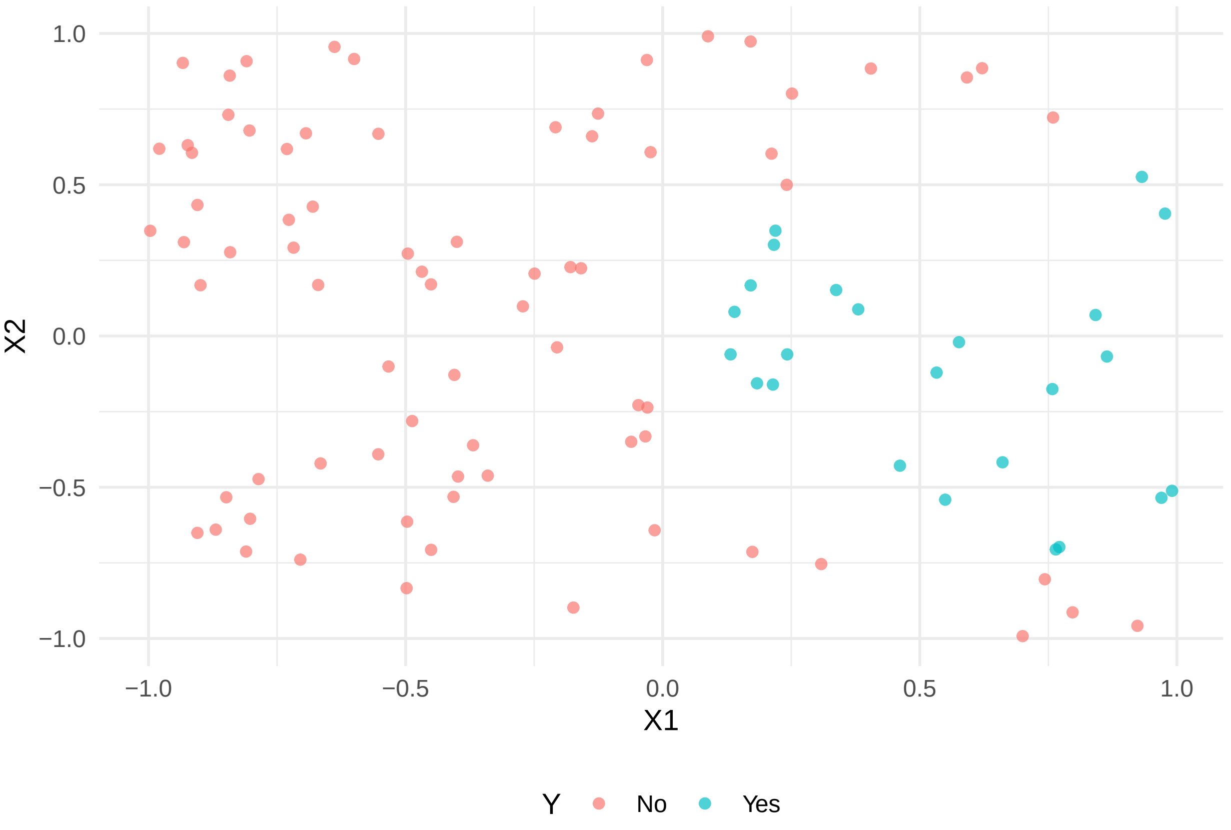
Homework 6: Support Vector Machines

Wanitchaya Poonpatanapricha

Conceptual exercises

Non-linear separation

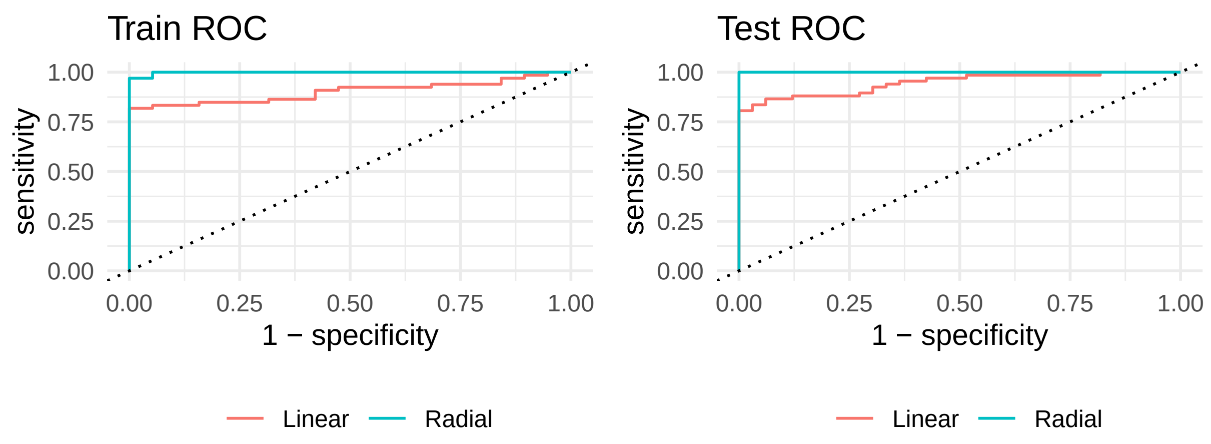
```r
# simulated data
X1 <- runif(100, -1, 1)
X2 <- runif(100, -1, 1)
Y <- X1 - X2^2 - X2^4 > 0
train <- data.frame(X1 = X1, X2 = X2, Y = Y) %>% mutate(Y = factor(Y, levels = c(FALSE,
    TRUE), labels = c("No", "Yes")))
train %>% ggplot(aes(X1, X2, color = Y)) + geom_point(alpha = 0.7) + theme(legend.position = "bottom")
```

