MACS30100 Homework 6

Takuyo Ozaki

3/7/2020

```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
knitr::opts_chunk$set(fig.width = 7, fig.height = 3.5, fig.align = 'center')

# load necessary package
library(tidyverse)
library(broom)
library(rsample)
library(tibble)
library(rcfss)
library(knitr)
library(caret)
library(elasticnet)
library(gridExtra)
library(e1071)
library(kernlab)
library(yardstick)
library(doParallel)
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

# Set the theme as minimal
theme_set(theme_minimal())
```

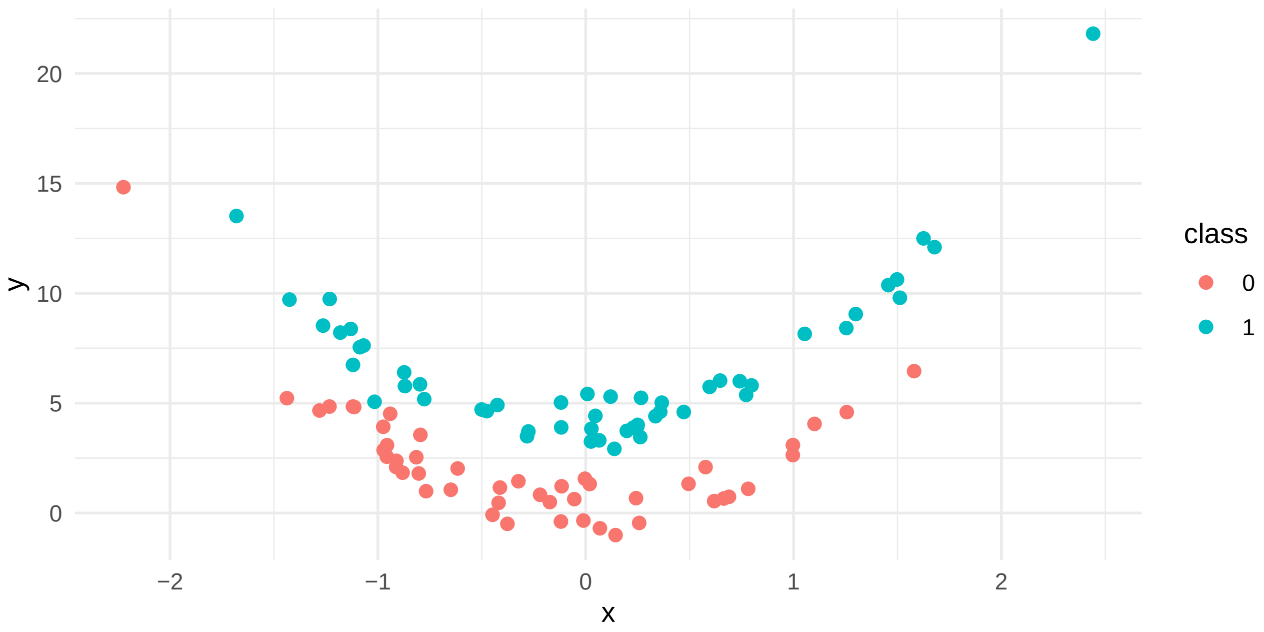
Conceptual exercises

Non-linear separation

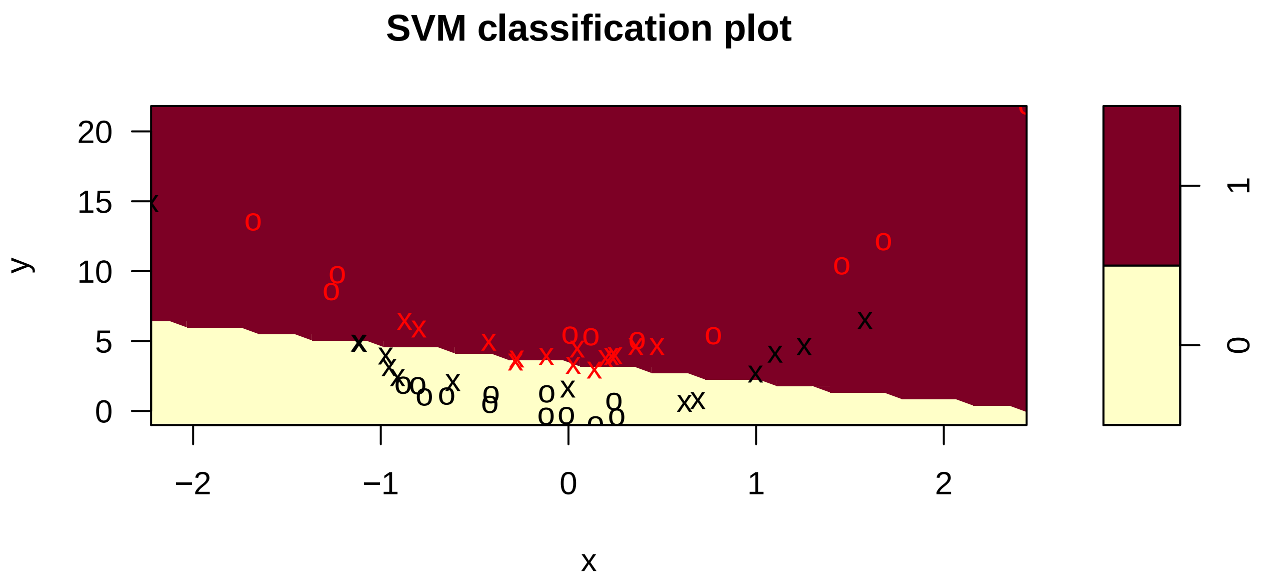
Question 1

```
set.seed(0) # Ensure the reproducibility

# Generate a simulated two-class data set with 100 observations and two features
# Note: we need to see the clear separation between two classes
class <- sample(0:1, size = 100, replace = TRUE)
x <- rnorm(100)
y <- (1 - class) * 3*x^2 +  class * (3*x^2 + 4) + 0.7 * rnorm(100)
data1 <- data.frame(x, y, class)
data1$class <- as.factor(data1$class)
```

