PS10

Minyoung Do

3/7/2020

Conceptual exercises

Non-linear separation

1. (15 points) *Generate* a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear separation between the two classes. *Show* that in this setting, a support vector machine with a radial kernel will outperform a support vector classifier (a linear kernel) on the training data. *Which* technique performs best on the test data? Make plots and report training and test error rates in order to support your conclusions.

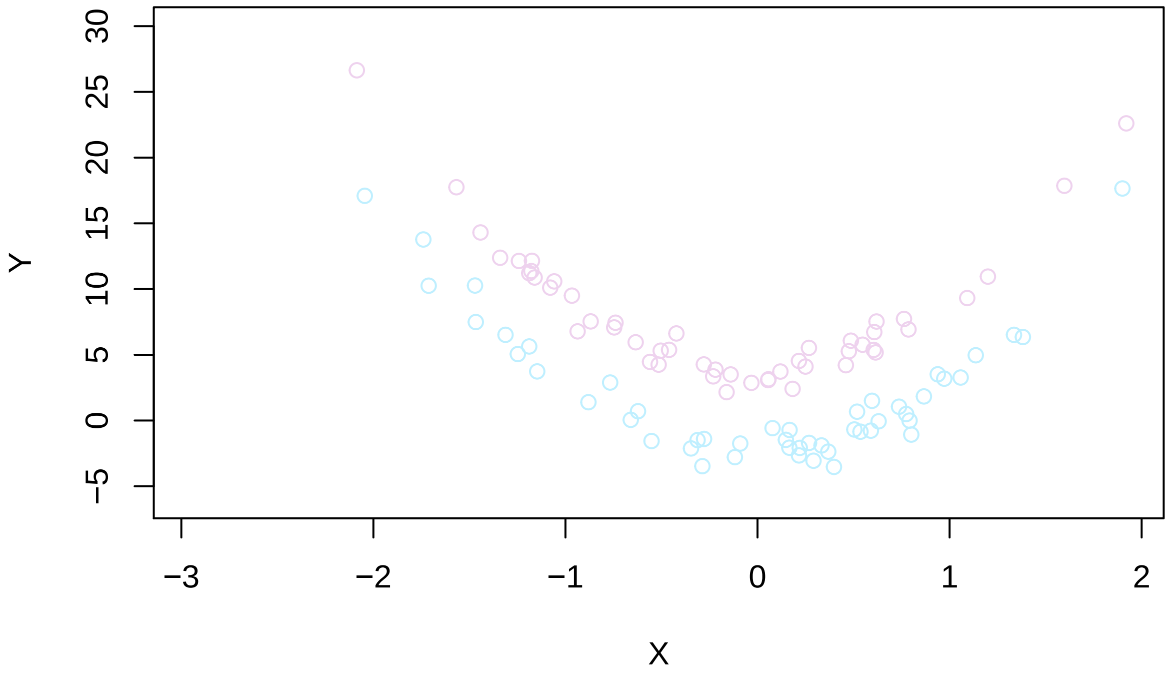
```r
set.seed(420)

x <- rnorm(100)
y <- 5 * x^2 + 1 + rnorm(100)

class <- sample(100, 50)

y[class] = y[class] + 3
y[-class] = y[-class] - 3

plot(x[class], y[class], col = "thistle2", xlab = "X", ylab = "Y", ylim = c(-6, 30))
points(x[-class], y[-class], col = "lightblue1")
```

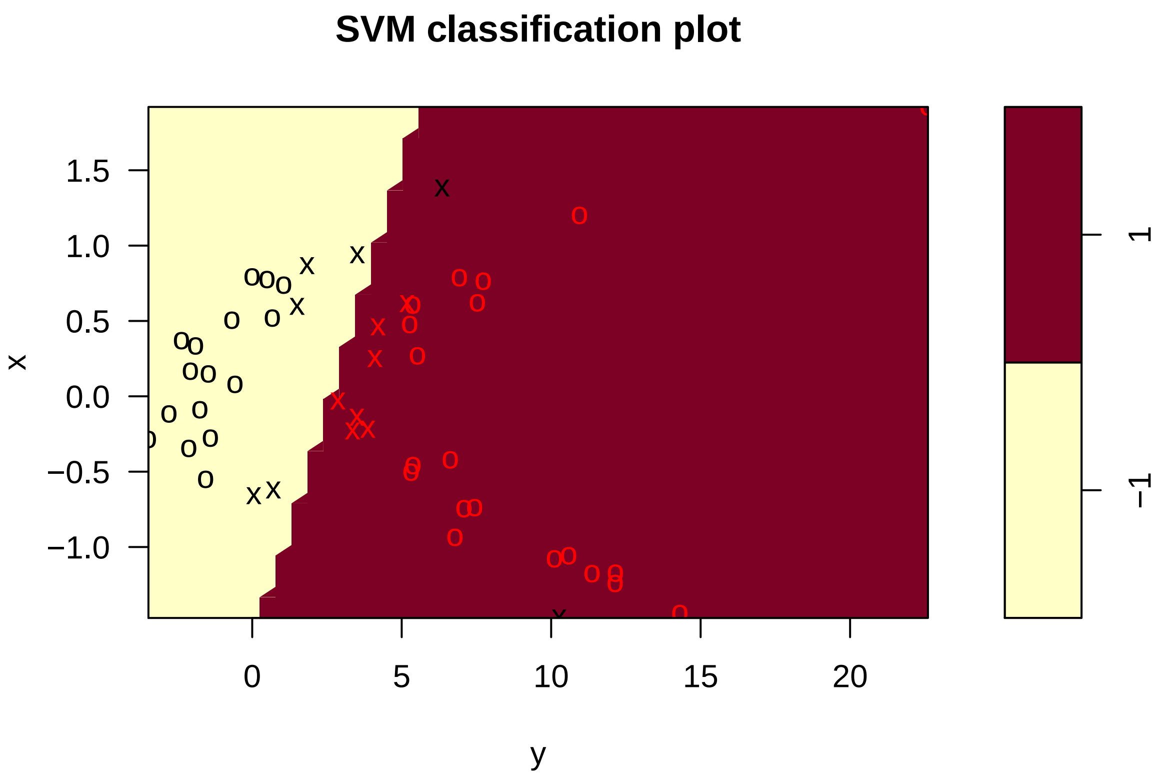


```r
z <- rep(-1, 100)
z[class] = 1

data <- data.frame(x = x, y = y, z = as.factor(z))
train <- sample(100, 50)

data.train <- data[train, ]
data.test <- data[-train, ]

svm.linear <- svm(z ~ ., data = data.train, kernel = "linear", cost = 10)
plot(svm.linear, data.train)
```

