Satinitigan_Karl_HW6

Karl Satinitigan

3/8/2020

Conceptual exercises

Non-linear separation

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------ tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts --------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(tidymodels)
```

```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## -- Attaching packages ----------------------------------------------------- tidymodels 0.0.3 --
```

```
## v broom     0.5.4     v recipes   0.1.9
## v dials     0.0.4     v rsample   0.0.5
## v infer     0.5.1     v yardstick 0.0.4
## v parsnip   0.0.5
```

```
## -- Conflicts -------------------------------------------------------- tidymodels_conflicts() --
## x scales::discard()   masks purrr::discard()
## x dplyr::filter()     masks stats::filter()
## x recipes::fixed()    masks stringr::fixed()
## x dplyr::lag()        masks stats::lag()
## x dials::margin()     masks ggplot2::margin()
## x yardstick::spec()   masks readr::spec()
## x recipes::step()     masks stats::step()
## x recipes::yj_trans() masks scales::yj_trans()
```

```r
library(patchwork)
library(rcfss)
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:yardstick':
##
```

```
##     precision, recall
## The following object is masked from 'package:purrr':
##
##     lift
```

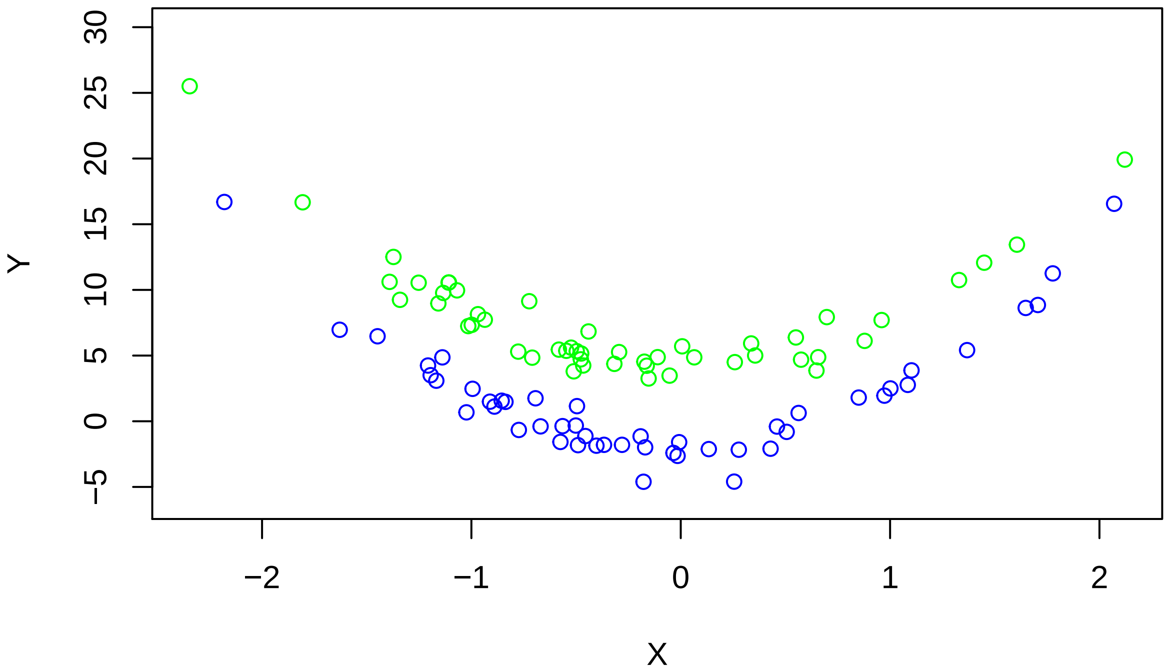
```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

