Problem Set 6

Akira Masuda

2020/3/8

```
Course: MACS30100 Perspectives on Computational Modeling (Winter 2020)
Author: Akira Masuda (ID: alakira)
```

```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
knitr::opts_chunk$set(fig.width=6,fig.height=3.4,fig.align='center')
```

```
library(tidyverse)
library(tidymodels)
library(knitr)
library(patchwork)
library(rcfss)
library(e1071)
library(caret)
library(pROC)
library(kernlab)
rm(list=ls())
set.seed(1100)
```

```
# set up parallelization for quicker computation time
library(doParallel)
cl <- makePSOCKcluster(3)
registerDoParallel(cl)

# set seed
set.seed(110)
```

Non-linear separation

1.

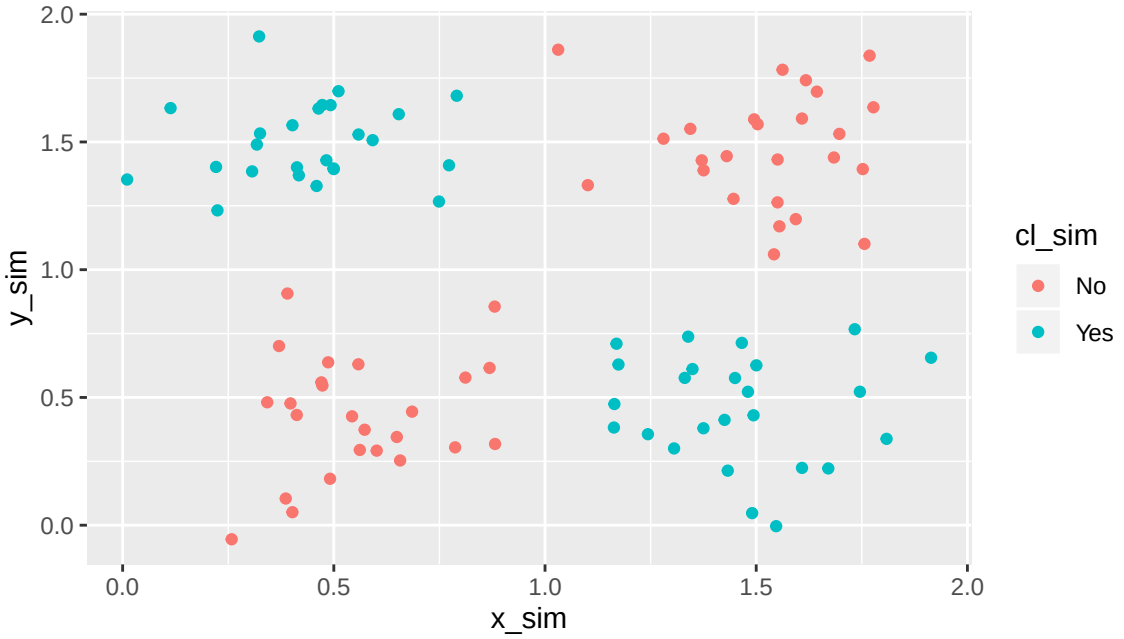
I generate a simulated dataset that is based on XOR space which has clear but non-linear separation.

```
# Generate dataset
x_sim = c()
y_sim = c()
cl_sim = c()
for (i in 1:100) {
    x_sim[i] = case_when(i <= 50 ~ rnorm(1, 0.5, 0.2),
                         i > 50 ~ rnorm(1, 1.5, 0.2))
    y_sim[i] = case_when(i <= 25 ~ rnorm(1, 0.5, 0.2),
                         i > 25 & i <= 50 ~ rnorm(1, 1.5, 0.2),
                         i > 50 & i <= 75 ~ rnorm(1, 0.5 , 0.2),
                         i > 75 ~ rnorm(1, 1.5, 0.2))
```

```
    cl_sim[i] = case_when(i <= 25 ~ '0',
                          i > 25 & i <= 50~ '1',
                          i > 50 & i <= 75 ~ '1',
                          i > 75 ~ '0',)
}
df_sim <- data.frame(x_sim, y_sim, cl_sim)

df_sim <- df_sim %>%
  mutate(cl_sim = factor(cl_sim, levels = c(0, 1), labels = c("No", "Yes")))

# Plot original dataset
df_sim %>%
  ggplot(aes(x=x_sim, y=y_sim, color=cl_sim)) +
  geom_point()
```