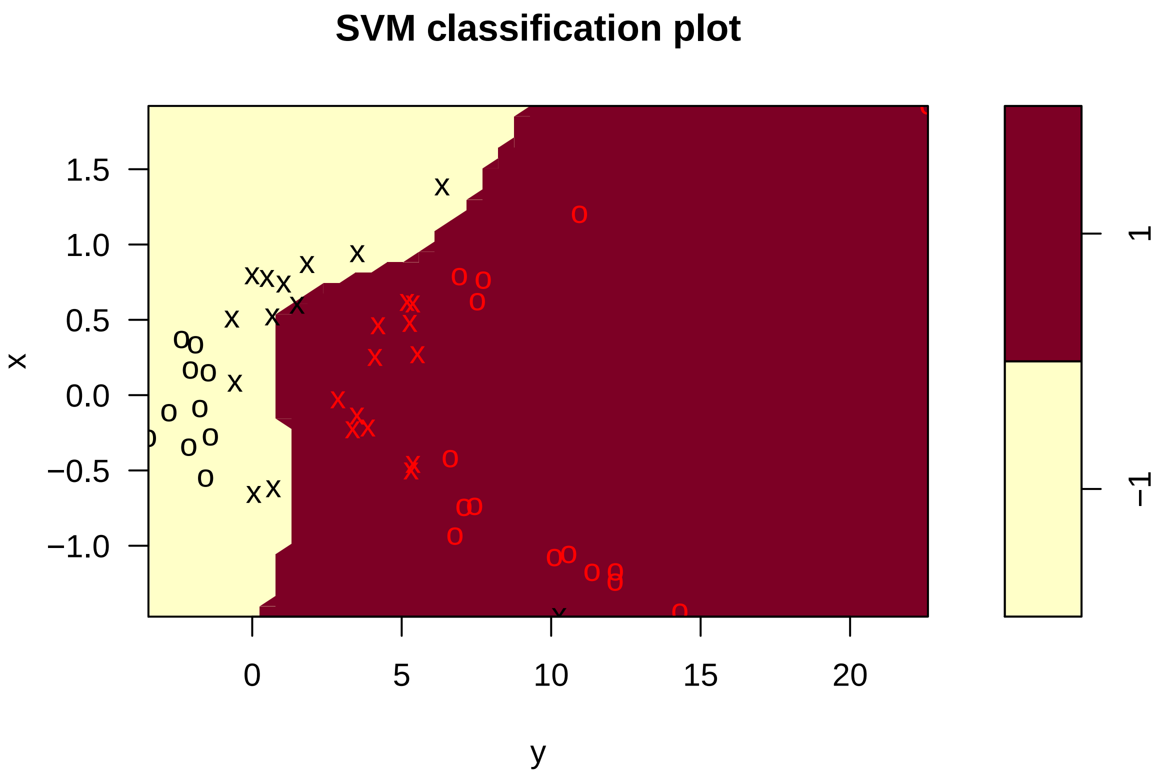


```r
table(predict = predict(svm.linear, data.train), truth = data.train$z)
```

```
##        truth
## predict -1  1
##      -1 21  0
##      1   2 27
```

The support vector classifier only makes 2 errors on the training data. Let's fit a support vector machine with a polynomial kernel now.

```r
svm.poly = svm(z ~ ., data = data.train, kernel = "polynomial", cost = 10)
plot(svm.poly, data.train)
```

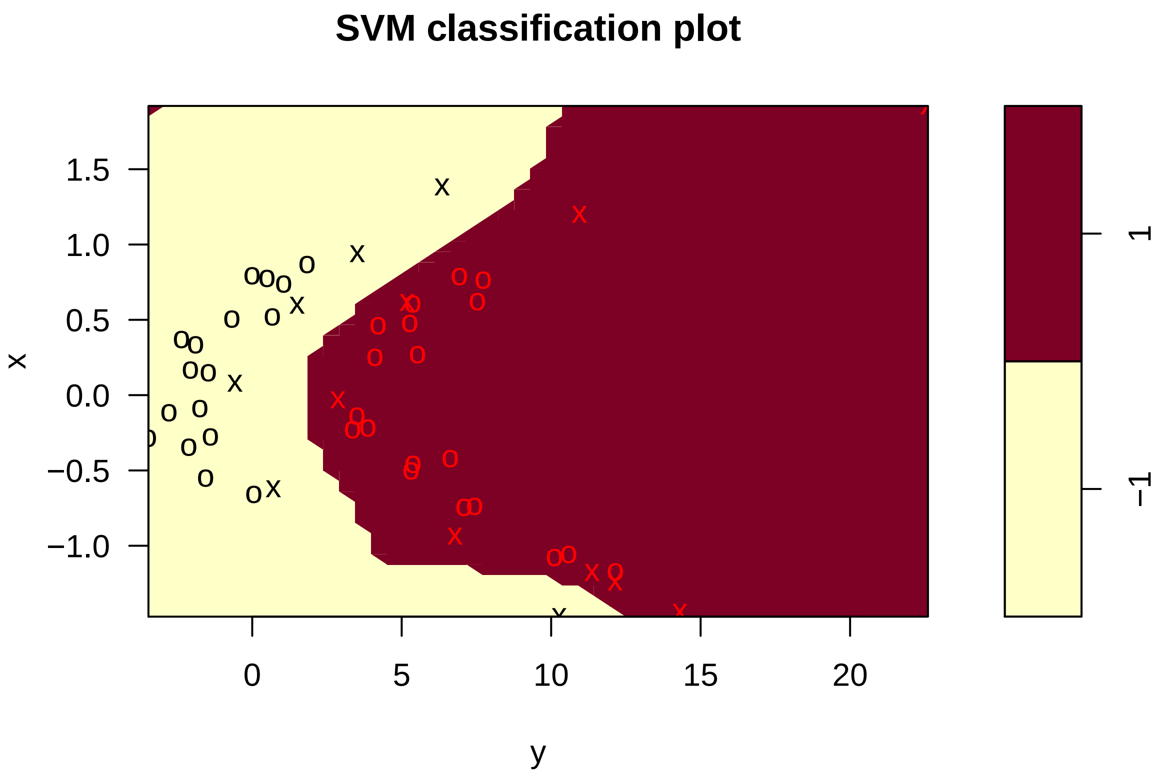


```r
table(predict = predict(svm.poly, data.train), truth = data.train$z)
```

```
##         truth
## predict -1  1
##      -1 21  0
##      1   2 27
```

The support vector machine with a polynomial kernel of degree 3 makes 2 errors on the training data, which is the same with the SVM with a linear kernel. a support vector machine with a radial kernel and a gamma of 1.

```r
svm.radial = svm(z ~ ., data = data.train, kernel = "radial", gamma = 1, cost = 10)
plot(svm.radial, data.train)
```

