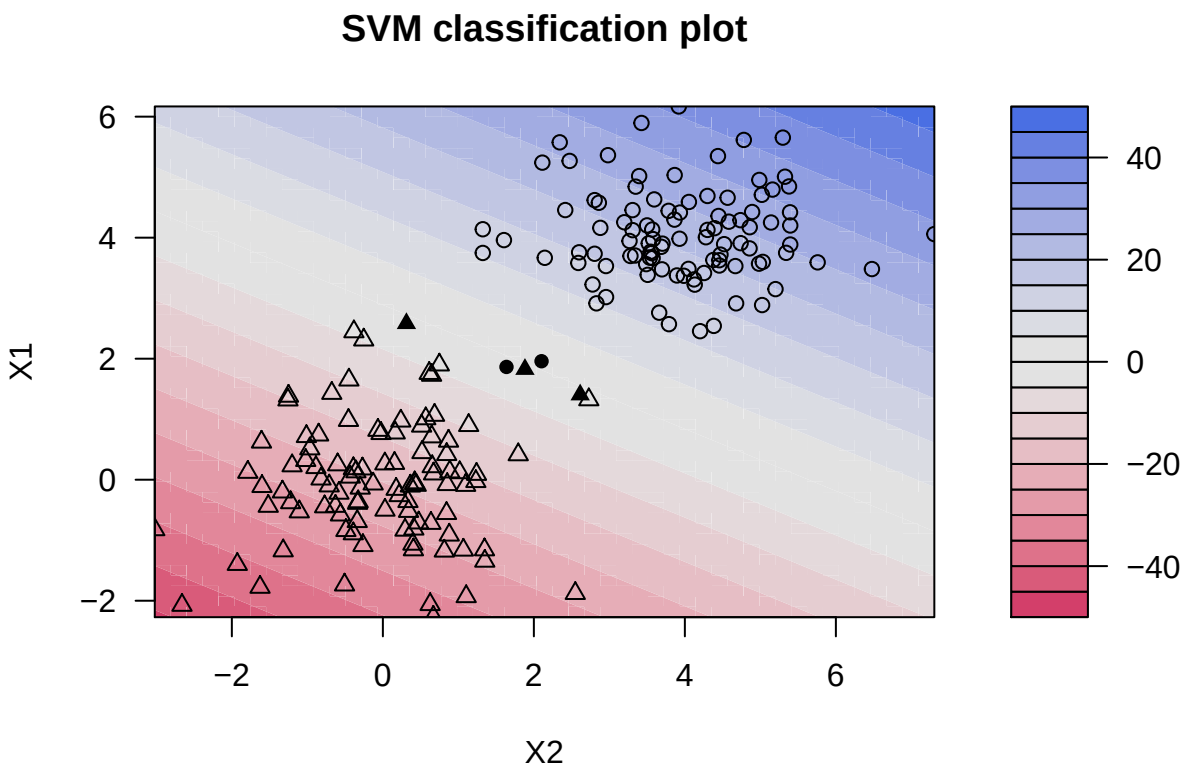

```
##  Setting default kernel parameters
```



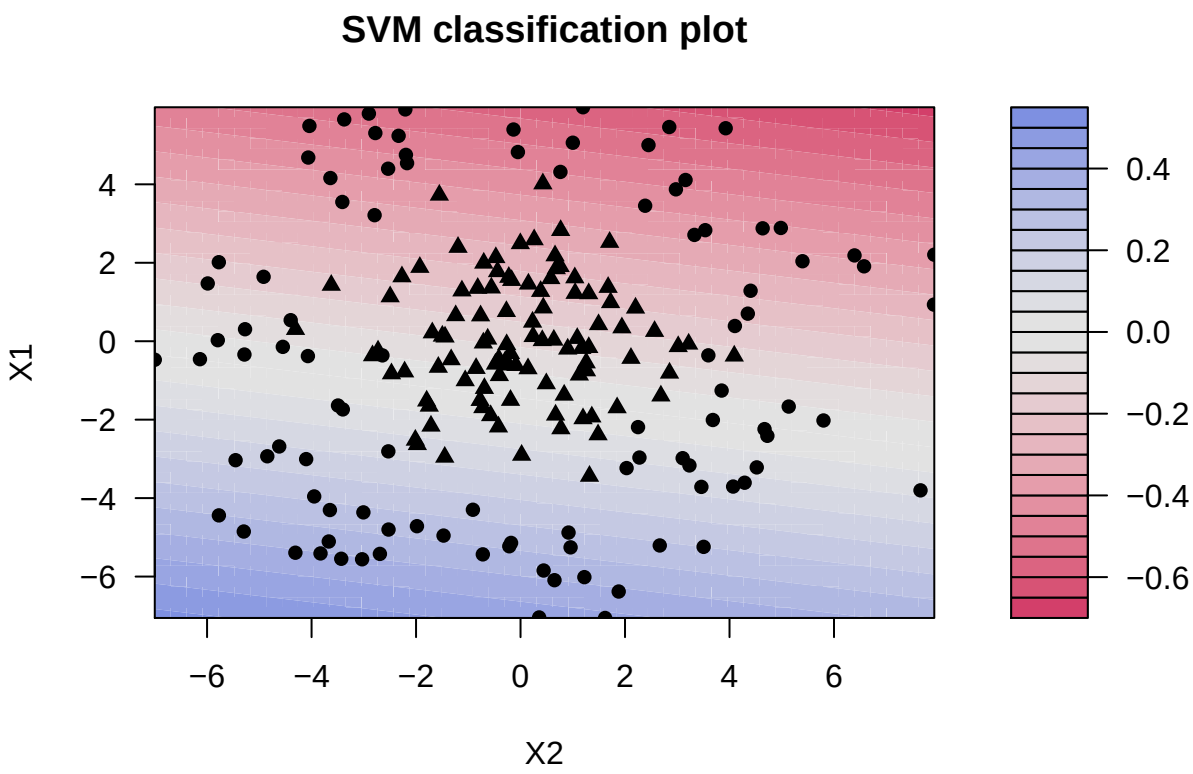

```
##  Setting default kernel parameters
```



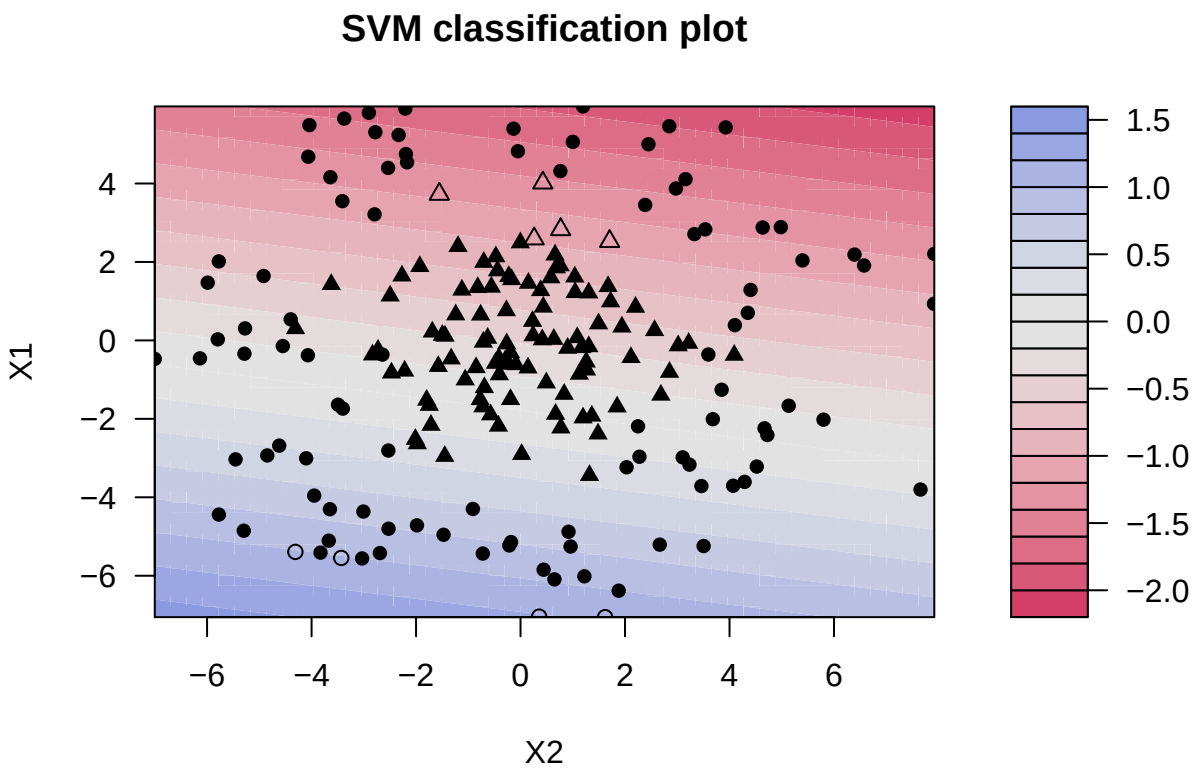

```
##  Setting default kernel parameters
```



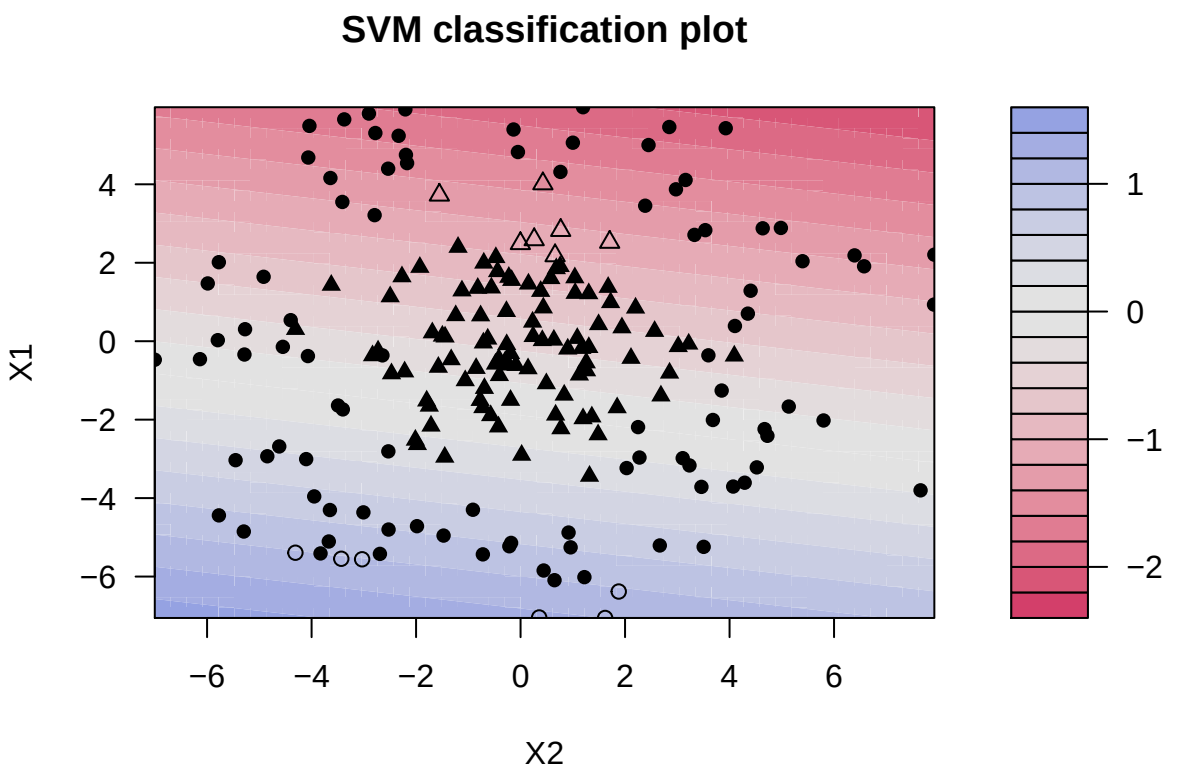

```
##  Setting default kernel parameters
```



