Problem Set #6

Kevin McAlister

March 17th, 2022

This is the sixth problem set for QTM 385 - Intro to Statistical Learning. This homework will cover applied exercises related to support vector machines and other classification approaches.

Please use the intro to RMarkdown posted in the Intro module and my .Rmd file as a guide for writing up your answers. You can use any language you want, but I think that a number of the computational problems are easier in R. Please post any questions about the content of this problem set or RMarkdown questions to the corresponding discussion board.

Your final deliverable should be two files: 1) a .Rmd/.ipynb file and 2) either a rendered HTML file or a PDF. Students can complete this assignment in groups of up to 3. Please identify your collaborators at the top of your document. All students should turn in a copy of the solutions, but your solutions can be identical to those of your collaborators.

This assignment is due by March 28th, 2022 at 11:59 PM EST.

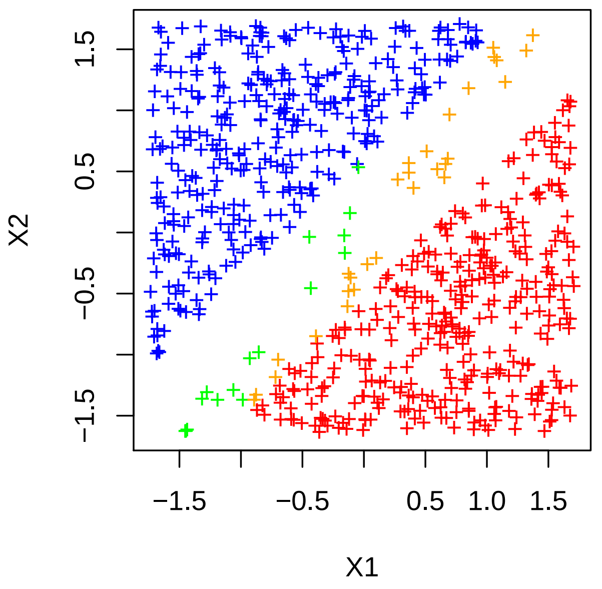
Problem 1: An Unintuitive Result

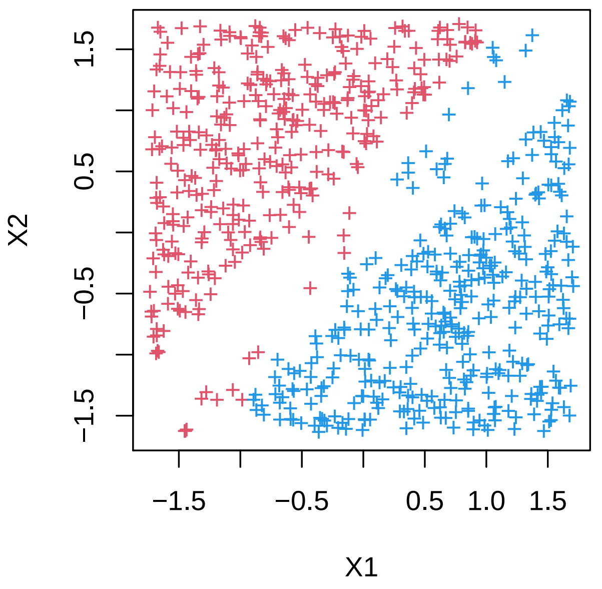
The support vector classifier generalizes the optimal separating hyperplane by allowing *slack variables* to handle cases where the points are misclassified (or within the margin). A somewhat uninuitive result related to the support vector classifier arises when we have a two-class problem where the data is only *barely separable* - a SVC with a small cost that misclassifies a few data points may outperform an optimally separating hyperplane (with no points inside the margin, let alone misclassified) on out-of-sample data. Let's show this via a simulated example.

Part 1

`simSVMTrain.csv` contains 729 instances of two predictors and a binary class label. In this simulated example, the classes are **linearly separable**. However, they are generated from two separate data generating processes. As shown in the below figure, the first DGP generates data that is separable with a large margin. The second DGP generates data that is still linearly separable, but with less of a margin than the first. This kind of conditional separating behavior is "exaggerated" in this data set, but it does occur quite frequently in real data sets when there is an important omitted predictor that conditions how classes are separated.

```r
setwd("/Users/zejiachen/Desktop/Sspring 2022/Statstical Learning/problemSet6")
train <- read.csv("simSVMTrain.csv")
par(mfrow = c(1,2))
plot(train$X1,train$X2, col = train$class + 3, type = "n", xlab = "X1", ylab = "X2")
points(train$X1[train$class == -1 & train$group == 1],train$X2[train$class == -1 & train$group == 1], p
points(train$X1[train$class == 1 & train$group == 1],train$X2[train$class == 1 & train$group == 1], pch
points(train$X1[train$class == -1 & train$group == 2],train$X2[train$class == -1 & train$group == 2], p
points(train$X1[train$class == 1 & train$group == 2],train$X2[train$class == 1 & train$group == 2], pch
plot(train$X1,train$X2, col = as.numeric(train$class) + 3, pch = "+", xlab = "X1", ylab = "X2")
```





```r
par(mfrow = c(1,1))
```