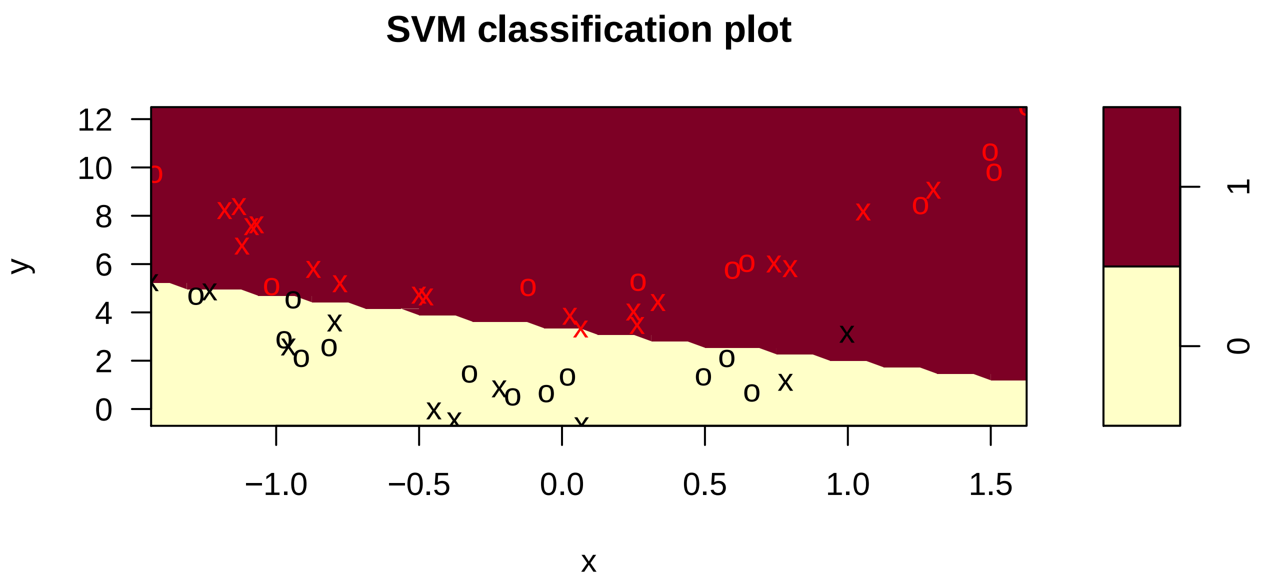
```
# Plot the data to check if there is a visible but non-linear separation
data1 %>%
  ggplot(aes(x = x, y = y, color = class)) +
  geom_point(size = 2)
```



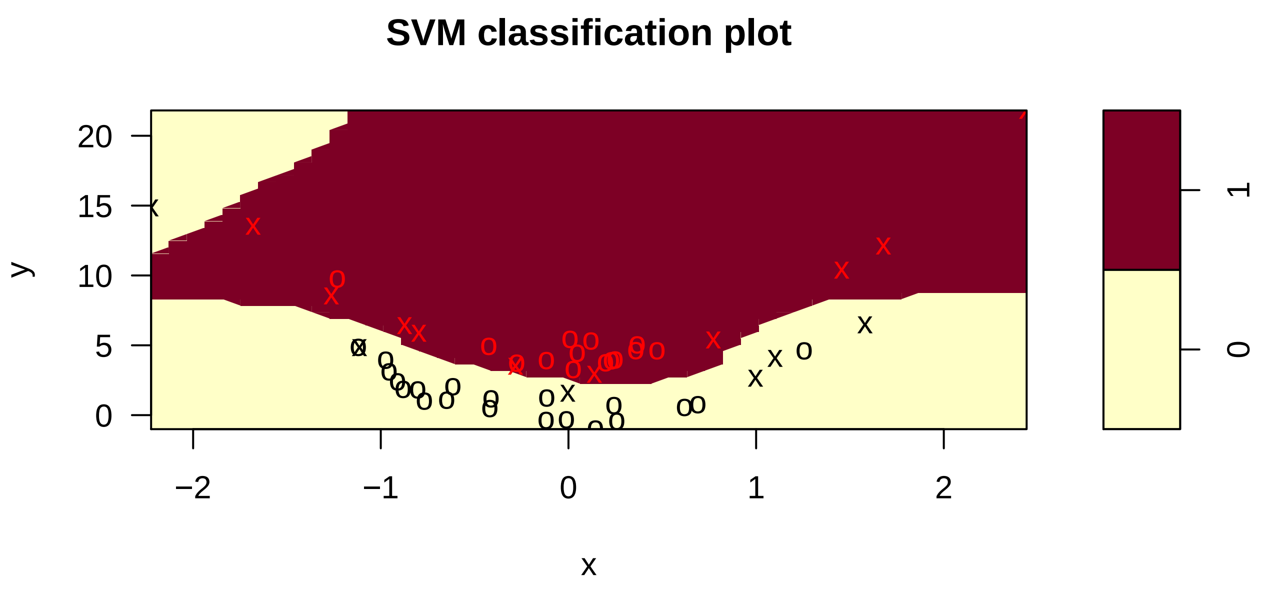
```
# Split the data to train and test
split1 <- initial_split(data1, 0.5)
train1 <- training(split1) #50%
test1 <- testing(split1) #50%

# Fit the train data by svm_linear
svm_linear1 <- svm(class ~ y + x, data = train1, kernel = "linear", cost = 10)

# Plot SVM linear with train data
plot(svm_linear1, train1)
```