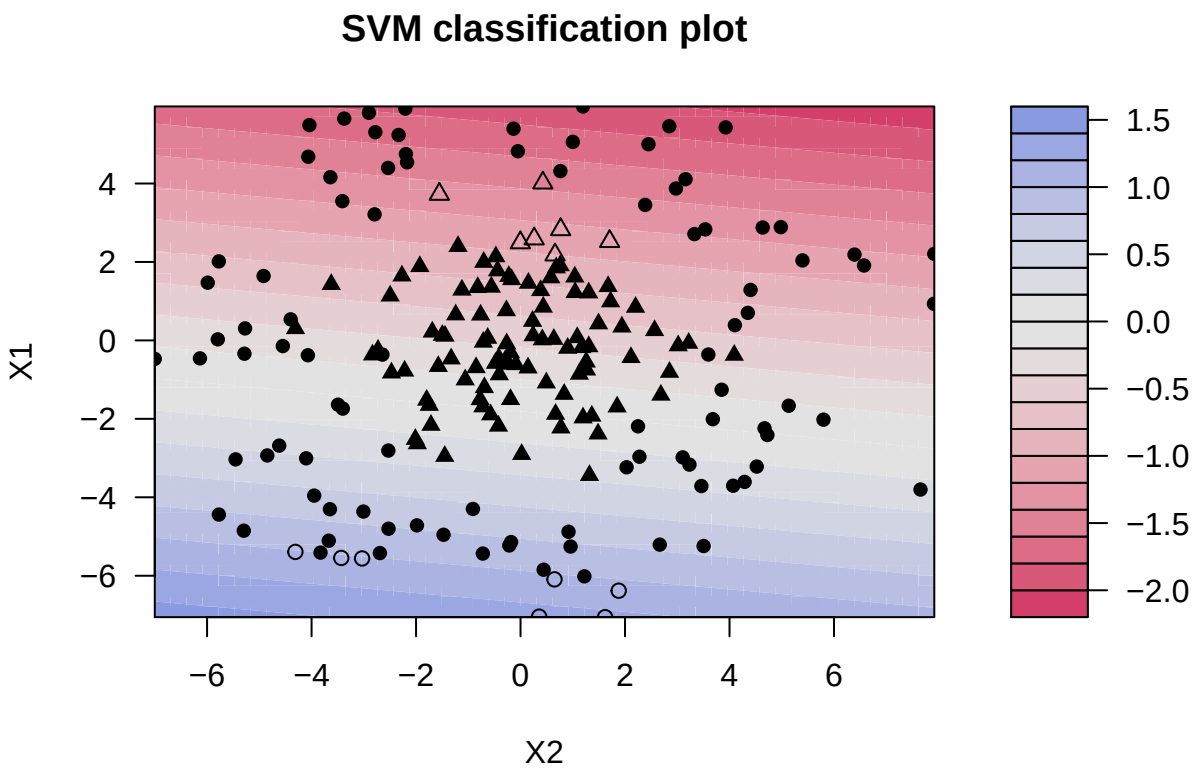

```
##  Setting default kernel parameters
```



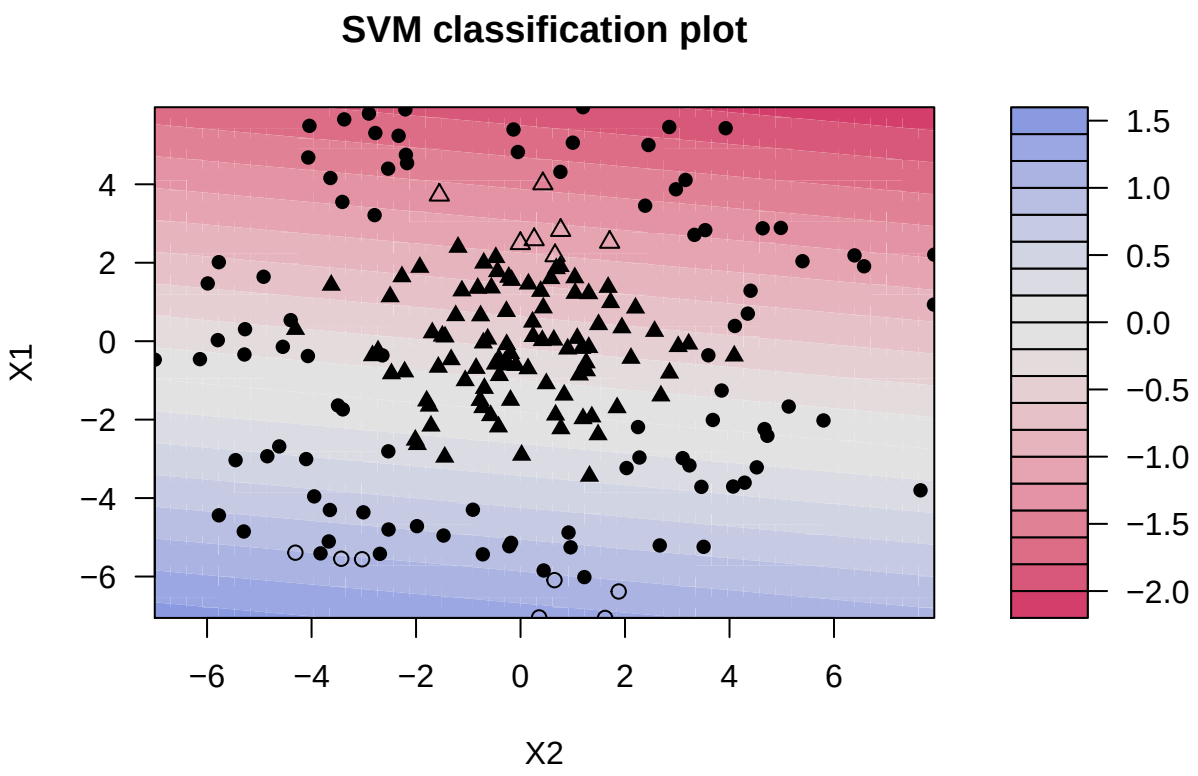

```
##  Setting default kernel parameters
```



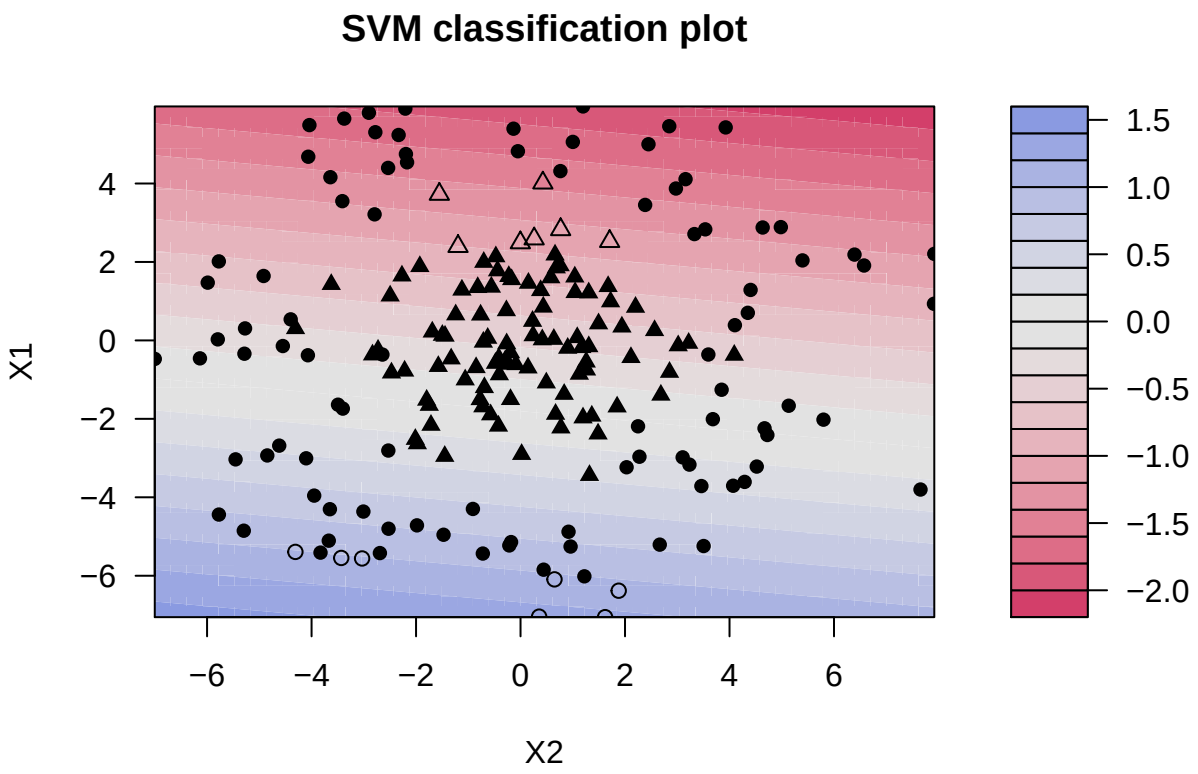

```
##  Setting default kernel parameters
```



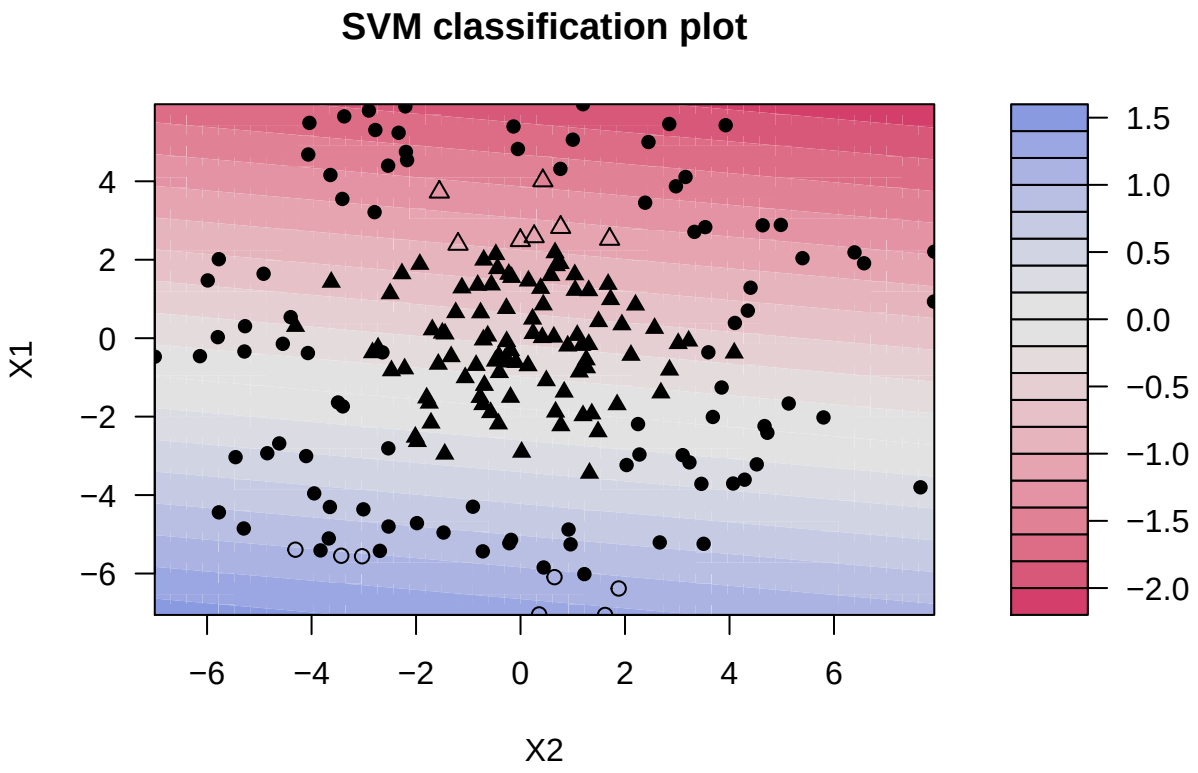

```
##  Setting default kernel parameters
```




```
##  Setting default kernel parameters
```




```
##  Setting default kernel parameters
```