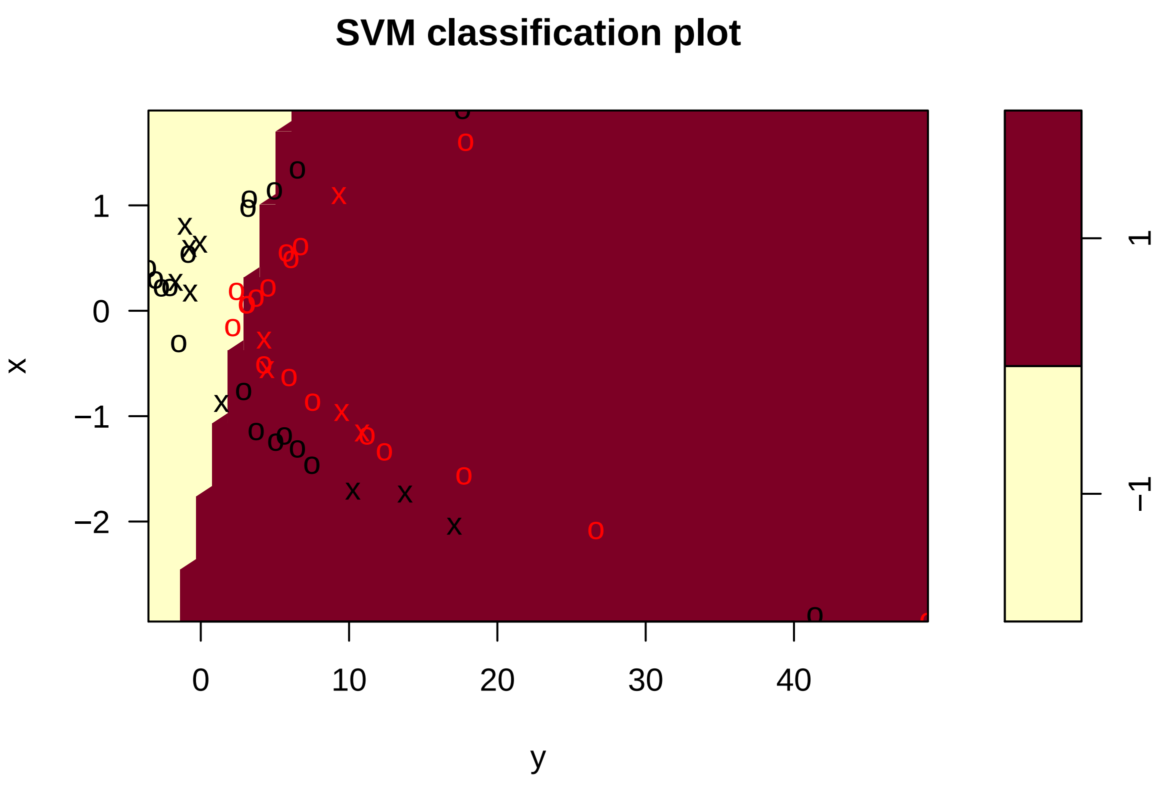


```r
table(predict = predict(svm.radial, data.train), truth = data.train$z)
```

```
##        truth
## predict -1  1
##      -1 23  0
##      1   0 27
```

The support vector machine with a radial kernel makes 0 error on the training data. Now applying these SVMs on the test set!

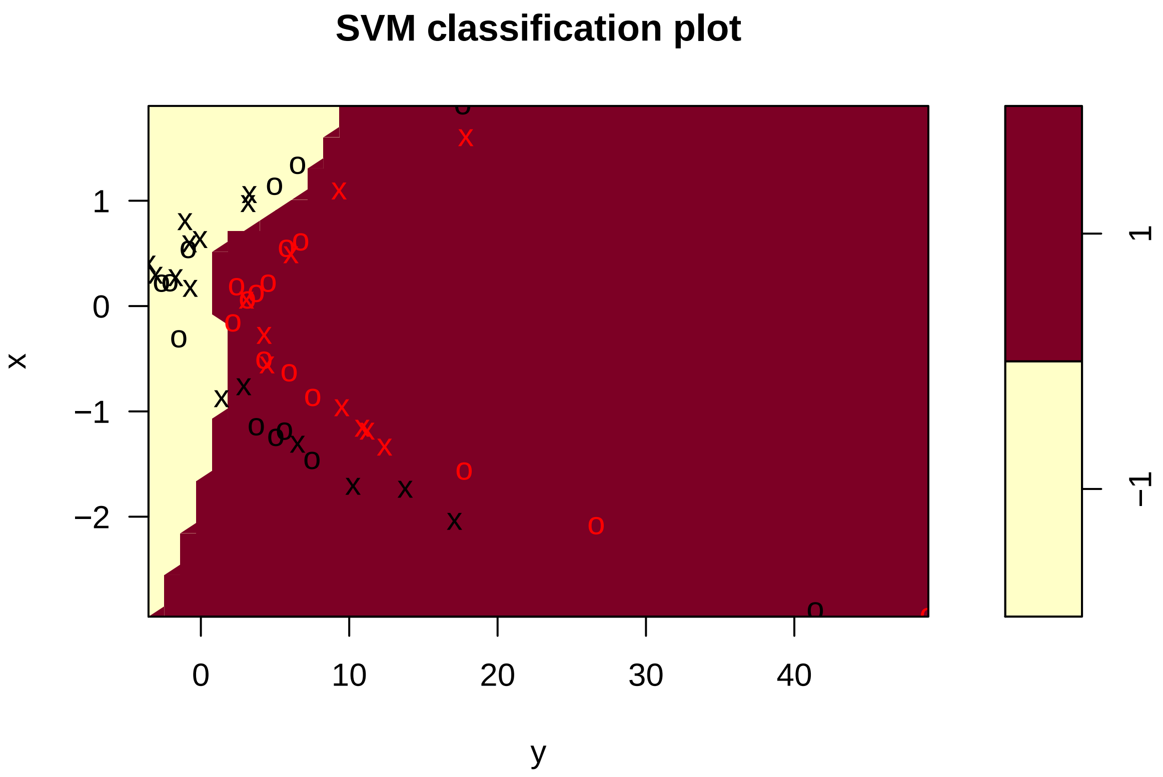
```r
# linear kernel
plot(svm.linear, data.test)
```



```r
table(predict = predict(svm.linear, data.test), truth = data.test$z)
```

```
##        truth
## predict -1  1
##      -1 13  1
##      1  14 22
```