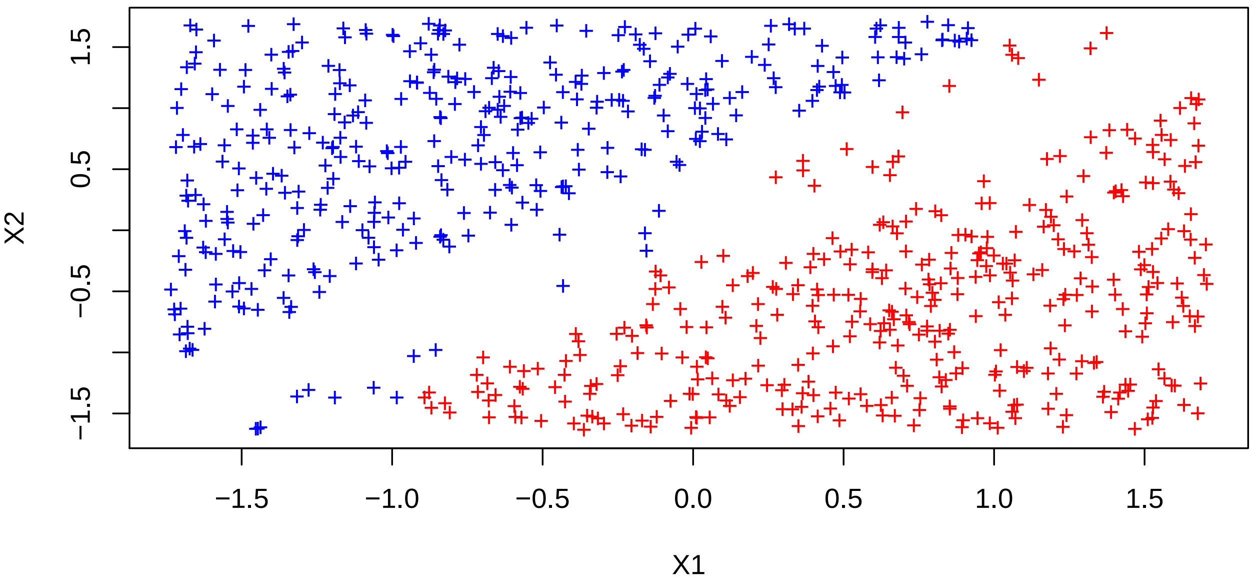
While it may not immediately appear to be the case in the combined data set, the classes are actually linearly separable! Prove that this is true by 1) plotting the data and corresponding labels, 2) computing the optimal separating hyperplane and corresponding margins and plotting them against the data, and 3) showing that the **signed distance** from the hyperplane multiplied by the class label is greater than or equal to one for all observations in the data set. (Note: When you compute the signed distance, you may find that a few observations have a signed distance of .999—. This can happen due to a combination of rounding and our approximation to the perfectly separable case with a finite cost.)

Some hints:

1) We can use SVM software to get the separating hyperplane and margins in the perfectly separable case when we set `cost` to be very high. `cost = 10000` should be sufficient.
2) There's no need for a nonlinear kernel here since the decision boundary is linear. Make sure to set the kernel to linear or vanilladot since most SVM software sets the kernel to radial/Gaussian by default.
3) For the two predictor case, the separating hyperplane can be computed by finding all values of $\boldsymbol{x} \in \mathbb{R}^2$ such that $\hat{\alpha} + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0$. Given the coefficients (which can usually be found using `coef()` or a similar argument in R or `svm.decision_function` in `sklearn`) and a value of x_1, we can uniquely compute x_2 that solves the equation.
4) The predictors are already normalized, so don't worry about that here.
5) Since the SVM coefficients are normalized in the actual computation, we can compute the margins by finding values of $\boldsymbol{x}$ that solve $\hat{\alpha} + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = \pm 1$
6) For most SVM implementations, you can force it to treat any response as a classification problem by ensuring that the method used is *C-classification*. This means that you don't necessarily convert `y` to a factor in R. Check your chosen implementation for this option!
7) It is possible that all of the signed distances will be less than or equal to -1! Because there is no real requirement that the SVC assigns $y = -1$ to be the negative distance class, we could just have everything reversed. This is equivalent, so don't think that there's something wrong!

Step 1: plot the combined training data.

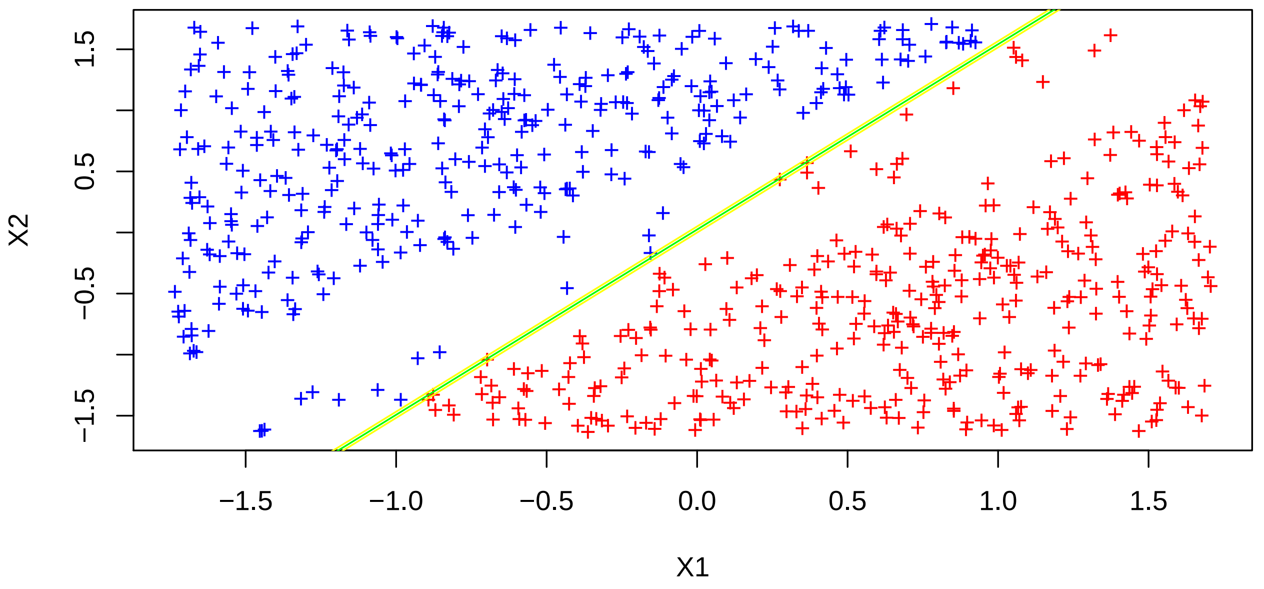
```r
par(mfrow = c(1, 1))
plot(train$X1, train$X2, col = train$class + 3, type = "n", xlab = "X1",
     ylab = "X2")
points(train$X1[train$class == -1], train$X2[train$class ==-1], pch = "+", col = "blue")
points(train$X1[train$class == 1], train$X2[train$class == 1], pch = "+", col = "red")
```



While it is not immediately obvious, there seems to be a line that perfectly separates the two classes.