Support Vector Machine (SVM)

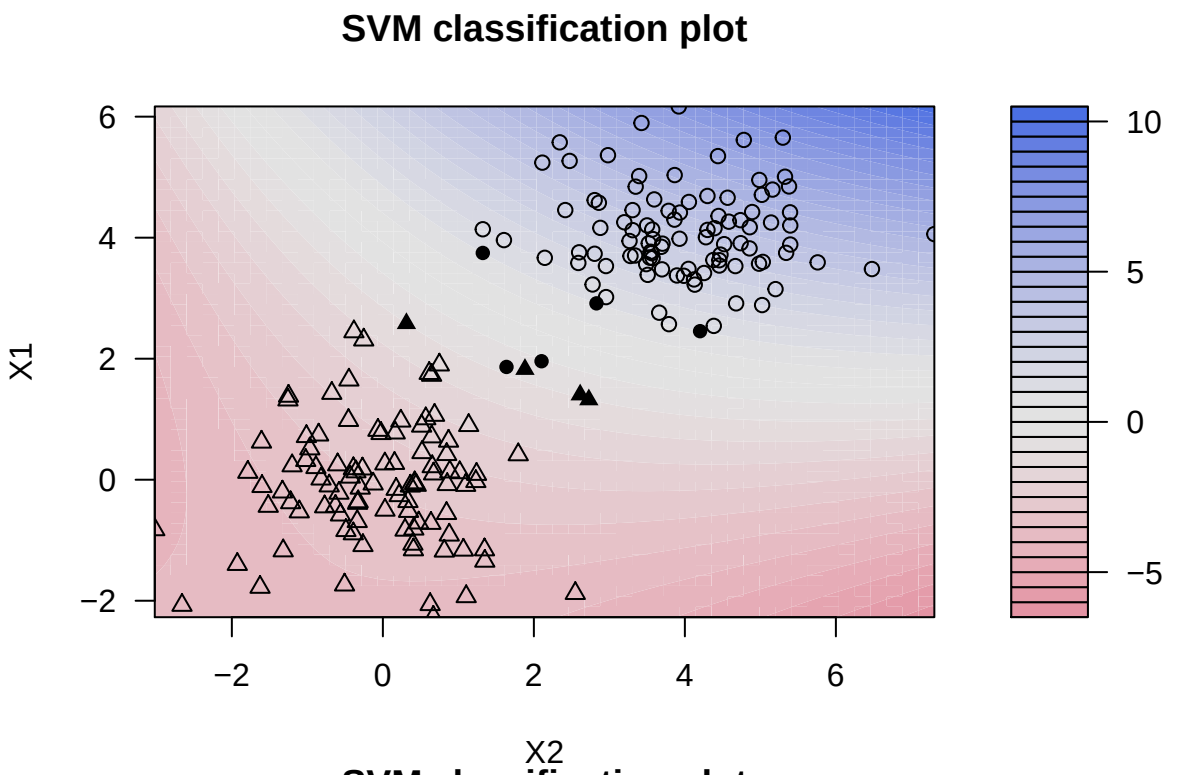
The support vector classifier uses the plain vanilla dot product (and hence the argument `kernel="vanilladot"`) when measuring similarity between two observations. In many scenarios, we might want to use a different similarity measure instead. In SVMs, the dot product is replaced by a *kernel*, $K(x_i, x_{i'})$. There are several commonly used kernels:

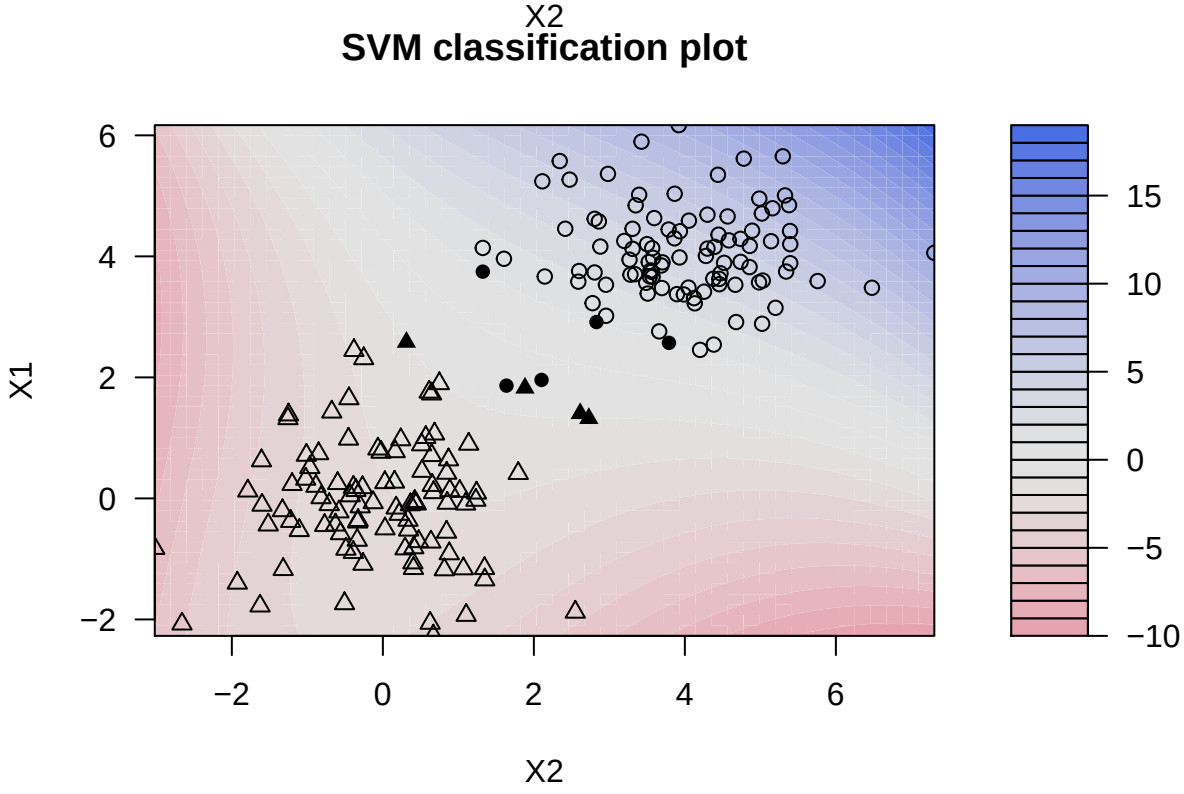
- Linear kernel: $K(x, x') = \langle x_i, x_{i'} \rangle$.
- Polynomial kernel: $K(x, x') = (1 + \langle x_i, x_{i'} \rangle)^d$, where d is a positive integer greater than or equal to 2.
- Radial basis kernel: $K(x, x') = \exp(-\gamma ||x_i - x_{i'}||_2^2)$, where γ is a positive constant.

