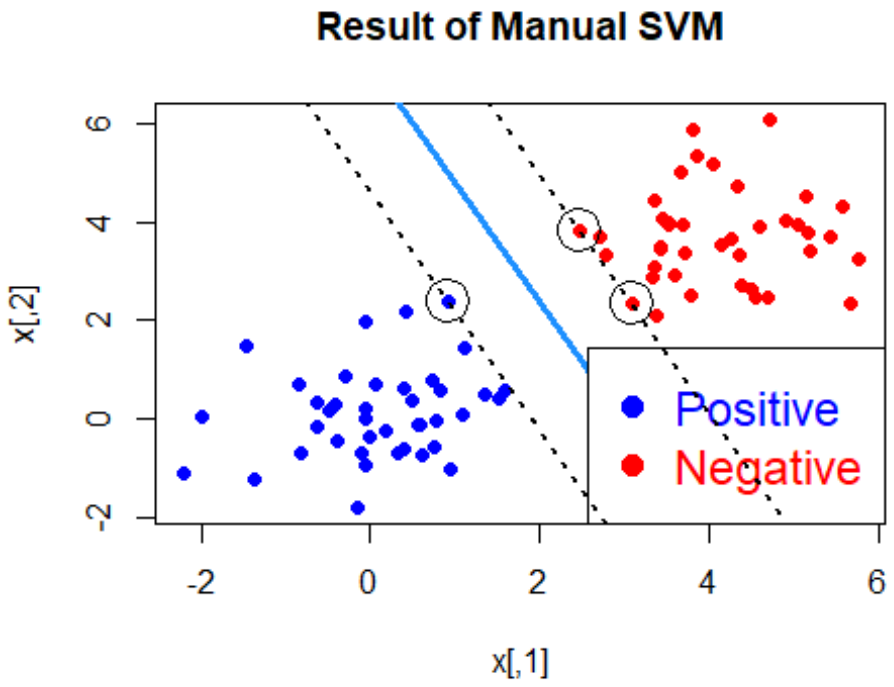
Josh Liu

October 12, 2017

Question 1

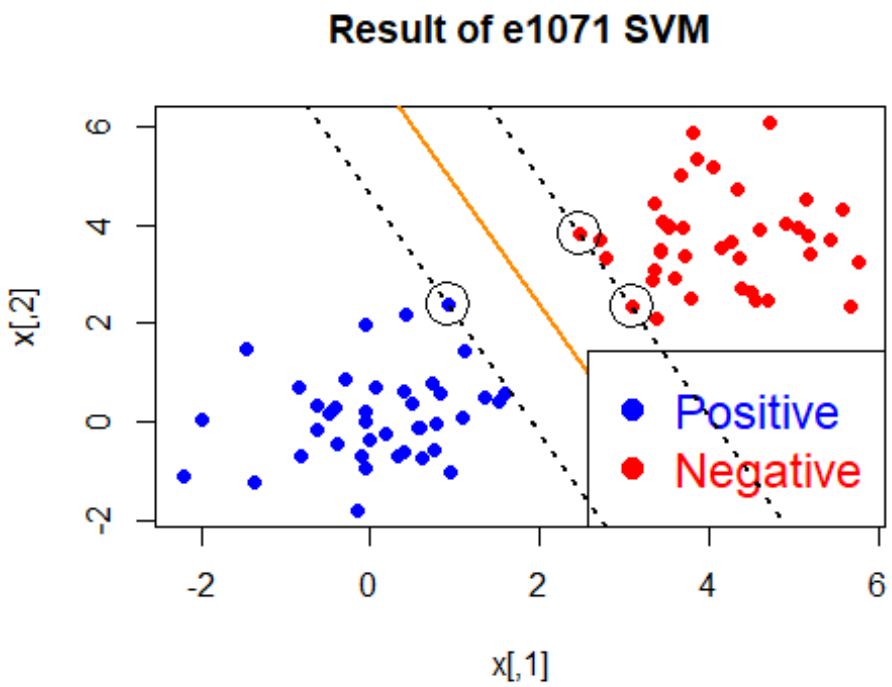
a)

```
set.seed(1)
n <- 40; p = 2
xpos = matrix(rnorm(n*p, mean = 0, sd = 1), n, p)
xneg = matrix(rnorm(n*p, mean = 4, sd = 1), n, p)
x = rbind(xpos, xneg)
Y = matrix(c(rep(1, n), rep(-1, n)))
X = cbind(x, rep(1, n*2))
D = diag(1,3)
D[3, 3] = .00000011 # We want to minimize the norm of the first two elements
of Beta. However, because D is required to be positive definite, we make the
3rd diagonol element to be small but not 0.
G = diag(1, 80)
G[41:80,] = -1 * G[41:80,] # Let G be a 80 by 80 diagonal matrix, the first
40 diagonol elements being 1, and the last 40 diagonol elements being -1.
A = t(G %*% X)
b = rep(1, 80)
d = as.vector(rep(0, 3))
result = solve.QP(Dmat = D, dvec = d, Amat = A, bvec = b, meq=0,
factorized=FALSE)
beta = result$solution
supportV = result$iact
beta

## [1] -0.9334294 -0.3849838  2.7823844

plot(x, col = Y+3, pch = 16, main = "Result of Manual SVM")
abline(beta[3]/-beta[2], -beta[1] / beta[2], col = "dodgerblue", lwd = 3)
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
abline(a = (beta[3] + 1)/-beta[2], b = -beta[1] / beta[2], col="black",
lty=3, lwd = 2)
abline(a = (beta[3] - 1)/-beta[2], b = -beta[1] / beta[2], col="black",
lty=3, lwd = 2)
points(x[supportV, ], col="black", cex=3)
```