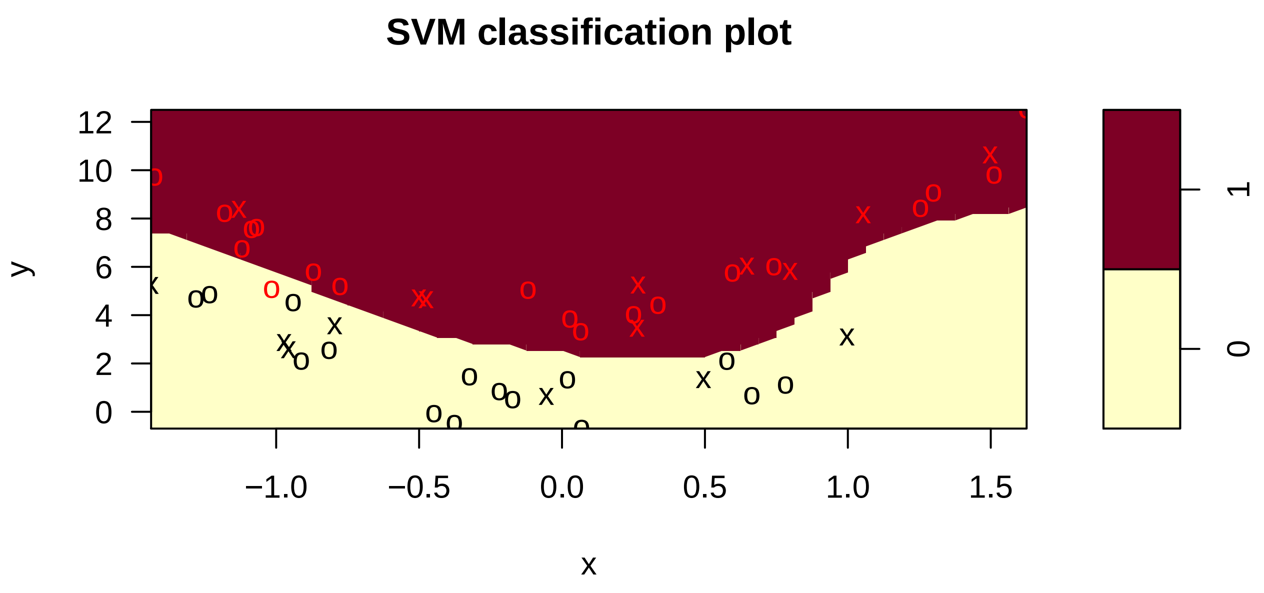
```r
# Plot SVM linear with test data
plot(svm_linear1, test1)
```



```r
# Fit the train data by svm_radial
svm_radial1 <- svm(class ~ y + x, data = train1, kernel = "radial", gamma = 1, cost = 10)

# Plot SVM radial with train data
plot(svm_radial1, train1)
```



```
# Plot SVM radial with test data
plot(svm_radial1, test1)
```



```
# Generate a function to estimate MSE
mse <- function(model, data){
  1 - data %>%
    mutate(.pred = predict(model, data)) %>%
    accuracy(truth = class, estimate = .pred) %>%
    select(.estimate) %>%
    as.numeric()
}

# Calcurate the train/test error rate
kable(data.frame(
  "Error rate" = c("Train error rate", "Test error rate"),
  "SVM linear" = c(mse(svm_linear1, train1),
                   mse(svm_linear1, test1)),
  "SVM radial" = c(mse(svm_radial1, train1),
                   mse(svm_radial1, test1))), digits = 3)
```

Error.rate	SVM.linear	SVM.radial
Train error rate	0.18	0.00
Test error rate	0.06	0.02

When we create the facke data by non-linear separation between the two classes, a support vector machine with a radial kernel will clearly outperform a support vector classifier (a linear kernel) both on the test and training data.

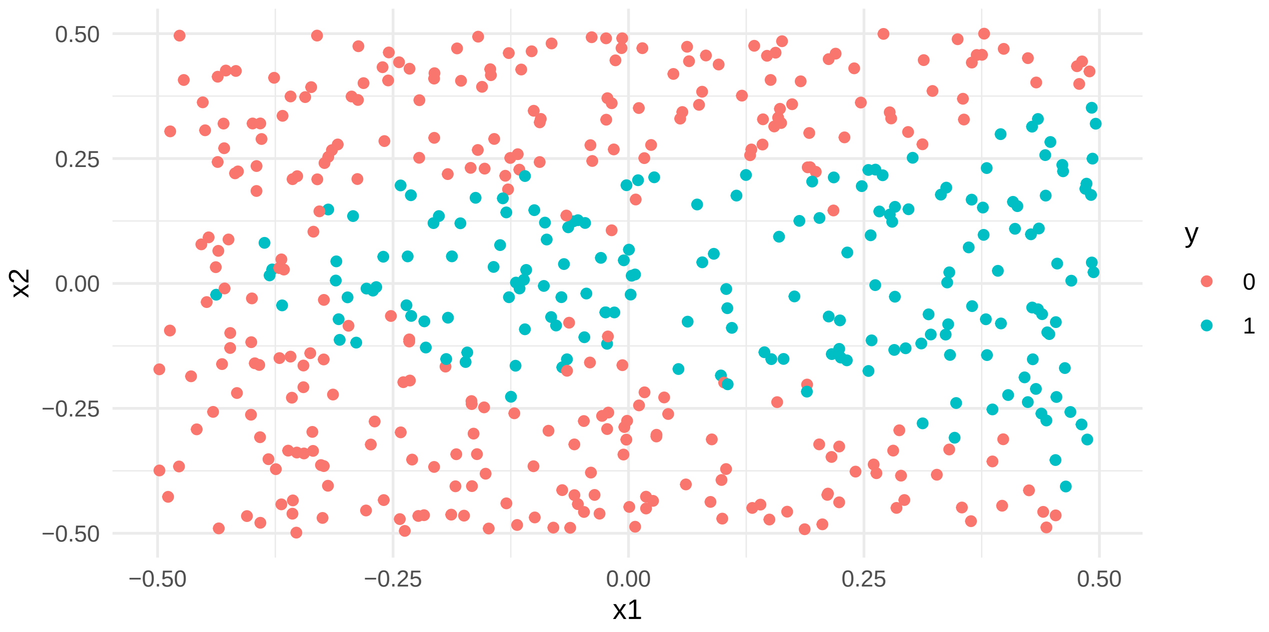
SVM vs. logistic regression

Question 2

```
set.seed(0) # Ensure the reproducibility

# Generate a simulated two-class data set with 500 observations and two features
# Note: we need to see the overlap but non-linear boundary
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- as.factor(as.numeric(x1^3 - x2^2 - 0.02*rnorm(500) + 0.04 > 0))
```

Question 3

```
# Plot the data to check the overlapping but non-linear boundary
data2 <- data.frame(x1, x2, y)
data2 %>%
  ggplot(aes(x = x1, y = x2, color = y))+
  geom_point()
```



This plot has clearly non-linear boundary and some overlapping.

