HW6 - Support Vector Machines [MACS 30100]

Adarsh Mathew

3/8/2020

Conceptual Exercises

```
set.seed(03072020)
n_obs0 <- 200

simdata_df0 <- tibble(x1 = runif(n_obs0, -1, 1), x2 = runif(n_obs0, -1, 1),
                      err =  rnorm(n_obs0, mean = 0, sd = 0.15)) %>%
  mutate(f_x = x1 + x1^2 +x2 +x2^2 + err, y = exp(f_x)/(1+exp(f_x)),
         y_class = as.factor(y > 0.5)) %>%
  mutate(y_class = factor(y_class, labels = make.names(levels(y_class))))

simdata_df0 %>% ggplot() +
  geom_point(aes(x = x1, y = x2, colour = y_class)) + theme_minimal() +
  ggtitle("True Class assignments of Simulated Data")
```

```
simdata0_dfsplit <- initial_split(simdata_df0, prop = 0.5)
simdata0_df_train <- training(simdata0_dfsplit)
simdata0_df_test <- testing(simdata0_dfsplit)

model_gen <- function(model_type){
  tr_control <- trainControl(method = "cv", number = 10,
                             # summaryFunction=postResample,
                             classProbs=T,
                             savePredictions = T)

  simdata0_model <- train(form = y_class ~ x1+x2, data = simdata0_df_train,
                      method = model_type, preProcess = c("center", "scale"),
                      metric = "Accuracy", trControl = tr_control)

  return(simdata0_model)
}
```

```
model_df0 <- c("Linear kernel" = "svmLinear",
               "Radial kernel" = "svmRadial") %>%
  enframe(name = "model_name", value = "model_io") %>%
  mutate(model_obj = map(model_io, ~model_gen(.x)),
         model_accuracy = map_dbl(model_obj, ~max(.x$results$Accuracy)),
         model_accuracySD = map_dbl(model_obj, ~.x$results$AccuracySD[which.max(.x$results$Accuracy)]),
         model_testAccuracy = map_dbl(model_obj, ~postResample(predict(.x, newdata = simdata0_df_test),
```

```
model_df0 %>% ggplot(aes(x = model_name, y = model_accuracy)) +
  geom_point(size = 2) +
```