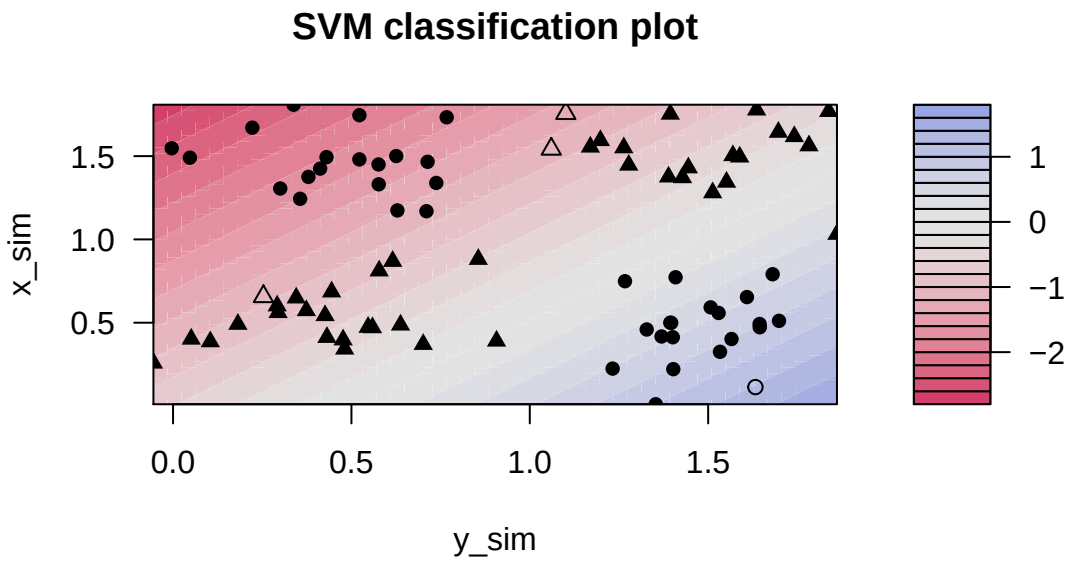

```
# Split dataset; 80% to training and 20% to testing
split_set = initial_split(df_sim, 0.8)
train_set = training(split_set)
train_x = train_set %>%
  select(-cl_sim)
test_set = testing(split_set)
test_x = test_set %>%
  select(-cl_sim)

# Fit linear svm model
(svm_linear <- ksvm(
  cl_sim ~ .,
  data = train_set,
  type = "C-svc",
  kernel="vanilladot"
))
```

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

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 77
##
## Objective Function Value : -74.9919
## Training error : 0.259259
```

```
# Classification plot
plot(svm_linear, data=train_x)
```

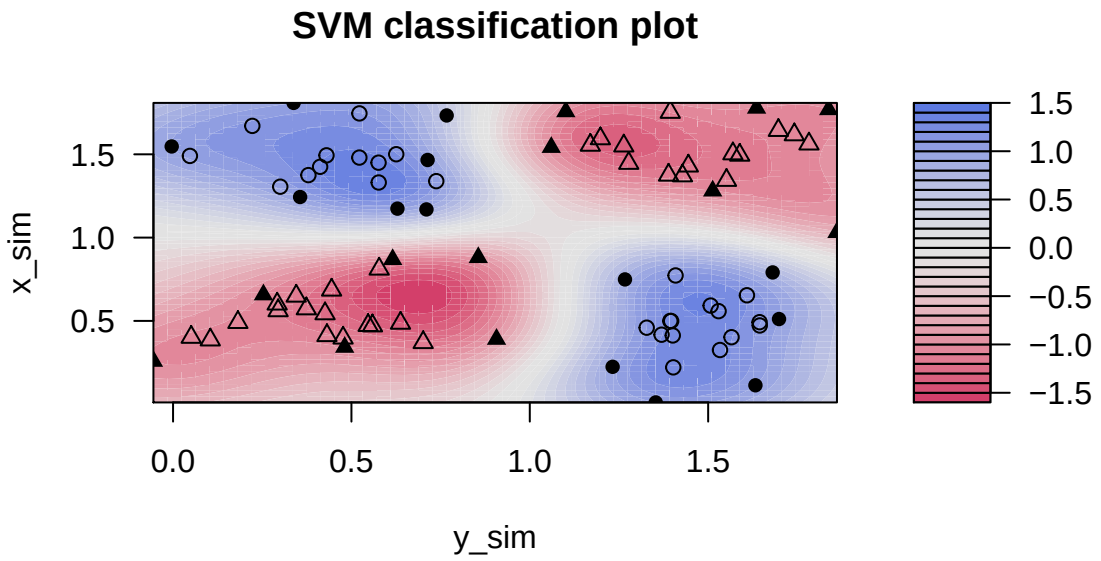



```
# Fit radial svm model
(svm_radial <- ksvm(
  cl_sim ~ .,
  data = train_set,
  type = "C-svc",
  kernel = "rbfdot",
))
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  2.09341034954089
##
## Number of Support Vectors : 25
##
## Objective Function Value : -9.3575
```

