Homework 4

Jingyi Guo(jingyig1), Pittsburgh Campus

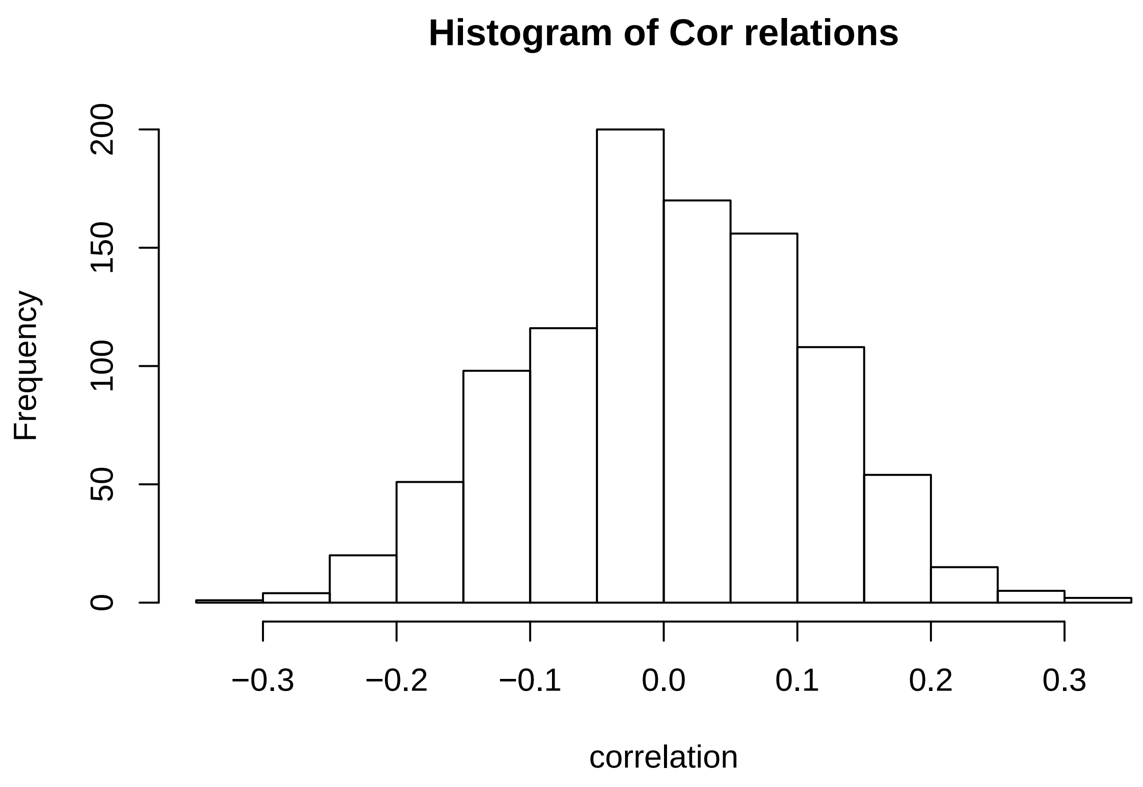
11/28/2017

```r
setwd("/Users/apple/Desktop/ML2/Homework 4")
library(mgcv)
library(glmnet)
library(randomForest)
library(e1071)
library(MASS)
```

1

(a)

```r
#You get to observe a bunch of data (that turns out to be worthless)
#Returns a data set with 1000 variables and n data points
#The first 1000 columns are the X variables (called X1,...X100)
#The last column is the response (called y)
get_data = function(n){
  p=1000
  X = matrix(rnorm(n*p),ncol=p)
  y = rnorm(n)
  dat = data.frame(X,y)
  dat
}

#Generate some data
n = 100
set.seed(1)
#Note, your y and X variables are all in this data frame. See the comments for get_data for details.
dataset = get_data(100)

cor_Xy = apply(as.matrix(dataset[,1:1000]),2,cor,dataset$y)
hist(cor_Xy, xlab = "correlation", main = "Histogram of Cor relations")
```



The histogram forms a bell shape curve of correlations around 0.