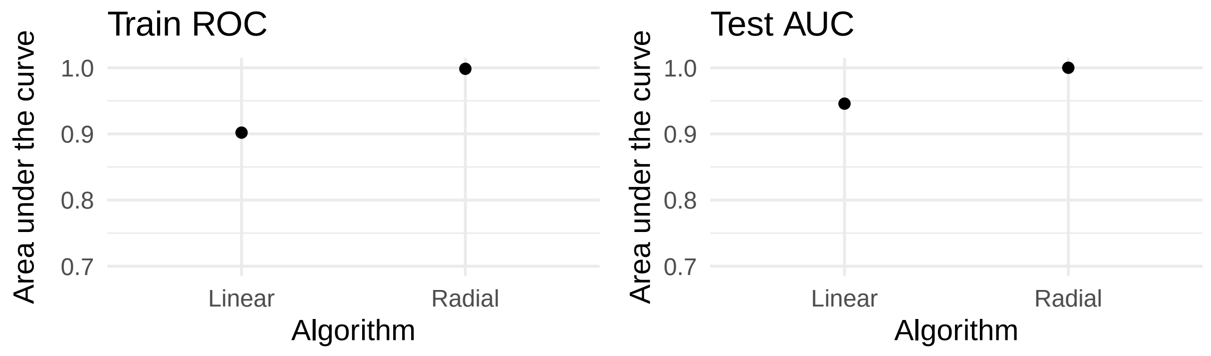


```r
# trainControl
cv_ctrl <- trainControl(method = "cv", number = 10, savePredictions = "final", classProbs = TRUE)

# linear svm
svm_linear <- train(Y ~ ., data = train, method = "svmLinear", trControl = cv_ctrl,
    tuneLength = 10)

# radial svm
svm_radial <- train(Y ~ ., data = train, method = "svmRadial", trControl = cv_ctrl)
```

```r
# Train

# ROC
p1 <- bind_rows(Linear = svm_linear$pred, Radial = svm_radial$pred, .id = "kernel") %>%
    group_by(kernel) %>% roc_curve(truth = obs, estimate = Yes) %>% ggplot(aes(x = 1 -
    specificity, y = sensitivity, color = kernel)) + geom_path() + geom_abline(lty = 3) +
    labs(color = NULL, title = "Train ROC") + theme(legend.position = "bottom")

# AUC
p2 <- bind_rows(Linear = svm_linear$pred, Radial = svm_radial$pred, .id = "kernel") %>%
    group_by(kernel) %>% roc_auc(truth = obs, Yes) %>% group_by(kernel) %>% summarize(.estimate = mean(
    ggplot(aes(fct_reorder(kernel, .estimate), .estimate)) + geom_point() + ylim(0.7,
    1) + labs(title = "Train ROC", x = "Algorithm", y = "Area under the curve")
```

```r
X1 <- runif(100, -1, 1)
X2 <- runif(100, -1, 1)
Y <- X1 - X2^2 - X2^4 > 0
test <- data.frame(X1 = X1, X2 = X2, Y = Y) %>% mutate(Y = factor(Y, levels = c(FALSE,
    TRUE), labels = c("No", "Yes")))

# Test
lin <- data.frame(Yes = predict(svm_linear, test, type = "prob")$Yes, obs = test$Y)
radial <- data.frame(Yes = predict(svm_radial, test, type = "prob")$Yes, obs = test$Y)
# ROC
p3 <- bind_rows(Linear = lin, Radial = radial, .id = "kernel") %>% group_by(kernel) %>%
    roc_curve(truth = obs, estimate = Yes) %>% ggplot(aes(x = 1 - specificity, y = sensitivity,
    color = kernel)) + geom_path() + geom_abline(lty = 3) + labs(color = NULL, title = "Test ROC") +
    theme(legend.position = "bottom")

# AUC
p4 <- bind_rows(Linear = lin, Radial = radial, .id = "kernel") %>% group_by(kernel) %>%
    roc_auc(truth = obs, Yes) %>% group_by(kernel) %>% summarize(.estimate = mean(.estimate)) %>%
    ggplot(aes(fct_reorder(kernel, .estimate), .estimate)) + geom_point() + ylim(0.7,
    1) + labs(title = "Test AUC", x = "Algorithm", y = "Area under the curve")

(p1 + p3)/(p2 + p4)
```





```
data.frame(Kernel = c("Linear", "Radial"), `CV error` = c(1 - max(svm_linear$results$Accuracy),
    1 - max(svm_radial$results$Accuracy)), `Test error` = c(mean(predict(svm_linear,
    test) != test$Y), mean(predict(svm_radial, test) != test$Y))) %>% kable()
```

Kernel	CV.error	Test.error
Linear	0.1778	0.2667
Radial	0.0111	0.0000

Based on CV/test error rates and ROC/AUCs, we can see that in both train and test sessions, radial SVM performs better than linear SVM. This is expected because the underlying pattern in the data is non-linear.