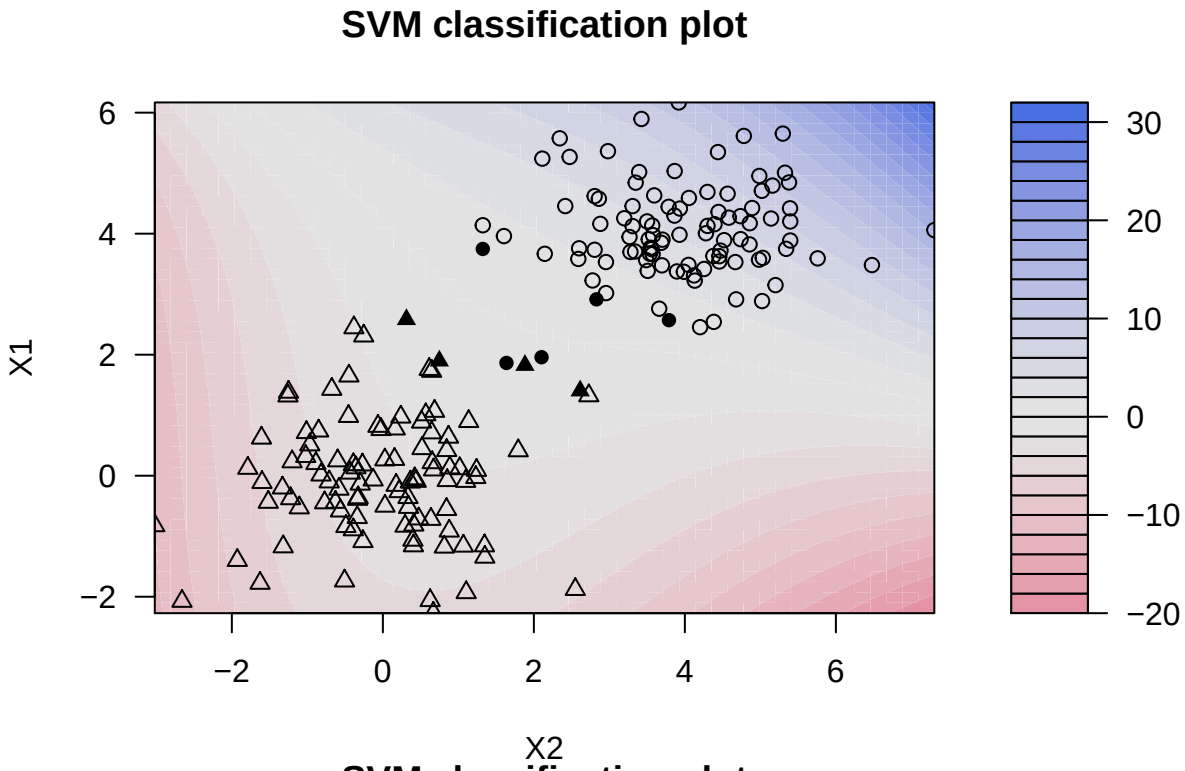
Your turn

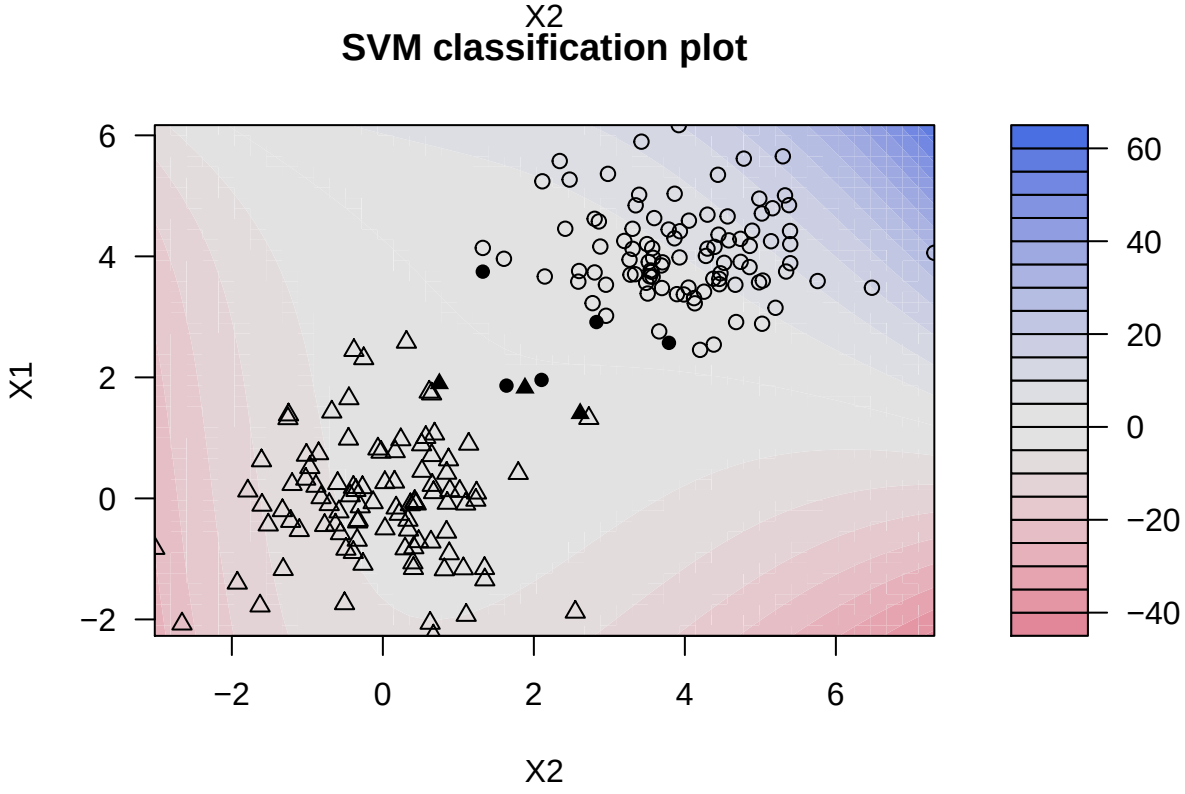
- For each dataset,
  1. fit a support vector classifier using `ksvm()` with the polynomial kernel of degree $d = 2$ and save the `ksvm` object. You might find the argument `kpar` with `degree` useful.
  2. use `plot()` with the `ksvm()` object and the argument `data=dfx`, where `dfx` is the dataset. Describe what you see in the plot.
  3. repeat 1. and 2. with $d \in \{3, 4, 5\}$.
- For each dataset,
  1. fit a support vector classifier using `ksvm()` with the radial basis kernel with $\gamma = 0.01$ and save the `ksvm` object. You might find the argument `kpar` with `sigma` useful. Here `sigma` is essentially γ.
  2. use `plot()` with the `ksvm()` object and the argument `data=dfx`, where `dfx` is the dataset. Describe what you see in the plot.
  3. repeat 1. and 2. with $\gamma \in \{0.1, 1, 10, 100, 1000, 10000\}$.
- As γ increases, how does the number of support vectors change?

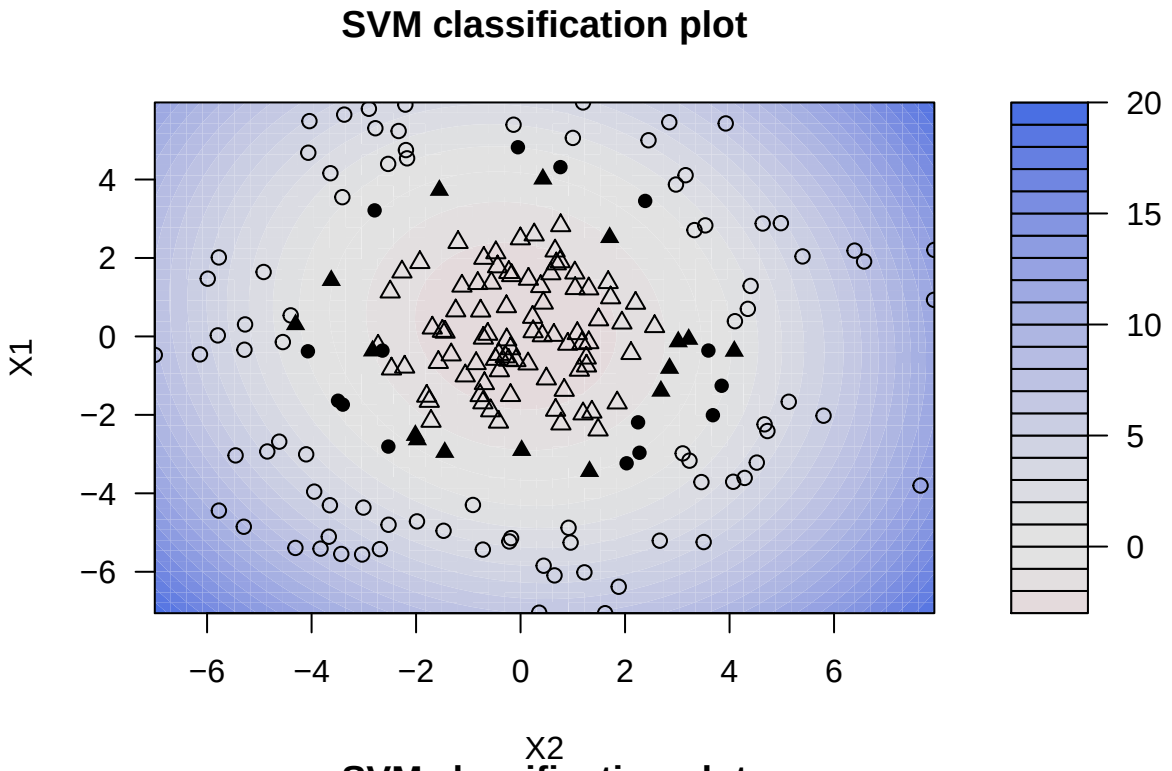
```
deg_vector <- 2:5
gam_vector <- c(0.01, 0.1, 1, 10, 100, 1000, 10000)

for (df in dataset_list) {
  for (deg in deg_vector) {
    fit <- ksvm(y~X1+X2, data = df, kernel = "polydot", kpar=list(degree=deg))
    plot(fit, data=df)
  }
}
```