(b)

Variables in Xgood include:

```
Xgood = cbind(dataset[,which(cor_Xy>0.25)],dataset$y)
print(names(Xgood))
```

```
## [1] "X63"       "X273"      "X383"      "X879"      "X918"      "X939"
## [7] "X972"      "dataset$y"
```

(c)

```
train = Xgood[1:(0.8*nrow(Xgood)),]
test = Xgood[(0.8*nrow(Xgood)+1):nrow(Xgood),]
lasso = glmnet(as.matrix(train[,-ncol(train)]),as.matrix(train[,ncol(train)]),family = "gaussian",alpha
lasso.cv = cv.glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),family = "gaussian",alpha = 1)

train_lasso = predict(lasso.cv,as.matrix(train[,-ncol(train)]))
mse_lasso_train = mean((train_lasso-as.matrix(train[,ncol(train)]))^2)

test_lasso = predict(lasso.cv,as.matrix(test[,-ncol(test)]))
mse_lasso_test = mean((test_lasso-as.matrix(test[,ncol(test)]))^2)

# train error
print(mse_lasso_train)
```

```
## [1] 0.8689179
```

```
# test error
print(mse_lasso_test)
```

```
## [1] 0.8693674
```

So the train error of lasso model is 0.8689179, the test error is 0.8693674.

(d)

```
dataset_new = get_data(100)
Xgood_new = cbind(dataset_new[,which(cor_Xy>0.25)],dataset_new$y)
new_lasso = predict(lasso.cv,as.matrix(Xgood_new[,-ncol(Xgood_new)]))
mse_lasso_new = mean((new_lasso-as.matrix(Xgood_new[,ncol(Xgood_new)]))^2)
print(mse_lasso_new)
```

```
## [1] 0.9544102
```

So the test error of lasso model on new generated data is 1.016396. The lasso model prediction error on the new dataset is larger than the previous one. This is because we extract the same columns as we did for the previous part, but these columns do not have correlation > 0.25 with y, therefore these columns has less prediction power than in the previous part.

2

(a)

```
load('problem2.RData')

# linear model
fit_lm = lm(y~.,data=train)
mse_lm_train = mean((fit_lm$residuals)^2)
test_lm = predict(fit_lm,test)
mse_lm_test = mean((test$y-test_lm)^2)

# train error
print(mse_lm_train)
```

```
## [1] 0.04692521
```

```
# test error
print(mse_lm_test)
```

```
## [1] 2.444134
```

So the train error of linear model is 0.04692521, the test error is 2.444134.

(b)

```
fit_ridge = glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=0)
fit_ridge_cv =cv.glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=0)
train_ridge = predict(fit_ridge_cv,as.matrix(train[,-1]))

mse_ridge_train = mean((train_ridge-as.matrix(train[,1]))^2)
```

```r
test_ridge = predict(fit_ridge_cv,as.matrix(test[,-1]))

mse_ridge_test = mean((test_ridge-as.matrix(test[,1]))^2)

# train error
print(mse_ridge_train)
```

```
## [1] 0.1135416
```

```r
# test error
print(mse_ridge_test)
```

```
## [1] 2.118056
```

So the train error of ridge regression model is 0.1039836, the test error is 2.131166. The train error increase a little and the test error decrease a little compared with the linear model.

(c)

```r
fit_lasso = glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=1)
fit_lasso_cv = cv.glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=1)

train_lasso = predict(fit_lasso_cv,as.matrix(train[,-1]))
mse_lasso_train = mean((train_lasso-as.matrix(train[,1]))^2)

test_lasso = predict(fit_lasso_cv,as.matrix(test[,-1]))
mse_lasso_test = mean((test_lasso-as.matrix(test[,1]))^2)

# train error
print(mse_lasso_train)
```

```
## [1] 0.05102891
```

```r
# test error
print(mse_lasso_test)
```

```
## [1] 2.361904
```

So the train error of lasso regression model is 0.0492735, the test error is 2.380025. The train error increase a little and the test error decrease a little compared with the linear model, however the improvement is not as much as that of the ridge regression model.

3

(a) logistic regression

```r
load('problem3.RData')
# logistic regression
fit_logistic = glm(y~.,data=train,family = "binomial")
train_logistic = fit_logistic$fitted.values>0.5
mse_logistic_train = mean(train_logistic != train[,1])
test_logistic = predict(fit_logistic,test[,-1],type = "response")>0.5
mse_logistic_test = mean(test_logistic != test[,1])
```

```
# train error
print(mse_logistic_train)
```

```
## [1] 0.1433333
```

```
# test error
print(mse_logistic_test)
```

```
## [1] 0.2383
```

So the train error of logistic regression model is 0.1433333, the test error is 0.2383.