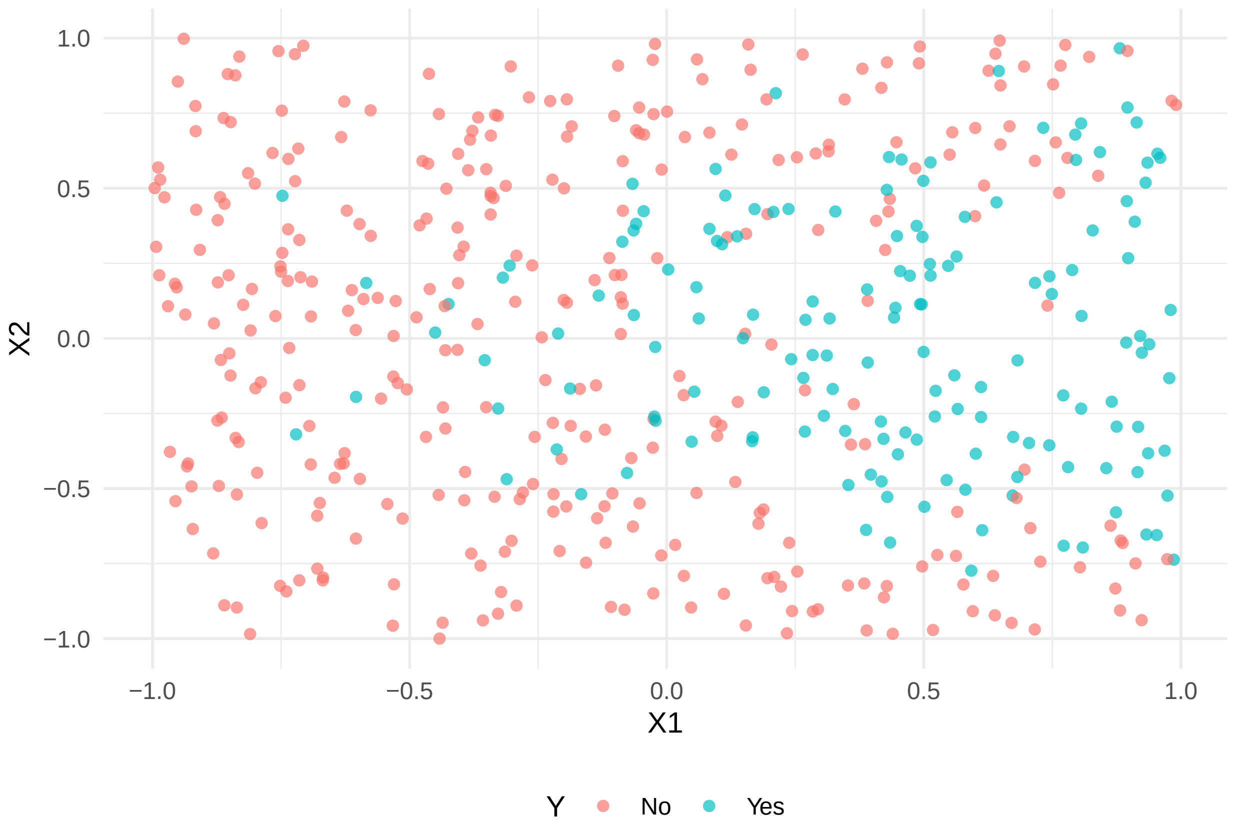
SVM vs. logistic regression

(2)

```
# simulated data
X1 <- runif(500, -1, 1)
X2 <- runif(500, -1, 1)
Y <- X1 - X2^2 - X2^4 > rnorm(500, 0, 0.5)
df <- data.frame(X1 = X1, X2 = X2, Y = Y) %>% mutate(Y = factor(Y, levels = c(FALSE,
    TRUE), labels = c("No", "Yes")))
```

(3)

```
df %>% ggplot(aes(X1, X2, color = Y)) + geom_point(alpha = 0.7) + theme(legend.position = "bottom")
```



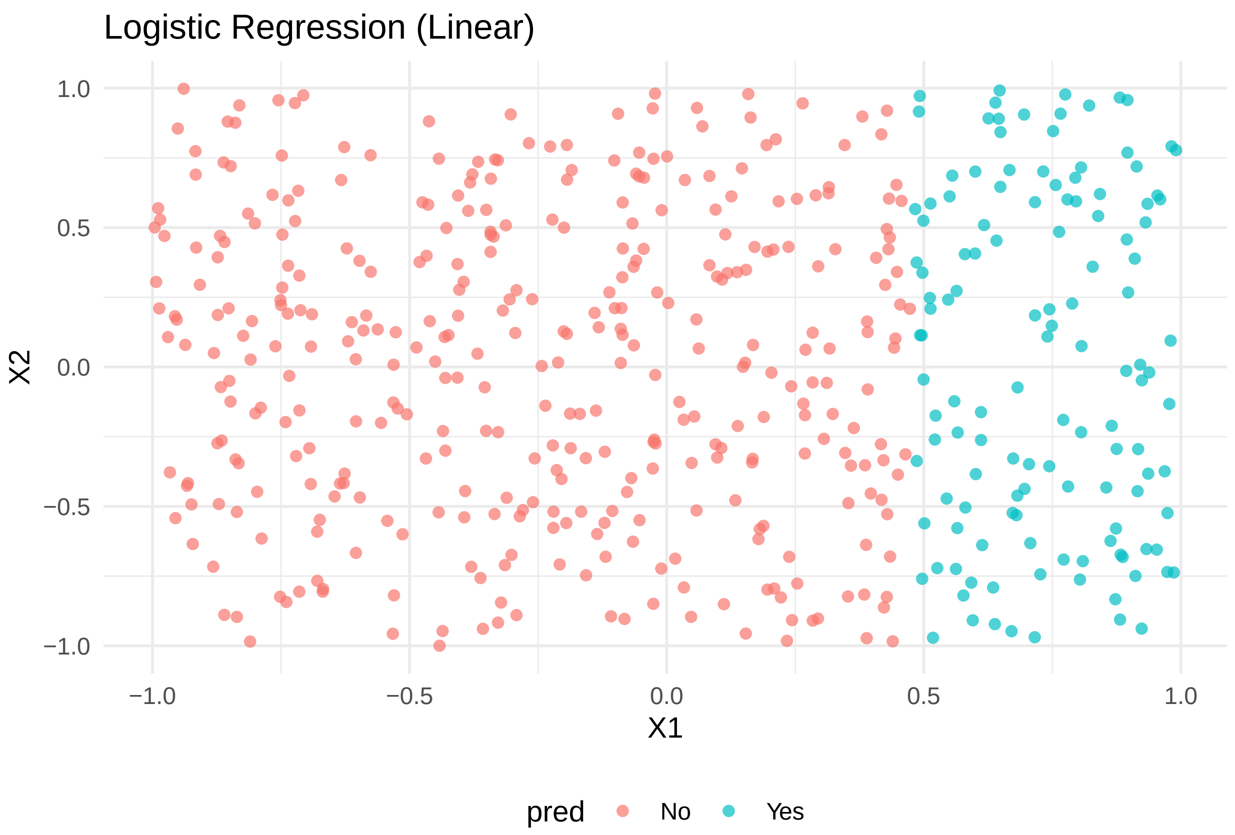
(4)

```
logit <- train(Y ~ ., data = df, method = "glm")
logit
```

```
## Generalized Linear Model
##
## 500 samples
##   2 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 500, 500, 500, 500, 500, 500, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.7345    0.3514
```

(5)

```
df$pred <- predict(logit, df)

(p1 <- df %>% ggplot(aes(X1, X2, color = pred)) + geom_point(alpha = 0.7) + ggtitle("Logistic Regressio
    theme(legend.position = "bottom"))
```