```
## Training error : 0
```

```
# Classification plot
plot(svm_radial, data=train_x)
```




```
# Showing the results
data.frame(
  "Error rate" = c("Train set", "Test set"),
  "SVM linear" = c(sum(predict(svm_linear, train_set) == train_set[,'cl_sim']) / nrow(train_set),
                   sum(predict(svm_linear, test_set) == test_set[,'cl_sim']) / nrow(test_set)),
  "SVM radial" = c(sum(predict(svm_radial, train_set) == train_set[,'cl_sim']) / nrow(train_set),
                   sum(predict(svm_radial, test_set) == test_set[,'cl_sim']) / nrow(test_set))
)
```

Error.rate	SVM.linear	SVM.radial
Train set	0.7407407	1
Test set	0.6842105	1

The above results show that radial svm model outperforms linear svm model both training and test set. The radial svm model classifies the observation perfectly in this case.

SVM vs. logistic regression

2. Data Generation

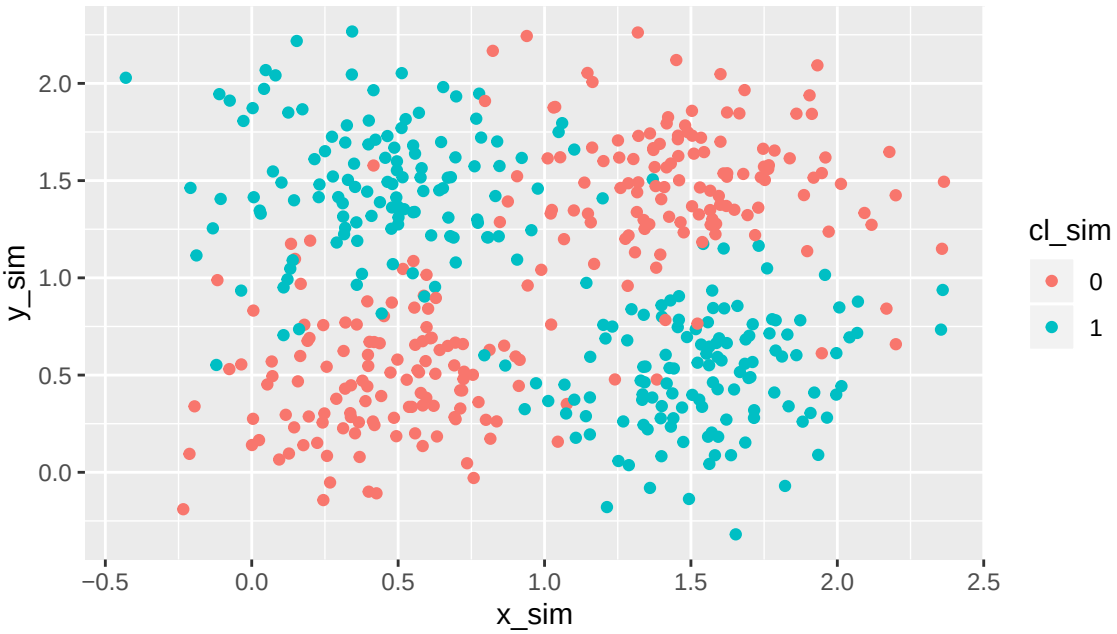
```
# Generate a dataset with some overlapping, non-linear boundary
x_sim = c()
y_sim = c()
cl_sim = c()
for (i in 1:500) {
  # i data generation
    x_sim[i] = case_when(i <= 250 ~ rnorm(1, 0.5, 0.3),
```

```
                          i > 250 ~ rnorm(1, 1.5, 0.3))
    y_sim[i] = case_when(i <= 125 ~ rnorm(1, 0.5, 0.3),
                          i > 125 & i <= 250 ~ rnorm(1, 1.5, 0.3),
                          i > 250 & i <= 375 ~ rnorm(1, 0.5 , 0.3),
                          i > 375 ~ rnorm(1, 1.5, 0.3))
    cl_sim[i] = case_when(i <= 125 ~ '0',
                           i > 125 & i <= 250~ '1',
                           i > 250 & i <= 375 ~ '1',
                           i > 375 ~ '0',)
}
df_sim2 <- data.frame(x_sim, y_sim, cl_sim)

df_sim2 <- df_sim2 %>%
  mutate(cl_sim = factor(cl_sim, levels = c(0, 1), labels = c(0, 1)))
```

3. Plot