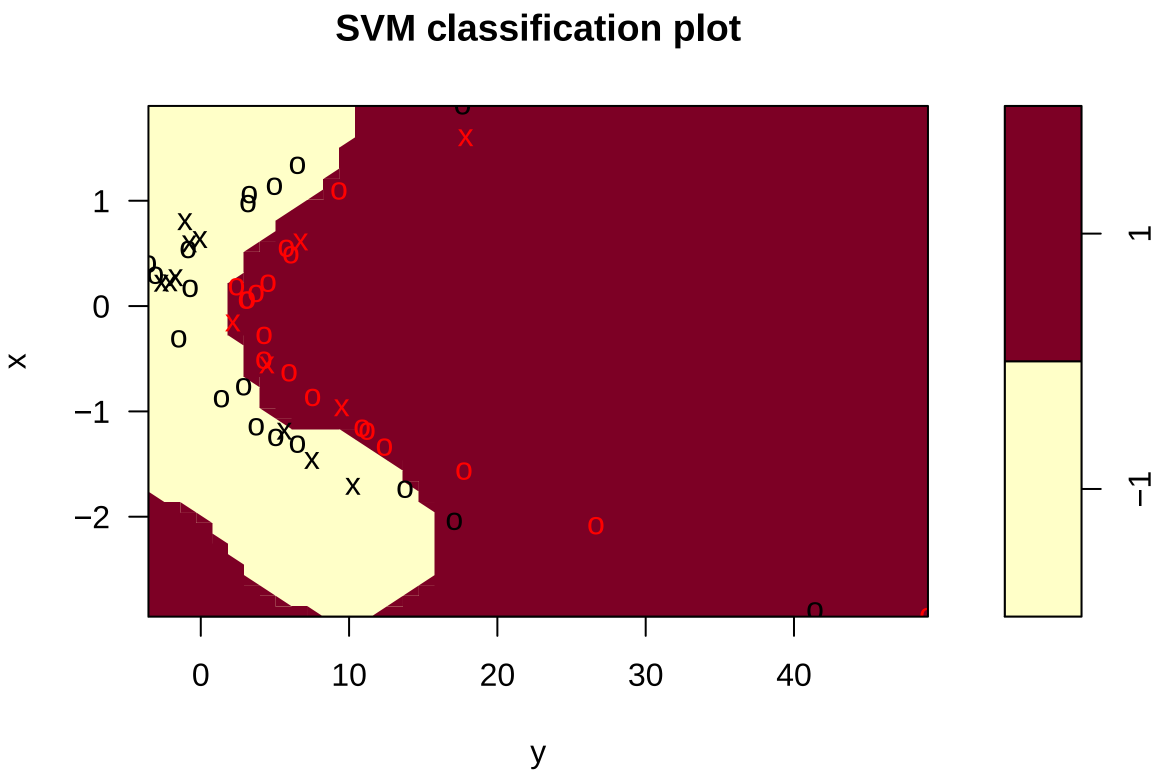
```r
# polynomial kernel
plot(svm.poly, data.test)
```



```r
table(predict = predict(svm.poly, data.test), truth = data.test$z)
```

```
##         truth
## predict -1  1
##      -1 15  0
##      1  12 23
```

```r
# radial
plot(svm.radial, data.test)
```



```r
table(predict = predict(svm.radial, data.test), truth = data.test$z)
```

```
##         truth
## predict -1  1
##      -1 24  0
##      1   3 23
```

We see that the linear, polynomial and radial support vector machines each classify 15, 12 and 3 observations incorrectly. Radial kernel has the smallest number of misclassification, thus the best model in the setting.

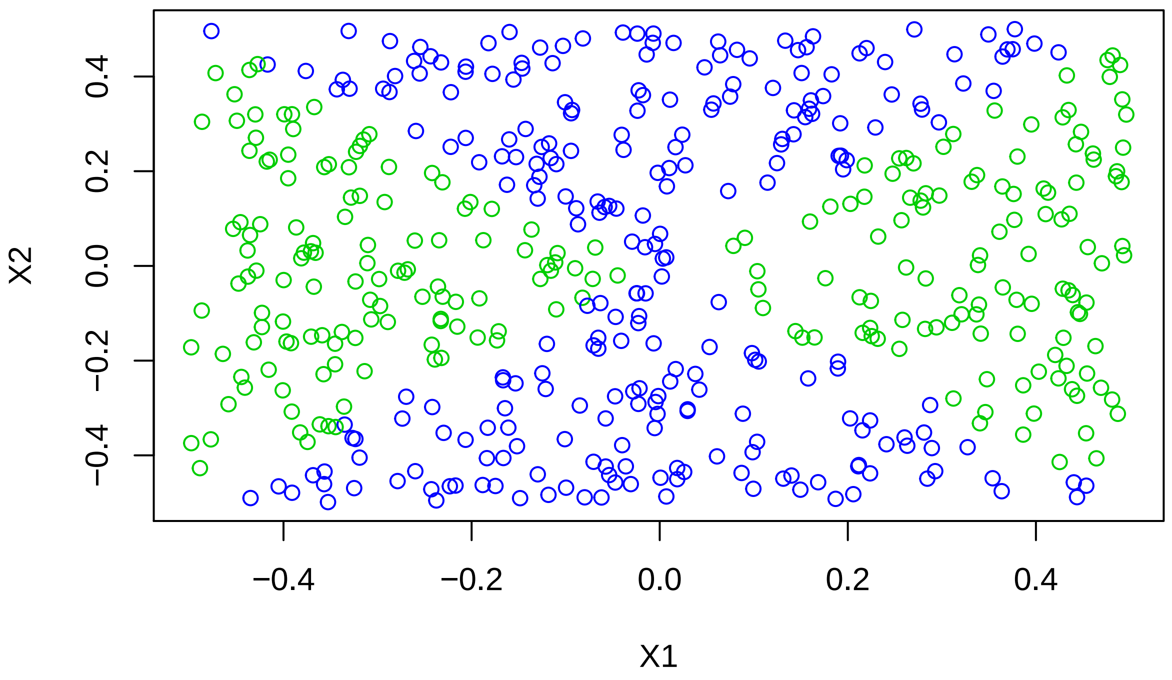
SVM vs. logistic regression

2. (5 points) Generate a data set with $n = 500$ and $p = 2$, such that the observations belong to two classes with some **overlapping, non-linear** boundary between them.

```r
set.seed(1)
X1 = runif(500) - 0.5
X2 = runif(500) - 0.5
Y = 1*(X1^2-X2^2 > 0)
```

3. (5 points) Plot the observations with colors according to their class labels (y). Your plot should display X_1 on the x-axis and X_2 on the y-axis.

```r
plot(X1, X2, col = (4 - Y))
```