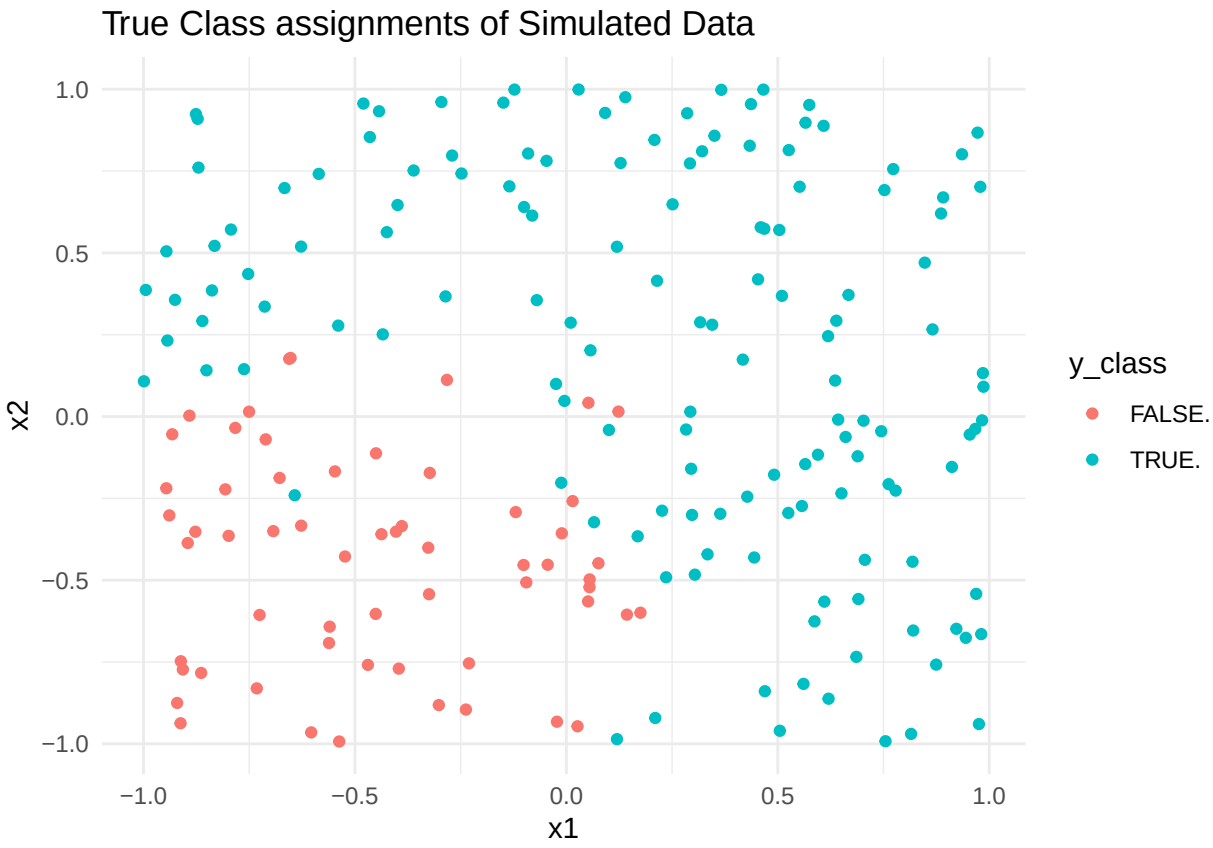


Figure 1: True Class Assignment of Simulated Data

```
  geom_errorbar(aes(ymin = model_accuracy - model_accuracySD,
                    ymax = model_accuracy + model_accuracySD,
                    colour = model_accuracySD), width = 0.1, alpha = 0.5) +
  theme_minimal() +
  ggtitle("Training Accuracy by kernel")
```