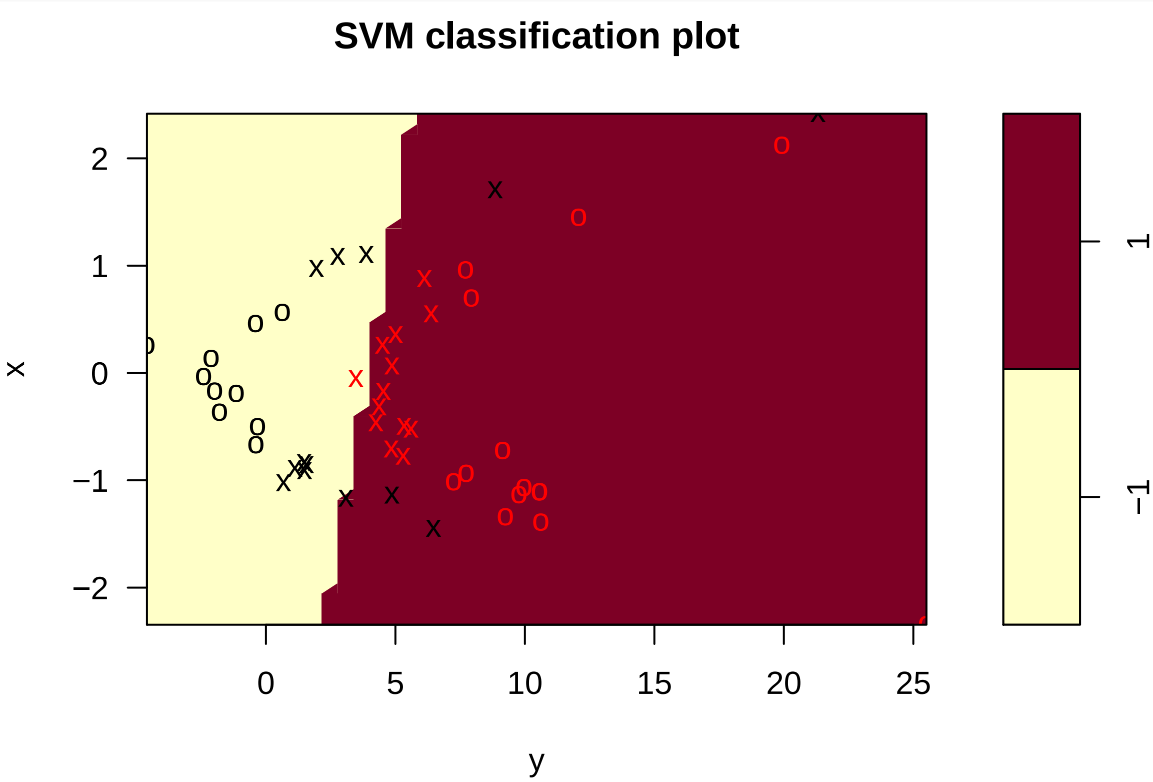
```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:scales':
##
##     alpha
## The following object is masked from 'package:purrr':
##
##     cross
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
set.seed(1234)
theme_set(theme_minimal())

x = rnorm(100)
y = 4 * 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 = "green", xlab = "X", ylab = "Y", ylim = c(-6, 30))
points(x[-class], y[-class], col = "blue")
```




```
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)
```

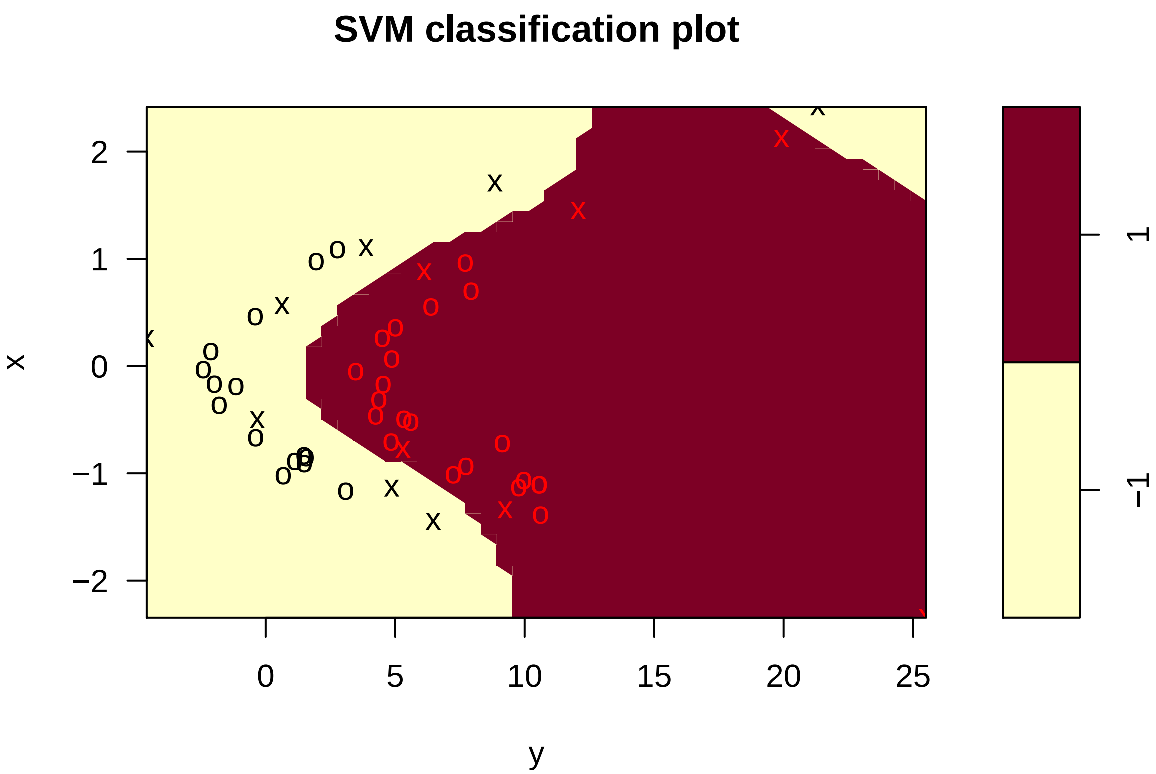


```r
table(predict = predict(svm.linear, data.train), truth = data.train$z)
```

```
##         truth
## predict -1  1
##      -1 18  1
##      1   5 26
```

The support vector classifier makes 6 errors on the training data.