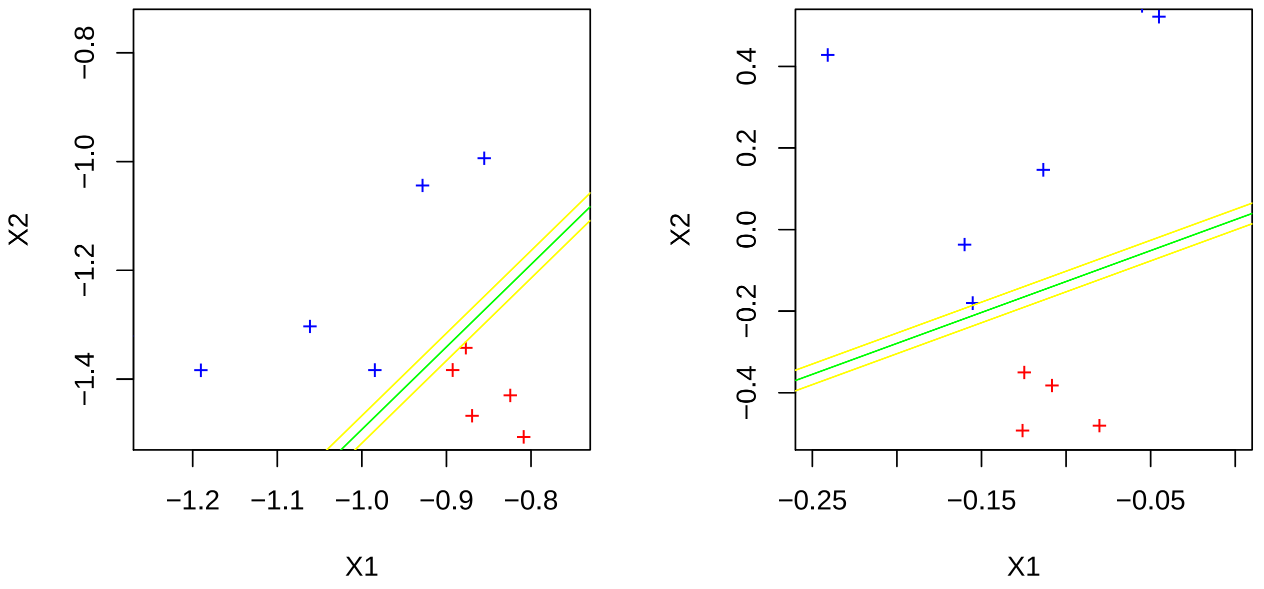
Step 2: find the optimal separating hyperplane and plot it on top of the graph

```
library(e1071)
svm1 <- svm(as.factor(class) ~ X1 + X2, data = train, kernel = "linear", cost = 10000, scale = FALSE)
cf1 <- coef(svm1)
par(mfrow = c(1, 1))
plot(train$X1, train$X2, col = train$class + 3, type = "n", xlab = "X1",
     ylab = "X2")
points(train$X1[train$class == -1], train$X2[train$class ==-1], pch = "+", col = "blue")
points(train$X1[train$class == 1], train$X2[train$class == 1], pch = "+", col = "red")
abline(-cf1[1]/cf1[3], -cf1[2]/cf1[3], col = "green")
abline(-(cf1[1] + 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
abline(-(cf1[1] - 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
```



The margin is very small but we can zoom in at areas where observations of different classes are close to see if the hyperplane is perfectly separating.

```r
par(mfrow = c(1, 2))
plot(train$X1, train$X2, col = train$class + 3, type = "n", xlab = "X1",
     ylab = "X2", xlim = c(-1.25,-0.75), ylim = c(-1.5,-0.75))
points(train$X1[train$class == -1], train$X2[train$class ==-1], pch = "+", col = "blue")
points(train$X1[train$class == 1], train$X2[train$class == 1], pch = "+", col = "red")
abline(-cf1[1]/cf1[3], -cf1[2]/cf1[3], col = "green")
abline(-(cf1[1] + 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
abline(-(cf1[1] - 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
plot(train$X1, train$X2, col = train$class + 3, type = "n", xlab = "X1",
     ylab = "X2", xlim = c(-0.25,0), ylim = c(-0.5,0.5))
points(train$X1[train$class == -1], train$X2[train$class ==-1], pch = "+", col = "blue")
points(train$X1[train$class == 1], train$X2[train$class == 1], pch = "+", col = "red")
abline(-cf1[1]/cf1[3], -cf1[2]/cf1[3], col = "green")
abline(-(cf1[1] + 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
abline(-(cf1[1] - 1)/cf1[3], -cf1[2]/cf1[3], col = "yellow")
```



Step 3: compute $y_i(\hat{\alpha} + x_i'\hat{\beta})$ and show that the result is greater than or equal to 1.

```
library(dplyr)
betanorm <- sqrt(cf1[2]^2+cf1[3]^2)
signdist <- train$class*(cf1[1]+cf1[2]*train$X1+cf1[3]*train$X2)
signdist[signdist <1]
```

```
## [1] 0.9998710 0.9999288 0.9999320
```

```
cf1
```

```
## (Intercept)          X1          X2
##   0.9629595  60.0253001 -39.5731694
```

```
svm1
```