4. (5 points) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

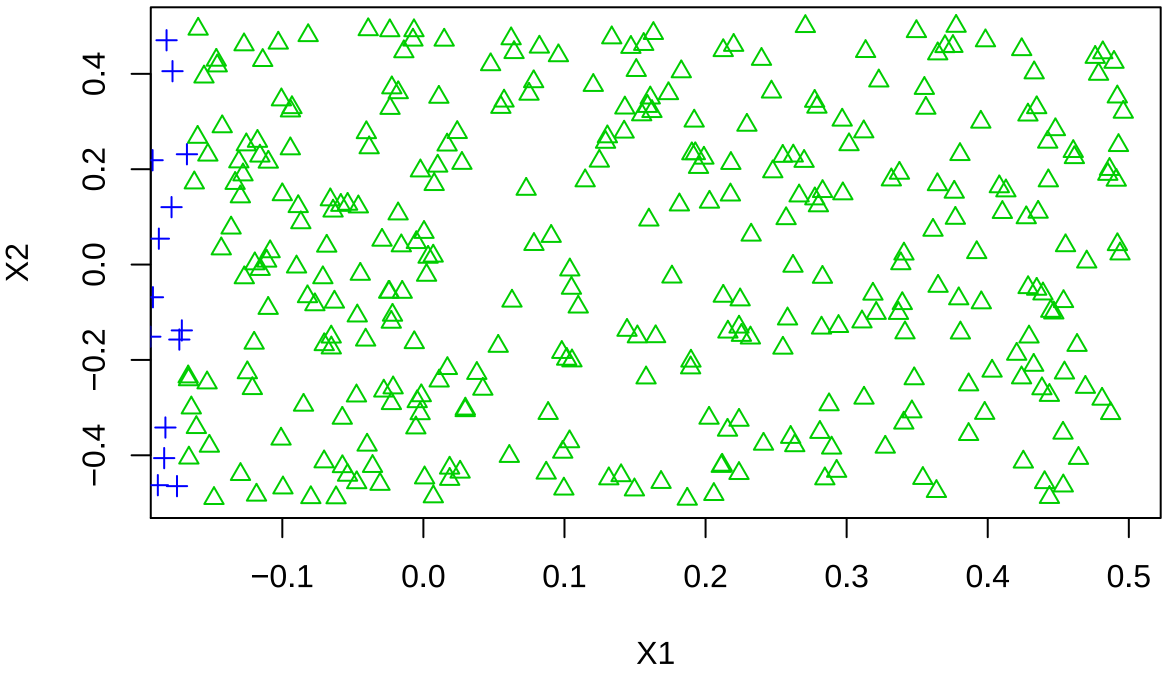
```
logit.fit = glm(Y ~ X1 + X2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = Y ~ X1 + X2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## X1           0.196199   0.316864   0.619    0.536
## X2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

All features are statistially insignificant.

5. (5 points) Obtain a predicted class label for each observation based on the logistic model previously fit. Plot the observations, colored according to the *predicted* class labels (*the predicted decision boundary should look linear*).

```
data = data.frame(X1 = X1, X2 = X2, Y = Y)
probs = predict(logit.fit, data, type = "response")
preds = rep(0, 500)
preds[probs > 0.47] = 1
plot(data[preds == 1, ]$X1, data[preds == 1, ]$X2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X
points(data[preds == 0, ]$X1, data[preds == 0, ]$X2, col = (4 - 0), pch = (3 - 0))
```



The decision boundary is linear and on the left side of the plot.

6. (5 points) Now fit a logistic regression model to the data, but this time using some *non-linear function* of both X_1 and X_2 as predictors (e.g. $X_1^2, X_1 \times X_2, \log(X_2)$, and so on).

```
logitnl.fit <- glm(Y ~ poly(X1, 2) + poly(X2, 2) + I(X1 * X2), family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

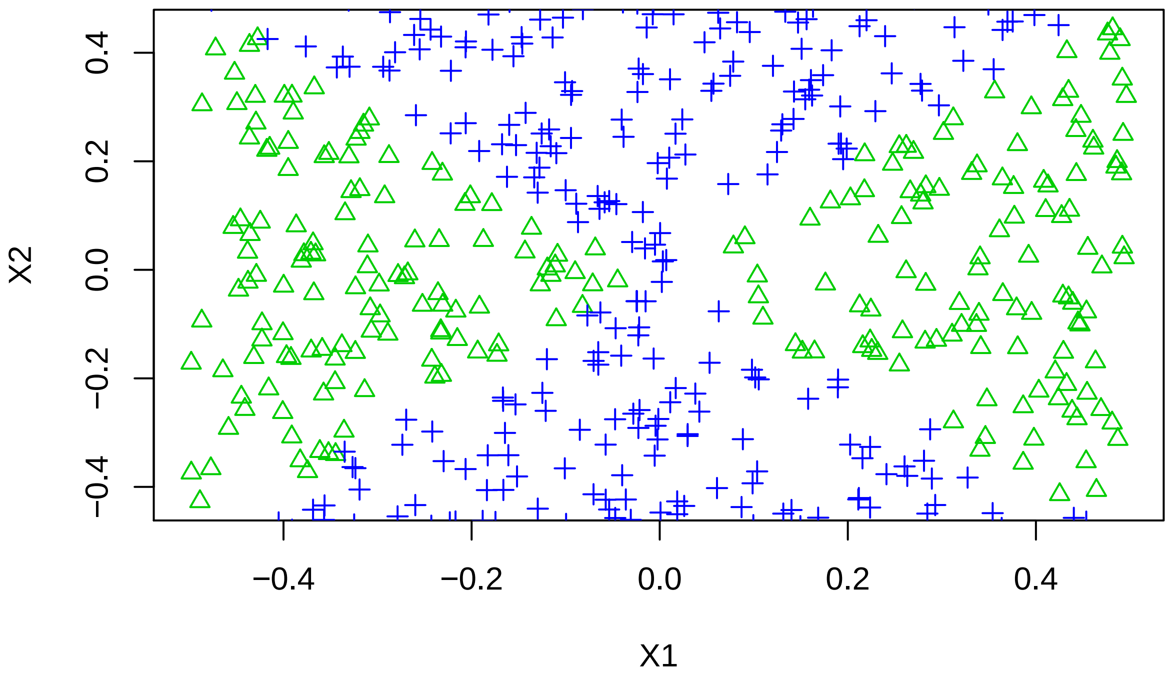
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logitnl.fit)
```

```
##
## Call:
## glm(formula = Y ~ poly(X1, 2) + poly(X2, 2) + I(X1 * X2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -8.240e-04  -2.000e-08  -2.000e-08   2.000e-08   1.163e-03
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -102.2     4302.0  -0.024    0.981
## poly(X1, 2)1     2715.3   141109.5   0.019    0.985
## poly(X1, 2)2    27218.5   842987.2   0.032    0.974
## poly(X2, 2)1     -279.7    97160.4  -0.003    0.998
## poly(X2, 2)2  -28693.0   875451.3  -0.033    0.974
## I(X1 * X2)       -206.4    41802.8  -0.005    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
## Residual deviance: 3.5810e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

7. (5 points) Now, obtain a predicted class label for each observation based on the fitted model with non-linear transformations of the X features in the previous question. Plot the observations, colored according to the new *predicted* class labels from the non-linear model (*the decision boundary should now be obviously non-linear*). If it is not, then repeat earlier steps until you come up with an example in which the predicted class labels and the resultant decision boundary are clearly non-linear.