```
# Plot data
df_sim2 %>%
  ggplot(aes(x=x_sim, y=y_sim, color=cl_sim)) +
  geom_point()
```



4. Logistic regression model (linear)

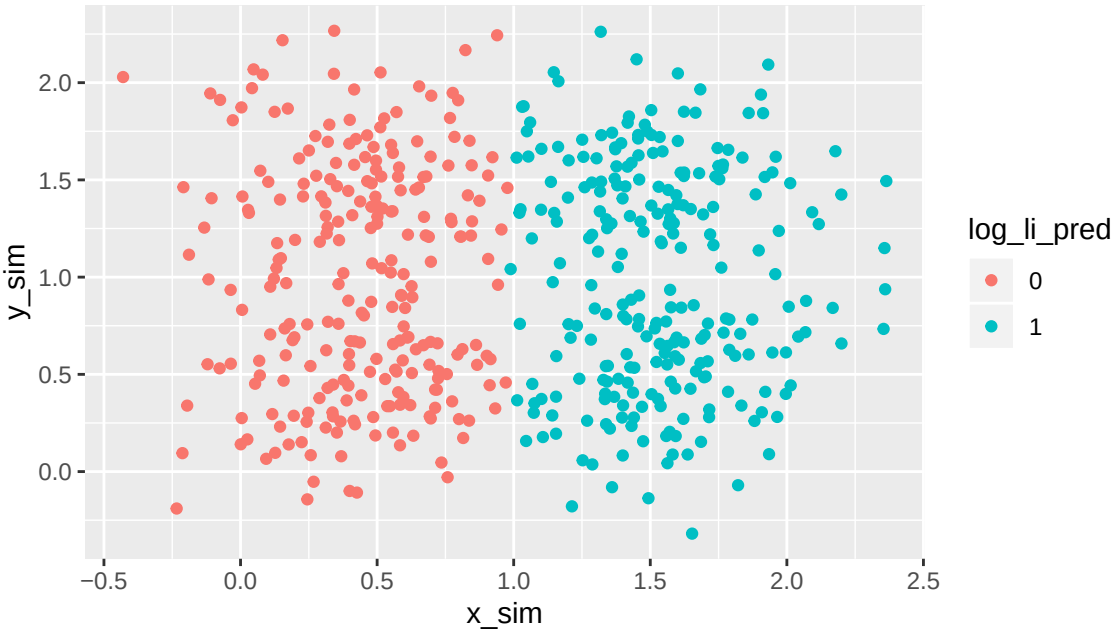
```
# Fit a logistic regression model to the data
log_li_model <- glm(cl_sim ~ x_sim + y_sim, data=df_sim2, family=binomial('logit'))
summary(log_li_model)
```

```
##
## Call:
## glm(formula = cl_sim ~ x_sim + y_sim, family = binomial("logit"),
##     data = df_sim2)
```

```
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.19608  -1.17683  -0.00107   1.17665   1.19651
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.121e-02  2.262e-01  -0.138    0.890
## x_sim        3.174e-02  1.447e-01   0.219    0.826
## y_sim        5.161e-06  1.532e-01   0.000    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 693.15  on 499  degrees of freedom
## Residual deviance: 693.10  on 497  degrees of freedom
## AIC: 699.1
##
## Number of Fisher Scoring iterations: 3
```

5. Plot

```
# Plot the predicted class labels
df_sim2 <- df_sim2 %>%
  mutate(log_li_prob = predict(log_li_model, type='response'),
         log_li_pred = as.factor(ifelse(log_li_prob >= 0.5, 1, 0)))
df_sim2 %>%
  ggplot(aes(x=x_sim, y=y_sim, color=log_li_pred)) +
  geom_point()
```



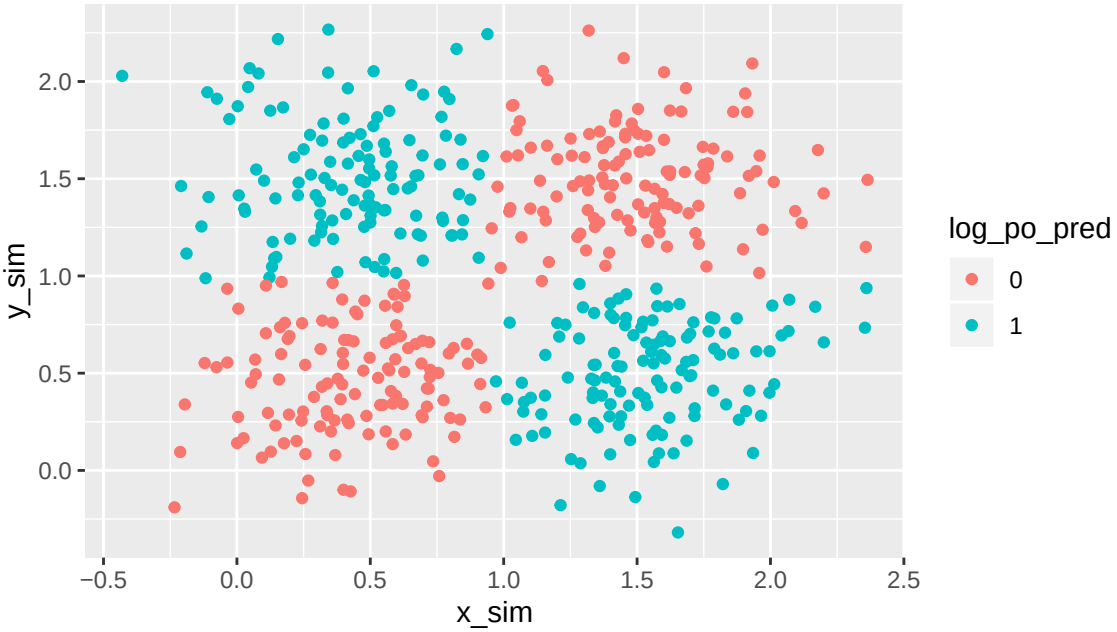
6. Logistic regression model (non-linear)

```r
log_po_model <- glm(cl_sim ~ x_sim*y_sim + x_sim + y_sim, data=df_sim2, family=binomial('logit'))
summary(log_po_model)
```