Now, let's take a look at the `e1071` solution.

```
svm.fit <- svm(Y ~ x, type='C-classification', kernel='linear',scale=FALSE,
cost = 10)
w <- t(svm.fit$coefs) %*% svm.fit$SV
b <- -svm.fit$rho
# b <- -(max(x[y == -1, ] %*% t(w)) + min(x[y == 1, ] %*% t(w)))/2
plot(x, col = Y+3, pch = 16, main = "Result of e1071 SVM")
abline(a= -b/w[1,2], b=-w[1,1]/w[1,2], col="darkorange", lty=1, lwd = 2)
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
abline(a= (-b-1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
abline(a= (-b+1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
points(x[svm.fit$index, ], col="black", cex=3)
```



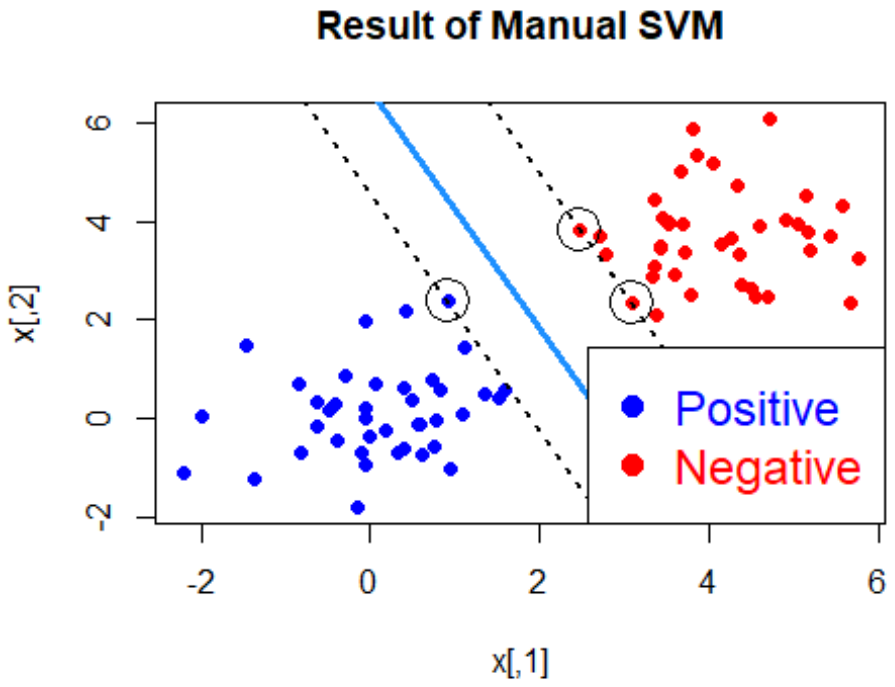

```
result1a = (rbind("Manual fit" = beta, "e1071 fit" = c(w, b)))
colnames(result1a) = c("beta_1", "beta_2", "beta_0")
kable(result1a)
```

	beta_1	beta_2	beta_0
Manual fit	-0.9334294	-0.3849838	2.782384
e1071 fit	-0.9338740	-0.3844957	2.781953

b)