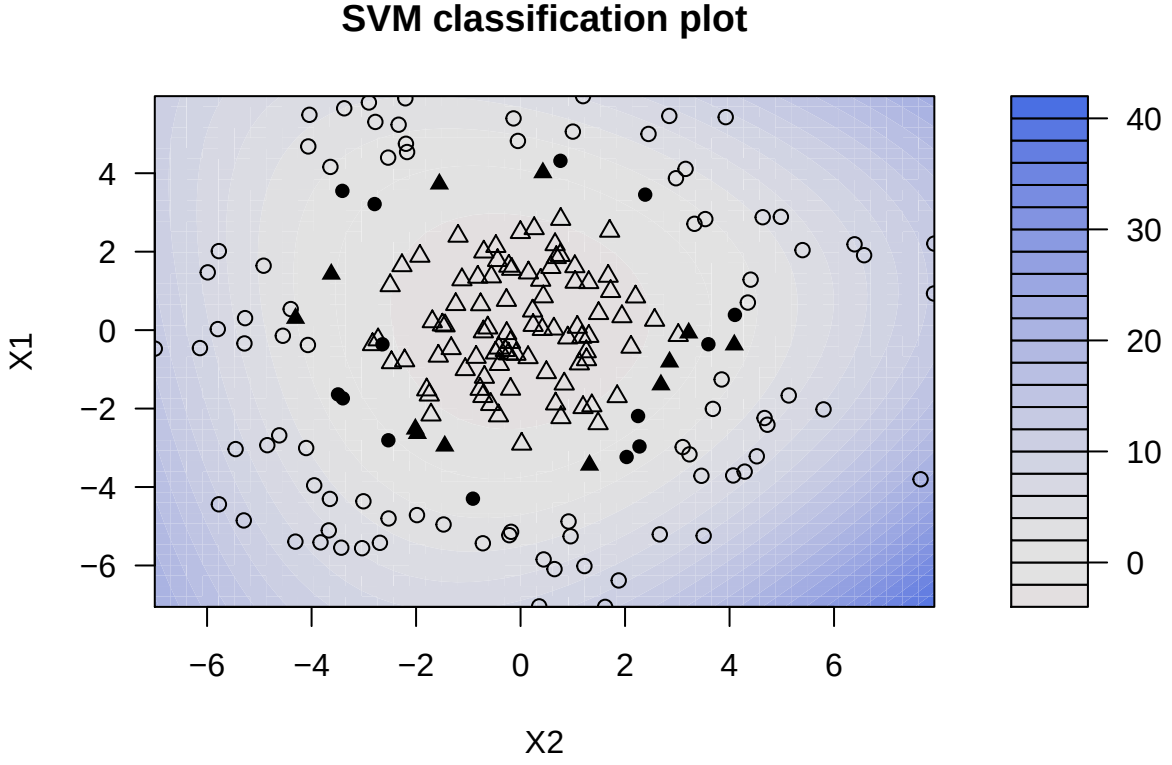

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
10
5
0
−5

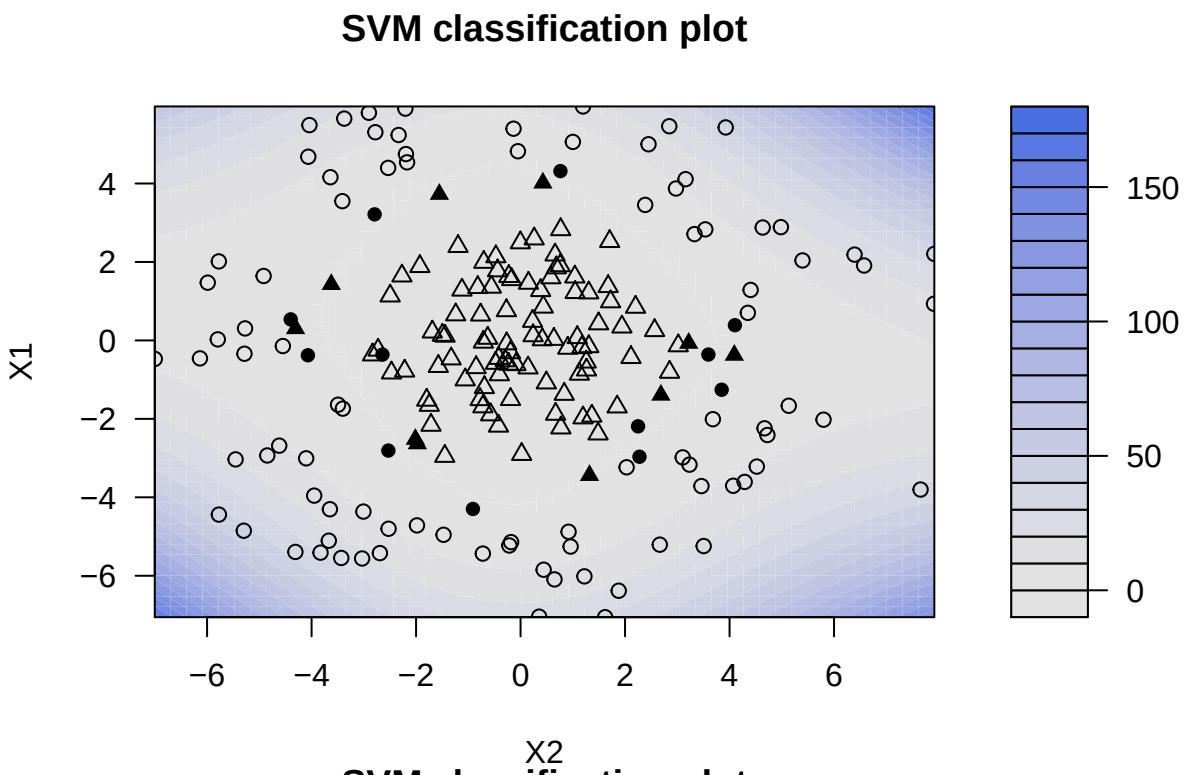

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
15
10
5
0
−5
−10

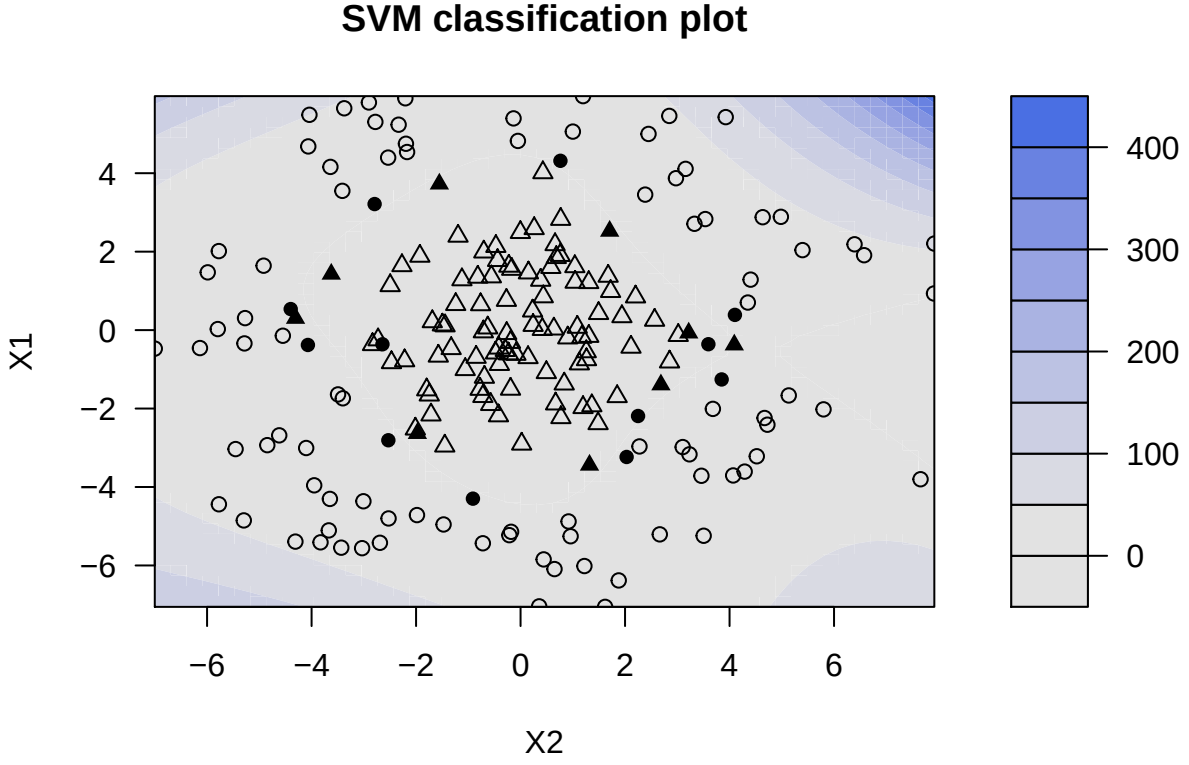

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
30
20
10
0
−10
−20


SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
60
40
20
0
−20
−40


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