```
##
## Call:
## glm(formula = cl_sim ~ x_sim * y_sim + x_sim + y_sim, family = binomial("logit"),
##     data = df_sim2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1581  -0.3462   0.0000   0.3612   3.4301
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -11.992      1.220   -9.83   <2e-16 ***
## x_sim         12.430      1.195   10.40   <2e-16 ***
## y_sim         12.252      1.205   10.17   <2e-16 ***
## x_sim:y_sim  -12.728      1.194  -10.66   <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: 693.15  on 499  degrees of freedom
## Residual deviance: 272.72  on 496  degrees of freedom
## AIC: 280.72
##
## Number of Fisher Scoring iterations: 7
```

7. Plot

```r
# Plot the predicted class labels
df_sim2 <- df_sim2 %>%
  mutate(log_po_prob = predict(log_po_model, type='response'),
         log_po_pred = as.factor(ifelse(log_po_prob >= 0.5, 1, 0)))
df_sim2 %>%
  ggplot(aes(x=x_sim, y=y_sim, color=log_po_pred)) +
  geom_point()
```



The interaction term plays a role.

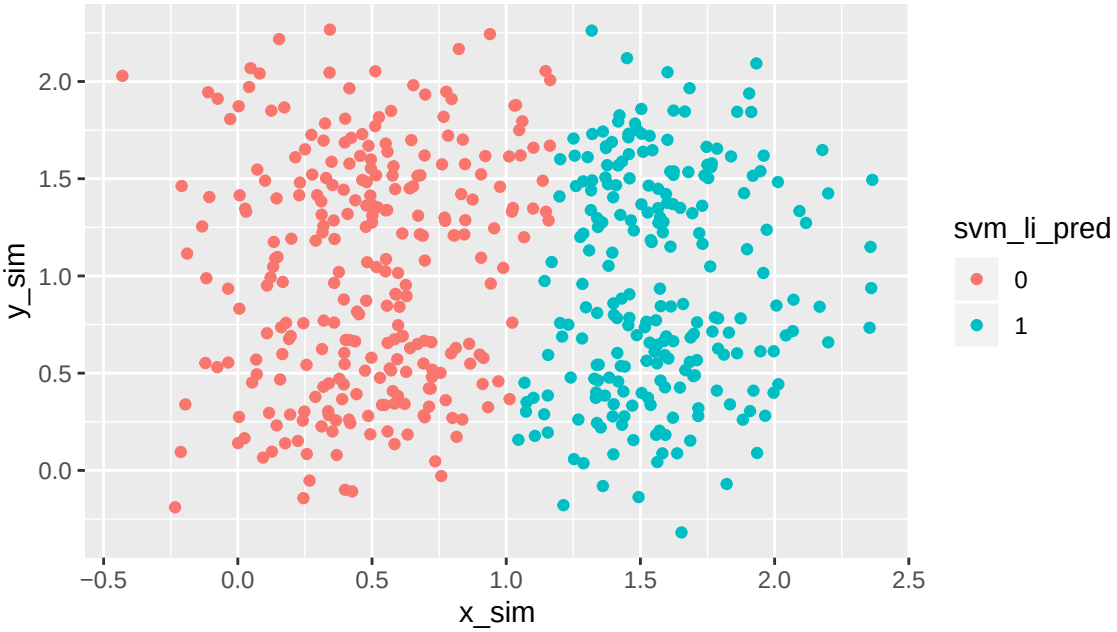
8. SVM linear

```
(svm_linear2 <- ksvm(
  cl_sim ~ x_sim + y_sim,
  data = df_sim2,
  type = "C-svc",
  kernel = "vanilladot",
))
```

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

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 499
##
## Objective Function Value : -497.6982
## Training error : 0.472
```

```
df_sim2 <- df_sim2 %>%
  mutate(svm_li_pred = predict(svm_linear2, type='response'))
# Plot the predicted class labels
df_sim2 %>%
  ggplot(aes(x=x_sim, y=y_sim, color=svm_li_pred)) +
  geom_point()
```