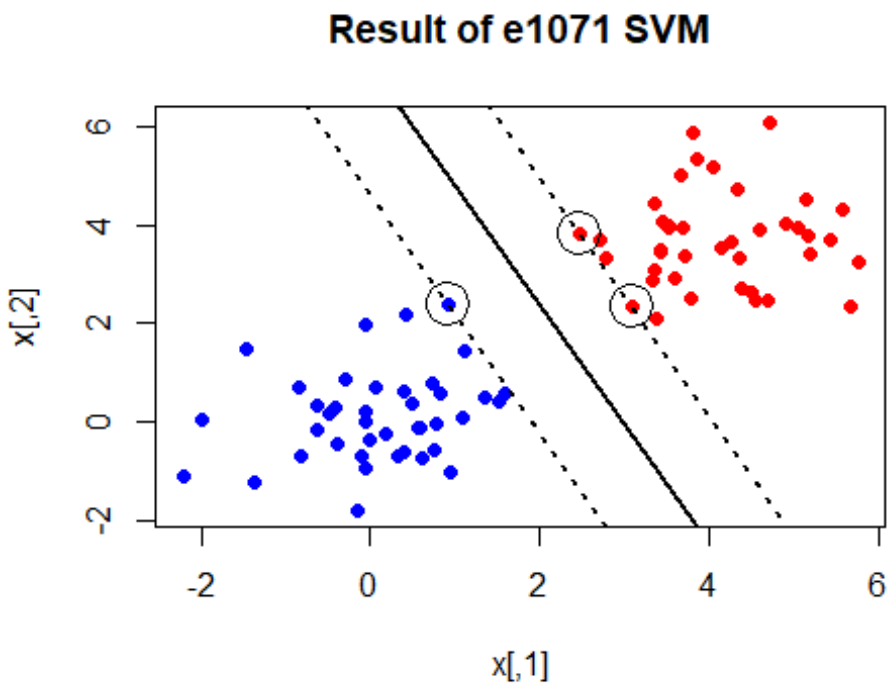
```
One = rep(1, 80)
G = diag(1, 80)
diag(G) = Y
D =  t(G) %*% x %*% t(x) %*% G
D = D + .00001 * diag(1, dim(D)[1])
d = One
A =t( rbind(t(Y), diag(1, 2 * n)) )
b = rep(0, 81)
result_1b = solve.QP(Dmat = D, dvec = d, Amat = A, bvec = b, meq = 1,
factorized=FALSE)
alpha = result_1b$solution
supportV1b = seq(1,80)[!seq(1, 80) %in% result_1b$iact]
beta = as.vector(t(alpha) %*% t(G) %*% x)
beta0 = mean(G %*% x %*% beta)
beta = c(beta, beta0)
plot(x, col = Y+3, pch = 16, main = "Result of Manual SVM")
```

```
abline(beta[3]/-beta[2], -beta[1] / beta[2], col = "dodgerblue", lwd = 3)
abline(a = (beta[3] + 1.23)/-beta[2], b = -beta[1] / beta[2], col="black",
lty=3, lwd = 2)
abline(a = (beta[3] - .8)/-beta[2], b = -beta[1] / beta[2], col="black",
lty=3, lwd = 2)
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
points(x[supportV, ], col="black", cex=3)
```



Now, again, let's take a look at the `e1071` solution.

```
svm.fit <- svm(Y ~ x, type='C-classification', kernel='linear',scale=FALSE,
cost = 1)
w <- t(svm.fit$coefs) %*% svm.fit$SV
b <- -svm.fit$rho
plot(x, col = Y+3, pch = 16, main = "Result of e1071 SVM")
abline(a= -b/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=1, lwd = 2)
abline(a= (-b-1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
abline(a= (-b+1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
points(x[svm.fit$index, ], col="black", cex=3)
```




```
result1b = (rbind("Manual fit" = beta, "e1071 fit" = c(w, b)))
colnames(result1b) = c("beta_1", "beta_2", "beta_0")
kable(result1b)
```

	beta_1	beta_2	beta_0
Manual fit	-0.933426	-0.3849820	2.565499
e1071 fit	-0.933874	-0.3844957	2.781953

(c)

In this part, I will use the same X matrix, but different Y matrix.