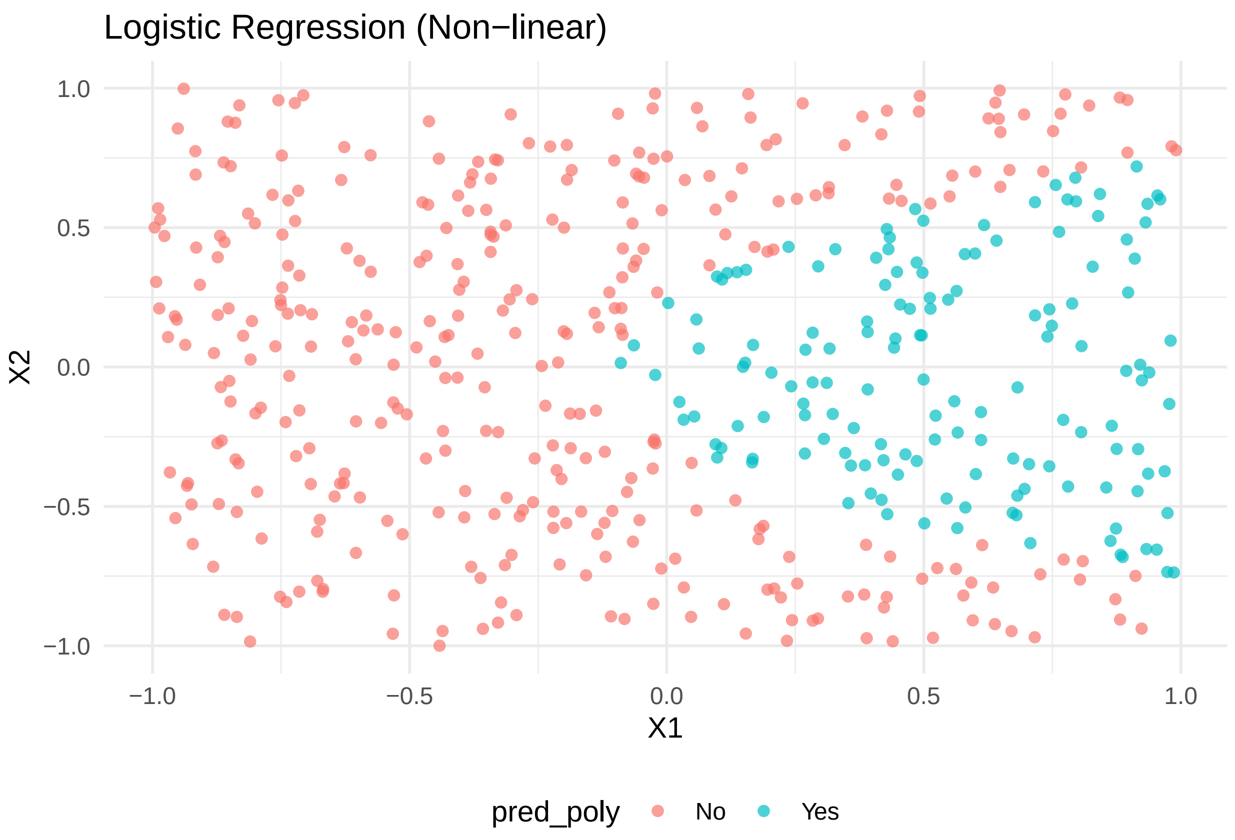
(6)

```
poly_logit <- train(Y ~ sin(X1) + cos(X2), data = df, method = "glm")
poly_logit
```

```
## Generalized Linear Model
##
## 500 samples
##   2 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 500, 500, 500, 500, 500, 500, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.8349    0.6054
```

(7)

```
df$pred_poly <- predict(poly_logit, df)

(p2 <- df %>% ggplot(aes(X1, X2, color = pred_poly)) + geom_point(alpha = 0.7) +
    ggtitle("Logistic Regression (Non-linear)") + theme(legend.position = "bottom"))
```

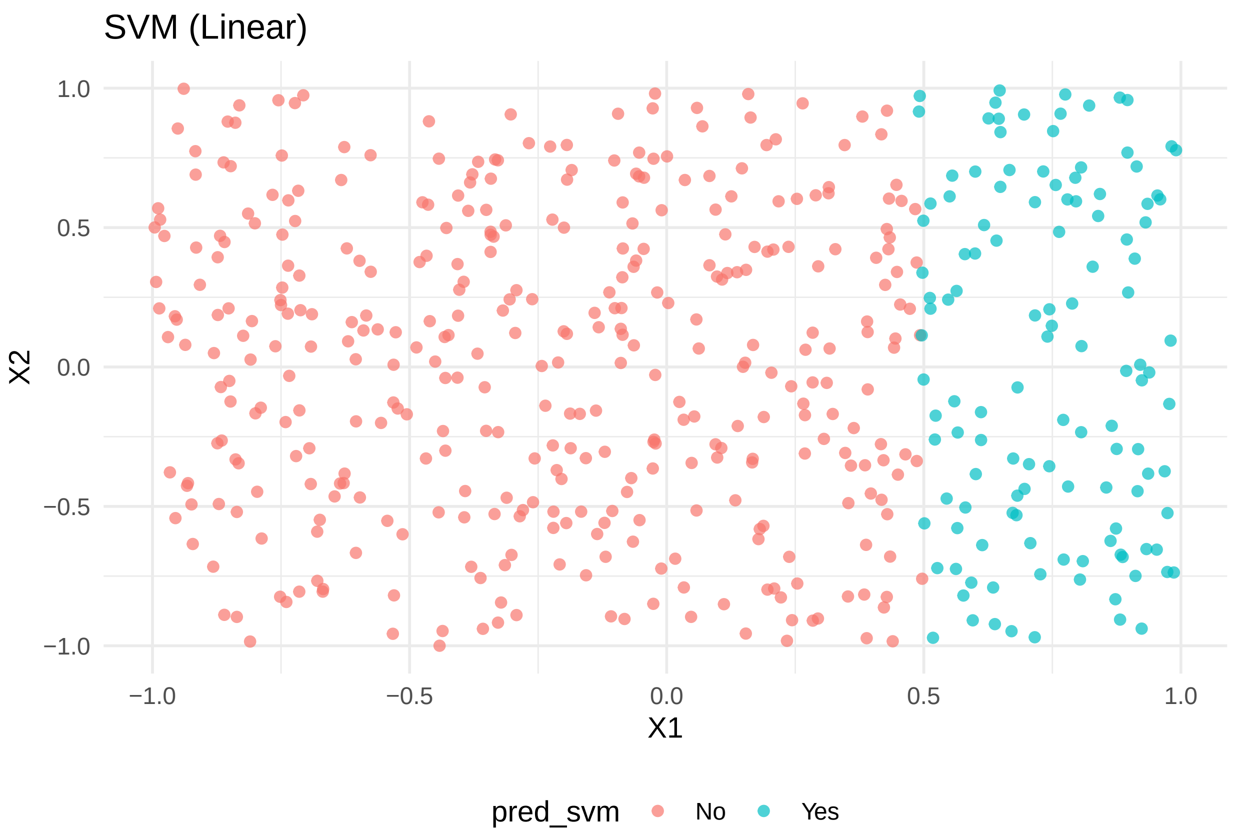


(8)

```
svm_linear <- train(Y ~ ., data = df %>% select(X1, X2, Y), method = "svmLinear",
    trControl = cv_ctrl, tuneLength = 10)
svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 500 samples
##   2 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 450, 449, 450, 450, 449, 450, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.738     0.3474
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
df$pred_svm <- predict(svm_linear, df)

(p3 <- df %>% ggplot(aes(X1, X2, color = pred_svm)) + geom_point(alpha = 0.7) + ggtitle("SVM (Linear)") +
    theme(legend.position = "bottom"))
```