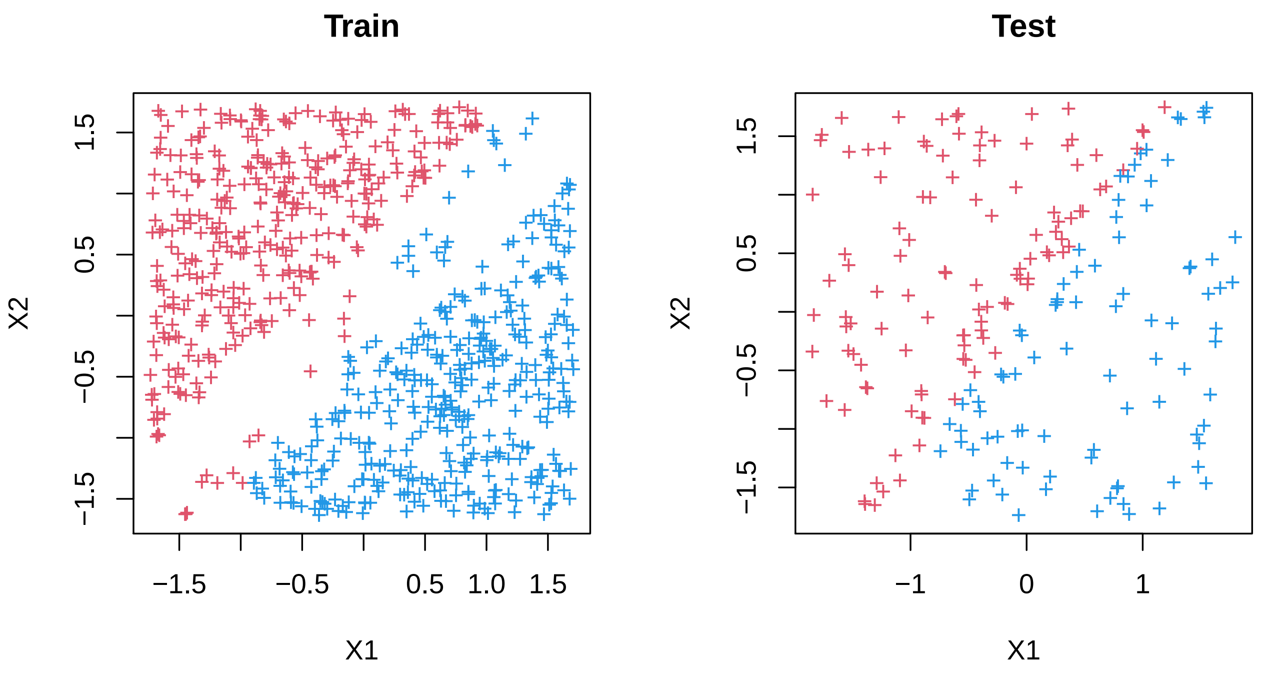
```
##
## Call:
## svm(formula = as.factor(class) ~ X1 + X2, data = train, kernel = "linear",
##     cost = 10000, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10000
##
## Number of Support Vectors:  3
```

There are only 3 out of 729 that have $y_i(\hat{\alpha} + x_i'\hat{\beta})$ less than 1, all of which are 0.9999. This happened because of a combination of rounding. Overall, we see that $y_i(\hat{\alpha} + x_i'\hat{\beta})$ is larger or equal to 1, which means that our line is perfectly separating the two classes

Part 2

Now, let's think about out of sample performance. `simSVMTestF.csv` includes 200 observations that are generated using the same DGP used to generate the training data just with slightly different levels of occurrence for each group (feel free to verify this yourself!).

```
test <- fread("simSVMTestF.csv")
par(mfrow = c(1,2))
plot(train$X1,train$X2, pch = "+", col = as.numeric(train$class) + 3, xlab = "X1", ylab = "X2", main =
plot(test$X1,test$X2, col = as.numeric(test$class) + 3, pch = "+", xlab = "X1", ylab = "X2", main = "Te
```



```r
par(mfrow = c(1,1))
```

Using `cost = 10000`, compute the proportion of observations misclassified in the training set, a 10-fold cross validation measure of the expected proportion of observations misclassified in a test set, and the actual proportion of observations misclassified using the trained SVC to create predictions for the test set. What do you see from these measures? How can it be the case that data generated from the exact same process can yield training errors? Think about the SVC and **overfitting**.

Now repeat this process for 250 different values of `cost` ranging from very small to very large - a reasonable set of cost values can be generated using `exp(seq(-10,10,length = 250))` in R. Make a plot that shows the training error, 10 fold CV estimate, and actual proportion of misclassified points in the test set plotted against the cost values.

There's a decent amount of information to unpack from this graph!