Probability mass funciton for the first 40 elements are:

$$f(1)=0.9\\ f(-1)=0.1 $$

.

Pmf for the last 40 elements is

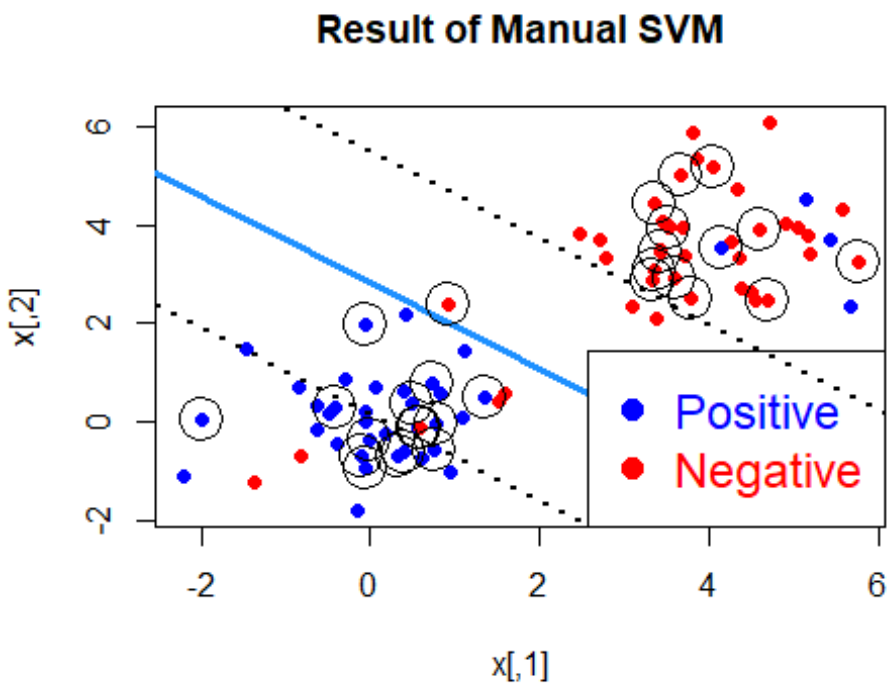
$$f(1)=0.1\\ f(-1)=0.9 $$

.

All elements in Y are assumed to be independent.