We can try adding bias to the model to improve the test error.

(b) logistic lasso

```
fit_logisticlasso = glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=1
,family = "binomial")
fit_logisticlasso_cv = cv.glmnet(as.matrix(train[,-1]),as.matrix(train[,1]),alpha=1,family = "binomial"
train_logisticlasso = predict(fit_logisticlasso_cv,as.matrix(train[,-1]),type ="response")>0.5
mse_logisticlasso_train = mean(train_logisticlasso != train[,1])
test_logisticlasso = predict(fit_logisticlasso_cv,as.matrix(test[,-1]),type="response")>0.5
mse_logisticlasso_test = mean(test_logisticlasso != test[,1])

# train error
print(mse_logisticlasso_train)
```

```
## [1] 0.1833333
```

```
# test error
print(mse_logisticlasso_test)
```

```
## [1] 0.2054
```

So the train error of logistic lasso model is 0.1833333, the test error is 0.2023. The test error improved from the model in (a).

Non-zero variables:

```
lambda_pos = which(fit_logisticlasso_cv$glmnet.fit$lambda == fit_logisticlasso_cv$lambda.1se)
nonzero_pos = which(fit_logisticlasso$beta[,lambda_pos] != 0)
logisticlasso_variables = fit_logisticlasso$beta[nonzero_pos,lambda_pos]
print(logisticlasso_variables)
```

```
##          X9         X20         X22         X31         X40
## -0.01350482 -0.62533542  0.02848308  0.39420103 -0.55350689
```

(c)

In (b), the train error and the test error are already quite close. In addition, it is hard to further improve the model adding bias because we have already shrunken 51 variables to 7.

(d) random forest

```r
fit_rf = randomForest(factor(y)~.,data=train)
oob_rf = fit_rf$predicted
mse_rf_oob = mean(oob_rf!=train[,1])
train_rf = predict(fit_rf,train[,-1])
mse_rf_train = mean(train_rf!=train[,1])
test_rf = predict(fit_rf,test[,-1])
mse_rf_test = mean(test_rf!=test[,1])

# train error
print(mse_rf_train)
```

```
## [1] 0
```

```r
# out of bag error
print(mse_rf_oob)
```

```
## [1] 0.2266667
```

```r
# test error
print(mse_rf_test)
```

```
## [1] 0.2308
```

So the train error of random forest model is 0, out-of-bag error is 0.24 and the test error is 0.2199. The out-of-bag error and test error are larger than that of logistic lasso, so random forest model does not improve the prediction performance.

```r
print(fit_rf)
```

```
##
## Call:
##  randomForest(formula = factor(y) ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 22.67%
## Confusion matrix:
##    0   1 class.error
## 0 34  64  0.65306122
## 1  4 198  0.01980198
```

This is probably because in random forest, at each split, 7 variables are randomly tried, which may not be a good prediction for y.

One possible way to improve the random forest is to restrict the variables used.

Here I pick variables as we selected in logistic lasso model:

```r
fit_rf_improve = randomForest(factor(y)~X3+X9+X20+X22+X24+X31+X40,data=train)
oob_rf_improve = fit_rf_improve$predicted
mse_rf_improve_oob = mean(oob_rf_improve!=train[,1])
train_rf_improve = predict(fit_rf_improve,train[,-1])
mse_rf_improve_train = mean(train_rf_improve!=train[,1])
test_rf_improve = predict(fit_rf_improve,test[,-1])
mse_rf_improve_test = mean(test_rf_improve!=test[,1])

# train error
```

```
print(mse_rf_improve_train)
```

```
## [1] 0
```

```
# out of bag error
print(mse_rf_improve_oob)
```

```
## [1] 0.1733333
```

```
# test error
print(mse_rf_improve_test)
```

```
## [1] 0.1698
```

So the train error of improved random forest model is 0, out-of-bag error is 0.1733333 and the test error is 0.174. After selecting 7 variables out of 51 variables to be included in the tree, the out-of-bag error and test error both improved compared to the previous random forest model.