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

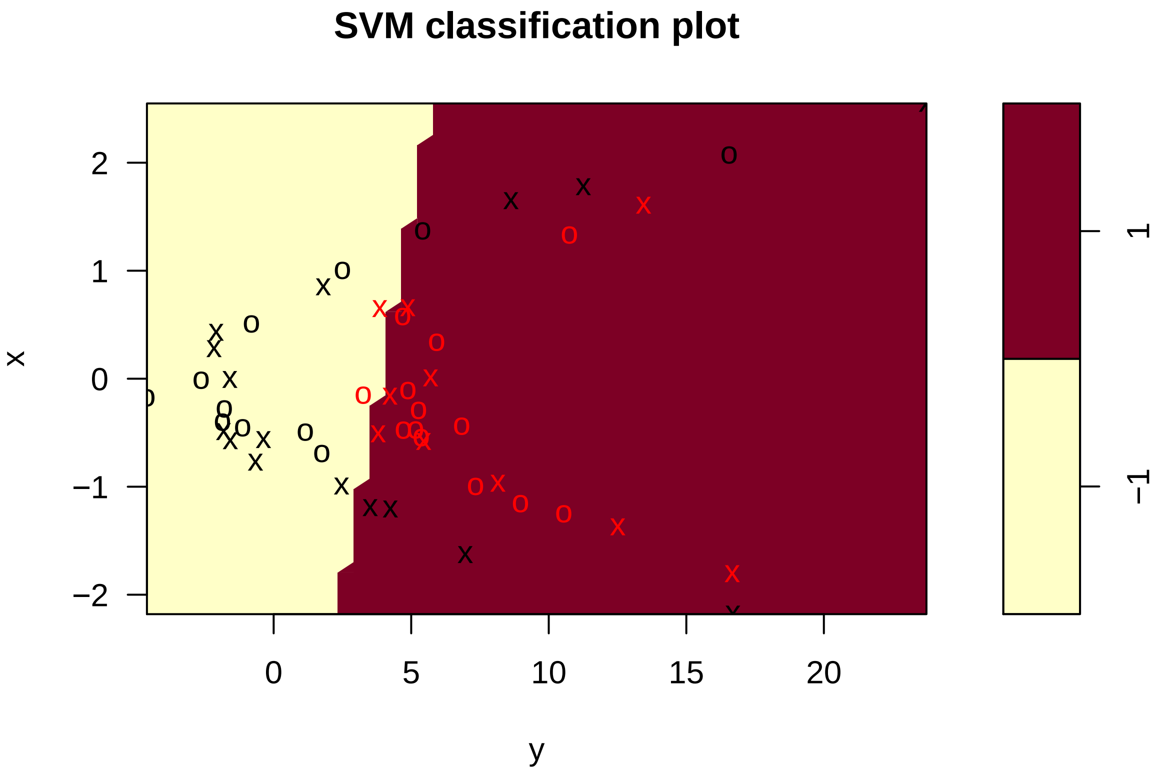



```r
table(predict = predict(svm.radial, data.train), truth = data.train$z)
```

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

The support vector classifier makes no errors on the training data.

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

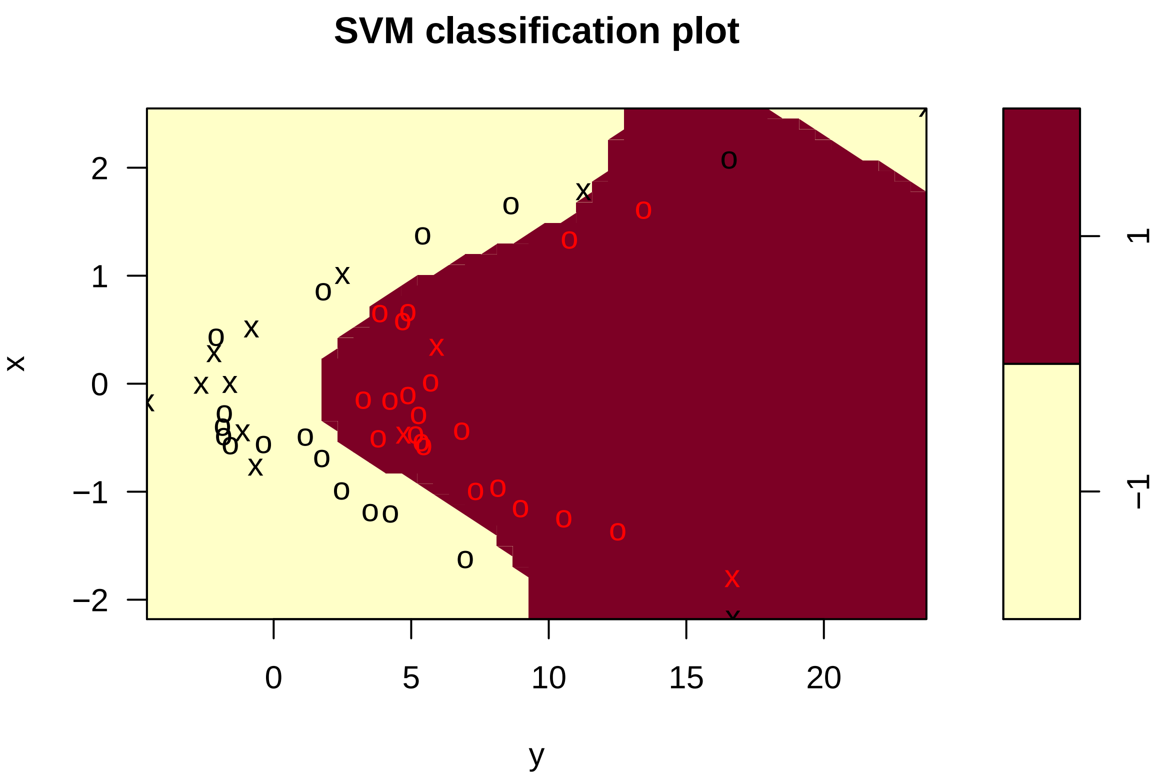



```r
table(predict = predict(svm.linear, data.test), truth = data.test$z)
```

```
##          truth
## predict -1  1
##      -1 18  2
##      1   9 21
```

The support vector classifier makes 11 errors on the test data.