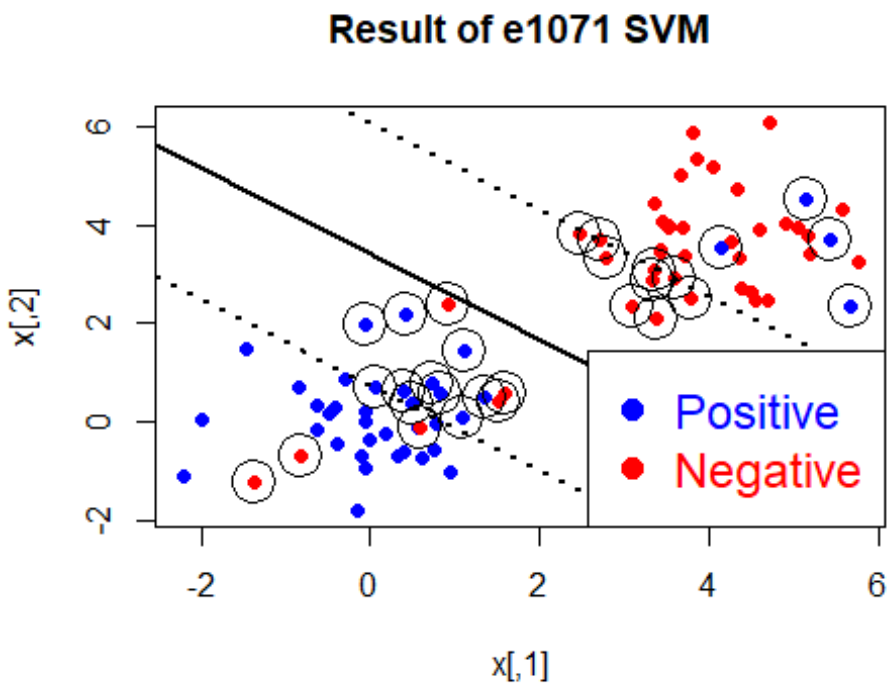
Code in this part is improved based on code in 1(b). For documentation, please see handwritten proof.

```
Y = as.vector(rep(0, 80))
Y[1:40] = rbinom(40, 1, .9)
Y[41:80] = rbinom(40, 1, .1)
Y = ifelse(Y == 0, -1, 1)
One = rep(1, 80)
G = diag(1, 80)
diag(G) = Y
D = t(G) %*% x %*% t(x) %*% G
D = D + .00001 * diag(1, dim(D)[1])
d = One
A =t( rbind(t(Y), diag(1, 80), diag(-1, 80)) )
c = 1
b = c(rep(0, 81), rep(-c, 80))
result_1c = solve.QP(Dmat = D, dvec = d, Amat = A, bvec = b, meq =1,
factorized = FALSE)
alpha_1c = result_1c$solution
supportV_1c = seq(1,80)[!seq(1, 80) %in% result_1c$iact]
beta = as.vector(t(alpha_1c) %*% G %*% x)
beta0 = mean(G %*% x %*% beta)
beta = c(beta, beta0)
plot(x, col = Y+3, pch = 16, main = "Result of Manual SVM")
abline(beta[3]/-beta[2], -beta[1] / beta[2], col = "dodgerblue", lwd = 3)
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
abline(a= (beta[3]-1)/-beta[2],-beta[1] / beta[2], col="black", lty=3, lwd =
2)
abline(a= (beta[3]+1)/-beta[2],-beta[1] / beta[2], col="black", lty=3, lwd =
2)
points(x[supportV_1c, ], col="black", cex=3)
```



Now, again, let's take a look at the `e1071` solution.

```
svm.fit <- svm(Y ~ x, type='C-classification', kernel='linear',scale=FALSE,
cost = 1)
w <- t(svm.fit$coefs) %*% svm.fit$SV
b <- -svm.fit$rho
plot(x, col = Y+3, pch = 16, main = "Result of e1071 SVM")
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
abline(a= -b/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=1, lwd = 2)
abline(a= (-b-1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
abline(a= (-b+1)/w[1,2], b=-w[1,1]/w[1,2], col="black", lty=3, lwd = 2)
points(x[svm.fit$index, ], col="black", cex=3)
legend("bottomright", c("Positive","Negative"),col=c( "blue",
"red"),pch=c(19, 19),text.col=c("blue", "red"), cex = 1.5)
```




```
result1c = (rbind("Manual fit" = beta, "e1071 fit" = c(w, b)))
colnames(result1c) = c("beta_1", "beta_2", "beta_0")
kable(result1c)
```

	beta_1	beta_2	beta_0
Manual fit	-0.3287252	-0.3752557	1.062682
e1071 fit	-0.3286521	-0.3753764	1.283473

(d)

```
library(ElemStatLearn)
data(SAheart)
x_1d = SAheart[, 1:9]
x_1d$famhist = as.numeric(x_1d$famhist)
x_1d = as.matrix(x_1d)
Y_1d = SAheart[, c("chd")]
Y_1d = ifelse(Y_1d == 0, -1, 1)
n_1d = dim(SAheart)[1]
```

Trying different tuning parameters, and testing each value with prediction accuracy with 10-fold cross validation.