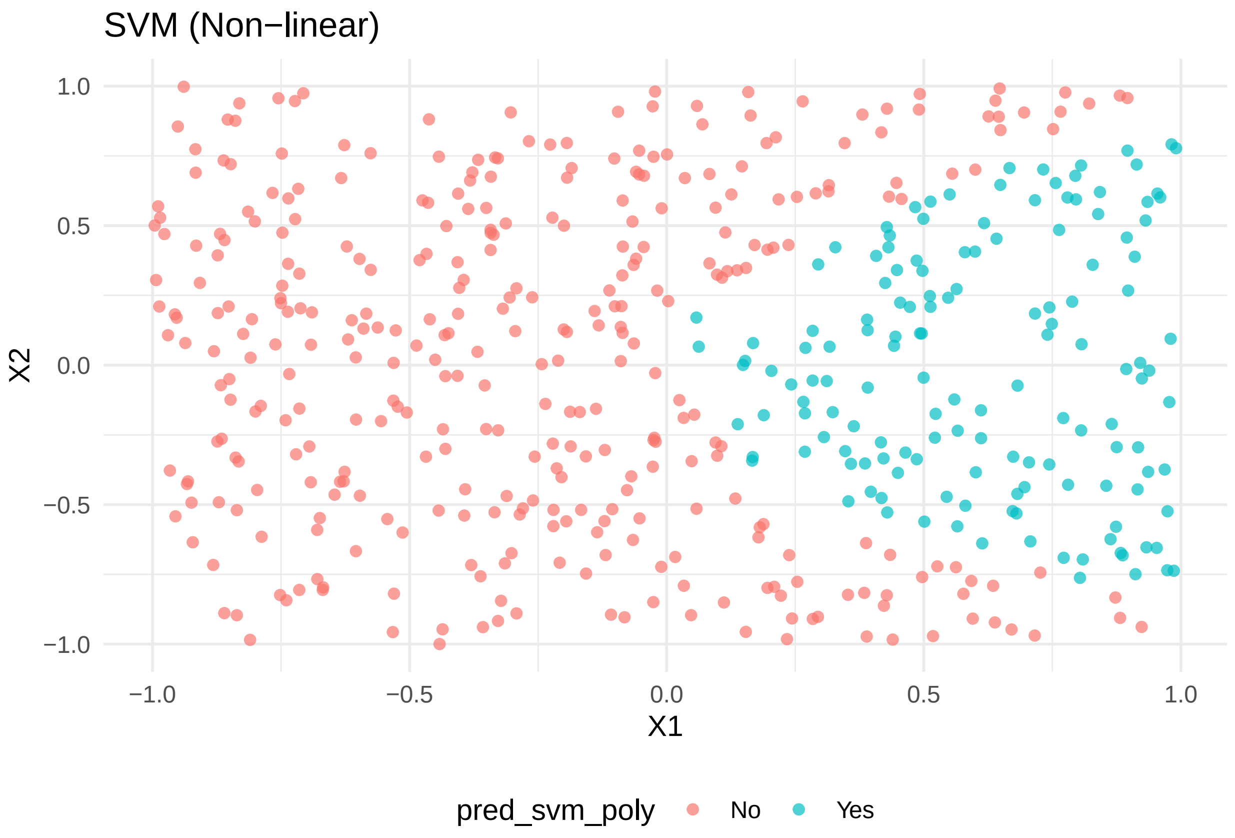


(9)

```
svm_poly <- train(Y ~ ., data = df %>% select(X1, X2, Y), method = "svmPoly", trControl = cv_ctrl)
svm_poly
```