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



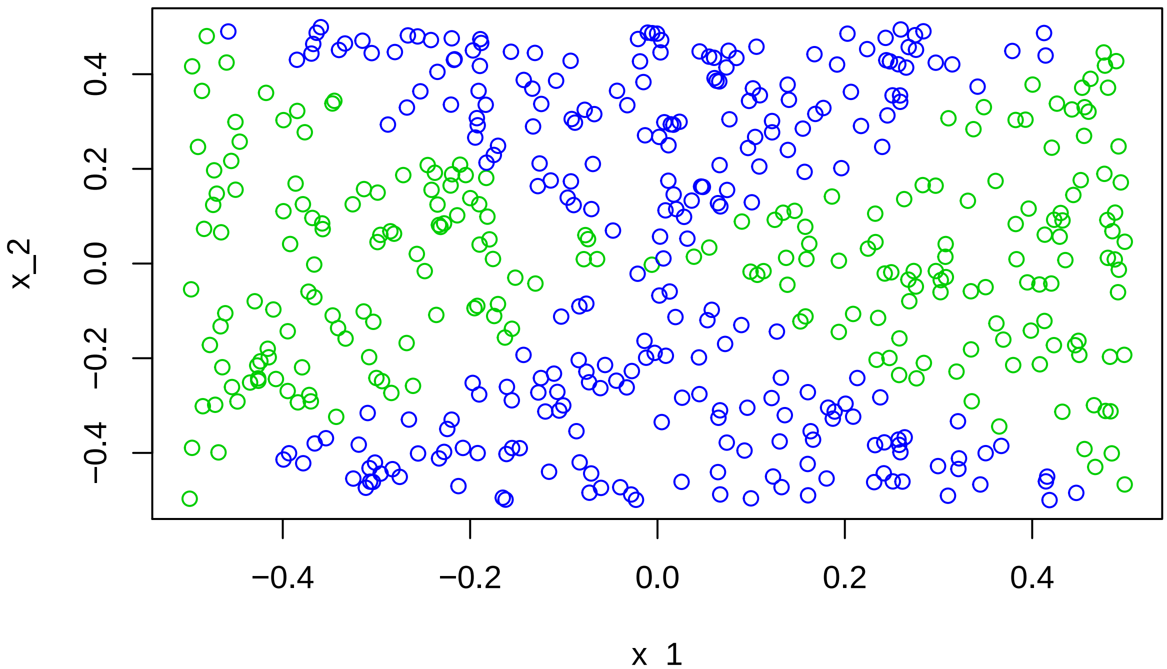

```r
table(predict = predict(svm.radial, data.test), truth = data.test$z)
```

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

The support vector classifier with radial kernel makes 3 errors on the test data. On both training and test data, the support vector classifier with radial kernel outperformed the one with a linear kernel.

SVM vs logistic regression

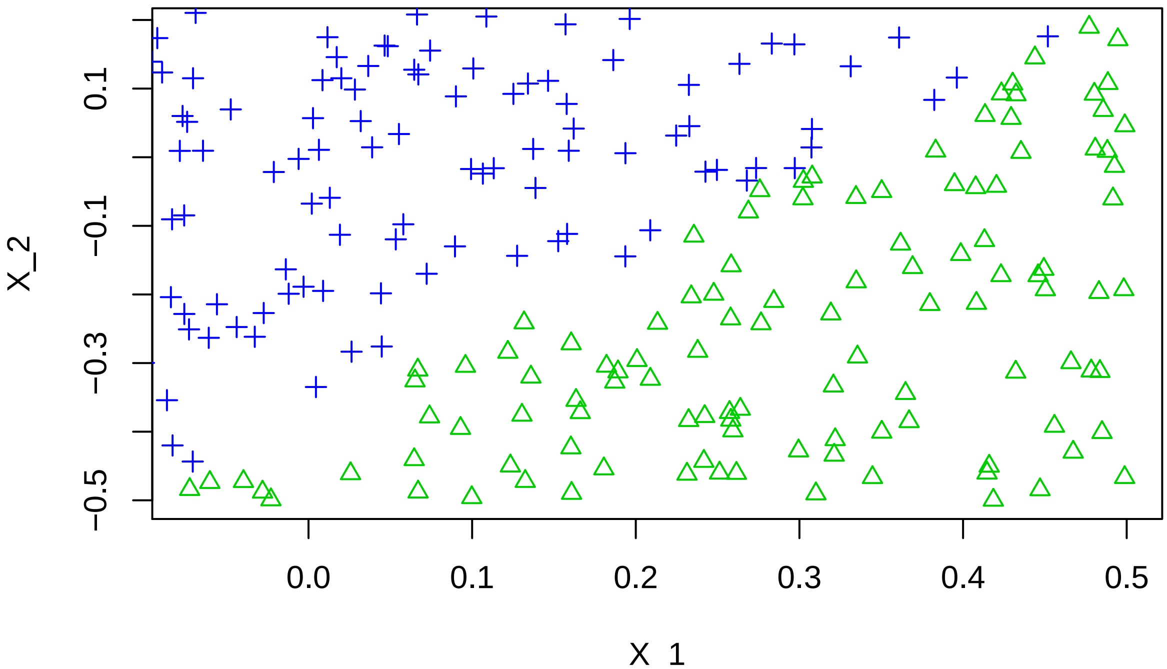
```r
set.seed(1234)
x_1 = runif(500)-0.5
x_2 = runif(500)-0.5
y = 1 * (x_1^2 - x_2^2 > 0)

plot (x_1, x_2, col = (4-y))
```




```r
logit.fit = glm(y ~ x_1 + x_2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = y ~ x_1 + x_2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.161  -1.107  -1.072   1.243   1.308
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.16814    0.08985  -1.871   0.0613 .
## x_1          0.15254    0.31736   0.481   0.6308
## x_2         -0.12672    0.30048  -0.422   0.6732
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 689.62  on 499  degrees of freedom
## Residual deviance: 689.21  on 497  degrees of freedom
## AIC: 695.21
##
## Number of Fisher Scoring iterations: 3
```

```r
data = data.frame(x_1 = x_1, x_2 = x_2, y = y)
probs = predict(logit.fit, data, type = "response")
preds = rep(0, 500)
preds[probs > 0.47] = 1
plot(data[preds == 1, ]$x_1, data[preds == 1, ]$x_2, col = (4 - 1), pch = (3 - 1), xlab = "X_1", ylab =
points(data[preds == 0, ]$x_1, data[preds == 0, ]$x_2, col = (4 - 0), pch = (3 - 0))
```




```
logitn1.fit <- glm(y ~ poly(x_1, 2) + poly(x_2, 2) + I(x_1 * x_2), family = "binomial")
```