(e) GAM

```
fit_gam = gam(y~s(X3)+s(X9)+s(X20)+s(X22)+s(X24)+s(X31)+s(X40),data=train,family = "binomial")
train_gam = (fit_gam$fitted.values>0.5)
mse_gam_train = mean(train_gam!=train[,1])
test_gam = predict(fit_gam,test[,-1])>0.5
mse_gam_test = mean(test_gam!=test[,1])

# train error
print(mse_gam_train)
```

```
## [1] 0.07333333
```

```
# test error
print(mse_gam_test)
```

```
## [1] 0.1295
```

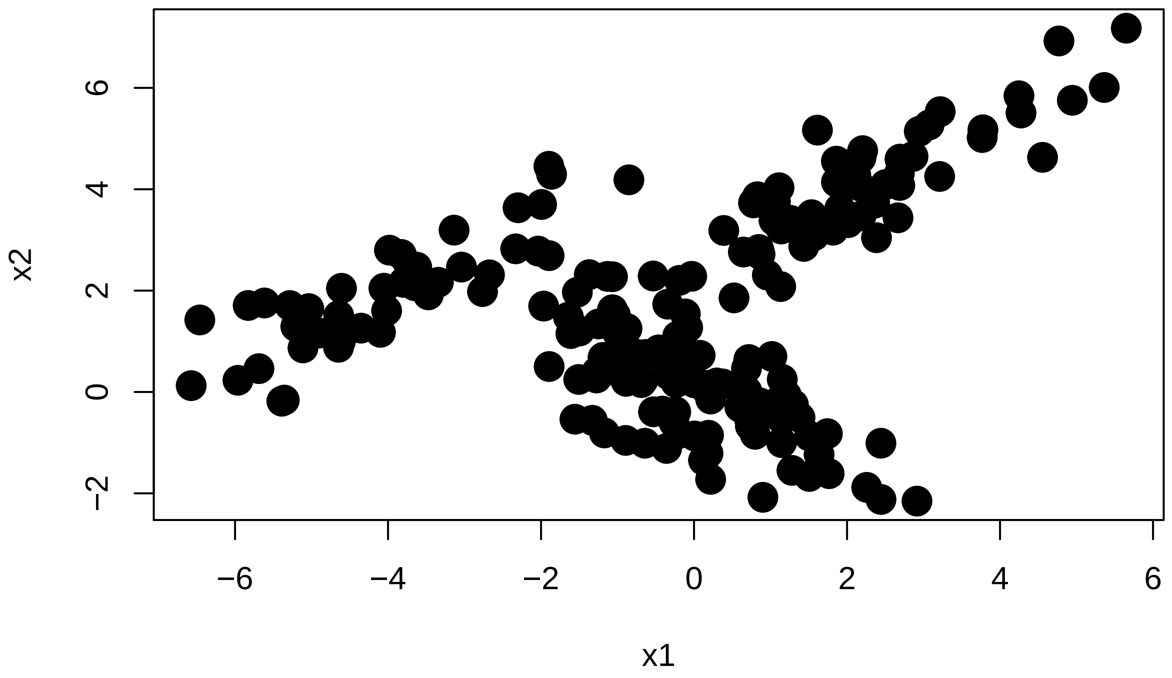
So the train error of GAM model is 0.07333333 and the test error is 0.1295. The GAM model improved a lot compared to the models above.

4

(a)