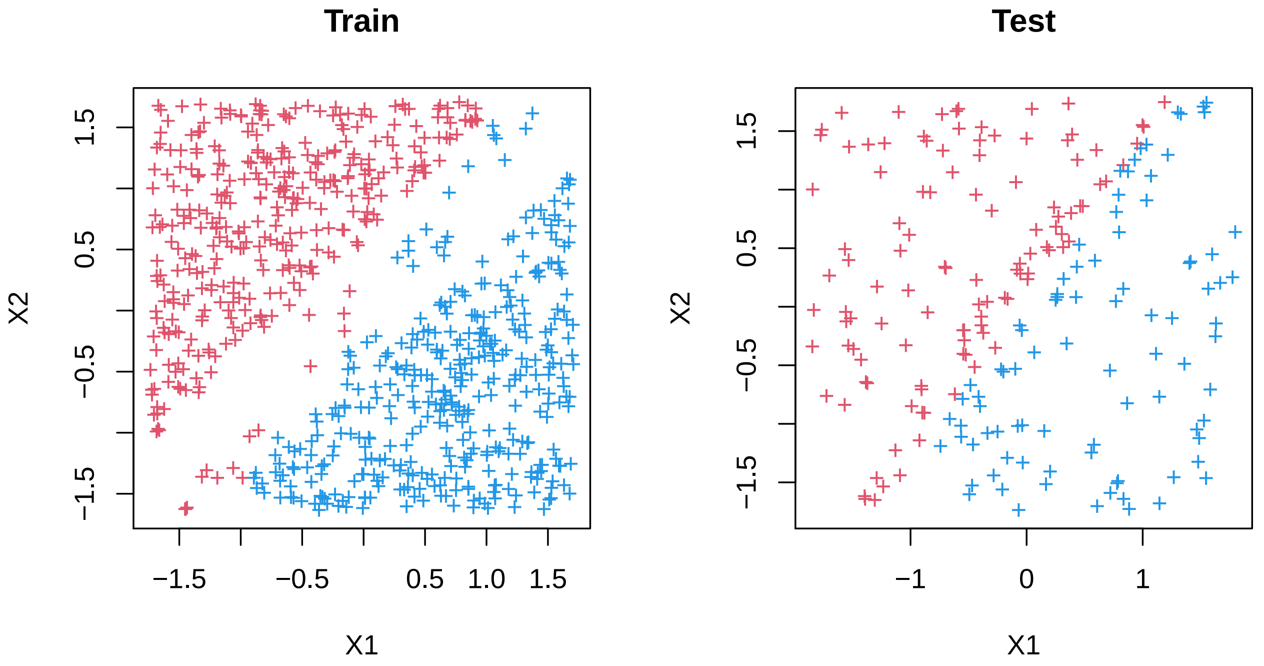
1) What value of cost returns the lowest proportion of misclassified points in the test set? How does this cost value compare to the optimal separating hyperplane with minimal slackness? Provide some reasoning for this result.
2) Does the K-fold CV error estimate more closely correspond to the training error or the test error? In a few sentences, provide some reasoning for this result. Think very carefully about how cross validation would work for the SVC and why we might see this relationship - how does drawing the decision boundary for **supprt vector** classification differ from the other models we've discussed in class?

Note that your findings for this problem are most prevalent when the data are perfectly separable and this relationship tends to disappear as there is more crossover between classes. This may seem like the most common scenario, but an SVM using the radial kernel is **guaranteed* to perfectly separate any training data in the two class problem so long as there are no two points with the same $\boldsymbol{x}_i$ that have different classes! Just something to be wary of and why a combined K-fold and validation set approach is preferable to just using K-fold cross validation!

Solution

```r
test <- read.csv("simSVMTestF.csv")
par(mfrow = c(1,2))
```

```
plot(train$X1,train$X2, pch = "+", col = as.numeric(train$class) + 3, xlab = "X1", ylab = "X2", main =
plot(test$X1,test$X2, col = as.numeric(test$class) + 3, pch = "+", xlab = "X1", ylab = "X2", main = "Te
```



```
par(mfrow = c(1,1))
```

(1) Proportion of observations misclassified in the *training set*

```
svm_train_10000 <- svm(as.factor(class) ~ X1 + X2, data = train, kernel = "linear", cost = 10000, scale

trainX <- train[,c("X1", "X2")]
svm_train_10000.pred <- predict(svm_train_10000, trainX)

print("Misclassificaiton rate")
```

```
## [1] "Misclassificaiton rate"
```

```
mean(svm_train_10000.pred!=train$class)
```

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

10-fold cross validation measure of the expected proportion of observations misclassified in the *test set*

```
misclass_cv_err <- c()
#Randomly shuffle the data
set.seed(123)
df <- cbind(train$class, as.matrix(cbind(train$X1,train$X2)))
rand_df<-data.frame(df[sample(nrow(df)),])
names(rand_df) <- c("class", "X1", "X2")

#Create n folds equally size folds
n_folds <- cut(seq(1,nrow(rand_df)),breaks=10,labels=FALSE)

# perform the k-fold cross validation
for (i in 1:10){
  #Segment the data by fold using the which() function
  index <- which(n_folds == i, arr.ind = T)
  test_data <- rand_df[index,]
  train_data <- rand_df[-index,]

  y_name = names(rand_df)[1]
  x_name = names(rand_df)[-1]

  test_x <- as.matrix(test_data[, x_name])
  test_y <- as.matrix(test_data[, y_name])

  # construct the model and predict the outcome
  svm_test_10000_cv <- svm(as.factor(class) ~ X1 + X2, data = train_data, kernel = "linear", cost = 100
  pred_cv = predict(svm_test_10000_cv, newdata=test_x)

  # get the misclassfied error
  misclass_cv_err[i] <- mean(pred_cv!=test_y)

}
print("Misclassified rate w/ 10 fold on training data")
```

```
## [1] "Misclassified rate w/ 10 fold on training data"
```

```
print(misclass_cv_err)
```

```
##  [1] 0.00000000 0.00000000 0.00000000 0.01369863 0.00000000 0.00000000
##  [7] 0.00000000 0.00000000 0.00000000 0.00000000
```

```
print("Mean misclassified rate")
```

```
## [1] "Mean misclassified rate"
```

```
mean(misclass_cv_err)
```

```
## [1] 0.001369863
```

The actual proportion of obs misclassified using train SVC to create prediciton for the test set

```
testX <- test[,c("X1", "X2")]
svm_train_10000.predTest <- predict(svm_train_10000, testX)

print("Misclassified rate")
```

```
## [1] "Misclassified rate"
```

```
mean(svm_train_10000.predTest!=test$class)
```

```
## [1] 0.055
```

Even though this data is almost separable in the training sample, it may not be in new observations. As a result, while we have nearly perfect prediction rate for the train data as well as the 10-fold, we still misclassfied a small portion of the observations as we do train SVM on the test data.