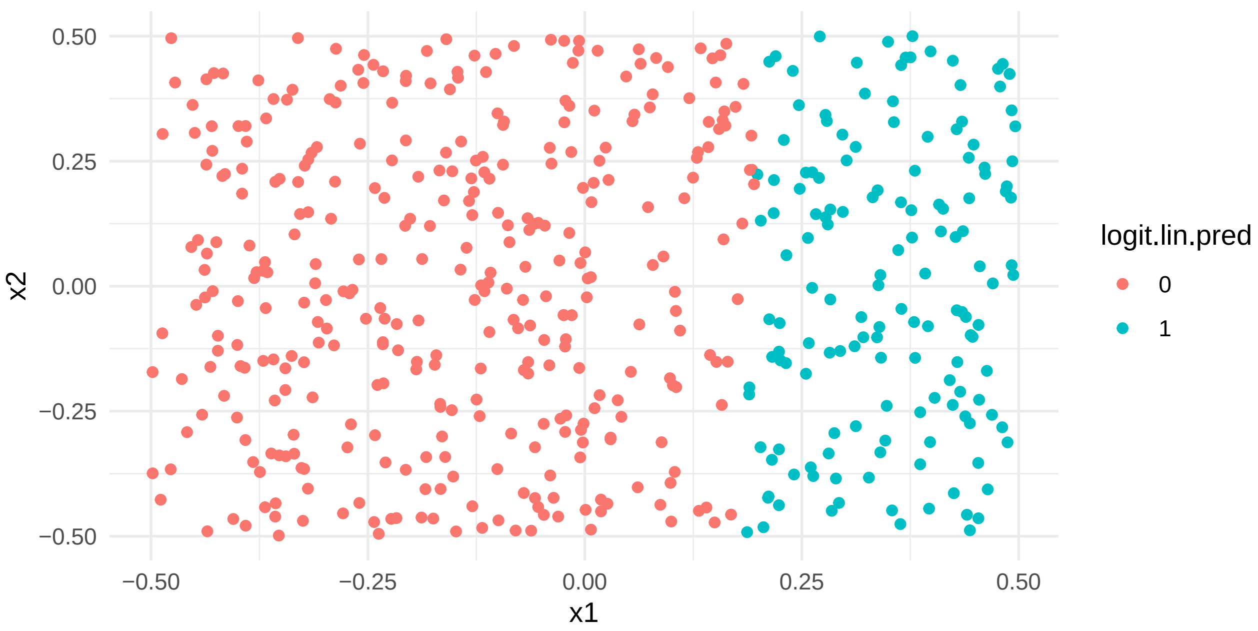
Question 4

```r
# Fit a logistic regression model to the data
logit.linear <- glm(y ~ x1 + x2, data = data2, family = binomial("logit"))
summary(logit.linear)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial("logit"), data = data2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5647  -0.9169  -0.5952   1.0092   2.0232
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.58179    0.10153  -5.730 1.00e-08 ***
## x1           3.03344    0.37755   8.035 9.39e-16 ***
## x2          -0.06362    0.33832  -0.188    0.851
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 662.07  on 499  degrees of freedom
## Residual deviance: 587.22  on 497  degrees of freedom
## AIC: 593.22
##
## Number of Fisher Scoring iterations: 4
```

Question 5

```r
# Obtain a predicted class label for each observation based on the linear logistic model
data2 <- data2 %>%
  mutate(logit.lin.prob = predict(logit.linear, type = "response"),
         logit.lin.pred = as.factor(ifelse(logit.lin.prob > 0.5, 1, 0)))

# Plot the observations, colored according to the predicted class labels
data2 %>%
  ggplot(aes(x = x1, y = x2, color = logit.lin.pred)) +
  geom_point()
```



The prediction plot has linear boundary, which is different from the true boundary.

Question 6

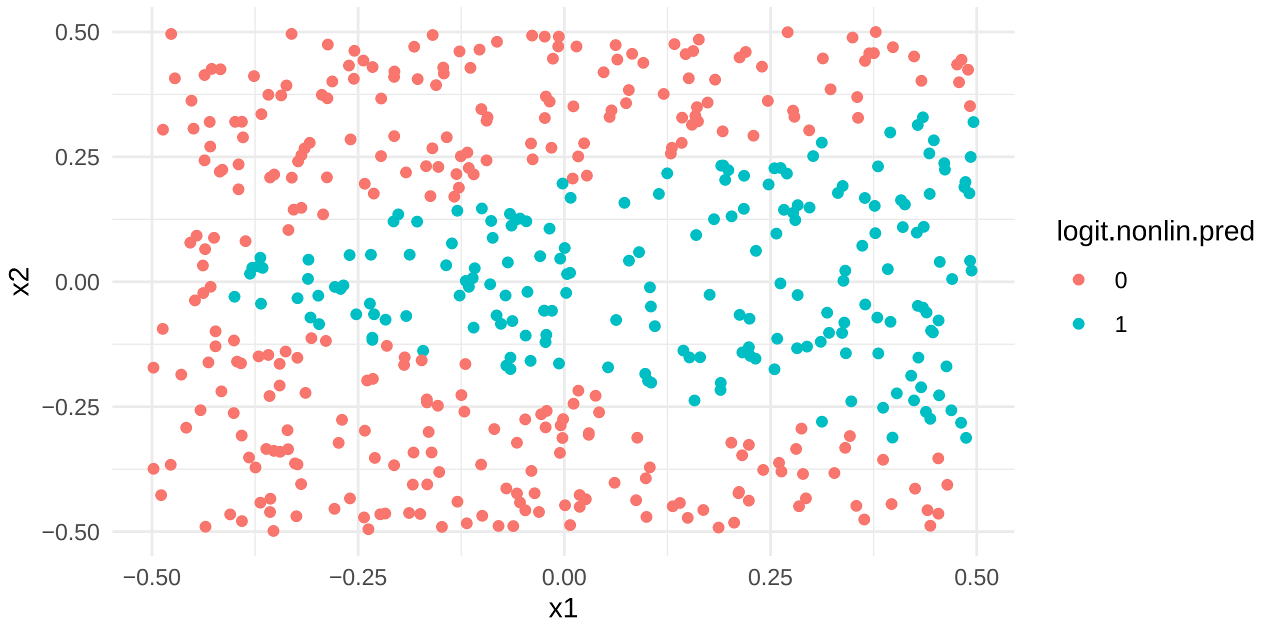
```r
# Fit a logistic regression model using polynomial function
logit.nonlinear <- glm(y ~ poly(x1, 2) + poly(x2, 2),
                       data = data2, family = binomial("logit"))
summary(logit.nonlinear)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2), family = binomial("logit"),
##     data = data2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.92323  -0.13060  -0.00193   0.14663   2.93386
```

```
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.3729     0.4641  -7.268 3.65e-13 ***
## poly(x1, 2)1   68.2349     8.5473   7.983 1.43e-15 ***
## poly(x1, 2)2   10.4130     4.6702   2.230   0.0258 *
## poly(x2, 2)1    4.3390     6.5448   0.663   0.5074
## poly(x2, 2)2 -143.7926    17.1246  -8.397  < 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: 662.07  on 499  degrees of freedom
## Residual deviance: 197.56  on 495  degrees of freedom
## AIC: 207.56
##
## Number of Fisher Scoring iterations: 8
```