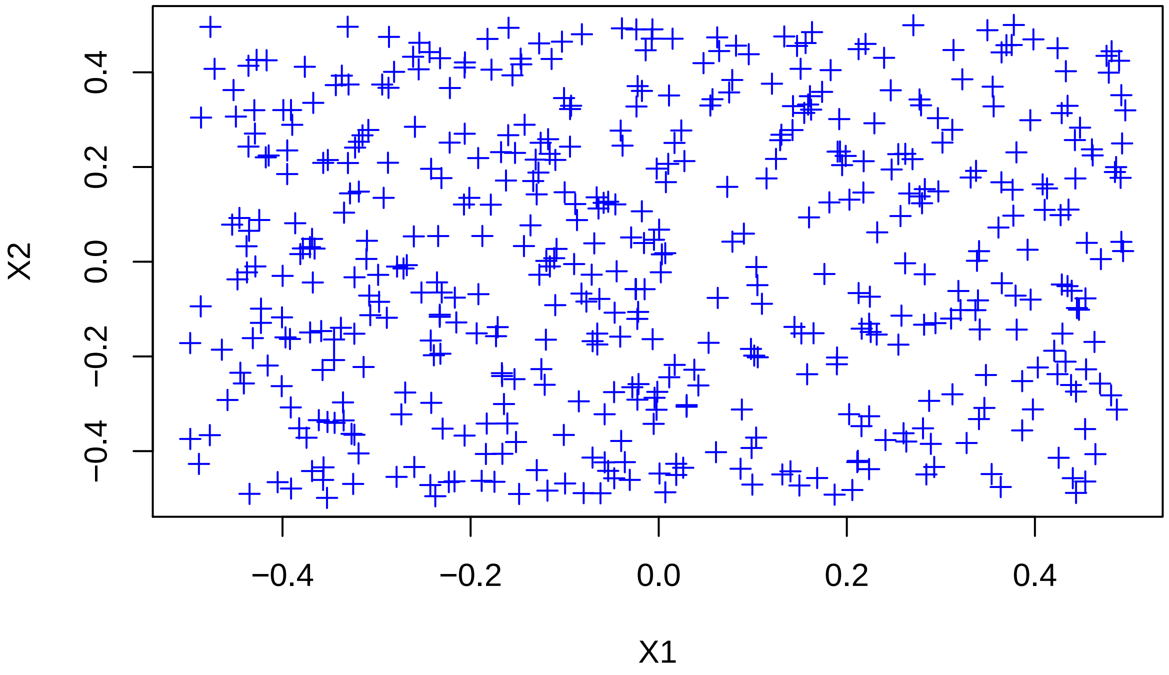
```
probs <- predict(logitnl.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$X1, data[preds == 1, ]$X2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X
points(data[preds == 0, ]$X1, data[preds == 0, ]$X2, col = (4 - 0), pch = (3 - 0))
```



The non-linear decision boundary seems similar to the true decision boundary presented in the first question.

8. (5 points) Now, fit a support vector classifier (linear kernel) to the data with *original* X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the *predicted* class labels.

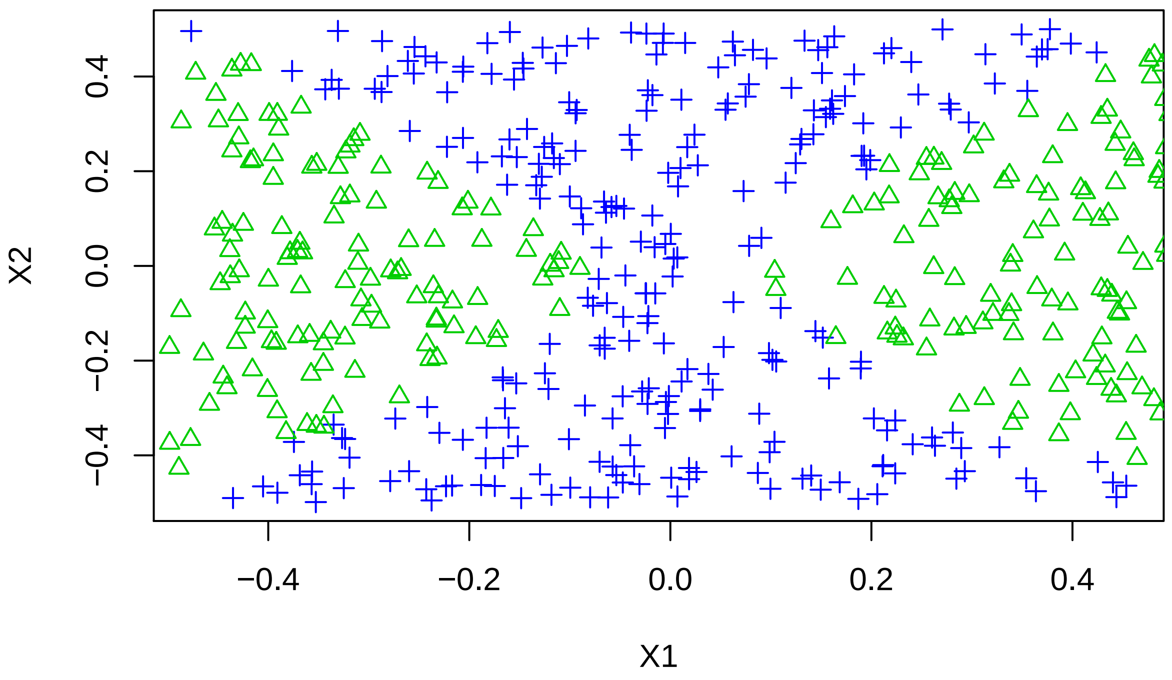
```
data$Y <- as.factor(data$Y)
svm.fit <- svm(Y ~ X1 + X2, data, kernel = "linear", cost = 0.01)
preds <- predict(svm.fit, data)
plot(data[preds == 0, ]$X1, data[preds == 0, ]$X2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X
points(data[preds == 1, ]$X1, data[preds == 1, ]$X2, col = (4 - 1), pch = (3 - 1))
```



It classified all data points as one single class.

9. (5 points) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each observation. Plot the observations, colored according to the *predicted* class labels.

```
data$Y <- as.factor(data$Y)
svmnl.fit <- svm(Y ~ X1 + X2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 0, ]$X1, data[preds == 0, ]$X2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X
points(data[preds == 1, ]$X1, data[preds == 1, ]$X2, col = (4 - 1), pch = (3 - 1))
```



Again, this non-linear decision boundary seems very similar to the true boundary.

10. (5 points) Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

The result suggests that logistic regression with interaction terms and SVM with non-linear kernel are the best models for finding non-linear decision boundaries. As seen in the plots, logistic regression without interaction terms and SVM with linear kernel have less power in predicting the true decision boundary. When using SVM, we only need to tune γ(gamma), whereas logistic regression requires some degree of tuning to find the correct interaction terms.