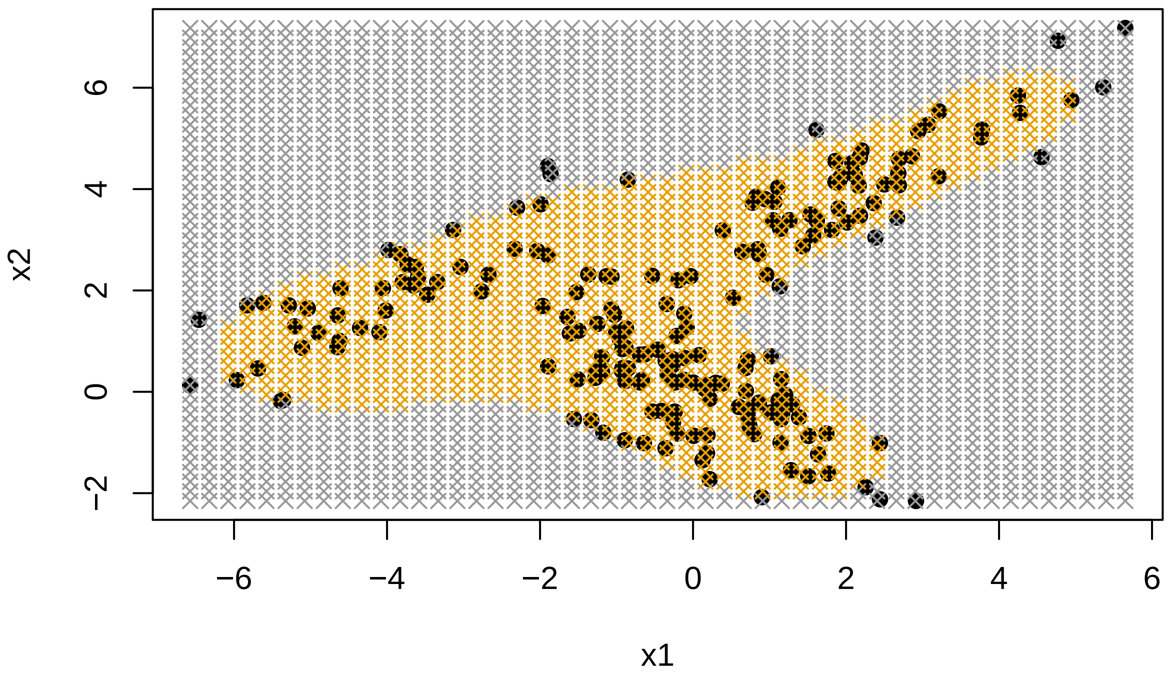
```
set.seed(1)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

#Generates crazy looking X data.  This has two columns.
get_data = function(n){
  K = 3
  pi0 = c(.5,.3,.2)
  mu = cbind(c(0,0),c(1,2),c(-2,1))*2
  Sigma = matrix(c(1,.8,.8,.9),2)*1.6
  Sigma2 = matrix(c(1,-.7,-.7,.9),2)*1.6
  Sigma3 = matrix(c(1,0,0,.9),2)*1
```

```r
  X = rbind(
    mvrnorm(pi0[1]*n,mu[,1],Sigma2),
    mvrnorm(pi0[2]*n,mu[,2],Sigma),
    mvrnorm(pi0[3]*n,mu[,3],Sigma)
  )
  X = as.data.frame(X)
  names(X) = c('x1','x2')
  X
}

#Plots the one-class svm results.  Pass in the svm fitted model and a data matrix.
plot_results = function(fit, X){
  xlim = range(X[,1])
  ylim = range(X[,2])
  Xgrid = as.matrix(expand.grid(seq(xlim[1],xlim[2],length.out=50),seq(ylim[1],ylim[2],length.out=50)))
  b = predict(fit,Xgrid)
  plot(X,pch=19, cex=1)
  points(Xgrid[,1],Xgrid[,2],col=cbPalette[as.numeric(b)+1],pch=4,cex=1)
}

#Generate data
set.seed(1)
X = get_data(200)

#Plot data
plot(X,pch=20, cex=3)
```



```r
### TODO: Really the only thing you have to do is add the SVM fitting function in here
###       and then play with it
fit = svm(X,kernel = "radial",nu = '0.1',gamma = .5,scale = T)

#Draw the decision regions by classifying a grid of points
```