Question 7

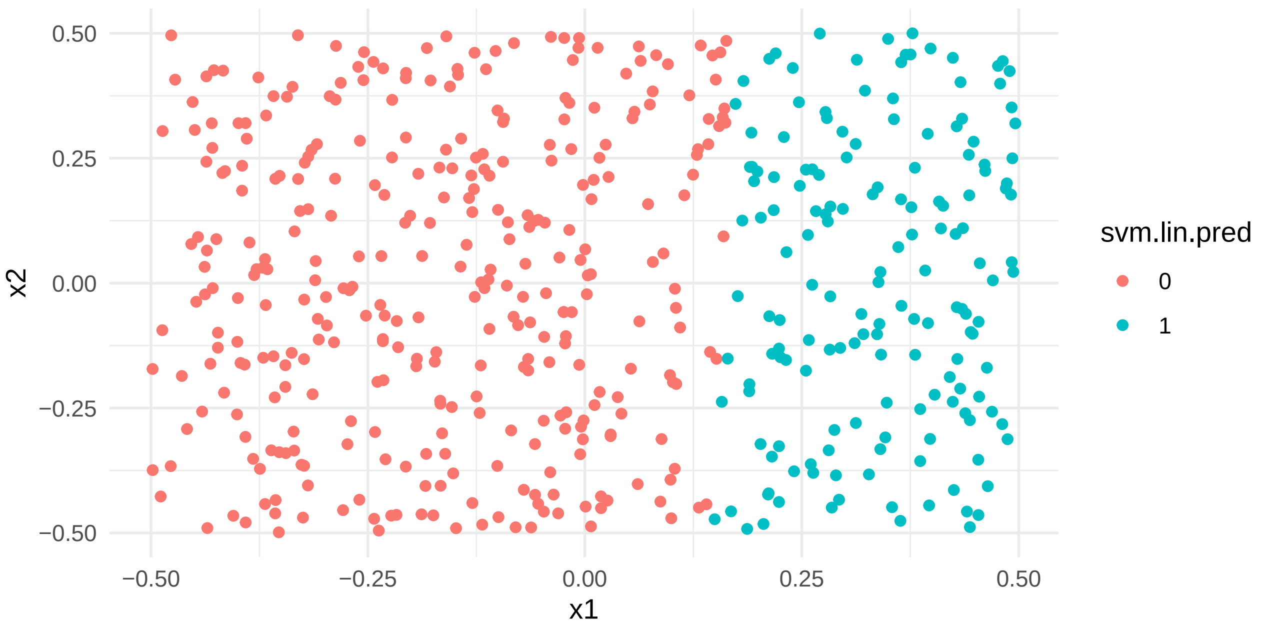
```
# Obtain a predicted class label for each observation based on the nonlinear logistic model
data2 <- data2 %>%
  mutate(logit.nonlin.prob = predict(logit.nonlinear, type = "response"),
         logit.nonlin.pred = as.factor(ifelse(logit.nonlin.prob > 0.5, 1, 0)))

# Plot the observations, colored according to the predicted class labels
data2 %>%
  ggplot(aes(x = x1, y = x2, color = logit.nonlin.pred)) +
  geom_point()
```



The prediction plot is similar to the true plot without overlapping parts.

Question 8

```r
set.seed(0) # Ensure the reproducibility

# Fit a support vector classifier (linear kernel)
svm.linear <- svm(y ~ x1 + x2, data = data2, kernel = "linear", cost = 0.1)

# Obtain a predicted class label for each observation based on the linear SVm
data2 <- data2 %>%
  mutate(svm.lin.pred = predict(svm.linear, type = "response"))

# Plot the observations, colored according to the predicted class labels
data2 %>%
  ggplot(aes(x = x1, y = x2, color = svm.lin.pred)) +
  geom_point()
```

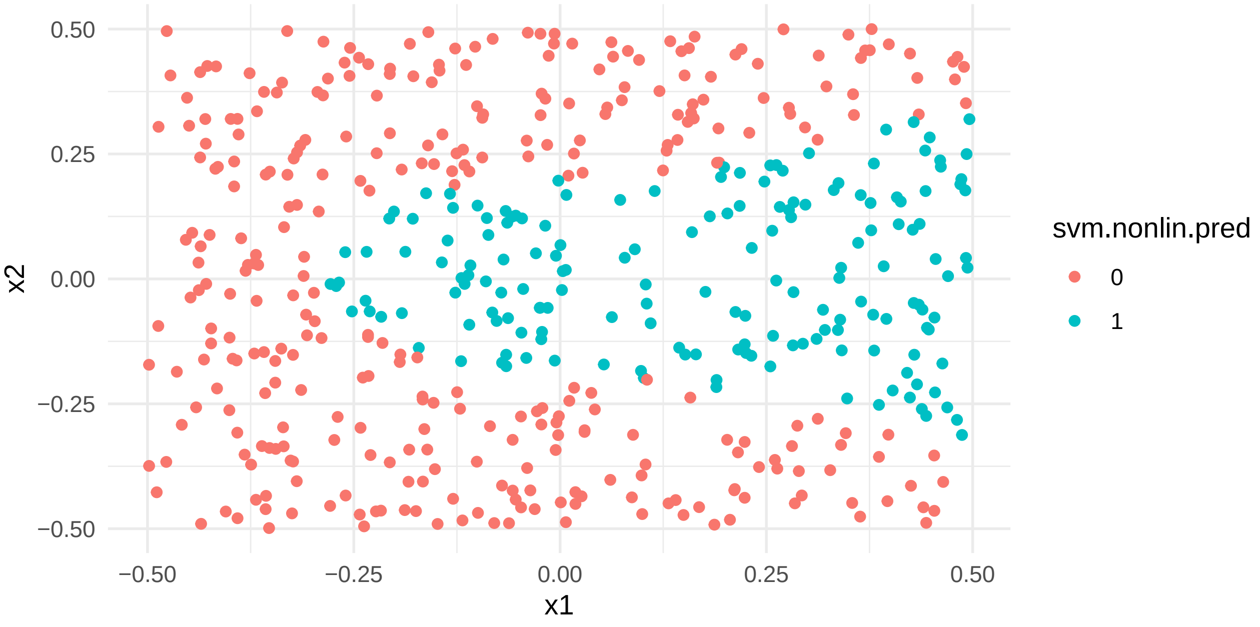


The prediction plot has linear boundary, which is similar to the prediction plot based on the linear logit model.