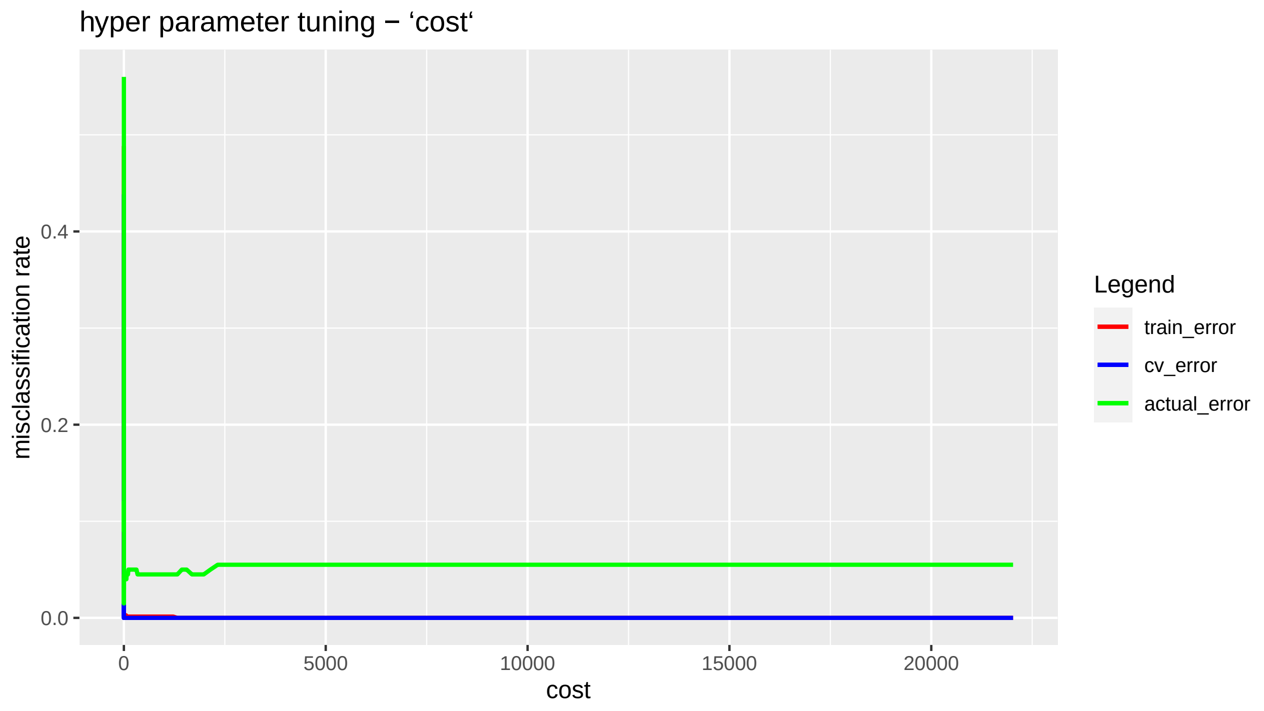
The `cost` parameter allows us to find out the most optimal hyperplane in a non-discrinate data set, as `cost` gets bigger however, the model become more strict about the observation that it intends to classify. The SVM really want to be very certain about its decision which ensure unbiasedness but sacrifice some flexibility when predicting OOS observations.

As a result, a better approach is to find the C which reasonably increase the margin of the SVM model so that ensure accurasy for the OOS sample. (Sacrificing some bias for variance)