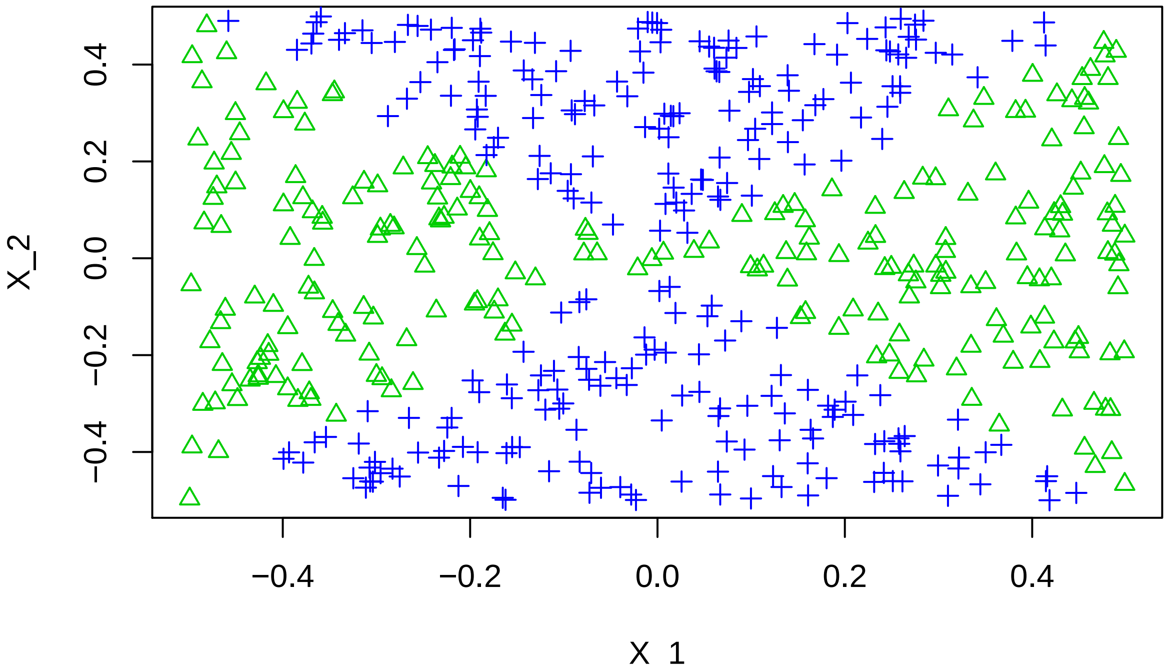
```
## Warning: glm.fit: algorithm did not converge

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