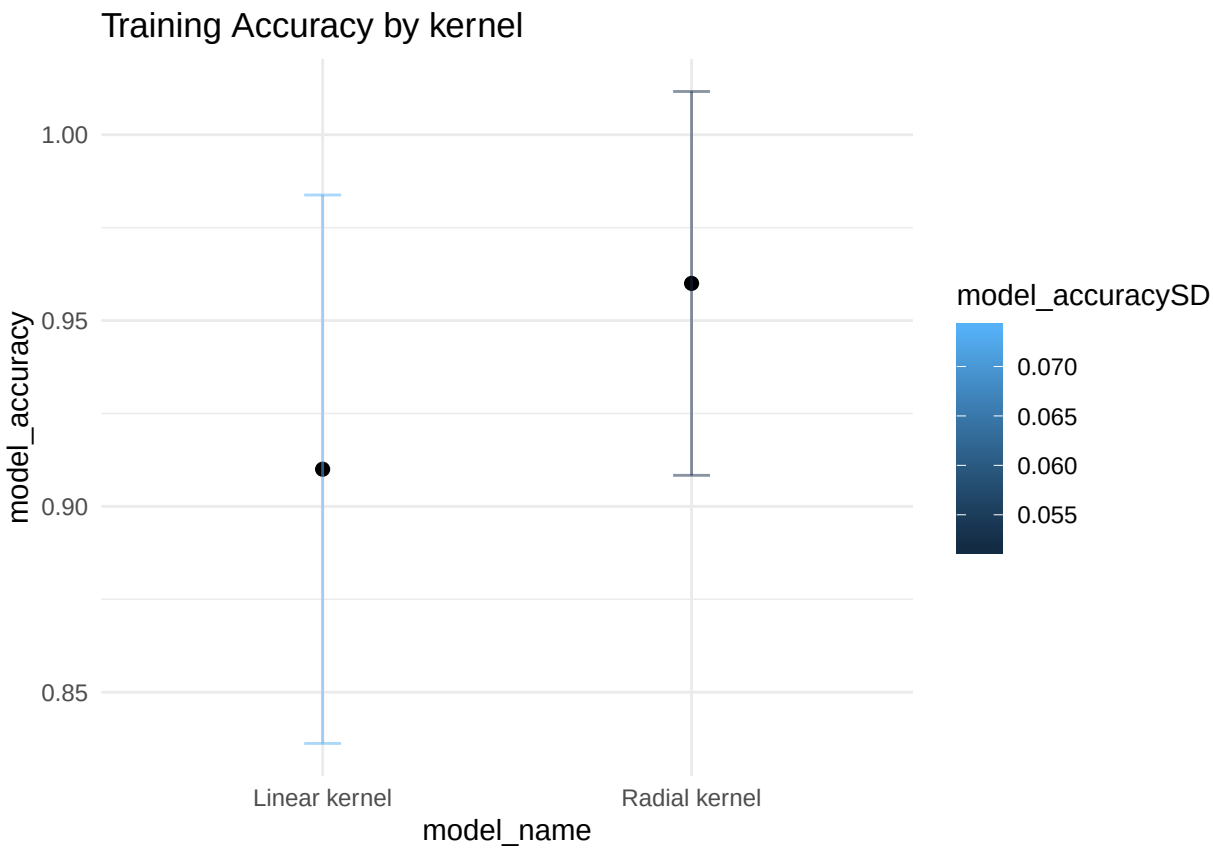


Figure 2: Training Accuracy by kernel

```
simdata_df0_aug <- simdata0_df_train %>%
  bind_cols(svmLinear_class = model_df0$model_obj[[1]]$finalModel@fitted,
            svmRadial_class = model_df0$model_obj[[2]]$finalModel@fitted)

simdata_df0_aug %>%
  pivot_longer(cols = 6:8, names_to = "model_type", values_to = "pred_class") %>%
  ggplot(aes(x = x1, y = x2, colour = pred_class)) + geom_point() +
  facet_wrap(vars(model_type)) + theme_minimal() +
  ggtitle("Predicted Classes by Model", "Training Data") +
  theme(legend.position = "bottom")
```

We see how the Radial kernel performs better with this data, but the Linear kernel doesn't perform too badly. The clear separation in the true data reduces the number of misclassified instances in the linear kernel.

```
model_df0 %>% ggplot(aes(x = model_name, y = model_testAccuracy)) +
  geom_point(size = 2) +
  theme_minimal() +
  ggtitle("Testing Accuracy by kernel")
```