20
15
10
5
0

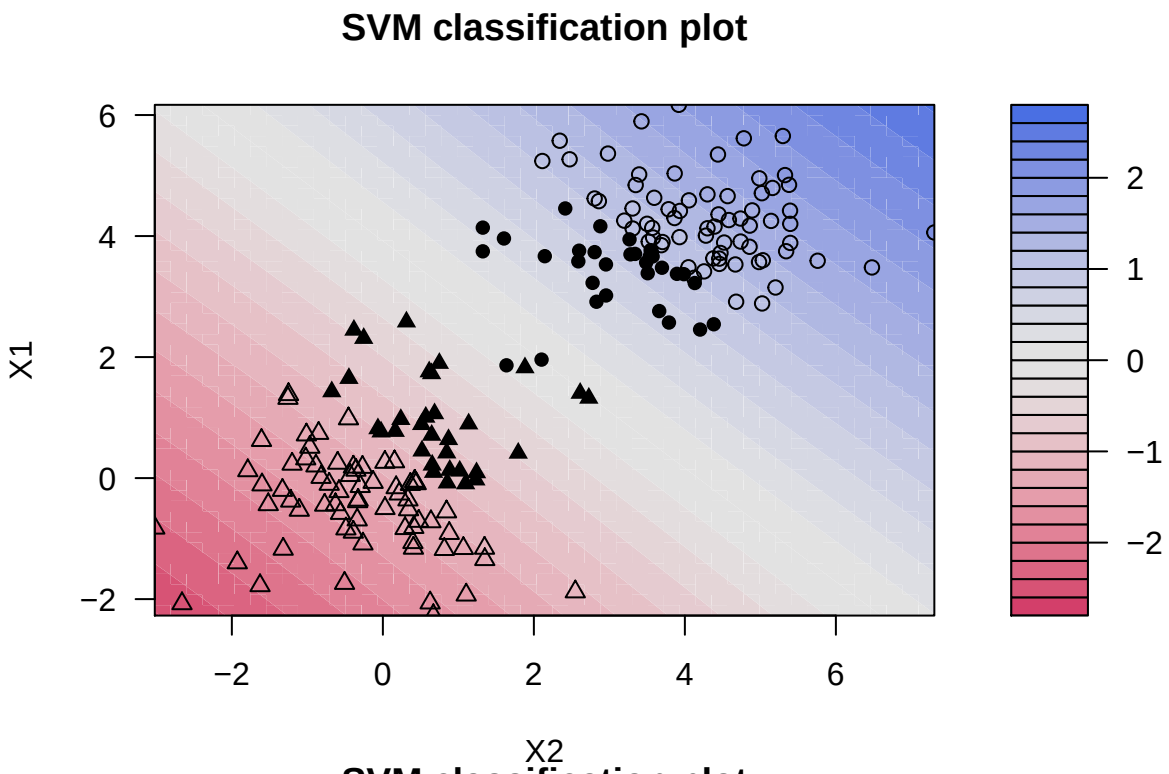

SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
40
30
20
10
0

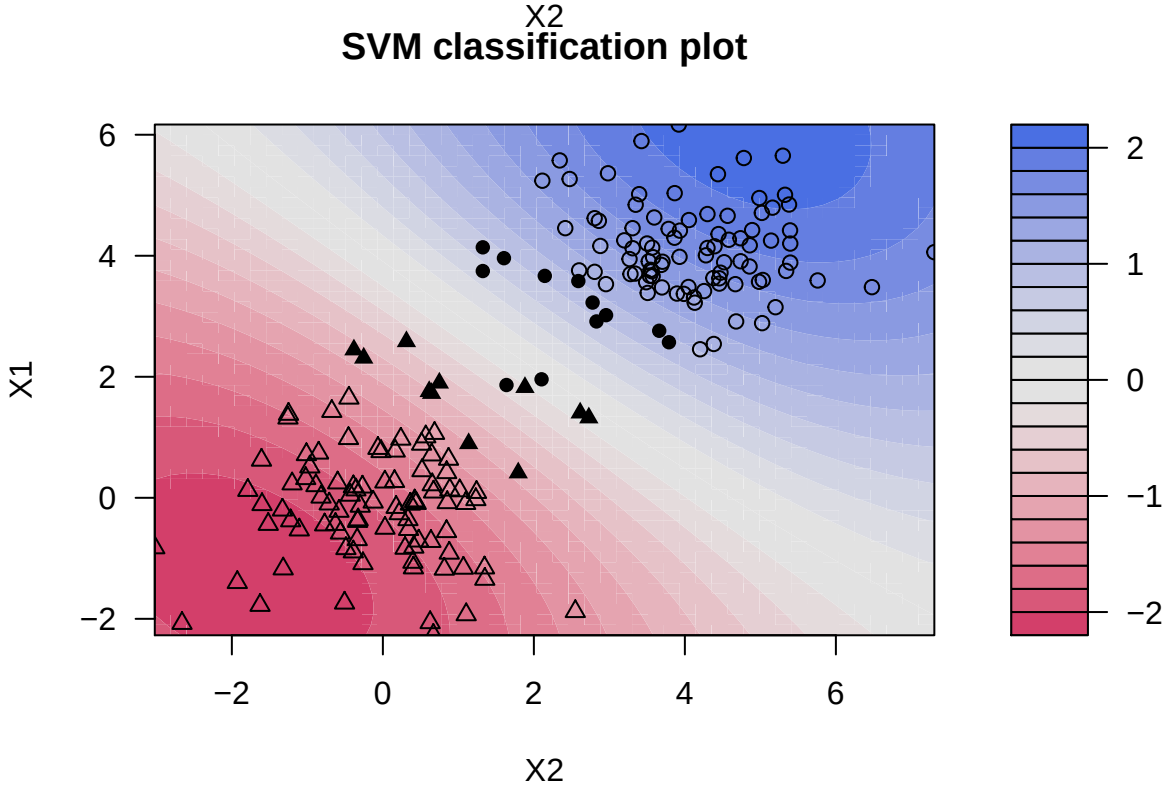


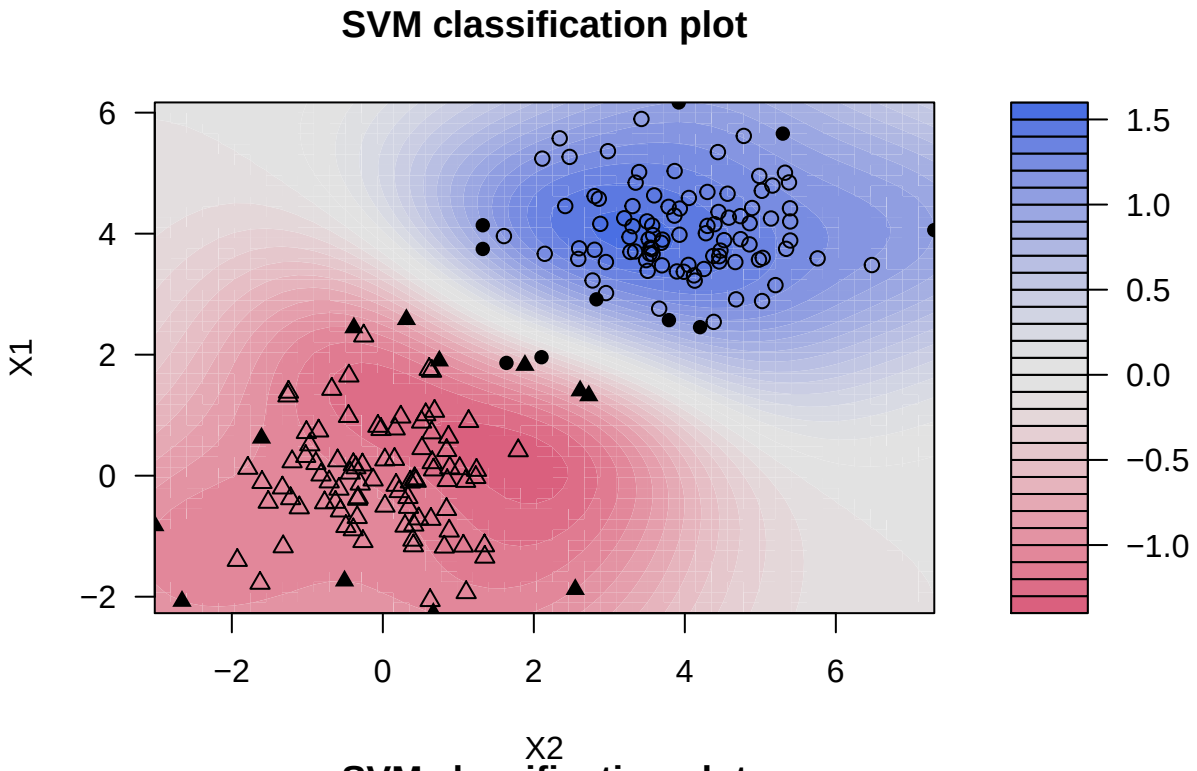


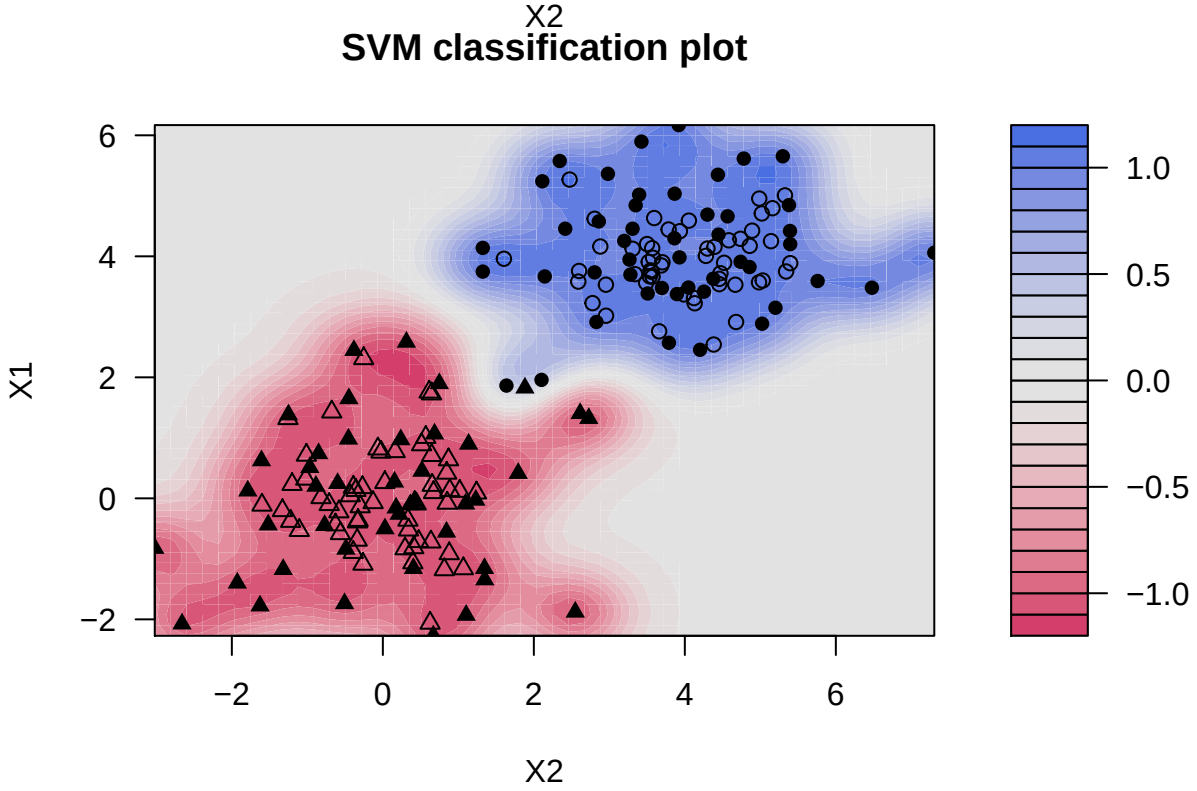

```
for (df in dataset_list) {
  for (gam in gam_vector) {
    fit <- ksvm(y~X1+X2, data = df, kernel = "rbfdot", kpar=list(sigma=gam))
    plot(fit, data=df)
  }
```