```
summary(logitn1.fit)
```

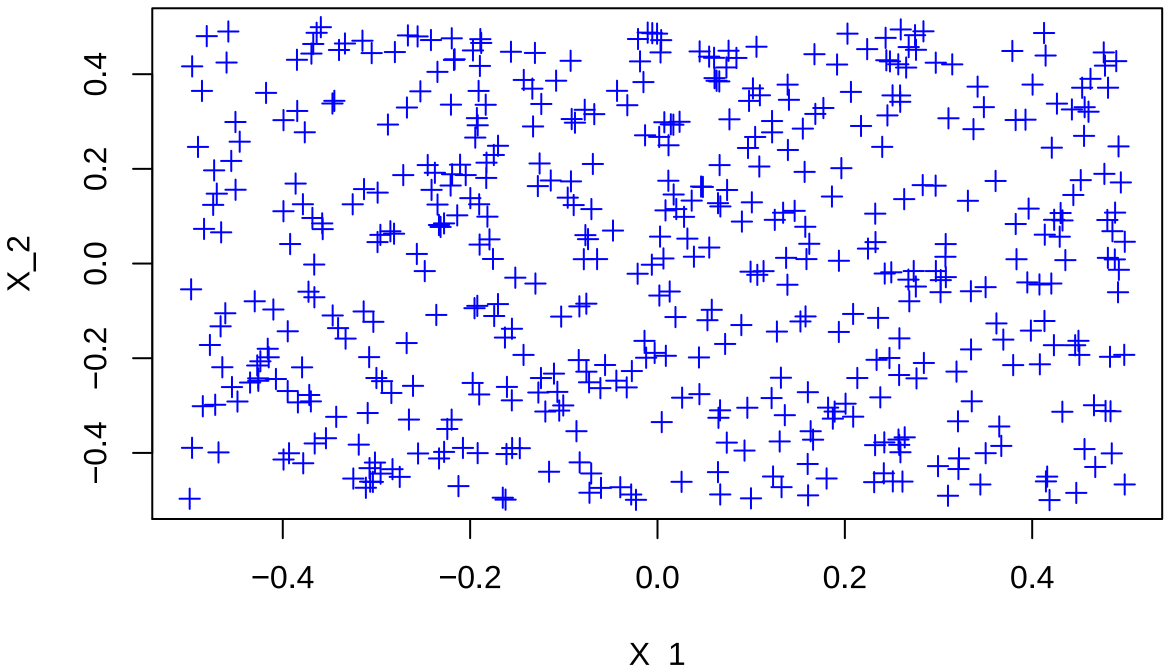
```
##
## Call:
## glm(formula = y ~ poly(x_1, 2) + poly(x_2, 2) + I(x_1 * x_2),
##     family = "binomial")
##
## Deviance Residuals:
##    Min       1Q   Median       3Q      Max
##  -8.49     0.00     0.00     0.00     0.00
##
## Coefficients:
##                 Estimate Std. Error    z value Pr(>|z|)
## (Intercept)   -3.251e+14  3.001e+06 -108315297   <2e-16 ***
## poly(x_1, 2)1  1.364e+15  6.712e+07   20314872   <2e-16 ***
## poly(x_1, 2)2  5.502e+16  6.721e+07  818622478   <2e-16 ***
## poly(x_2, 2)1 -3.402e+14  6.715e+07   -5065879   <2e-16 ***
## poly(x_2, 2)2 -5.745e+16  6.721e+07 -854764051   <2e-16 ***
## I(x_1 * x_2)   8.402e+14  3.607e+07   23290795   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 689.62  on 499  degrees of freedom
## Residual deviance: 144.17  on 494  degrees of freedom
## AIC: 156.17
##
```

```
## Number of Fisher Scoring iterations: 25
```

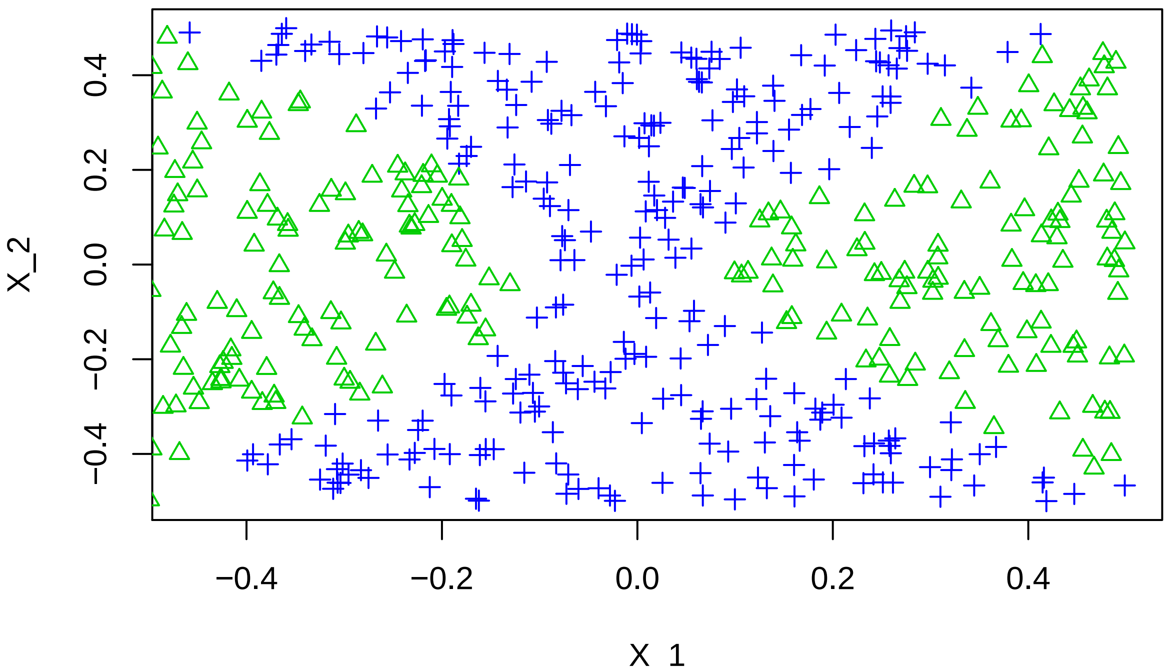
```
probs <- predict(logitn1.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x_1, data[preds == 1, ]$x_2, col = (4 - 1), pch = (3 - 1), xlab = "X_1", ylab =
points(data[preds == 0, ]$x_1, data[preds == 0, ]$x_2, col = (4 - 0), pch = (3 - 0))
```