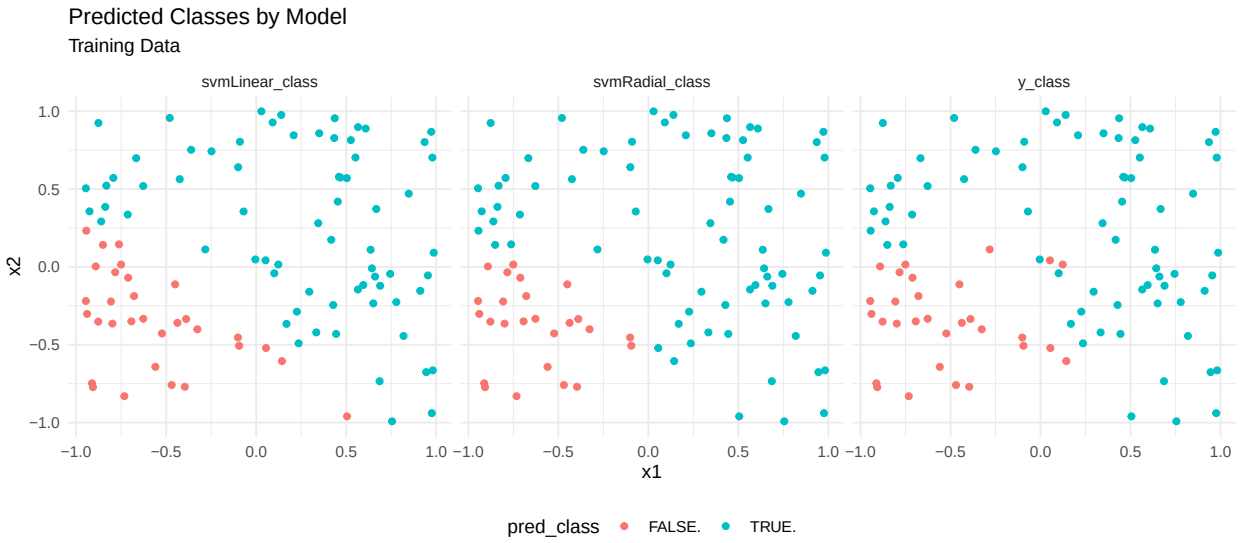


Figure 3: Classification of data by kernel

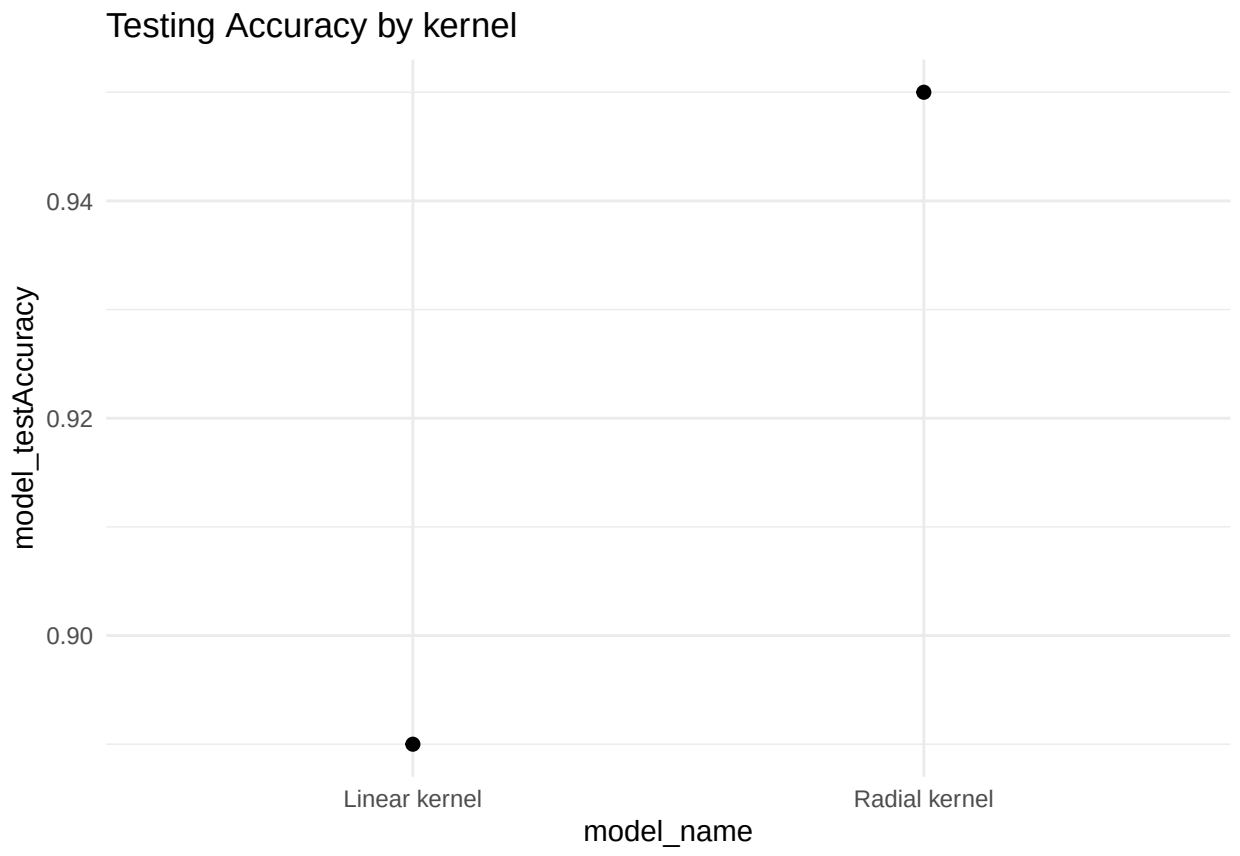


Figure 4: Testing Accuracy by kernel

We see how the Radial kernel performs so much better than the linear kernel on test data, indicating that it generalizes well for our non-linear decision boundary.