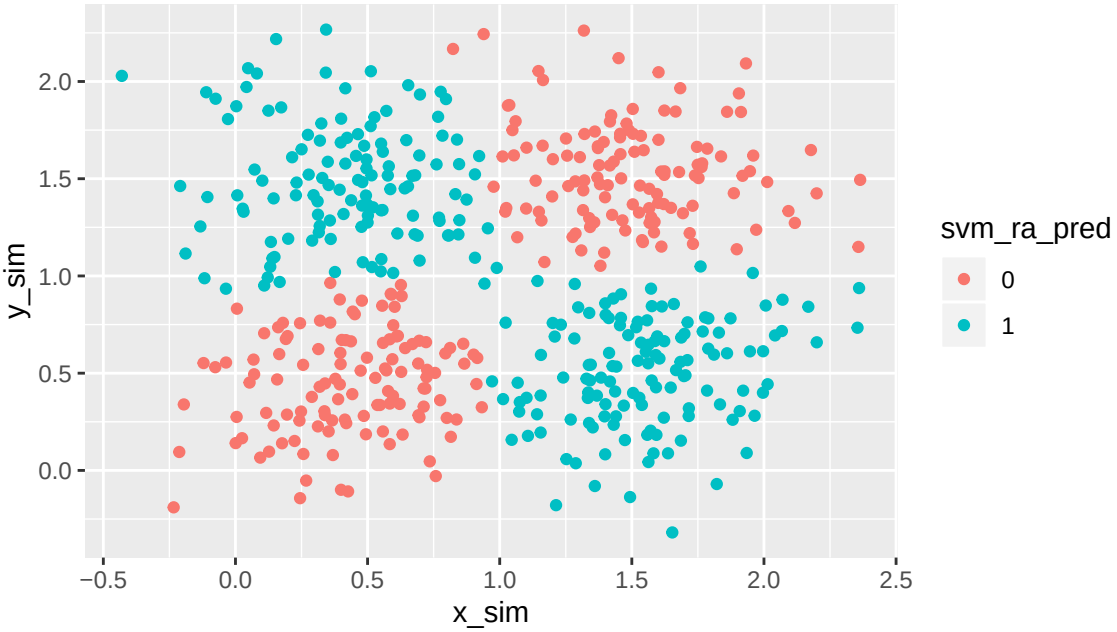


9. SVM radial

```
(svm_radial2 <- ksvm(
  cl_sim ~ x_sim * y_sim,
  data = df_sim2,
  type = "C-svc",
  kernel = "rbfdot",
))
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.841817574026438
##
## Number of Support Vectors : 149
##
## Objective Function Value : -123.048
## Training error : 0.09
```

```
df_sim2 <- df_sim2 %>%
  mutate(svm_ra_pred = predict(svm_radial2, type='response'))
# Plot the predicted class labels
df_sim2 %>%
  ggplot(aes(x=x_sim, y=y_sim, color=svm_ra_pred)) +
  geom_point()
```



10.

```
data.frame('Model' = c('Logistic (linear)', 'Logistic (non-linear)', 'SVM (linear)', 'SVM (non-linear)'
           'Error' = c(sum(df_sim2$cl_sim!=df_sim2$log_li_pred),
             sum(df_sim2$cl_sim!=df_sim2$log_po_pred),
             sum(df_sim2$cl_sim!=df_sim2$svm_li_pred),
             sum(df_sim2$cl_sim!=df_sim2$svm_ra_pred)
))
```

Model	Error
Logistic (linear)	248
Logistic (non-linear)	50
SVM (linear)	236
SVM (non-linear)	45

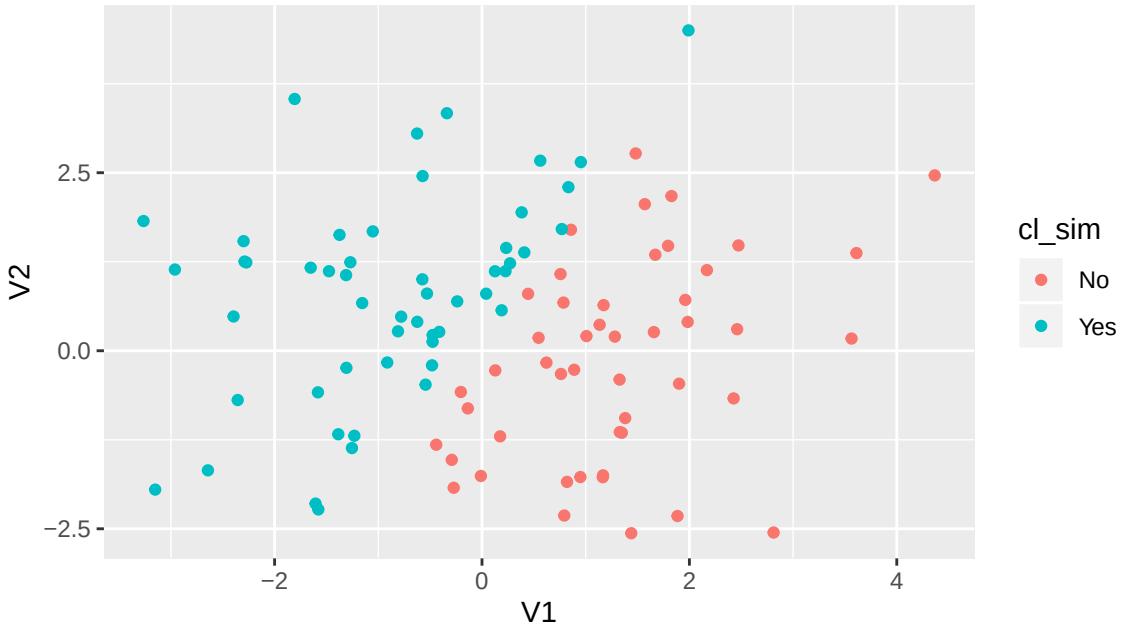
The above shows the number of error for each model. Both non-linear logistic model and non-linear SVM yields lower error rate. This is because the original dataset is distributed in non-linear. Also, the SVM models perform better than the logistic models both in linear and non-linear, for svm is more flexible adjusting the data. However, svm is more vulnerable to overfitting problem due to this flexibility, so if we want to check which model is preferable to certain dataset, we have to split the data and check its validation with test dataset. On the other hand, if we use logistic regression, we have to identify function form for certain dataset whereas we do not need to do that in svm model.