Question 9

```r
set.seed(0) # Ensure the reproducibility

# Fit a support vector classifier (non linear kernel)
svm.nonlinear <- svm(y ~ x1 + x2, data = data2, kernel = "radial", gamma = 1, cost = 0.1)
```

```r
# Obtain a predicted class label for each observation based on the linear SVm
data2 <- data2 %>%
  mutate(svm.nonlin.pred = predict(svm.nonlinear, type = "response"))

# Plot the observations, colored according to the predicted class labels
data2 %>%
  ggplot(aes(x = x1, y = x2, color = svm.nonlin.pred)) +
  geom_point()
```



The prediction plot has non-linear boundary, which is similar to the prediction plot based on the non-linear logit model.

Question 10

As we see the answer below, both the non-linear logit model and the non-linear SVM (radial kernel) could make a precise non-linear decision boundary while linear logit and linear SVM could not make it. The tradeoffs between estimating non-linear decision boundaries using these two different approaches are:

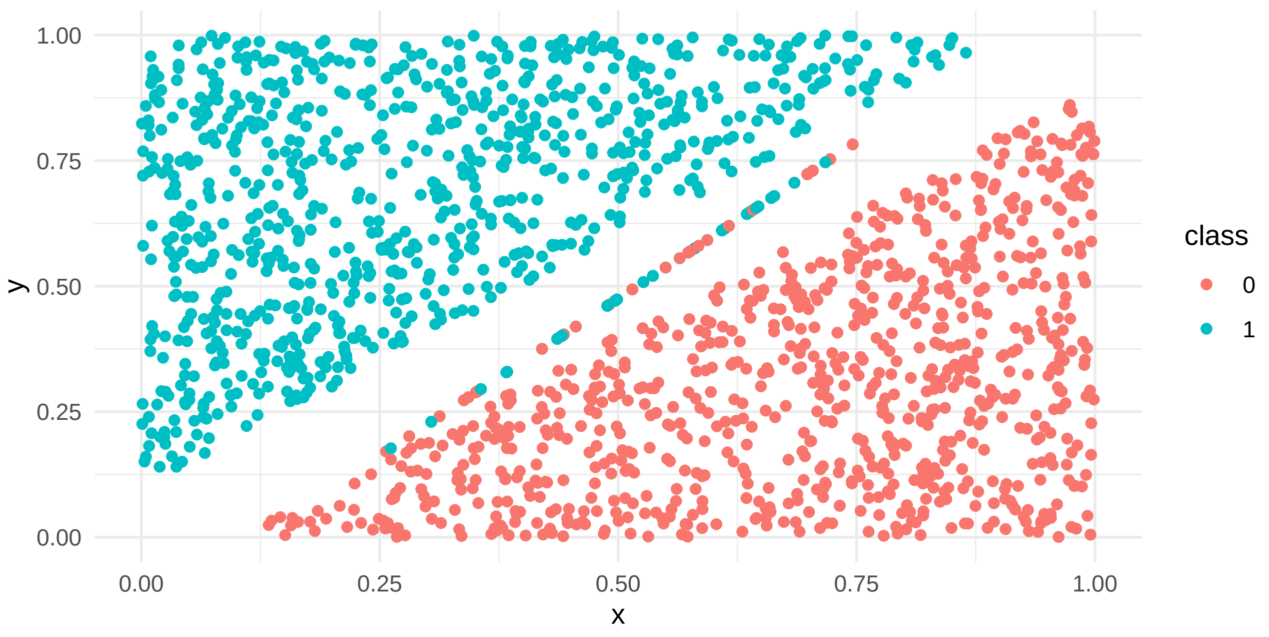
- As for logit model, we need to mannualy specify the function term (e.g., polinominal, interaction, log). There is a lot of effort for it. On the other hand, as for SVM, we can use hyperparameters (e.g., gammma) and easily check the fit by cross varidation.
- As for SVM, it is difficult to interpret the mechanism (relationship) between the feature and the outcome. On the other hand, as for the logit model, once we get appropriate function, we can interpret their relationship although there is some difficulty to interpret the log odds.

Tuning cost

Question 11

Although the meaning of "barely linearly separable" is vague, I generate the data: true boundary y = x, but there is a noise boundary y = - 1.25*x - 0.15, which includes random class. The plot of the data is as follows.

```r
set.seed(0) # Ensure the reproducibility

# Generate a boundary data y = x
x <- runif(50, min = 0.25, max = 0.75)
y <- 1.25*x - 0.15
class <- as.factor(sample(0:1, size = 100, replace = TRUE))
boundary_data <- data.frame(x, y, class)

# Generate a normal data but exclude the near boundary data
x <- runif(2000)
y <- runif(2000)
class <- as.factor(as.numeric(y  > x))
normal_data <- data.frame(x, y, class) %>%
  filter(abs(y - x) > 0.1)

# Marge the boundary and normal data
train_data <- rbind(boundary_data, normal_data)

# Plot the data to check that the classes are just barely linearly separable
train_data %>%
  ggplot(aes(x = x, y = y, color = class)) +
  geom_point()
```



Question 12

```r
set.seed(0) # Ensure the reproducibility

svm_linear3 <- train(
  class ~ x + y,
  data = train_data,
  method = "svmLinear",
  trControl =  trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10, 100, 1000))
)

svm_linear3
```

```
## Support Vector Machines with Linear Kernel
##
## 1732 samples
##    2 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1559, 1559, 1559, 1559, 1559, 1559, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-02  0.9682446  0.9365009
##   1e-01  0.9705468  0.9410975
##   1e+00  0.9705468  0.9410972
##   1e+01  0.9699655  0.9399236
##   1e+02  0.9740017  0.9479899
##   1e+03  0.9745798  0.9491456
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1000.
```

```r
# Show the error rate by each cost
kable(data.frame(svm_linear3$results) %>%
  mutate(error.rate = 1 - Accuracy) %>%
  select(C, error.rate))
```

C	error.rate
1e-02	0.0317554
1e-01	0.0294532
1e+00	0.0294532
1e+01	0.0300345
1e+02	0.0259983
1e+03	0.0254202

The largest accuracy = least error rate model is the model with cost 1000. The errors rate are 2.5% of the samples (about 50 data). This is exactly the same as the cross-validation errors.