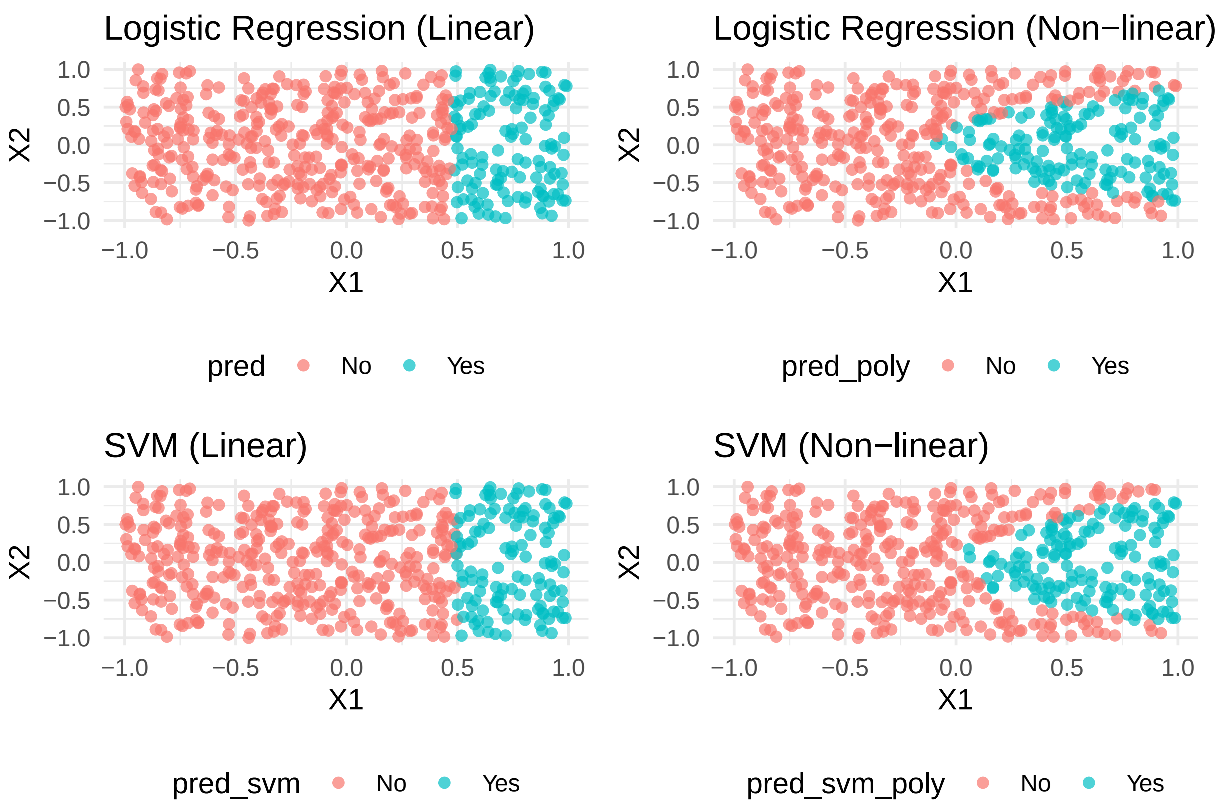
```
## Support Vector Machines with Polynomial Kernel
##
## 500 samples
##   2 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 449, 451, 450, 451, 449, 450, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy  Kappa
##   1       0.001  0.25  0.7176    0.3270
##   1       0.001  0.50  0.7300    0.3665
##   1       0.001  1.00  0.7257    0.3493
##   1       0.010  0.25  0.7557    0.4377
##   1       0.010  0.50  0.7336    0.3786
##   1       0.010  1.00  0.7276    0.3584
##   1       0.100  0.25  0.7259    0.3587
##   1       0.100  0.50  0.7497    0.4105
##   1       0.100  1.00  0.7356    0.3533
##   2       0.001  0.25  0.7377    0.4129
##   2       0.001  0.50  0.7536    0.4460
##   2       0.001  1.00  0.7416    0.4181
##   2       0.010  0.25  0.7437    0.3954
##   2       0.010  0.50  0.7517    0.4193
##   2       0.010  1.00  0.7516    0.4291
##   2       0.100  0.25  0.7836    0.4835
##   2       0.100  0.50  0.8015    0.5284
##   2       0.100  1.00  0.8156    0.5635
##   3       0.001  0.25  0.7357    0.3967
##   3       0.001  0.50  0.7416    0.4152
##   3       0.001  1.00  0.7538    0.4406
##   3       0.010  0.25  0.7517    0.4320
##   3       0.010  0.50  0.7418    0.4048
##   3       0.010  1.00  0.7516    0.4437
##   3       0.100  0.25  0.8175    0.5640
##   3       0.100  0.50  0.8334    0.6039
##   3       0.100  1.00  0.8274    0.5861
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 3, scale = 0.1 and C = 0.5.
```

```r
df$pred_svm_poly <- predict(svm_poly, df)

(p4 <- df %>% ggplot(aes(X1, X2, color = pred_svm_poly)) + geom_point(alpha = 0.7) +
    ggtitle("SVM (Non-linear)") + theme(legend.position = "bottom"))
```



(10)

```
(p1 + p2)/(p3 + p4)
```



```
data.frame(Method = c("Logistic (Linear)", "Logistic (Non-linear)", "SVM (linear)",
    "SVM (Non-linear)"), Accuracy = c(logit$results$Accuracy, poly_logit$results$Accuracy,
    svm_linear$results$Accuracy, max(svm_poly$results$Accuracy))) %>% kable()
```

Method	Accuracy
Logistic (Linear)	0.7345
Logistic (Non-linear)	0.8349
SVM (linear)	0.7380
SVM (Non-linear)	0.8334

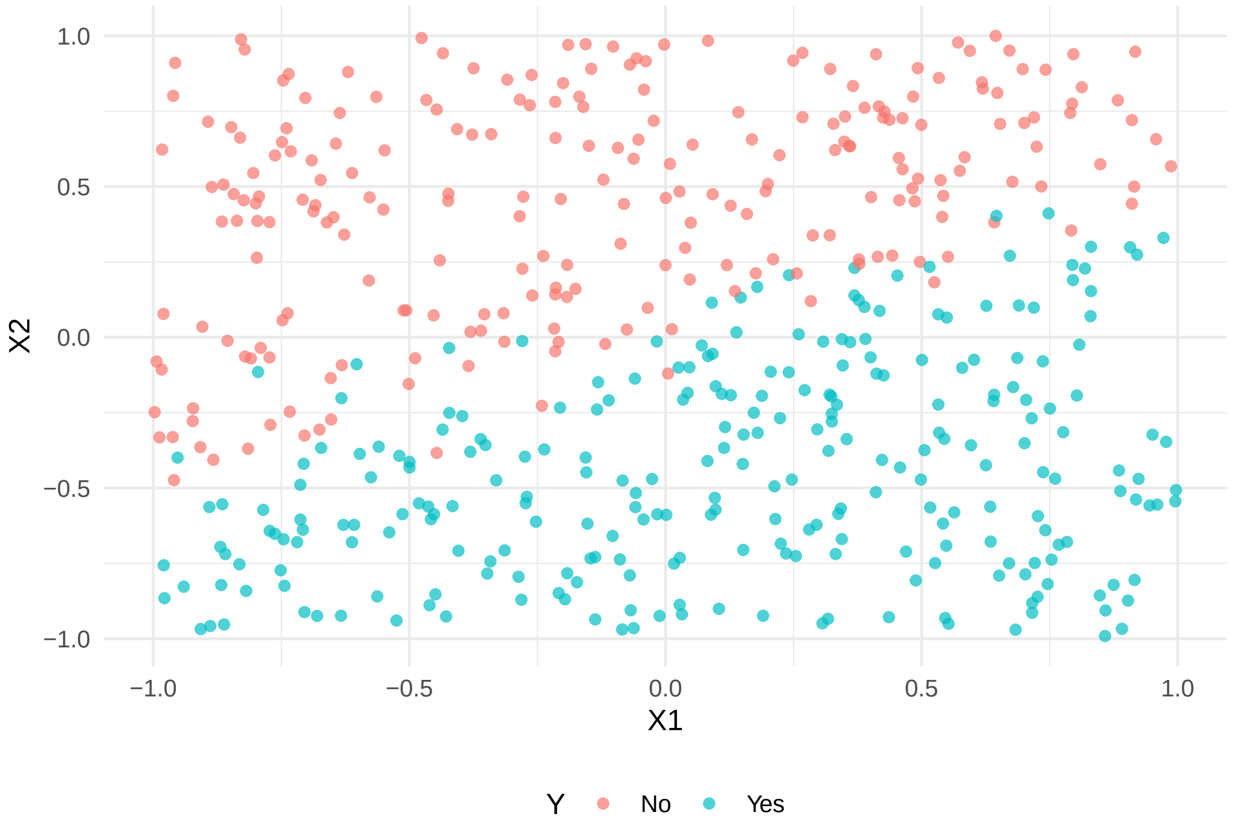
From the plots and the tables, we can see that logistic (linear) is almost the same as SVM (linear), and logistic (non-linear) is almost identical with SVM (non-linear)! In general, models with non-linear are better than those with linear (~10% higher in accuracy). This is expected because the underlying data is non-linear.