Tuning cost

11. Generate new data

```
df_sim3 <- as.data.frame(matrix(rnorm(200, 0.5, 1.5), ncol=2))
df_sim3 <- df_sim3 %>%
  mutate(cl_sim = ifelse(V2 - 2 * V1 > 0, 1, 0)) %>%
```

```
  mutate(cl_sim = factor(cl_sim, levels = c(0, 1), labels = c("No", "Yes"))
)
# Plot
df_sim3 %>%
  ggplot(aes(x=V1, y=V2, color=cl_sim)) +
  geom_point()
```



12. CV error with different cost values

```
cv_ctrl <- trainControl(method = "cv",
                        number = 10,
                        savePredictions = "final",
                        classProbs = TRUE)

# fit model
svm_linear3 <- train(
  cl_sim ~ V1 + V2,
  data = df_sim3,
  method = "svmLinear",
  trControl = cv_ctrl,
  tuneGrid = expand.grid(C = c(0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000)),
  tuneLength = 10
)

svm_linear3$results %>%
  select(C, Accuracy) %>%
  mutate(Error = round(1 - Accuracy, 2) * 100)
```

C	Accuracy	Error
1e-03	0.7506061	25
1e-02	0.8503030	15
1e-01	0.9297980	7

C	Accuracy	Error
1e+00	0.9809091	2
1e+01	0.9900000	1
1e+02	0.9788889	2
1e+03	0.9800000	2
1e+04	0.9788889	2

13. Test errors with different cost values

```
df_sim3_test <- as.data.frame(matrix(rnorm(200, 0.5, 1.5), ncol=2))
df_sim3_test <- df_sim3_test %>%
  mutate(cl_sim = ifelse(V2 - 2 * V1 > 0, 1, 0)) %>%
  mutate(cl_sim = factor(cl_sim, levels = c(0, 1), labels = c("No", "Yes"))
)

error_table = c()
for (i in c(0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000)) {
  # fit model
  svm_loop <- train(
    cl_sim ~ V1 + V2,
    data = df_sim3,
    method = "svmLinear",
    trControl = cv_ctrl,
    tuneGrid = expand.grid(C = i),
  )
  predict(svm_loop, newdata=df_sim3_test)

  error <- svm_loop$results %>%
    select(C, Accuracy) %>%
    mutate(Error = round(1 - Accuracy, 2) * 100)

  error_table <- rbind(error_table, error)
}
error_table
```

C	Accuracy	Error
1e-03	0.7389899	26
1e-02	0.9007071	10
1e-01	0.9609091	4
1e+00	0.9788889	2
1e+01	0.9909091	1
1e+02	0.9788889	2
1e+03	0.9800000	2
1e+04	0.9900000	1

14.

The best cost value for both train and test set is 10, which yields accuracy of 0.99, and only one error. When we see smaller cost value (0.01 and 0.1), we got higer accuracy in test set than in train set. This is because the models have wider margin which avoids overfitting.

Application: Predicting attitudes towards racist college professors

15. SVM linear

```
gss_train <- read_csv('data/gss_train.csv') %>%
  mutate(colrac = factor(colrac, levels = c(0,1), labels = c('No', 'Yes')))
gss_test <- read_csv('data/gss_test.csv') %>%
  mutate(colrac = factor(colrac, levels = c(0,1), labels = c('No', 'Yes')))

cv_ctrl <- trainControl(method = "cv",
                        number = 10,
                        savePredictions = "final",
                        classProbs = TRUE)

cost_grid <- expand.grid(C = c(0.001, 0.01, 0.1, 1, 10, 100, 1000))

svm_li_gss <- train(
  colrac ~ .,
  data = gss_train,
  method = "svmLinear",
  trControl = cv_ctrl,
  tuneGrid = cost_grid,
  tuneLength = 10
)

svm_li_gss$results %>%
  select(C, Accuracy) %>%
  mutate(Error = round((1 - Accuracy) * nrow(gss_train)))
```

C	Accuracy	Error
1e-03	0.7798821	326
1e-02	0.7940715	305
1e-01	0.7967695	301
1e+00	0.7933819	306
1e+01	0.7933819	306
1e+02	0.7953999	303
1e+03	0.7913503	309

The model yields highest accuracy when the cost value is 0.01, and its accuracy is 0.7968

16. SVM radial & polynomial

```
cost_grid <- expand.grid(C = c(0.001, 0.01, 0.1, 1, 10, 100),
                         scale = c(0.01, .005),
                         degree=c(3,4,5,6))

svm_po_gss <- train(
  colrac ~ .,
  data = gss_train,
  method = "svmPoly",
  trControl = cv_ctrl,
  tuneGrid = cost_grid
```

```
)

svm_po_gss$results %>%
  select(C, scale, degree, Accuracy) %>%
  mutate(Error = round((1 - Accuracy) * nrow(gss_train)))
```