Note that we have 462 observations. Because we will do 10-fold cross validation, two observations will not be used.

```
allCosts = seq(.0001, 1, length.out = 20)
x_1d = x_1d[1: 460, ]
x_1d = scale(x_1d)
Y_1d = Y_1d[1: 460]
n = 460 * 9 / 10
One = as.vector(rep(1, n))
results = matrix(rep(NA, length(allCosts) * 10), nrow = 10)
# This matrix is used to store result. Each row is a 'fold' in the original
dataset. Each column is a Cost value.
for (i in 1:10) {
    training.x = x_1d[-seq(46 * (i - 1) + 1, 46 * i), ]
    training.Y = Y_1d[-seq(46 * (i - 1) + 1, 46 * i)]
    test.x = x_1d[seq(46 * (i - 1) + 1, 46 * i), ]
    test.Y = Y_1d[seq(46 * (i - 1) + 1, 46 * i)]
    G = diag(1, n)
    diag(G) = training.Y
    D = t(G) %*% training.x %*% t(training.x) %*% G
    D = D + .0001 * diag(1, dim(D)[1])
    d = One
    A =t( rbind(t(training.Y), diag(1, n), diag(-1, n)))
    for (j in 1 : length(allCosts)){
      cost = allCosts[j]
      b = c(rep(0, n + 1), rep(-cost, n))
      alpha = solve.QP(Dmat = D, dvec = d, Amat = A, bvec = b, meq = 1,
factorized=FALSE)$solution
      beta = as.vector(t(alpha) %*% G %*% training.x)
      beta0 = mean(G %*% training.x %*% beta)
      linearpredictor = test.x %*% beta + beta0
      testresult = ifelse(test.Y * linearpredictor > 0, 1, 0)
      results[i, j] = mean(testresult)
}

}
accuracyByCost = apply(results, 2, mean)
accuracyByCost

##  [1] 0.6086957 0.5891304 0.5913043 0.5978261 0.6000000 0.5978261 0.5978261
0.5978261 0.5934783 0.5913043 0.5913043 0.5913043 0.5913043 0.5913043
0.5913043 0.5913043 0.5913043 0.5913043 0.5913043 0.5913043

which.max(accuracyByCost)

## [1] 1

allCosts[which.max(accuracyByCost)]

## [1] 1e-04
```

And then, I will use the optimal `cost` value to fit a SVM on the whole dataset.

```
n_1d = length(Y_1d)
One_1d = as.vector(rep(1, n_1d))
H_1d = diag(1, n_1d - 1)
H_1d = rbind(H_1d, - Y_1d[1 : n_1d - 1] / Y_1d[n_1d])
G_1d = diag(1, n_1d)
diag(G_1d) = Y_1d
D_1d = t(H_1d) %*% t(G_1d) %*% x_1d %*% t(x_1d) %*% G_1d %*% H_1d
D_1d = D_1d + .00001 * diag(1, dim(D_1d)[1])
d_1d = t(One_1d) %*% H_1d
J_1d = rbind(diag(1, n_1d), diag(-1, n_1d))
A_1d = t(J_1d %*% H_1d)
c = .0001
b_1d = c(rep(0, n_1d), rep(-c, n_1d))
result_1d = solve.QP(Dmat = D_1d, dvec = d_1d, Amat = A_1d, bvec = b_1d, meq
= 0, factorized= TRUE )
alpha_star_1d = result_1d$solution
alpha_1d = H_1d %*% alpha_star_1d
beta_1d = as.vector(t(alpha_1d) %*% G_1d %*% x_1d)
beta0_1d = mean(G_1d %*% x_1d %*% beta_1d)
linearpredictor_1d = x_1d %*% beta_1d + beta0_1d
mean(Y_1d * linearpredictor_1d > 0)
```

What about automatic fitting?

```
svm1d.fit <- svm(Y_1d ~ x_1d, type='C-classification',
kernel='linear',scale=FALSE, cost = .0001)
autofitaccuracy = mean(predict(svm1d.fit, x_1d) == Y_1d)
autofitaccuracy
```

```
## [1] 0.6543478
```