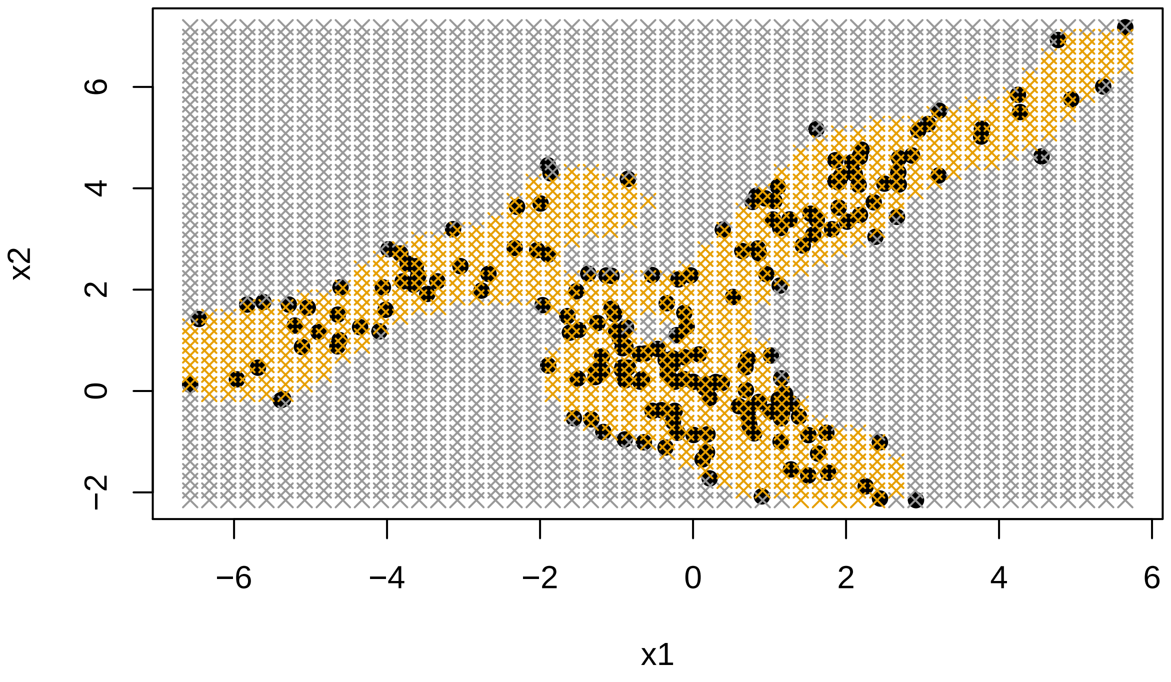
```r
plot_results(fit, X)
```



(b)

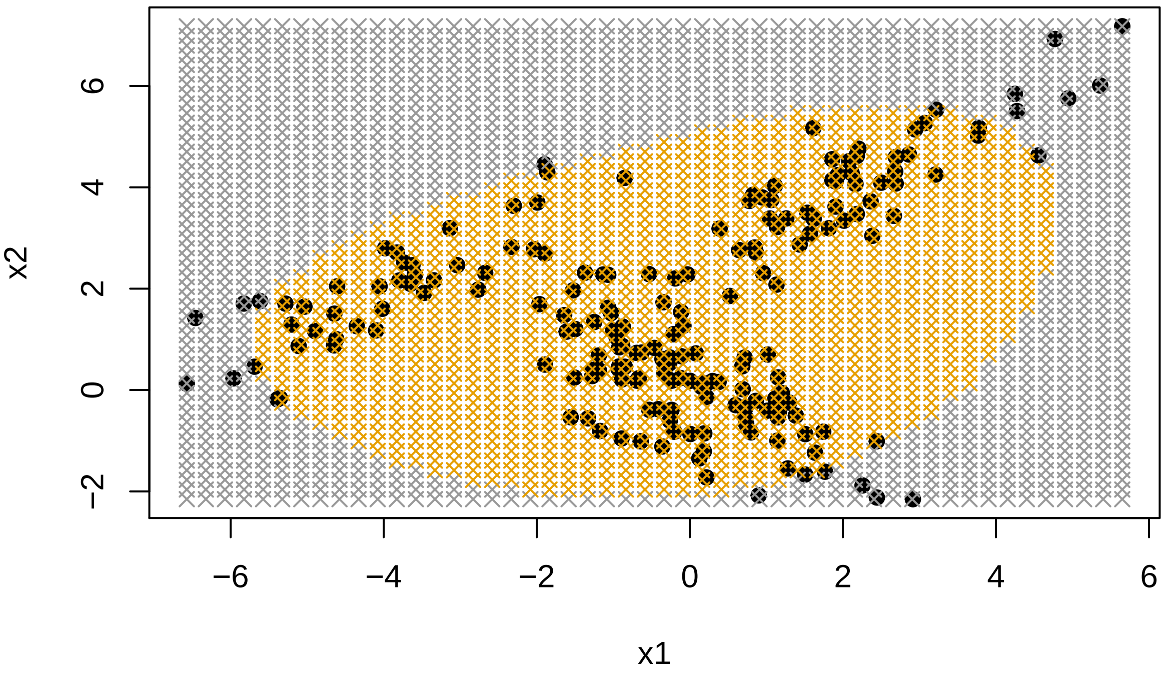
gamma=2:

```r
#gamma=2
fit = svm(X,kernel = "radial",nu = '0.1',gamma = 2,scale = T)
plot_results(fit, X)
```



gamma=0.1:

```r
#gamma=0.1
fit = svm(X,kernel = "radial",nu = '0.1',gamma = 0.1,scale = T)
plot_results(fit, X)
```