In terms of trade-offs, linear models are easier to interpret, but non-linear models perform a lot better. If we actually don't know that the underlying structure is non-linear, we need to see how far apart the accuracy between linear and non-linear models are along with plausible theoritical explanations. Between SVM and logistic model, SVM requires more computing costs (with CV and tuning). However, in non-linear case, we do not have to pre-specify the non-linear formula for SVM but we have to in logistic model. Importantly, the performance of logistic model depends heavily on what is the form of the non-linear formula. Hence, if we want linear model, logistic model is better, but if we want non-linear model, SVM is better.

Tuning cost

(11)

```r
X1 <- runif(500, -1, 1)
X2 <- runif(500, -1, 1)
Y <- X1 - 2 * X2 > rnorm(500, 0, 0.2)
df <- data.frame(X1 = X1, X2 = X2, Y = Y) %>% mutate(Y = factor(Y, levels = c(FALSE,
    TRUE), labels = c("No", "Yes")))
df %>% ggplot(aes(X1, X2, color = Y)) + geom_point(alpha = 0.7) + theme(legend.position = "bottom")
```



```r
# train/test split
split <- initial_split(data = df, prop = 0.85)
df <- training(split)
test_df <- testing(split)
```

(12)

```r
our_svm <- function(C = 0.1, kernel = "vanilladot", df) {
    ksvm(Y ~ ., data = df, type = "C-svc", kernel = kernel, C = C, scale = c(), cross = 10)
}

C_df <- tibble(C = c(0.001, 0.01, 0.1, 1)) %>% mutate(model = map(C, our_svm, df = df))
```

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

```r
for (i in 1:4) {
    print(C_df[i, ]$model)
}
```

```
## [[1]]
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.001
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 406
##
## Objective Function Value : -0.3588
## Training error : 0.461176
## Cross validation error : 0.4752
##
## [[1]]
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.01
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 250
##
## Objective Function Value : -1.963
## Training error : 0.082353
## Cross validation error : 0.08942
##
## [[1]]
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 136
##
## Objective Function Value : -10.65
## Training error : 0.047059
## Cross validation error : 0.05648
##
## [[1]]
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 76
##
## Objective Function Value : -65.54
## Training error : 0.044706
## Cross validation error : 0.04928
```

From the outputs, we can see that as `cost` increases, the training error decreases. The training error drops drastically when `cost` increases from 0.001 to 0.01. After than, the training error only slightly drops when `cost` increases from 0.01 to 0.1 and from 0.1 to 1. The CV error is tightly correlated to the training error. Mostly, the train and CV errors only differ by less than 1% when holding `cost` constant. `cost` = 1 yields the lowest training and CV errors.

(13)

```r
pred_test <- function(object) {
    predict(object, newdata = test_df)
}
# test error
C_df %>% mutate(pred = map(model, pred_test)) %>% unnest(pred) %>% cbind(test_df$Y) %>%
    group_by(C) %>% summarise(error = mean(pred != test_df$Y)) %>% kable()
```

C	error
0.001	0.3733
0.010	0.0400
0.100	0.0400
1.000	0.0400

Interestingly, the test errors for `cost` = {0.01, 0.1, 1} are all the same at 4%. Only when `cost` = 0.001 that the test error is significantly bigger (37.33%). `cost` = 1 still yields lowest errors for all train, CV, and test errors.

(14)

From train, CV, and test errors across different `cost` values, we can say that this suggests:

- The overlapping between classes are not that severe that `cost` = 0.01 is enough to reach good accuracy.
- However, there are still overlapping between classes such that `cost` = 0.001 is too low to counter the overlap.

These suggestions match the simulated data well: *the classes are just barely linearly separable.*

Application: Predicting attitudes towards racist college professors

```r
gss <- read_csv("data/gss_train.csv") %>% mutate(colrac = factor(colrac, levels = c(0,
    1), labels = c("Racist", "Okay")))
```

(15)

```r
# linear svm
svm_linear <- train(colrac ~ ., data = gss, method = "svmLinear", trControl = cv_ctrl,
    tuneGrid = expand.grid(C = c(0.001, 0.01, 0.1, 1)))

svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
```

```
## 1481 samples
##   55 predictor
##    2 classes: 'Racist', 'Okay'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1332, 1333, 1332, 1333, 1334, 1333, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy  Kappa
##   0.001  0.7819    0.5620
##   0.010  0.7988    0.5962
##   0.100  0.7974    0.5935
##   1.000  0.7927    0.5839
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The best `cost` value is 0.01 with accuracy of 80.09% or CV error rate of 19.91%. This rate is approximately the same as those of the models using logistic regression and boosting in HW5 (which were chosen as the best models). Hence, Linear SVM seems to do pretty well!

(16)