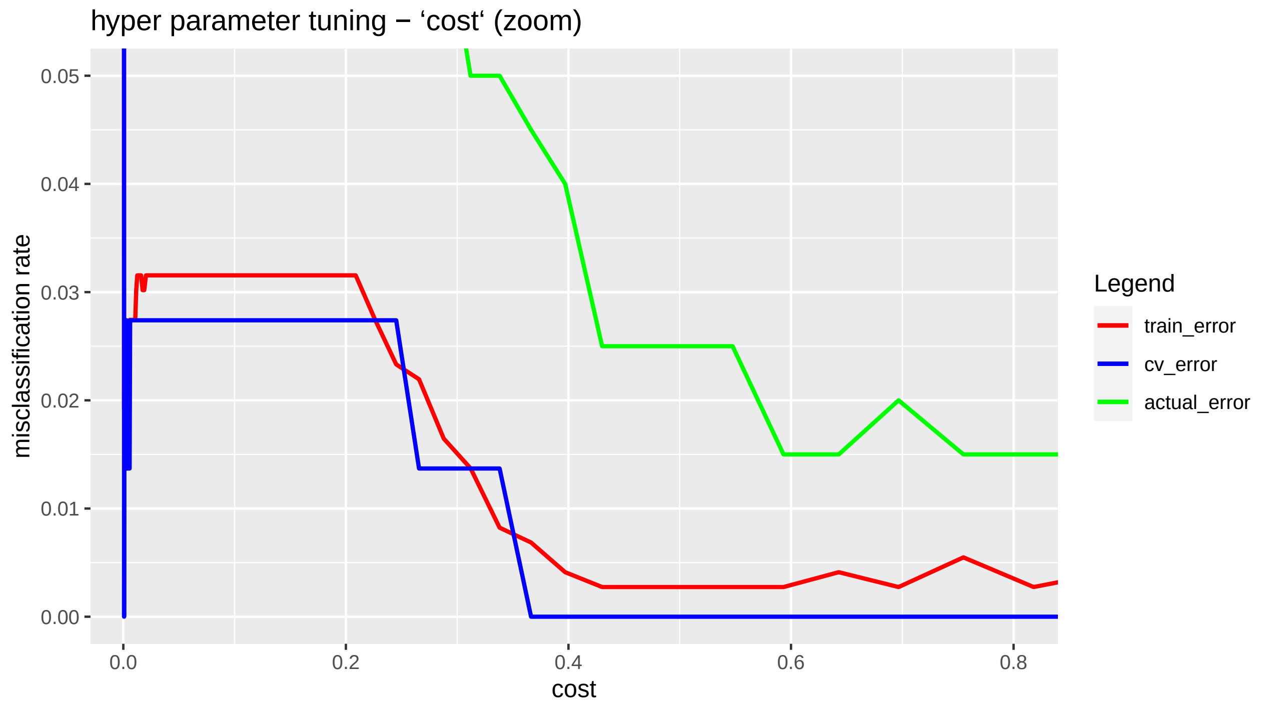
(2)

```r
cost <- exp(seq(-10,10,length = 250))
train_error <- c()
cv_error <- c()
misclass_err <- c()

# training error
for (i in seq(1,250)){

  cost_param <- cost[i]
  svm_train <- svm(as.factor(class) ~ X1 + X2, data = train, kernel = "linear", cost = cost_param, scal
  trainX <- train[,c("X1", "X2")]
  pred <- predict(svm_train, trainX)
  train_error[i] <- mean(pred != train$class)

}
# 10-fold CV
for (i in seq(1,250)){
  set.seed(123)
  df <- cbind(train$class, as.matrix(cbind(train$X1,train$X2)))
  rand_df<-data.frame(df[sample(nrow(df)),])
  names(rand_df) <- c("class", "X1", "X2")

  n_folds <- cut(seq(1,nrow(rand_df)),breaks=10,labels=FALSE)

  for (j in 1:10){
    index <- which(n_folds == j, arr.ind = T)
    test_data <- rand_df[index,]
    train_data <- rand_df[-index,]

    y_name = names(rand_df)[1]
    x_name = names(rand_df)[-1]

    test_x <- as.matrix(test_data[, x_name])
    test_y <- as.matrix(test_data[, y_name])

    cost_param <- cost[i]
    svm_test_10000_cv <- svm(as.factor(class) ~ X1 + X2, data = train_data, kernel = "linear", cost = c
    pred_cv = predict(svm_test_10000_cv, newdata=test_x)

    cv_error[i] <- mean(pred_cv!=test_y)
  }
}

# actual misclassification rate
for (i in seq(1, 250)){

  cost_param <- cost[i]
  svm_train <- svm(as.factor(class) ~ X1 + X2, data = train, kernel = "linear", cost = cost_param, scal
  testX <- test[,c("X1", "X2")]
  pred <- predict(svm_train, testX)
  misclass_err[i] <- mean(pred != test$class)
}
```

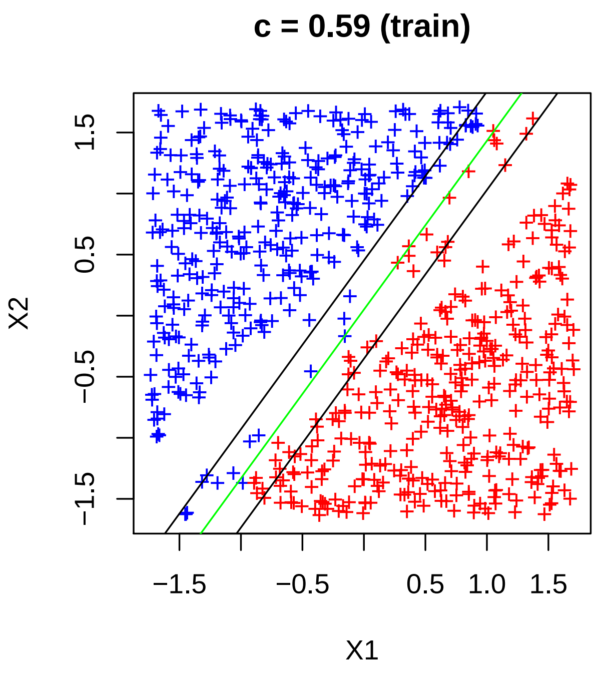
```
library(ggplot2)

error_df <- data.frame("cost" = cost, "train_error" = train_error, "train_error_cv" = cv_error, "actual


colors <- c("train_error" = "red", "cv_error" = "blue",
            "actual_error" = "green")

gg <- ggplot(error_df, aes(x = cost)) +
  geom_line(aes(y = train_error, color = "train_error"), size = 0.9) +
  geom_line(aes(y = train_error_cv, color = "cv_error"), size = 0.9) +
  geom_line(aes(y = actual_error, color = "actual_error"), size = 0.9) +
  labs(
       y="misclassification rate",
       x="cost",
       title="hyper parameter tuning - `cost`",
       color = "Legend"
       ) +
  scale_color_manual(values = colors)

plot(gg)
```