Question 3

```
shooting = read_csv('massShooting.csv')
# head(massShooting)
#Cleaning race:
shooting$Race = gsub(".*Black.*", ignore.case = TRUE, "Black", shooting$Race)
shooting$Race = gsub(".*White.*", ignore.case = TRUE, "White", shooting$Race)
shooting$Race = gsub(".*Asian.*", ignore.case = TRUE, "Asian", shooting$Race)
shooting$Race = gsub(".*other.*", ignore.case = TRUE, NA, shooting$Race)
shooting$Race = gsub(".*unclear.*", ignore.case = TRUE, NA, shooting$Race)
shooting$Race = gsub(".*native.*", ignore.case = TRUE, "Native",
shooting$Race)
shooting$Race = gsub(".*more*", ignore.case = TRUE, NA, shooting$Race)
shooting$Race = gsub(".*unknown.*", ignore.case = TRUE, NA, shooting$Race)
shooting$Race = ifelse(shooting$Race == "", NA, shooting$Race)
unique(shooting$Race)
```

```
## [1] NA       "Asian"  "White"  "Black"  "Latino" "Native"
```

```
#Cleaning `Mental.Health.Issues`:
colnames(shooting)[9] = 'Mental.Health.Issues'
shooting$Mental.Health.Issues = gsub(".*unclear.*", ignore.case = TRUE, NA,
shooting$Mental.Health.Issues)
shooting$Mental.Health.Issues = gsub(".*unknown*", ignore.case = TRUE, NA,
shooting$Mental.Health.Issues)
unique(shooting$Mental.Health.Issues)
```

```
## [1] NA    "Yes" "No"
```

```
#Cleaning `Gender`:
shooting$Gender = gsub(".*M/F.*", ignore.case = TRUE, NA, shooting$Gender)
shooting$Gender = gsub(".*Unknown.*", ignore.case = TRUE, NA,
shooting$Gender)
shooting$Gender = gsub(".*Male/Female.*", ignore.case = TRUE, NA,
shooting$Gender)
shooting$Gender = ifelse(shooting$Gender == "", NA, shooting$Gender)
shooting$Gender = ifelse(shooting$Gender == "M", "Male", shooting$Gender)
unique(shooting$Gender)
```

```
## [1] NA       "Male"   "Female"
```

```
# Processing age:
age = rep(NA, dim(shooting)[1])
ageindex1 = grep("[[:digit:]]+[ -]year", shooting$Summary)
agegroup1 = shooting$Summary[ageindex1]
group1Ages = str_extract(agegroup1, "[[:digit:]]+[ -]year")
group1Ages = as.numeric(gsub("\\D", "", group1Ages))
age[ageindex1] = group1Ages
ageindex2 = grep(", [[:digit:]]{2}, ", shooting$Summary)
agegroup2 = shooting$Summary[ageindex2]
group2Ages = str_extract(agegroup2, ", [[:digit:]]{2}, ")
group2Ages = as.numeric(gsub("\\D", "", group2Ages))
age[ageindex2] = group2Ages
shooting = cbind(shooting, age)
shooting[["Summary"]] = NULL # Summary column is removed.
```

In this part, I will construct a model to perform Support Vector Machine regression on the `Victims` of the shooter.

Because `Victim` is a linear combo of `Injured` and `Fatalities`, I will remove these two variables from the dataset.