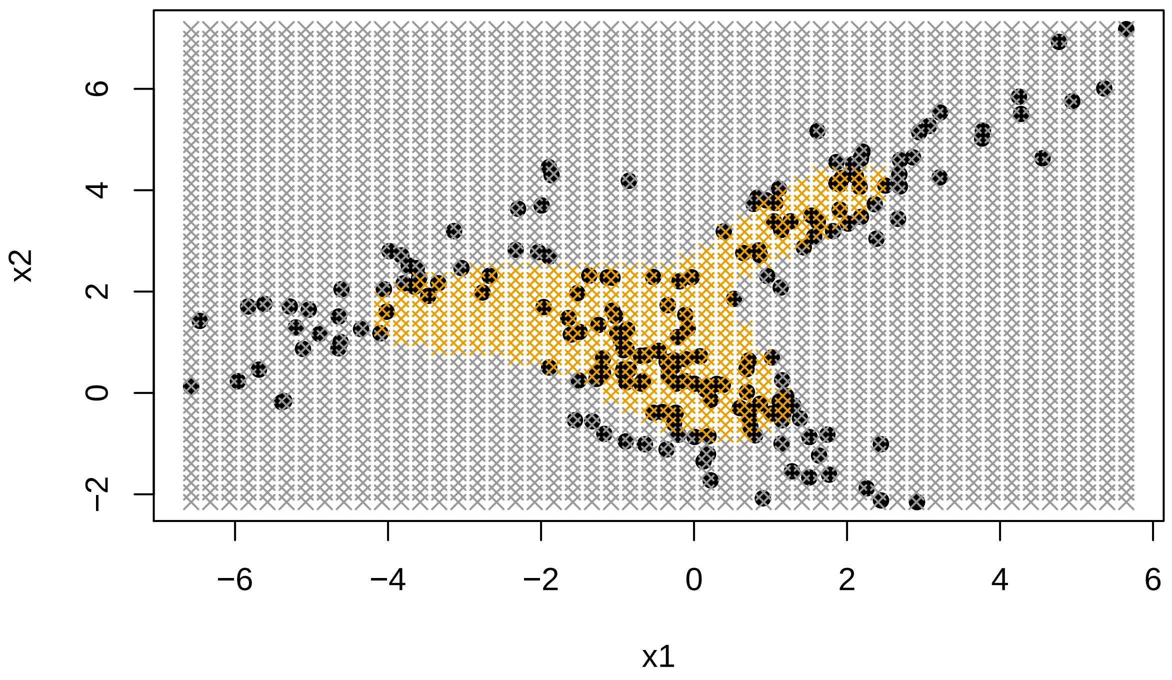
If we choose the alarger gamma, the boundaries will be more sensitive to the training data, and become closer to the shape of the training data.

A large gamma means a Gaussian with a large variance, therefore, the influence of x_j is larger, i.e. the support vector have a wide-spread influence. If gamma is small, then the variance is small, implying the support vector does not have wide-spread influence. So larger gamma leads to higher bias and lower variance models, and vice-versa.

(c)

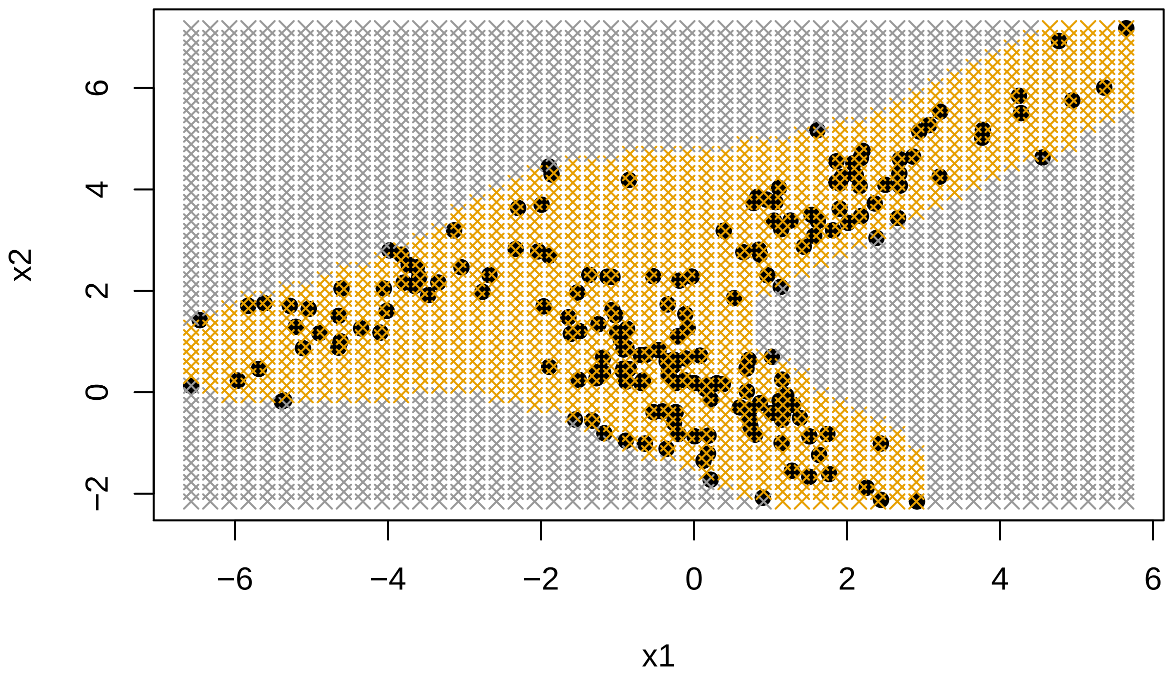
nu=0.5:

```r
#nu=0.5
fit = svm(X,kernel = "radial",nu = '0.5',gamma = 0.5,scale = T)
plot_results(fit, X)
```