```
data$y <- as.factor(data$y)
svm.fit <- svm(y ~ x_1 + x_2, data, kernel = "linear", cost = 0.01)
preds <- predict(svm.fit, data)
plot(data[preds == 0, ]$x_1, data[preds == 0, ]$x_2, col = (4 - 0), pch = (3 - 0), xlab = "X_1", ylab =
points(data[preds == 1, ]$x_1, data[preds == 1, ]$x_2, col = (4 - 1), pch = (3 - 1))
```



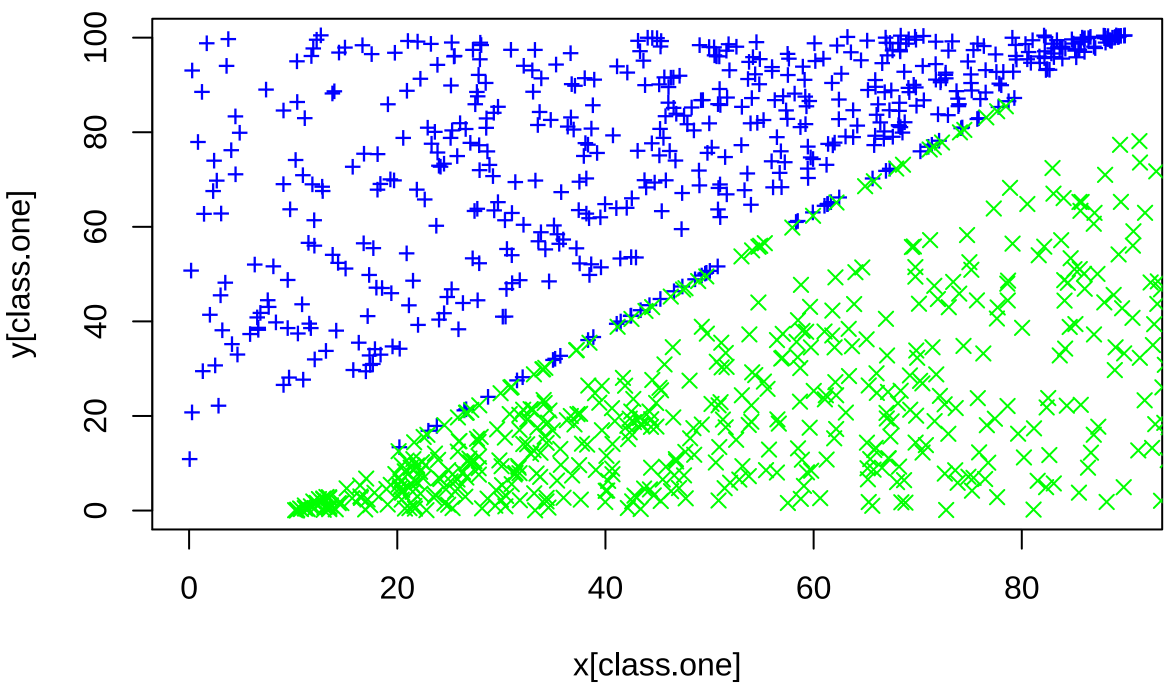

```r
data$y <- as.factor(data$y)
svmnl.fit <- svm(y ~ x_1 + x_2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 0, ]$x_1, data[preds == 0, ]$x_2, col = (4 - 0), pch = (3 - 0), xlab = "X_1", ylab =
points(data[preds == 1, ]$x_1, data[preds == 1, ]$x_2, col = (4 - 1), pch = (3 - 1))
```



Support vector machines with non-linear kernel and logistic regression with interaction terms are ideal for finding non-linear decision boundaries. Meanwhile, support vector machines with linear kernel and logistic regression without interaction terms are not ideal for finding non-linear decision boundaries.

Tuning cost

```
set.seed(1234)
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

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




```
set.seed(1234)
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:
```