Question 13

```r
set.seed(9) # Ensure the reproducibility

x <- runif(2000)
y <- runif(2000)
class <- as.factor(as.numeric(y  > x))
test_data <- data.frame(x, y, class)

svm_linear4 <- train(
  class ~ x + y,
  data = test_data,
  method = "svmLinear",
  trControl =  trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(C = c(0.01, 0.1, 1, 10, 100, 1000))
)
svm_linear4
```

```
## Support Vector Machines with Linear Kernel
##
## 2000 samples
##    2 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1801, 1800, 1799, 1800, 1800, 1800, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-02  0.9969925  0.9939842
##   1e-01  0.9985000  0.9969982
##   1e+00  0.9960000  0.9919980
##   1e+01  0.9995000  0.9989994
##   1e+02  0.9990000  0.9979988
##   1e+03  0.9990000  0.9979988
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 10.
```

```r
# Show the error rate
kable(data.frame(svm_linear4$results) %>%
  mutate(error.rate = 1 - Accuracy) %>%
  select(C, error.rate))
```

C	error.rate
1e-02	0.0030075
1e-01	0.0015000

C	error.rate
1e+00	0.0040000
1e+01	0.0005000
1e+02	0.0010000
1e+03	0.0010000

The model with cost = 10 has the fewest test errors while the model with cost = 1000 has the fewest the fewest training errors and the fewest cross-validation errors. Compared with those result, the appropriate cost is smaller (10 < 1000).

Question 14

There could be overfitting for SVM linear kernel with a large cost. This shows that a support vector classifier with a small value of cost that misclassifies a noise part of the training data may perform better on test data than one with a huge value of cost that does not misclassify the noise part.

Application: Predicting attitudes towards racist college professors

Question 15

```r
set.seed(0) # Ensure the reproducibility

# Import the data
gss_test <- read_csv("data/gss_test.csv")
gss_train <- read_csv("data/gss_train.csv")

# Report the CV errors associated with different values of cost
tune.lin = tune(svm, colrac ~ ., data = gss_train, kernel = "linear",
                ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.lin)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.1818715
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.1818715 0.02003760
## 2 1e-01 0.1999773 0.02263824
## 3 1e+00 0.2017925 0.02243359
## 4 5e+00 0.2019573 0.02244402
## 5 1e+01 0.2019910 0.02251040
## 6 1e+02 0.2043857 0.02234224
```

With a support vector classifier (linear kernel), the cross-validation error is minimized at 0.1818715 for cost = 0.01. We can see that as the cost increase, the error rate is increasing.

Question 16

Polinomial

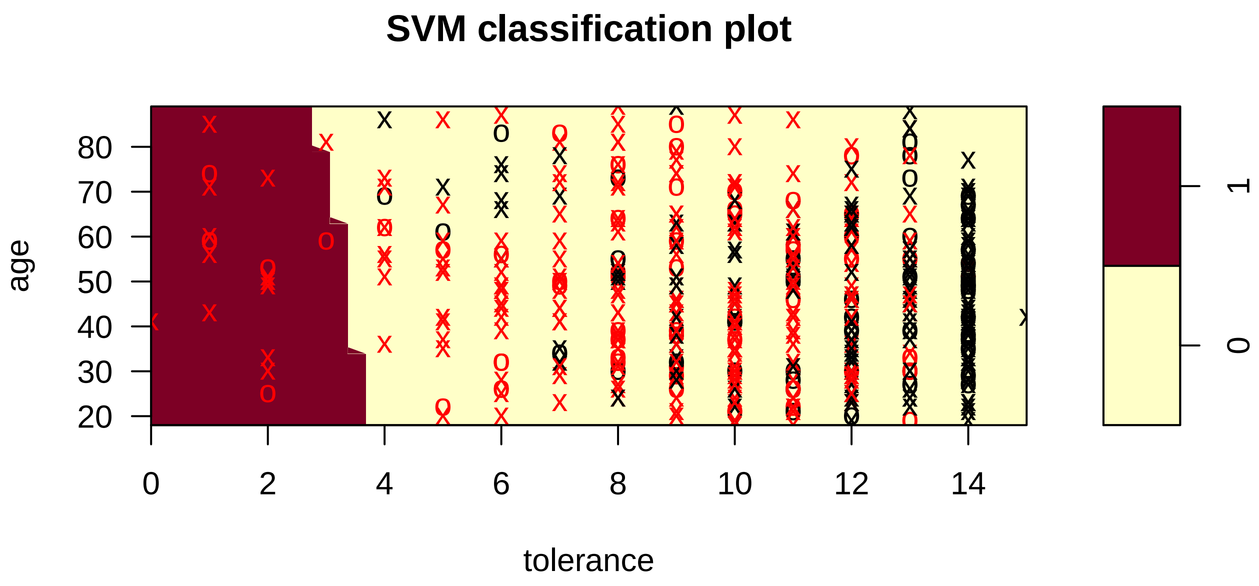
```r
set.seed(0) # Ensure the reproducibility

# Report the CV errors associated with different values of cost
tune.poly = tune(svm, colrac ~ ., data = gss_train, kernel = "polynomial",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10),
                               degree = c(2, 3, 4)))
summary(tune.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##     1      3
##
## - best performance: 0.1510676
##
## - Detailed performance results:
##     cost degree     error  dispersion
## 1   0.01      2 0.4070809 0.046167602
## 2   0.10      2 0.2658775 0.034926446
## 3   1.00      2 0.2173000 0.022199819
## 4   5.00      2 0.2833760 0.037647733
## 5  10.00      2 0.3357071 0.051379134
## 6   0.01      3 0.3706636 0.046921131
## 7   0.10      3 0.1809957 0.014495617
## 8   1.00      3 0.1510676 0.015506901
## 9   5.00      3 0.1592888 0.018099297
## 10 10.00      3 0.1617000 0.019723341
## 11  0.01      4 0.4157153 0.047268665
## 12  0.10      4 0.3118844 0.044970885
## 13  1.00      4 0.1880575 0.009502657
## 14  5.00      4 0.1863574 0.008988712
## 15 10.00      4 0.1880815 0.010098308
```

```r
# Calcurate the test error rate by the best model of polynominal SVM
1 - gss_test %>%
  mutate(.pred = predict(tune.poly$best.model, gss_test) > 0.5) %>%
  accuracy(truth = as.factor(colrac),
            estimate = as.factor(as.numeric(.pred))) %>%
  select(.estimate) %>%
  as.numeric()
```

```
## [1] 0.2434077
```

```
# Check the fit with age and tolerance (just for fun)
best.poly <- svm(as.factor(colrac) ~ ., data = gss_train,
                 kernel = "polynomial", cost = 1, degree = 3)
plot(best.poly, gss_test, age ~ tolerance)
```