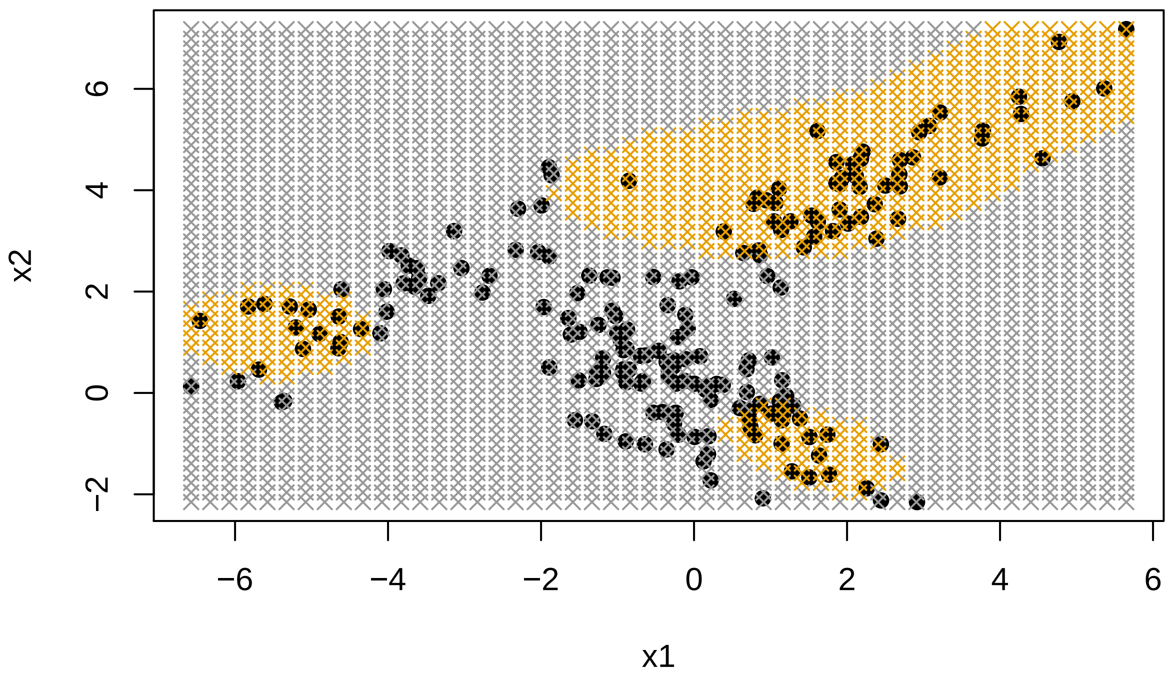
nu=0.01:

```
#nu=0.01
fit = svm(X,kernel = "radial",nu = '0.01',gamma = 0.5,scale = T)
plot_results(fit, X)
```



nu=0.0001:

```
#nu=0.0001
fit = svm(X,kernel = "radial",nu = '0.0001',gamma = 0.5,scale = T)
plot_results(fit, X)
```



For higher nu, the penalty for the data in soft margin is higher. So for higher nu, the clusters are more concentrated, and for lower nu, the clusters are more separated.