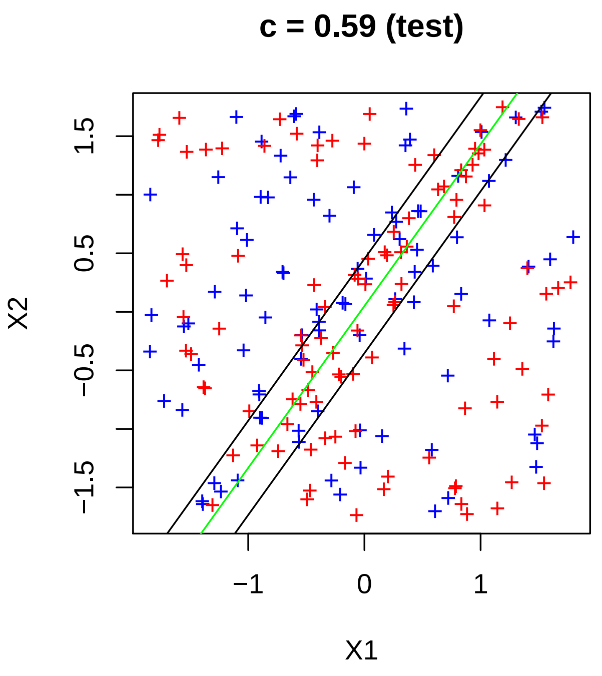


```
gg_zoom <- ggplot(error_df, aes(x = cost)) +
  geom_line(aes(y = train_error, color = "train_error"), size = 0.9) +
  geom_line(aes(y = train_error_cv, color = "cv_error"), size = 0.9) +
  geom_line(aes(y = actual_error, color = "actual_error"), size = 0.9) +
  labs(
       y="misclassification rate",
       x="cost",
       title="hyper parameter tuning - `cost` (zoom)",
        color = "Legend"
        ) +
  scale_color_manual(values = colors) + coord_cartesian(xlim=c(0.01,0.8), ylim=c(0, 0.05))

plot(gg_zoom)
```



```
svm_cv <- svm(as.factor(class) ~ X1 + X2, data = train, kernel = "linear", cost = 0.59, scale = FALSE)
cf1 <- coef(svm_cv)
par(mfrow = c(1, 2))
plot(train$X1, train$X2, col = train$class + 3, type = "n", xlab = "X1",
     ylab = "X2", main = "c = 0.59 (train)")
points(train$X1[train$class == -1], train$X2[train$class ==-1], pch = "+", col = "blue")
points(train$X1[train$class == 1], train$X2[train$class == 1], pch = "+", col = "red")
abline(-cf1[1]/cf1[3], -cf1[2]/cf1[3], col = "green")
abline(-(cf1[1] + 1)/cf1[3], -cf1[2]/cf1[3], col = "black")
abline(-(cf1[1] - 1)/cf1[3], -cf1[2]/cf1[3], col = "black")

plot(test$X1, test$X2, col = test$class + 3, type = "n", xlab = "X1",
     ylab = "X2", main = "c = 0.59 (test)")
points(test$X1[train$class == -1], test$X2[train$class ==-1], pch = "+", col = "blue")
points(test$X1[train$class == 1], test$X2[train$class == 1], pch = "+", col = "red")
abline(-cf1[1]/cf1[3], -cf1[2]/cf1[3], col = "green")
abline(-(cf1[1] + 1)/cf1[3], -cf1[2]/cf1[3], col = "black")
abline(-(cf1[1] - 1)/cf1[3], -cf1[2]/cf1[3], col = "black")
```





1. Judging the graph, it seem like $c = 0.59$ returns the lowest proportion of misclassified point in the test set. Compared to the optimal separating hyperplane with minimal slakcness (part 1), the margin is obviously greater. A greater margin makes our model to be more relaxed, allowing more flexibility when fitting with OOS observations. Hence, we see a better performance of the of OOS classification when introducing some slackness into our model.
2. The K-fold CV however more closely resembles training error when looking at the plot. One reasoning here is the relatively small validation test when we use cross validation to test the model prediction accuracy. Even though we are just doing 10-fold, the validation set is still quite small (73 observations.) Less observations means that the validation sample is more likely to be already perfectly separated and crossovers are less liekly to appear.

As a result, it is highly possible that the train model to draw a perfectly separated line given the validation set when doing CV, as we can see that the `cv_error` achieve a perfect misclassification rate when $c \approx 0.36$, performing even better than the `train_error` That is exactly why it a good practice to have both K-fold and validation set when tuning SVM.

Problem 2: Some Data on Cars

The data set `carsTrain.csv` contains data related to the overall fuel performance for a variety of cars. The main outcome variable is `mpg` - this is coded as 1 if the car gets greater than 20 miles per gallon and is coded as 0 for all other MPGs. There are 8 predictors with the country of origin being a set of two dummy coded variables indicating whether the car was produced in the U.S. or Europe. The variable for Japanese cars was dropped from the data set as a base category.

There is also a test set, `carsTest.csv`, that contains 92 observations that can be used to quantify out-of-sample predictive ability.

Using all available predictors, compute classification models for the `mpg` variable using the training data and compare them in terms of out of sample predictive ability. Use the following classification approaches:

1) Logistic regression
2) QDA
3) Logistic GAM (you don't need to worry about selecting spline interactions unless you really want to)
4) Logistic LASSO
5) Support Vector Classifier w/ Linear Separator

6) Support Vector Machine w/ Polynomial Kernel
7) Support Vector Machine w/ Radial Kernel

For some approaches, there are tuning parameters. Use cross validation approaches to select these tuning parameters and settle on a single final model.

For each classification approach, use your final model to compute 1) the training misclassification rate, 2) a 10-fold CV estimate of the misclassification rate for out of sample data, and 3) the misclassification rate for the held out test set. When appropriate, use the Bayes' classifier to convert probability to a class label. Create a table that summarizes your results.

Which method performs the best? Worst? Provide some intuition for this result.

Some notes:

1) For the Logistic GAM, remember that splines for binary and categorical variables make little sense. Be careful with `cylinders`, in particular.
2) GCV doesn't make much sense for logistic regression models. This is particularly relevant for GAMs, which are optimizing to a specific likelihood. Rather than using this measure, just compute the 10-fold estimate of misclassification rate for the Logistic GAM.
3) If using built in K-fold approaches, make sure that the measure being used to choose a model is the misclassification rate. In `glmnet`, `type.measure` allows you to change the measure that is estimated via cross validation.
4) For support vector methods, always tune `cost` using cross validation. For polynomial kernels, there is an additional parameter - `degree`. This is similar to the degree of a spline. Realistically, we only care about positive integers for this value (though we could consider other values). For the radial kernel, the `gamma` parameter controls the spread of the resulting Gaussian kernelization. This value need only be greater than zero, so try a range of values and zoom in on one that minimizes the CV error.