Tuning cost

In class we learned that in the case of data that is just barely linearly separable, a support vector classifier with a small value of `cost` that misclassifies a couple of training observations may perform better on test data than one with a huge value of `cost` that does not misclassify any training observations. You will now investigate that claim.

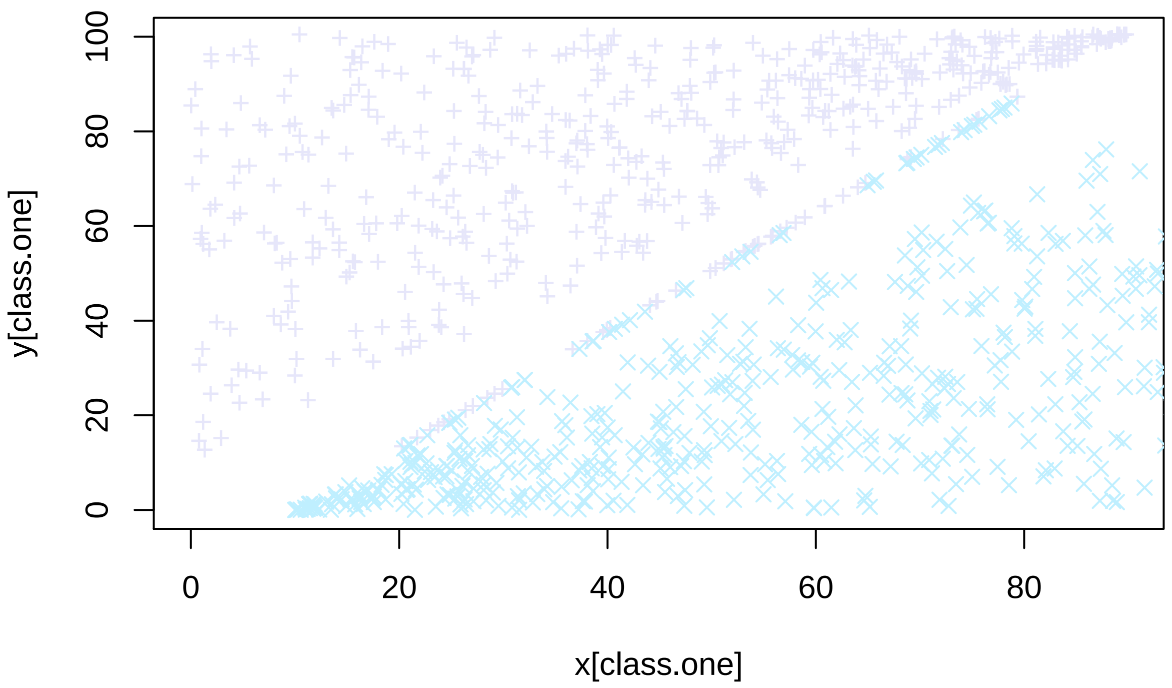
11. (5 points) Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

```r
set.seed(77)
x.one <- runif(500, 0, 90)
y.one <- runif(500, x.one + 10, 100)
x.one.noise <- runif(50, 20, 80)
y.one.noise <- 5/4 * (x.one.noise - 10) + 0.1

x.zero <- runif(500, 10, 100)
y.zero <- runif(500, 0, x.zero - 10)
x.zero.noise <- runif(50, 20, 80)
y.zero.noise <- 5/4 * (x.zero.noise - 10) - 0.1
```

```r
class.one <- seq(1, 550)
x <- c(x.one, x.one.noise, x.zero, x.zero.noise)
y <- c(y.one, y.one.noise, y.zero, y.zero.noise)

plot(x[class.one], y[class.one], col = "lavender", pch = "+", ylim = c(0, 100))
points(x[-class.one], y[-class.one], col = "lightblue1", pch = 4)
```



12. (5 points) Compute the cross-validation error rates for support vector classifiers with a range of `cost` values. How many training errors are made for each value of `cost` considered, and how does this relate to the cross-validation errors obtained?

```r
set.seed(33)
z <- rep(0, 1100)
z[class.one] <- 1
data <- data.frame(x = x, y = y, z = as.factor(z))
tune.out <- tune(svm, z ~ ., data = data, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1, 5, 10
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  10000
##
## - best performance: 0
##
## - Detailed performance results:
```

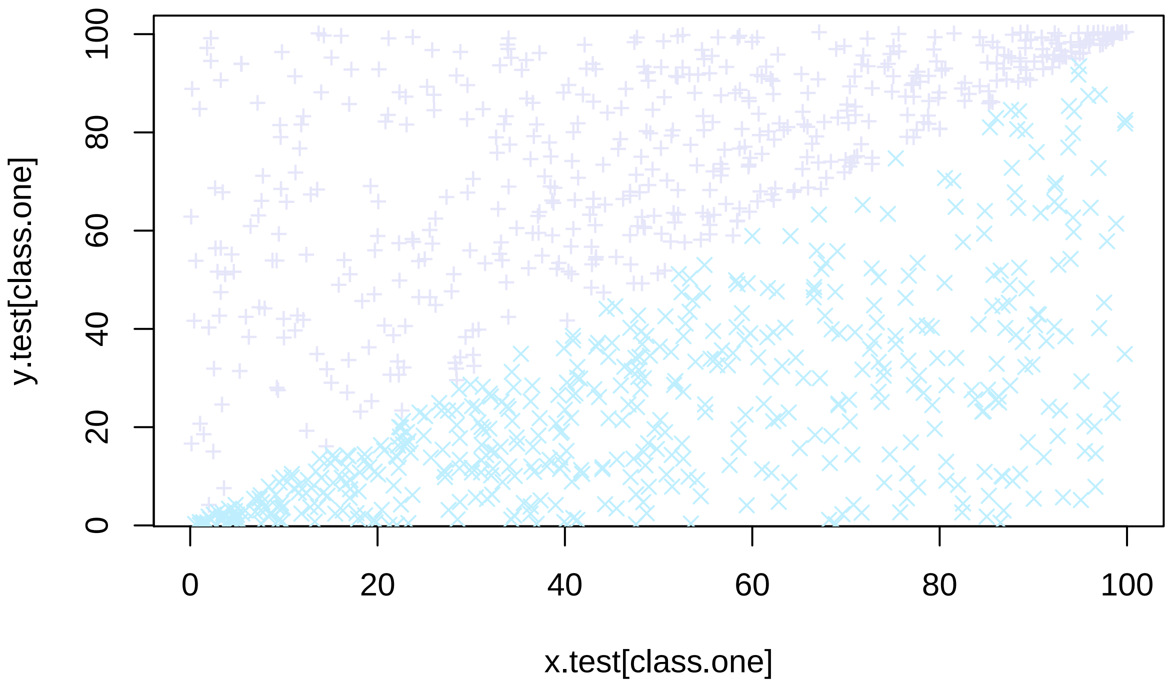
```
##    cost      error dispersion
## 1 1e-02 0.05454545 0.01714198
## 2 1e-01 0.04272727 0.01215903
## 3 1e+00 0.04272727 0.01215903
## 4 5e+00 0.04272727 0.01215903
## 5 1e+01 0.04181818 0.01067080
## 6 1e+02 0.04363636 0.01196874
## 7 1e+03 0.02363636 0.02355074
## 8 1e+04 0.00000000 0.00000000
```

```r
tibble(cost = tune.out$performance$cost, misclass = tune.out$performance$error * 1100)
```

```
## # A tibble: 8 x 2
##       cost misclass
##      <dbl>    <dbl>
## 1     0.01     60.0
## 2     0.1      47.0
## 3     1        47.0
## 4     5        47.0
## 5    10        46.0
## 6   100        48.0
## 7  1000        26.0
## 8 10000         0
```