Radial

```
set.seed(0) # Ensure the reproducibility

tune.radial = tune(svm, colrac ~ ., data = gss_train, kernel = "radial",
                   ranges = list(cost = c(0.01, 0.1, 1, 5, 10),
                                 gamma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.radial)
```

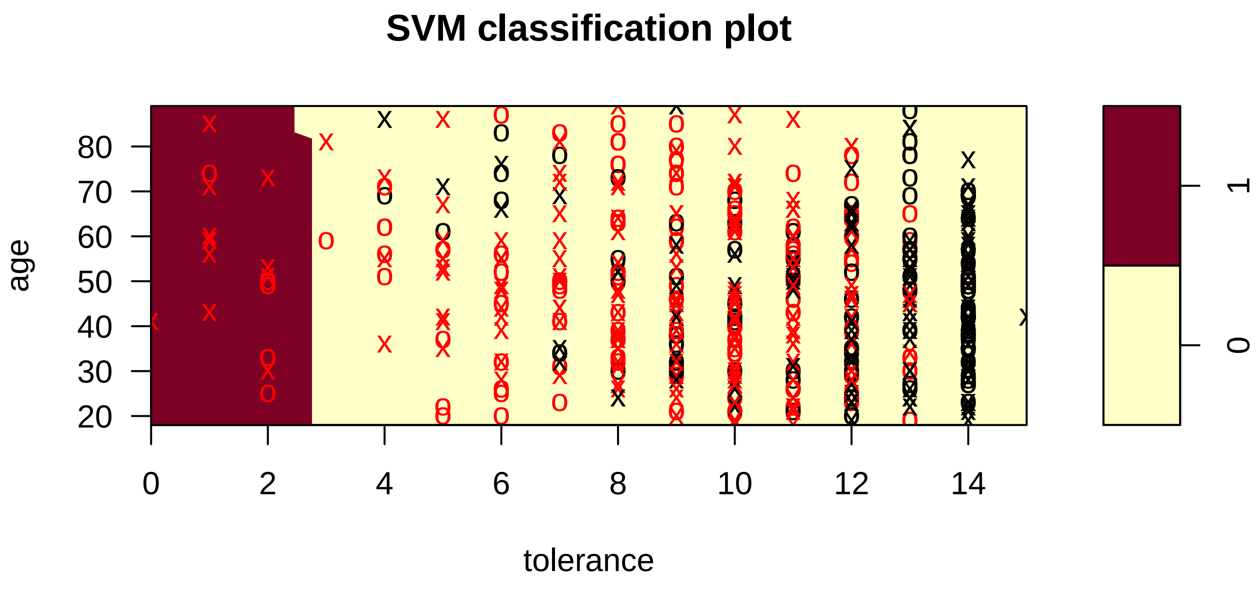
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1  0.01
##
## - best performance: 0.148392
##
```

```
## - Detailed performance results:
##     cost gamma     error  dispersion
## 1   0.01 1e-02 0.2815197 0.035785681
## 2   0.10 1e-02 0.1551127 0.012619291
## 3   1.00 1e-02 0.1483920 0.013795334
## 4   5.00 1e-02 0.1561731 0.016391815
## 5  10.00 1e-02 0.1651579 0.017168099
## 6   0.01 1e-01 0.4245548 0.046457647
## 7   0.10 1e-01 0.3807699 0.042441512
## 8   1.00 1e-01 0.2166007 0.005621240
## 9   5.00 1e-01 0.2172402 0.005649587
## 10 10.00 1e-01 0.2172402 0.005649587
## 11  0.01 1e+00 0.4258971 0.046536059
## 12  0.10 1e+00 0.3934960 0.043045762
## 13  1.00 1e+00 0.2498748 0.002526544
## 14  5.00 1e+00 0.2498748 0.002526544
## 15 10.00 1e+00 0.2498748 0.002526544
## 16  0.01 5e+00 0.4258971 0.046536059
## 17  0.10 5e+00 0.3934963 0.043045772
## 18  1.00 5e+00 0.2498765 0.002527117
## 19  5.00 5e+00 0.2498765 0.002527117
## 20 10.00 5e+00 0.2498765 0.002527117
## 21  0.01 1e+01 0.4258971 0.046536059
## 22  0.10 1e+01 0.3934963 0.043045772
## 23  1.00 1e+01 0.2498765 0.002527117
## 24  5.00 1e+01 0.2498765 0.002527117
## 25 10.00 1e+01 0.2498765 0.002527117
## 26  0.01 1e+02 0.4258971 0.046536059
## 27  0.10 1e+02 0.3934963 0.043045772
## 28  1.00 1e+02 0.2498765 0.002527117
## 29  5.00 1e+02 0.2498765 0.002527117
## 30 10.00 1e+02 0.2498765 0.002527117
```

```
# Calcurate the test error rate by the best model of polynominal SVM
1 - gss_test %>%
  mutate(.pred = predict(tune.radial$best.model, gss_test) > 0.5) %>%
  accuracy(truth = as.factor(colrac),
           estimate = as.factor(as.numeric(.pred))) %>%
  select(.estimate) %>%
  as.numeric()
```

```
## [1] 0.2190669
```

```
# Check the fit with age and tolerance (just for fun)
best.radial <- svm(as.factor(colrac) ~ ., data = gss_train,
                 kernel = "radial", cost = 1, gamma = 0.01)
plot(best.radial, gss_test, age ~ tolerance)
```



- Polynominal: When the cost is 1 and degree = 3, the polynominal model has a best (train) performance: the 10 fold CV train error rate 0.1510676. The train error rate clearly decreases as the degree increases. The test error rate by the best model is 0.2434077.
- Radial: When the cost is 1 and gamma = 0.01, the polynominal model has a best (train) performance: the 10 fold CV train error rate 0.148392. The train error rate clearly decreases as the gamma decreases. The test error rate by the model is 0.2190669.
- Compared with the two models, the radial kernels model is a slightly better model because of the smaller training and test error rate. Just for fun, I plotted the two variables (age and tolerance) in the test data to check the fit, but they do not show a good fit, partly because only the two variables are not good predictors to the colrac.