```r
tuneGrid <- expand.grid(C = c(0.001, 0.01, 0.1, 1), degree = c(1, 2, 3), scale = c(1))
# poly svm
svm_poly <- train(colrac ~ ., data = gss, method = "svmPoly", trControl = cv_ctrl,
    tuneGrid = tuneGrid)
svm_poly
```

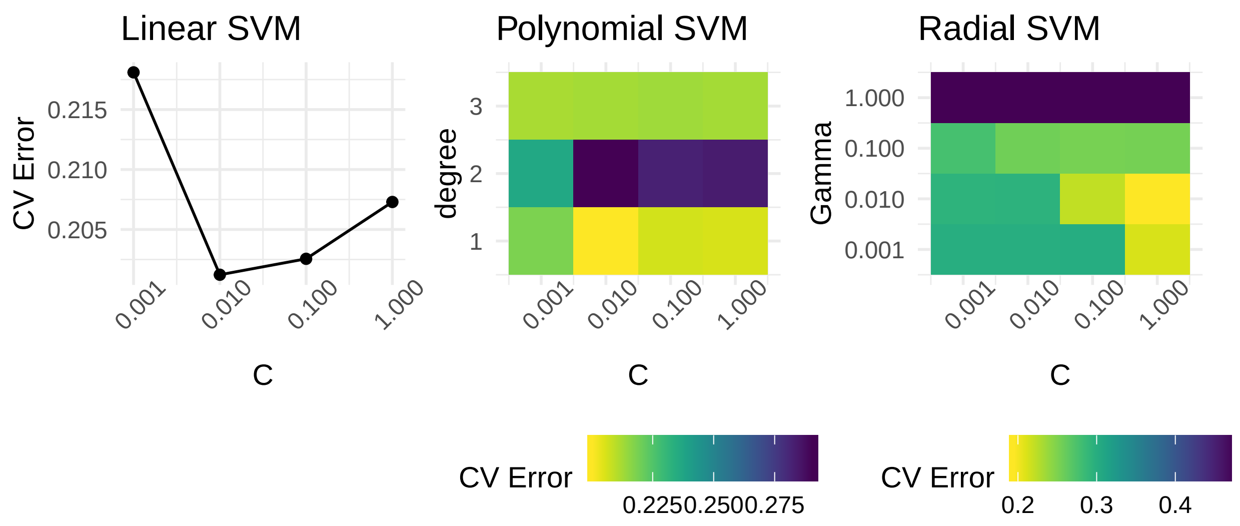
```
## Support Vector Machines with Polynomial Kernel
##
## 1481 samples
##   55 predictor
##    2 classes: 'Racist', 'Okay'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1333, 1333, 1333, 1333, 1333, ...
## Resampling results across tuning parameters:
##
##   C      degree  Accuracy  Kappa
##   0.001  1       0.7819    0.5619
##   0.001  2       0.7636    0.5252
##   0.001  3       0.7880    0.5746
##   0.010  1       0.7994    0.5980
##   0.010  2       0.7097    0.4181
##   0.010  3       0.7873    0.5732
##   0.100  1       0.7933    0.5856
##   0.100  2       0.7178    0.4338
##   0.100  3       0.7866    0.5718
##   1.000  1       0.7940    0.5868
##   1.000  2       0.7164    0.4312
##   1.000  3       0.7873    0.5733
##
```

```
## Tuning parameter 'scale' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 1, scale = 1 and C = 0.01.
```

```r
tuneGrid <- expand.grid(C = c(0.001, 0.01, 0.1, 1), sigma = c(0.001, 0.01, 0.1, 1))
# radial svm
svm_radial <- train(colrac ~ ., data = gss, method = "svmRadial", trControl = cv_ctrl,
    tuneGrid = tuneGrid)
svm_radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1481 samples
##   55 predictor
##    2 classes: 'Racist', 'Okay'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1332, 1333, 1333, 1333, 1333, 1334, ...
## Resampling results across tuning parameters:
##
##   C      sigma  Accuracy  Kappa
##   0.001  0.001  0.6996    0.4067
##   0.001  0.010  0.7050    0.4165
##   0.001  0.100  0.7219    0.4338
##   0.001  1.000  0.5253    0.0000
##   0.010  0.001  0.6996    0.4068
##   0.010  0.010  0.7044    0.4151
##   0.010  0.100  0.7435    0.4813
##   0.010  1.000  0.5253    0.0000
##   0.100  0.001  0.6983    0.4043
##   0.100  0.010  0.7779    0.5541
##   0.100  0.100  0.7468    0.4882
##   0.100  1.000  0.5253    0.0000
##   1.000  0.001  0.7874    0.5729
##   1.000  0.010  0.8036    0.6055
##   1.000  0.100  0.7455    0.4826
##   1.000  1.000  0.5253    0.0000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 1.
```

```r
p1 <- svm_linear$results %>% ggplot(aes(C, 1 - Accuracy)) + geom_point() + geom_line() +
    scale_x_log10() + labs(y = "CV Error", title = "Linear SVM") + theme(axis.text.x = element_text(ang

p2 <- svm_poly$results %>% ggplot(aes(C, degree)) + geom_tile(aes(fill = 1 - Accuracy)) +
    scale_fill_viridis(direction = -1) + scale_x_log10() + labs(title = "Polynomial SVM",
    fill = "CV Error") + theme(axis.text.x = element_text(angle = 45), legend.position = "bottom")

p3 <- svm_radial$results %>% ggplot(aes(C, sigma)) + geom_tile(aes(fill = 1 - Accuracy)) +
    scale_fill_viridis(direction = -1) + scale_x_log10() + scale_y_log10() + labs(y = "Gamma",
    title = "Radial SVM", fill = "CV Error") + theme(axis.text.x = element_text(angle = 45),
    legend.position = "bottom")

(p1 + p2 + p3)
```



```r
gss_test <- read_csv("data/gss_test.csv") %>% mutate(colrac = factor(colrac, levels = c(0,
    1), labels = c("Racist", "Okay")))

data.frame(Kernel = c("Linear", "Polynomial", "Radial"), `CV error` = c(1 - max(svm_linear$results$Accu
    1 - max(svm_poly$results$Accuracy), 1 - max(svm_radial$results$Accuracy)), `Test error` = c(mean(pre
    gss_test) != gss_test$colrac), mean(predict(svm_poly, gss_test) != gss_test$colrac),
    mean(predict(svm_radial, gss_test) != gss_test$colrac))) %>% kable()
```

Kernel	CV.error	Test.error
Linear	0.2012	0.215
Polynomial	0.2006	0.213
Radial	0.1964	0.213

From the plots and the table, we can see that the best kernel among the three kernels is radial. Radial SVM has the lowest CV error rate as well as test error rate. Interestingly, radial SVM's tuning is the only kernel that chooses `cost` equal to 1 instead of 0.01 as linear and polynomial kernels do. The `cost` choice may depend on the best `gamma` chosen by the radial SVM. It may also suggest that radial kernel is more sensitive to overlapping between classes than linear and polynomial kernels are.

Nevertheless, all kernels seem to perform very similarly to each other. That is, there doesn't seem to be a significant benefit using radial kernel over linear kernel. Hence, I think the best model is the linear SVM since it is easier to tune, requires less computing cost, and is easier to interpret in comparison to other kernels.