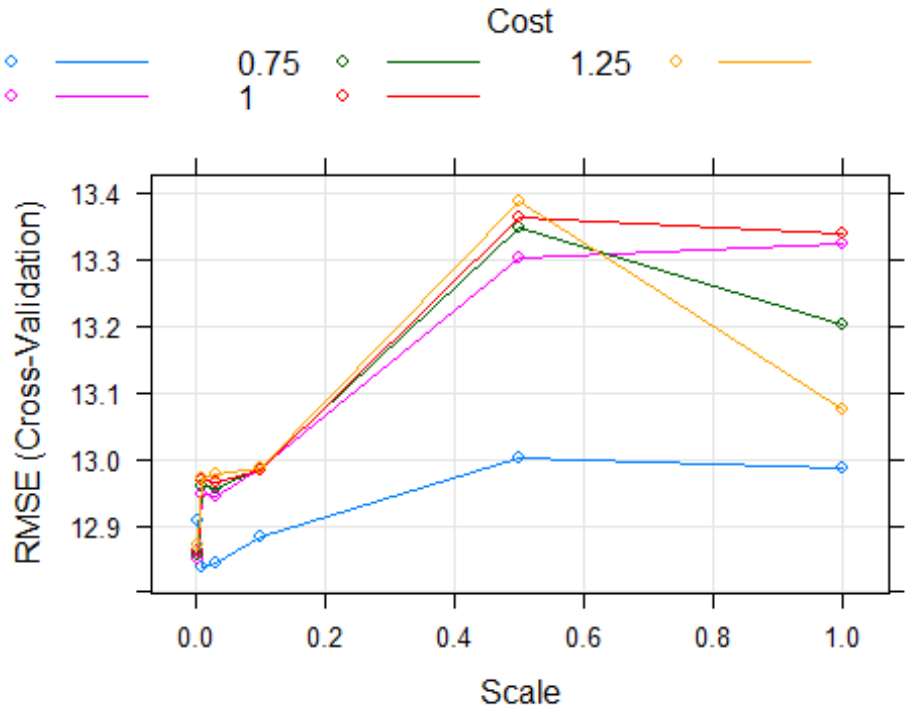
First, I will perform linear regression:

```
shooting$Title = NULL
shooting$Location = NULL
shooting$Date = NULL
shooting$Latitude = NULL
shooting$Longitude = NULL
shooting$Fatalities = NULL
```

```
shooting$Injured = NULL
shooting$'S#' = NULL
shooting = shooting[complete.cases(shooting),]
colnames(shooting)[5] = 'Victims'
```

Then I will do support vector machine regression with `Victim` as response variable, and with `Mental.Health.Issues`, `Race`, `Gender`, age as predictor variables.

```
# First, I am going to split the whole data into training and testing
dataset.
train_index = sample(nrow(shooting), 160)
train = shooting[train_index, ]
test = shooting[-train_index, ]
# Now, I will use different C values and sigma values for the radial support
vector machine.
svm_grid = expand.grid(   C = c(2 ^ (-5:5)),
                       sigma = c(2 ^ (-10:-5)))
# I will use the caret package to do 5-fold cross validation.
svm_control = trainControl(method = "cv", number = 5, returnResamp = "all",
verbose = FALSE)
rad_svm_fit = train(Victims ~ ., data = train, method = "svmRadial",
trControl = svm_control, tuneGrid = svm_grid, scale = FALSE)
# And then, I use different Cost values for the linear support vector machine
svm_grid =  expand.grid(C = c(2 ^ (-10: 1)))
lin_svm_fit = train(Victims ~ ., data = train, method = "svmLinear",
trControl = svm_control, tuneGrid = svm_grid, scale = FALSE)
# Tuning grid for Poly SVM
grid_poly = expand.grid(C = c(.01, .5, .75, 1, 1.25),  scale = c(0.001, 0.01,
.03, .1, .5, 1), degree = 2)
poly_svm_fit = train(Victims ~ ., data = train, method = "svmPoly", tuneGrid
= grid_poly, trControl = svm_control,scale = FALSE)
#Calculating Test, Train and CV root mean square of error:
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
test_rmse_rad = rmse(actual = test$Victims, predicted = predict(rad_svm_fit,
newdata=test))
test_rmse_lin = rmse(actual = test$Fatalities, predicted =
predict(lin_svm_fit, newdata=test))
test_rmse_poly = rmse(actual = test$Fatalities, predicted =
predict(poly_svm_fit, newdata=test))
plot(poly_svm_fit)
```



The plot shows that in our poly SVM, our model achieved minimum RMSE with Cost value being 1, and Scale being 1. The minimum RMSE is 12.84

I am going to compare my model with linear regression.

```
fit3 = lm(Victims ~ ., data = shooting)
mean((shooting$Victims - predict(fit3)) ^ 2)

## [1] 158.7598
```

Linear regression returned a much higher RMSE. In conclusion, the poly SVM works better in this case. It is probably because `Victim` as a counting variable is not a good candidate as a response variable in linear regression, and as a result, linear regression RMSE is greatly increased because of its rigidness. But in SVM regression, the fitted space is more flexible than that in the linear subspace produced by linear regression. So we get a higher RMSE from linear regression.