C	scale	degree	Accuracy	Error
1e-03	0.005	3	0.7029340	440
1e-02	0.005	3	0.7029340	440
1e-01	0.005	3	0.7886904	313
1e+00	0.005	3	0.8028569	292
1e+01	0.005	3	0.7967713	301
1e+02	0.005	3	0.7670778	345
1e-03	0.010	3	0.7042853	438
1e-02	0.010	3	0.7508752	369
1e-01	0.010	3	0.7940867	305
1e+00	0.010	3	0.8028433	292
1e+01	0.010	3	0.7697805	341
1e+02	0.010	3	0.7711364	339
1e-03	0.005	4	0.7036096	439
1e-02	0.005	4	0.7076592	433
1e-01	0.005	4	0.7900463	311
1e+00	0.005	4	0.8015146	294
1e+01	0.005	4	0.7846182	319
1e+02	0.005	4	0.7738346	335
1e-03	0.010	4	0.7036142	439
1e-02	0.010	4	0.7657265	347
1e-01	0.010	4	0.8015146	294
1e+00	0.010	4	0.7967713	301
1e+01	0.010	4	0.7751769	333
1e+02	0.010	4	0.7751769	333
1e-03	0.005	5	0.7056367	436
1e-02	0.005	5	0.7400644	385
1e-01	0.005	5	0.7934110	306
1e+00	0.005	5	0.8008208	295
1e+01	0.005	5	0.7778841	329
1e+02	0.005	5	0.7758571	332
1e-03	0.010	5	0.7056367	436
1e-02	0.010	5	0.7799157	326
1e-01	0.010	5	0.8035280	291
1e+00	0.010	5	0.7886768	313
1e+01	0.010	5	0.7825957	322
1e+02	0.010	5	0.7819200	323
1e-03	0.005	6	0.7036142	439
1e-02	0.005	6	0.7535779	365
1e-01	0.005	6	0.7967894	301
1e+00	0.005	6	0.8021721	293
1e+01	0.005	6	0.7771948	330
1e+02	0.005	6	0.7798975	326
1e-03	0.010	6	0.7069880	434
1e-02	0.010	6	0.7839606	320
1e-01	0.010	6	0.8042173	290
1e+00	0.010	6	0.7832714	321

C	scale	degree	Accuracy	Error
1e+01	0.010	6	0.7866543	316
1e+02	0.010	6	0.7873299	315

```
cost_grid <- expand.grid(C = c(0.001, 0.01, 0.1, 1, 10, 100),
                         sigma=c(0.001, 0.01, 0.1, 1, 10, 100))

svm_ra_gss <- train(
  colrac ~ .,
  data = gss_train,
  method = "svmRadial",
  tuneGrid = cost_grid,
  trControl = cv_ctrl,
)

svm_ra_gss$results %>%
  select(C, sigma, Accuracy) %>%
  mutate(Error = round((1 - Accuracy) * nrow(gss_train)))
```

C	sigma	Accuracy	Error
1e-03	1e-03	0.6981761	447
1e-02	1e-03	0.6968248	449
1e-01	1e-03	0.6961445	450
1e+00	1e-03	0.7872897	315
1e+01	1e-03	0.7987353	298
1e+02	1e-03	0.8006894	295
1e-03	1e-02	0.7069372	434
1e-02	1e-02	0.7062706	435
1e-01	1e-02	0.7784691	328
1e+00	1e-02	0.8007670	295
1e+01	1e-02	0.7973386	300
1e+02	1e-02	0.7790766	327
1e-03	1e-01	0.7203918	414
1e-02	1e-01	0.7480861	373
1e-01	1e-01	0.7447031	378
1e+00	1e-01	0.7399914	385
1e+01	1e-01	0.7508162	369
1e+02	1e-01	0.7494375	371
1e-03	1e+00	0.5253224	703
1e-02	1e+00	0.5253224	703
1e-01	1e+00	0.5253224	703
1e+00	1e+00	0.5253224	703
1e+01	1e+00	0.5253224	703
1e+02	1e+00	0.5253224	703
1e-03	1e+01	0.5253224	703
1e-02	1e+01	0.5253224	703
1e-01	1e+01	0.5253224	703
1e+00	1e+01	0.5253224	703
1e+01	1e+01	0.5253224	703
1e+02	1e+01	0.5253224	703
1e-03	1e+02	0.5253224	703
1e-02	1e+02	0.5253224	703

C	sigma	Accuracy	Error
1e-01	1e+02	0.5253224	703
1e+00	1e+02	0.5253224	703
1e+01	1e+02	0.5253224	703
1e+02	1e+02	0.5253224	703

```
stopCluster(cl)
```

For polynomial, the model with C = 0.1, scale=0.01, degree=6 yields the highest accuracy 0.8042
For radial, the model with C = 1, sigma = 0.01 yiedls the highest accuracy 0.8008
In sum, polynomial SVM has the highest accuracy, whose cost value is 0.1, meaning that the margin is not strict.