13. (5 points) Generate an appropriate test data set, and compute the test errors corresponding to each of the values of `cost` considered. Which value of `cost` leads to the fewest test errors, and how does this compare to the values of `cost` that yield the fewest training errors and the fewest cross-validation errors?

```r
x.test <- runif(1000, 0, 100)
class.one <- sample(1000, 500)
y.test <- rep(NA, 1000)
# Set y > x for class.one
for (i in class.one) {
    y.test[i] <- runif(1, x.test[i], 100)
}
# set y < x for class.zero
for (i in setdiff(1:1000, class.one)) {
    y.test[i] <- runif(1, 0, x.test[i])
}
plot(x.test[class.one], y.test[class.one], col = "lavender", pch = "+")
points(x.test[-class.one], y.test[-class.one], col = "lightblue1", pch = 4)
```



```r
set.seed(40)
z.test <- rep(0, 1000)
z.test[class.one] <- 1
data.test <- data.frame(x = x.test, y = y.test, z = as.factor(z.test))
costs <- c(0.01, 0.1, 1, 5, 10, 100, 1000, 10000)
test.err <- rep(NA, length(costs))
for (i in 1:length(costs)) {
    svm.fit <- svm(z ~ ., data = data, kernel = "linear", cost = costs[i])
    pred <- predict(svm.fit, data.test)
    test.err[i] <- sum(pred != data.test$z)
}
tibble(cost = costs, misclass = test.err)
```

```
## # A tibble: 8 x 2
##       cost misclass
##      <dbl>    <int>
## 1     0.01       44
## 2     0.1        12
## 3     1           1
## 4     5           0
## 5    10           0
## 6   100         185
## 7  1000         212
## 8 10000         214
```

14. (5 points) Discuss your results.

On the training set, it seems that a higher cost seems to correctly classify the nosiy data points, thus overfitting the data. The highest cost we have, 1e+04, has zero misclassification, which suggests the overfitting tendency. On the other hand, on the test set, it seems that higher costs have significantly more misclassifications than smaller ones.

Application: Predicting attitudes towards racist college professors

15. (5 points) Fit a support vector classifier to predict `colrac` as a function of all available predictors, using 10-fold cross-validation to find an optimal value for `cost`. Report the CV errors associated with different values of cost, and discuss your results.

```
gss_train <- read_csv("gss_train.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )

## See spec(...) for full column specifications.
```

```
gss_test <- read_csv("gss_test.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
## See spec(...) for full column specifications.
```

```
set.seed(77)
gss_tuned <- tune(svm, colrac ~ ., data = gss_train, kernel = "linear",
                  ranges = list(cost = c(0.01, 0.1, 1, 5)))
summary(gss_tuned)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.181181
##
## - Detailed performance results:
##   cost     error dispersion
## 1 0.01 0.1811810 0.01986164
## 2 0.10 0.1995654 0.02094616
## 3 1.00 0.2010758 0.02093637
## 4 5.00 0.2012605 0.02085335
```

This SVM performs the best when the cost equals 0.01 with the error rate of 0.1811. There seems to be a positive relationship between the cost and the error rate; as the cost increases, the error rate increases as well.

16. (15 points) Repeat the previous question, but this time using SVMs with **radial** *and* **polynomial** basis kernels, with different values for `gamma` and `degree` and `cost`. **Present and discuss your results** (e.g., fit, compare kernels, cost, substantive conclusions across fits, etc.).

```r
set.seed(33)
radial_tune <- tune(svm, colrac ~ ., data = gss_train, kernel = "radial",
                 ranges = list(gamma = c(0.01, 0.1, 1, 5), degree = c(2, 3, 4)))
summary(radial_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma degree
##   0.01      2
##
## - best performance: 0.1483846
##
## - Detailed performance results:
##    gamma degree     error  dispersion
## 1   0.01      2 0.1483846 0.018361305
## 2   0.10      2 0.2167099 0.007202750
## 3   1.00      2 0.2499691 0.002370117
## 4   5.00      2 0.2499696 0.002370058
## 5   0.01      3 0.1483846 0.018361305
## 6   0.10      3 0.2167099 0.007202750
## 7   1.00      3 0.2499691 0.002370117
## 8   5.00      3 0.2499696 0.002370058
## 9   0.01      4 0.1483846 0.018361305
## 10  0.10      4 0.2167099 0.007202750
## 11  1.00      4 0.2499691 0.002370117
## 12  5.00      4 0.2499696 0.002370058
```

For the SVM with radial kernel, the best performing SVM is the one with gamma = 0.01 and degree = 2, yielding the error rate of 0.1483.

```r
polynomial_tune <- tune(svm, colrac ~ ., data = gss_train, kernel = "polynomial",
                        ranges = list(cost = c(0.01, 0.1, 1, 5), degree = c(2, 3, 4)))
summary(polynomial_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##     1      3
##
## - best performance: 0.1512064
##
## - Detailed performance results:
##    cost degree     error dispersion
## 1  0.01      2 0.4069273 0.04065524
## 2  0.10      2 0.2645330 0.02847615
## 3  1.00      2 0.2128994 0.02935936
## 4  5.00      2 0.2790585 0.03579790
## 5  0.01      3 0.3705417 0.04147717
## 6  0.10      3 0.1802739 0.01729942
## 7  1.00      3 0.1512064 0.01948883
## 8  5.00      3 0.1587812 0.02265867
## 9  0.01      4 0.4158091 0.04135714
## 10 0.10      4 0.3120807 0.03894149
## 11 1.00      4 0.1868791 0.01665178
## 12 5.00      4 0.1854972 0.01842834
```

The SVM with a polynomial kernel, the lowest cross validation error is 0.1512 with a degree of 3 and a cost of 1.