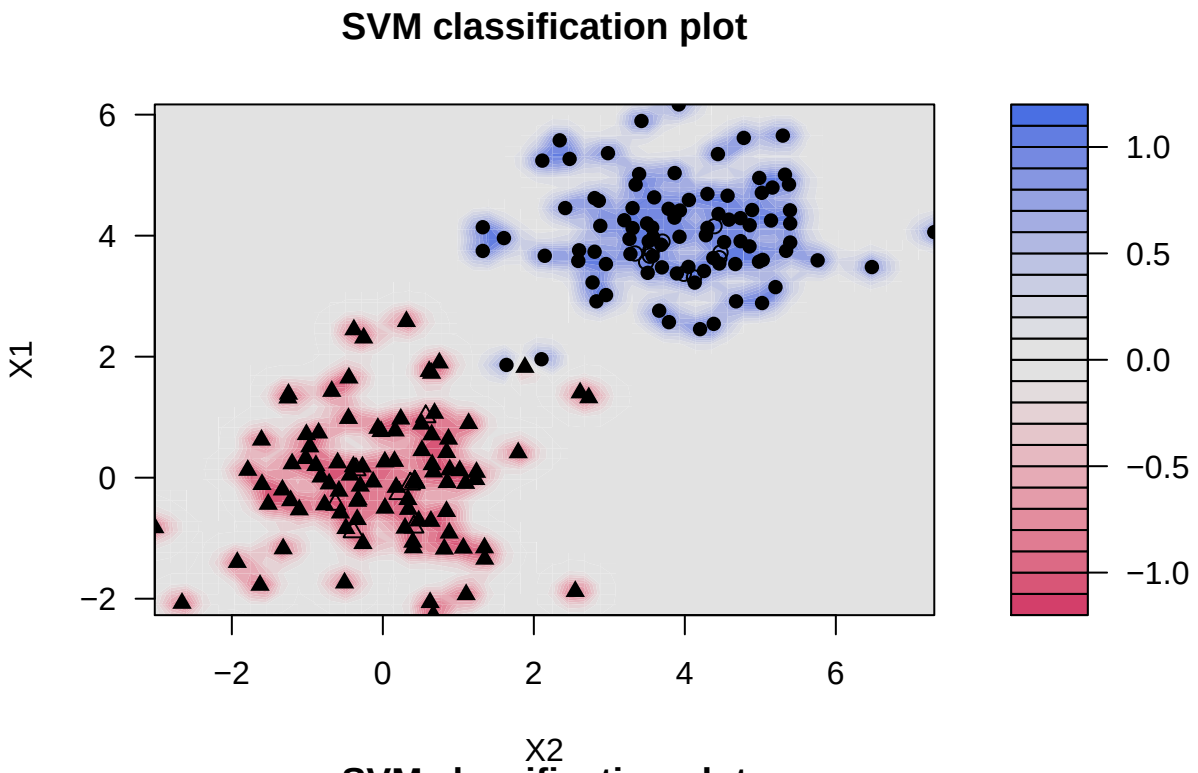
```
}
```

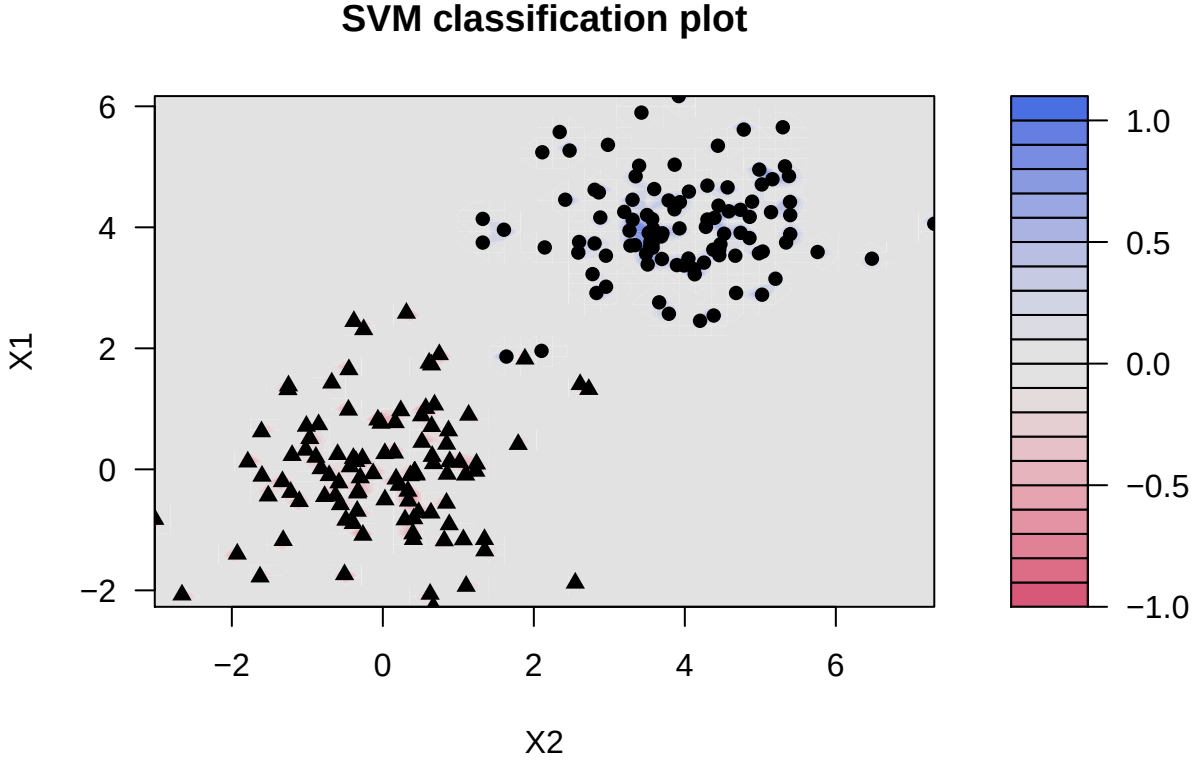

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
2
1
0
−1
−2

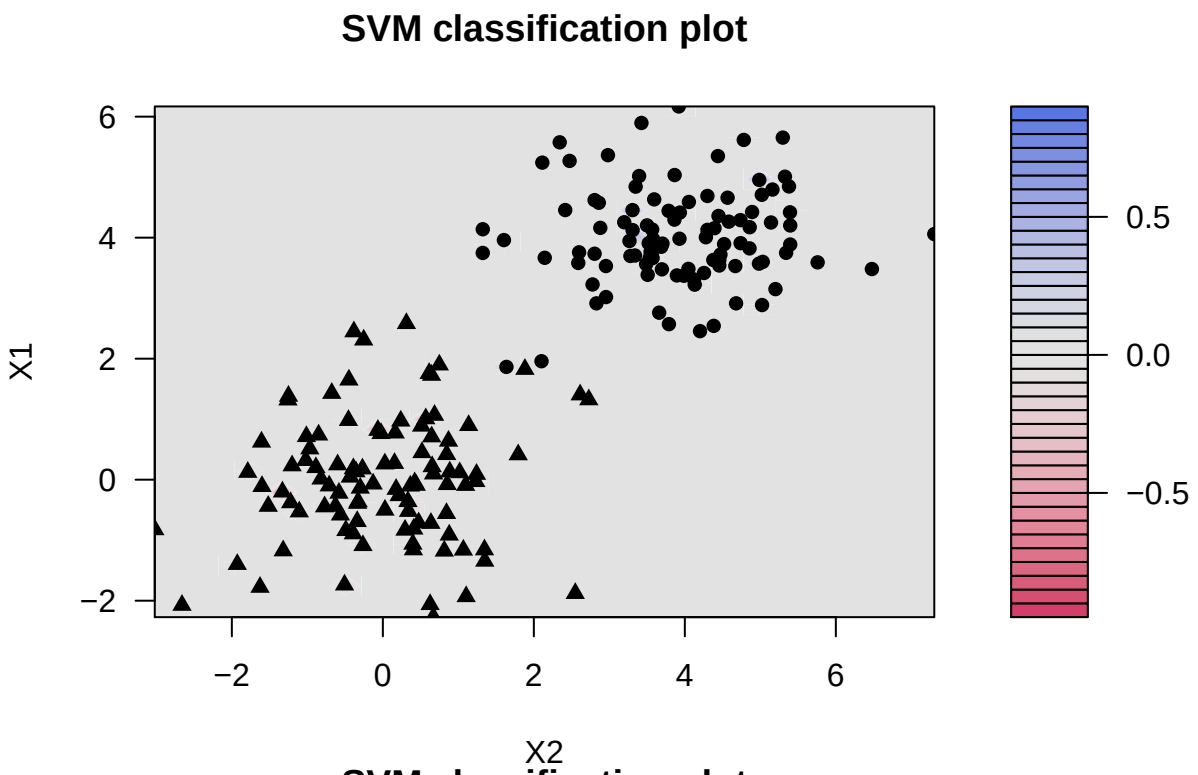

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
2
1
0
−1
−2

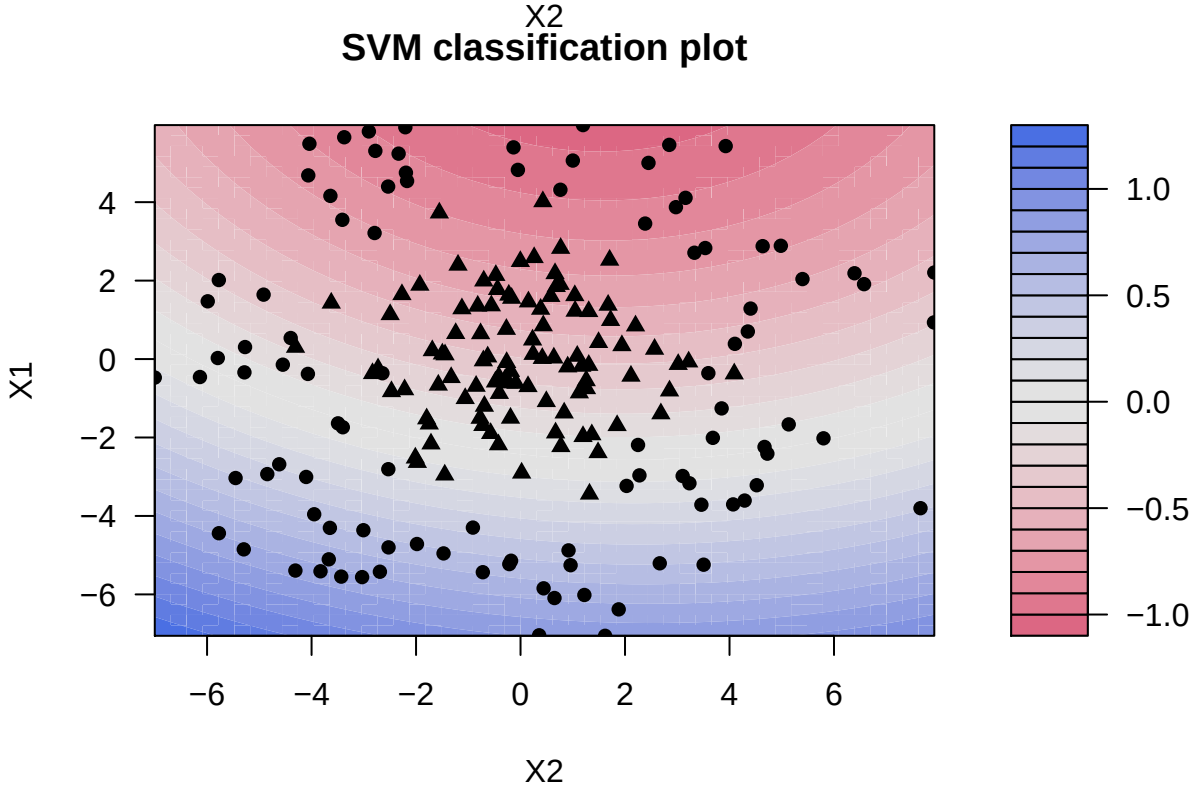

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
1.5
1.0
0.5
0.0
−0.5
−1.0

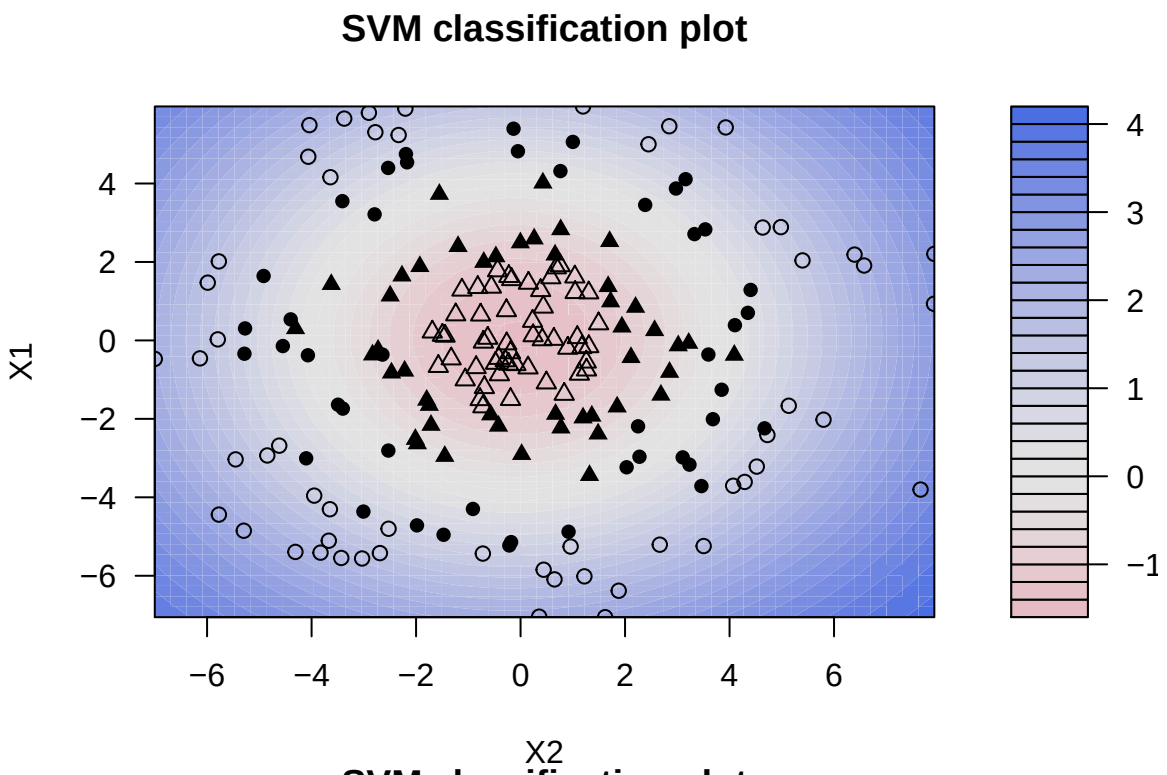

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0

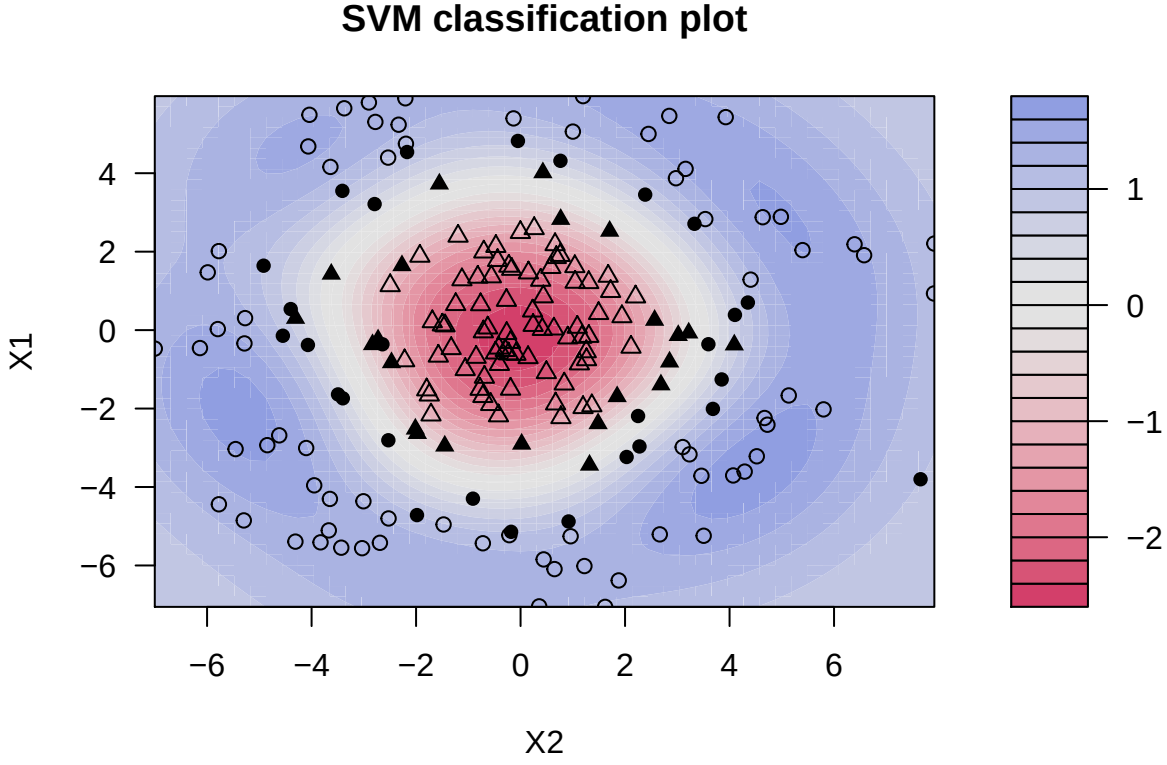

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0

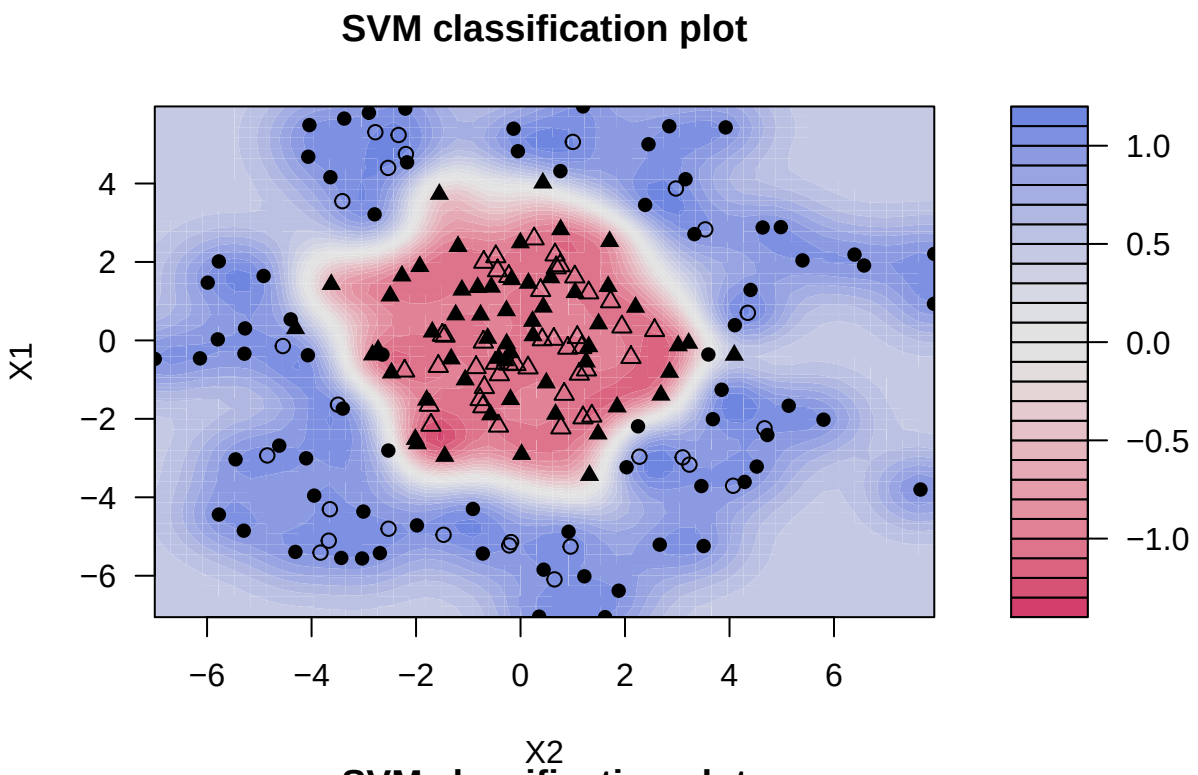

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0

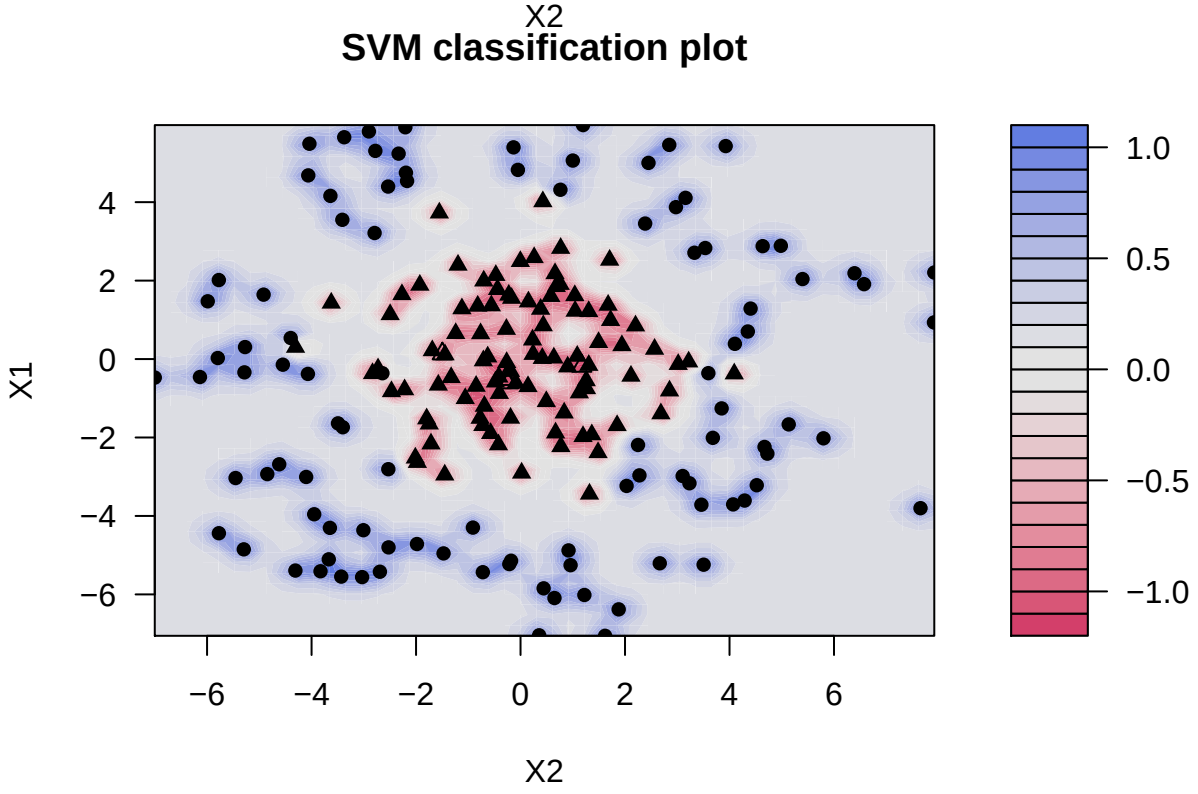

SVM classification plot
X1
X2
6
4
2
0
−2
−2
0
2
4
6
0.5
0.0
−0.5

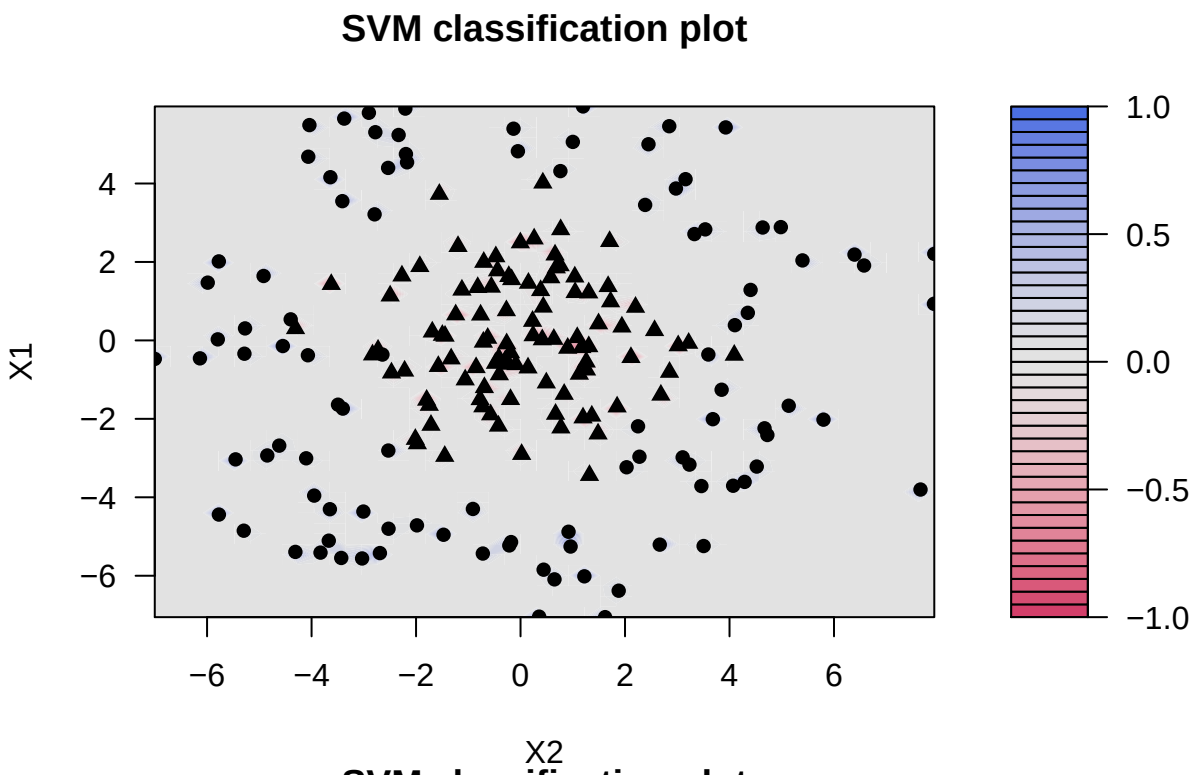

SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0

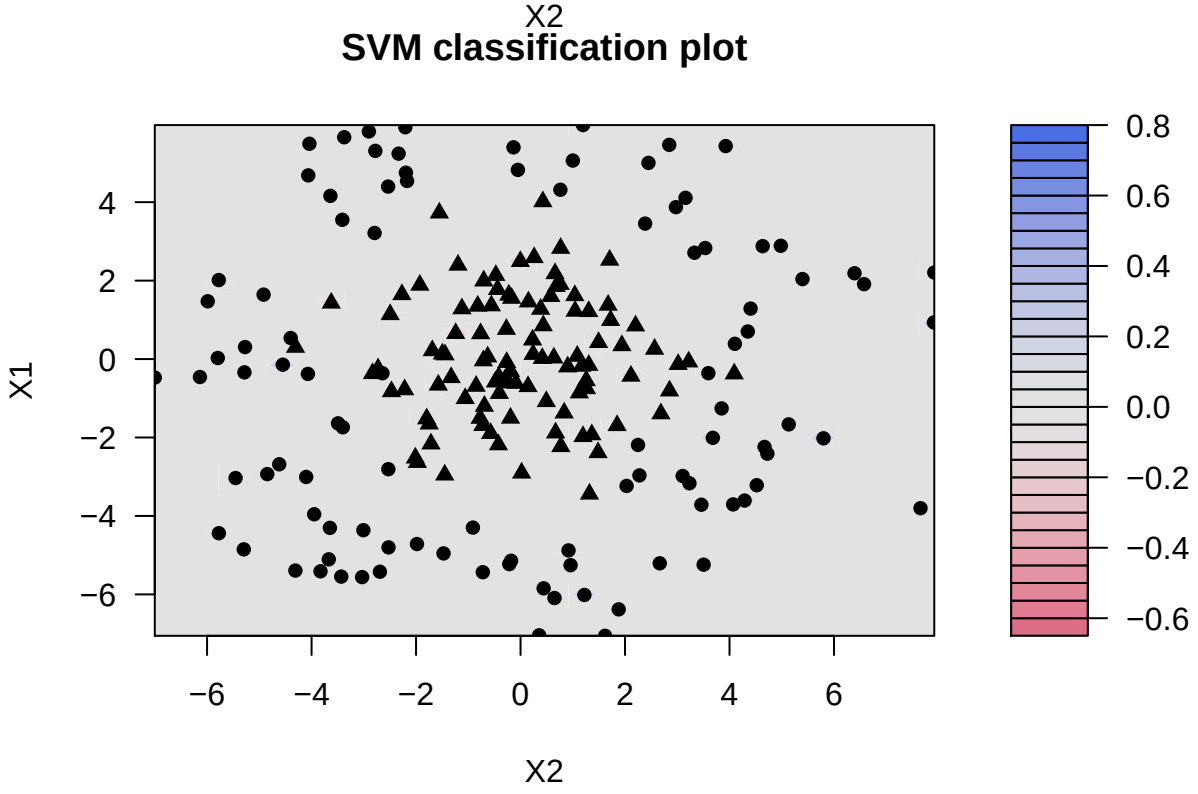

SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
4
3
2
1
0
−1


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
1
0
−1
−2


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
1.0
0.5
0.0
−0.5
−1.0


SVM classification plot
X1
X2
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

ROC curve

Let's restrict our discussion to a two-class classification problem. Recall that when we use the LDA or QDA or logistic regression in previous labs, we use 0.5 as the threshold for deciding whether a new observation is classified as the positive instance. In practice, we do not have to restrict ourselves to the threshold 0.5. As we alter the threshold, the true postive rate and the false positive rate change. The *receiver operating characteristics curve*, known as the *ROC curve*, traces out the true postive rates against the false positive rates as the threshold varies. The curve provides a graphical summary of the classifier performance across thresholds, providing a more comprehensive view of the classifier than a single point metric such as the misclassification rate or the accuracy.

Your turn

We will use `df2` for this part. Read https://www.r-bloggers.com/a-small-introduction-to-the-rocr-package/ to learn how to use the `ROCR` package.

- Train LDA using 70% of the data. Generate posterior class probabilities (for class 1 only) on the remaining 30%.
- Plot the ROC curve. Add the 45 degree dotted line to the plot.
- Compute the area under ROC (AUC). You might find `performance(..., measure="auc")` useful.
- Does LDA perform much better than a random classifier for this particular dataset?