Thus, a radial kernel outperforms the linear kernel comfortably for data with a non-linear separation between the two classes.

SVM vs. Logistic Regression

```
set.seed(03072020)
n_obs <- 500

simdata_df <- tibble(x1 = runif(n_obs, -1, 1), x2 = runif(n_obs, -1, 1),
                     err =  rnorm(n_obs, mean = 0, sd = 0.25)) %>%
  mutate(f_x = x1 + x1^2 +x2 +x2^2 + err, y = exp(f_x)/(1+exp(f_x)),
         y_class = as.factor(y > 0.5)) %>%
  mutate(y_class = factor(y_class, labels = make.names(levels(y_class))))
```

```
simdata_df %>% ggplot() +
  geom_point(aes(x = x1, y = x2, colour = y_class)) + theme_minimal() +
  ggtitle("True Class assignments of Simulated Data")
```

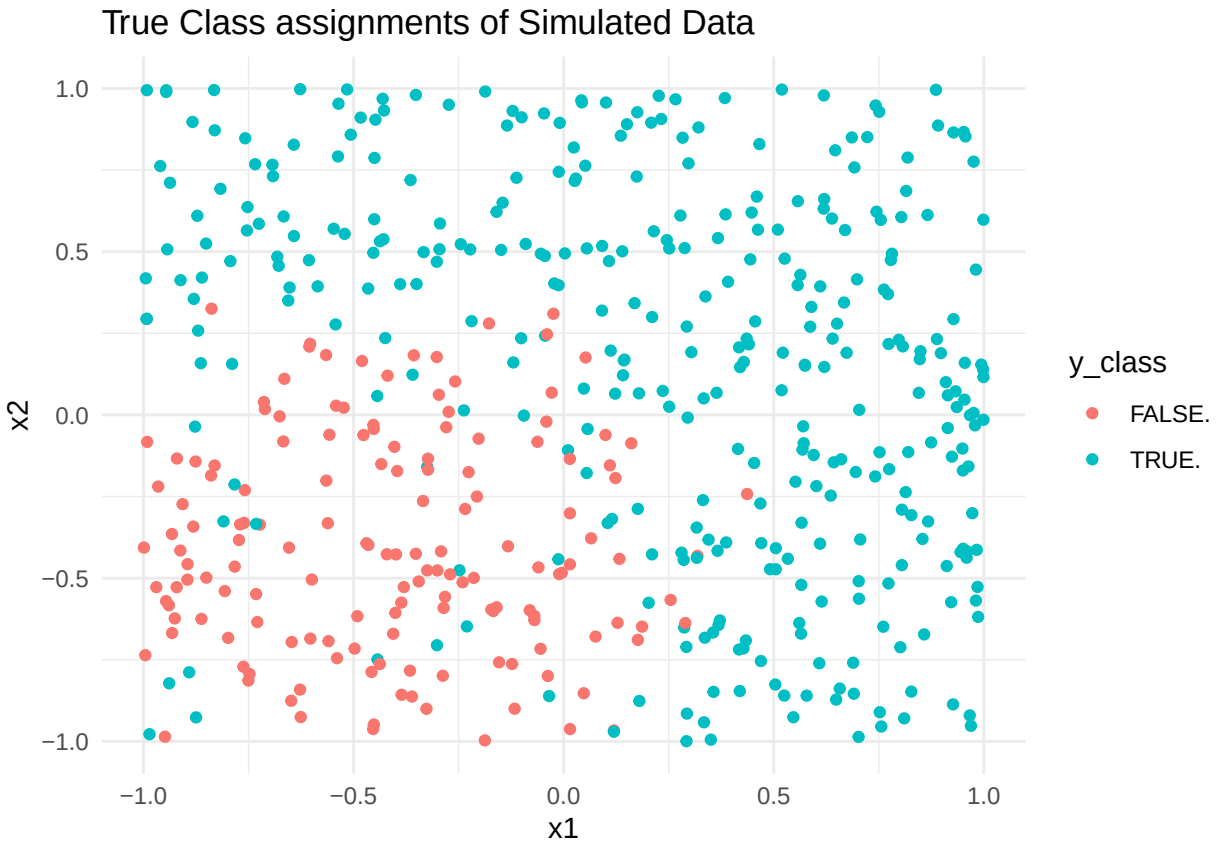


Figure 5: True Class Assignment of Simulated Data

```
set.seed(02242020)
tr_control <- trainControl(method = "cv", number = 10,
                           classProbs=T, savePredictions = T)

simdf_Logit <- train(form = y_class ~ x1+x2, data = simdata_df,
                     method = "glm", preProcess = c("center", "scale"),
                     metric = "Accuracy", trControl = tr_control)
simdf_trLogit <- train(form = y_class ~ x1+x2+(x1^2)+(x2^2)+(x1*x2), data = simdata_df,
                     method = "glm", preProcess = c("center", "scale"),
```

```
                      metric = "Accuracy", trControl = tr_control)
simdf_svmLinear <- train(form = y_class ~ x1+x2, data = simdata_df,
                      method = "svmLinear", preProcess = c("center", "scale"),
                      metric = "Accuracy", trControl = tr_control,
                      tuneGrid = expand.grid(C = 10^seq(-3, 1, by = 0.5)))
simdf_svmRadial <- train(form = y_class ~ x1+x2, data = simdata_df,
                      method = "svmRadial", preProcess = c("center", "scale"),
                      metric = "Accuracy", trControl = tr_control)
simdf_svmPoly <- train(form = y_class ~ x1+x2, data = simdata_df,