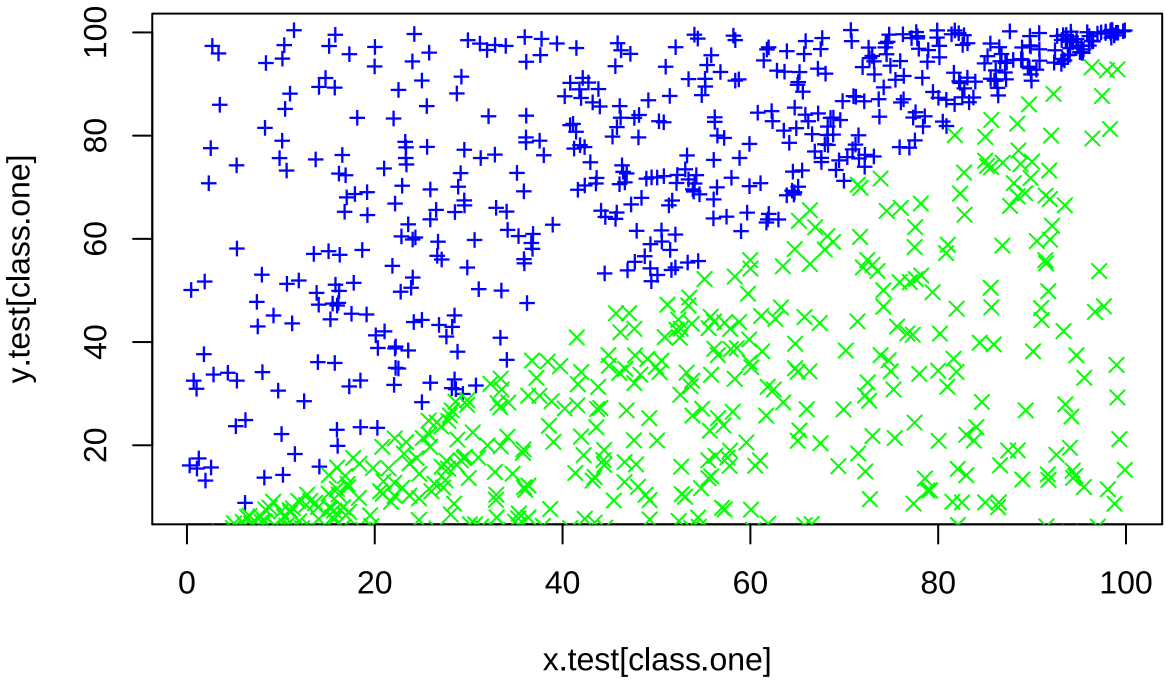
```
##   cost
##  10000
##
## - best performance: 0
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-02 0.04727273 0.014083576
## 2 1e-01 0.04000000 0.013686776
## 3 1e+00 0.04000000 0.016707938
## 4 5e+00 0.04090909 0.016735395
## 5 1e+01 0.04272727 0.017168746
## 6 1e+02 0.04181818 0.017248787
## 7 1e+03 0.01818182 0.007422696
## 8 1e+04 0.00000000 0.000000000
```

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

```
##    cost misclass
## 1 1e-02       52
## 2 1e-01       44
## 3 1e+00       44
## 4 5e+00       45
## 5 1e+01       47
## 6 1e+02       46
## 7 1e+03       20
## 8 1e+04        0
```

The table shows the training errors misclassified with the cost of 10000 classifying all training points correctly.

```r
x.test <- runif(1000, 0, 100)
class.one <- sample(1000, 500)
y.test <- rep(NA, 1000)

for (i in class.one) {
  y.test[i] <- runif(1, x.test[i], 100)
}

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 = "blue", pch = "+")
points(x.test[-class.one], y.test[-class.one], col = "green", pch = 4)
```




```r
set.seed(1234)
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)
}
data.frame(cost = costs, misclass = test.err)
```

```
##    cost misclass
## 1 1e-02       48
## 2 1e-01       19
## 3 1e+00       11
## 4 5e+00        7
## 5 1e+01        6
## 6 1e+02      169
## 7 1e+03      212
## 8 1e+04      214
```

Cost of 5 and 10 seem to perform better on test observations. Here we see that a large cost correctly classifies training points and overfits the training data. However, a small cost makes a few errors on the test points and performs better on the test data.

Application: Predicting attitudes towards racist college professors

```r
gsstrain <- read_csv(url("https://raw.githubusercontent.com/ksatinitigan/problem-set-6/master/data/gss_
```

```
## Parsed with column specification:
```

```
## cols(
##   .default = col_double()
## )

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

```
set.seed(1)
colrac.svmlinear <- tune(svm, colrac ~ ., data = gsstrain, kernel = "linear", ranges = list(cost = c(0.0

summary(colrac.svmlinear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.1818622
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.1818622 0.02880076
## 2 1e-01 0.2002275 0.03216013
## 3 1e+00 0.2018497 0.03216100
## 4 5e+00 0.2019705 0.03209954
## 5 1e+01 0.2019792 0.03214031
## 6 1e+02 0.2046538 0.03226920
## 7 1e+03 0.1881967 0.01827205
```

A cost of 0.01 seems to perform best.

```
colrac.svmpoly <- tune(svm, colrac ~ ., data = gsstrain, kernel = "polynomial", ranges = list(cost = c(0

summary(colrac.svmpoly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1504993
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.3705099 0.03346081
## 2 1e-01 0.1796360 0.01747162
## 3 1e+00 0.1504993 0.01802055
## 4 5e+00 0.1582809 0.01566215
## 5 1e+01 0.1609328 0.01508874
```

```
## 6 1e+02 0.1634561 0.01525228
## 7 1e+03 0.1634561 0.01525228
```

A cost of 1 seems to perform best.

```r
colrac.svmradial <- tune(svm, colrac ~ ., data = gsstrain, kernel = "radial", ranges = list(cost = c(0.0

summary(colrac.svmradial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1462846
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.3069969 0.03693685
## 2 1e-01 0.1573178 0.01545161
## 3 1e+00 0.1462846 0.01564342
## 4 5e+00 0.1556657 0.01516395
## 5 1e+01 0.1572079 0.01627300
## 6 1e+02 0.1593979 0.01689155
## 7 1e+03 0.1593979 0.01689155
```

A cost of 1 seems to perform best.