                      method = "svmPoly", preProcess = c("center", "scale"),
                      metric = "Accuracy", trControl = tr_control)
```

```
c("Logit" = simdf_Logit$results$Accuracy,
  "trLogit" = simdf_trLogit$results$Accuracy,
  "svmLinear" = (simdf_svmLinear$results %>% arrange(desc(Accuracy)) %>%
                   filter(row_number() == 1))$Accuracy,
  "svmRadial" = (simdf_svmRadial$results%>% arrange(desc(Accuracy)) %>%
                   filter(row_number() == 1))$Accuracy,
  "svmPoly" = (simdf_svmPoly$results %>% arrange(desc(Accuracy)) %>%
                 filter(row_number() == 1))$Accuracy) %>%
  enframe(name = "model_type", value = "Accuracy") %>% arrange(desc(Accuracy))
```

```
## # A tibble: 5 x 2
##   model_type Accuracy
##   <chr>         <dbl>
## 1 svmRadial     0.910
## 2 svmPoly       0.908
## 3 trLogit       0.888
## 4 Logit         0.860
## 5 svmLinear     0.860
```

We see that the `Accuracy` values are much higher for the `Radial` and `Polynomial` kernels, as is to be expected.

```
simdata_df_aug <- simdata_df %>%
  bind_cols(Logit_class = simdf_Logit$pred$pred,
            trLogit_class = simdf_trLogit$pred$pred,
            svmLinear_class = simdf_svmLinear$finalModel@fitted,
            svmRadial_class = simdf_svmRadial$finalModel@fitted,
            svmPoly_class = simdf_svmPoly$finalModel@fitted)

simdata_df_aug %>%
  pivot_longer(cols = 6:11, names_to = "model_type", values_to = "pred_class") %>%
  ggplot(aes(x = x1, y = x2, colour = pred_class)) + geom_point() +
  facet_wrap(vars(model_type)) + theme_minimal() + ggtitle("Predicted Classes by Model")
```

Comparing the predicted classes to the true class, we see the shortcomings of the Logit and `Linear` SVM kernel. The logistic models don't seem to be tease out the true decision boundary and seem to be assigning classes at random almost. The `Linear` SVM kernel draws a strict line as the decision boundary, which fails to capture the true boundary. So while the Accuracy drop-offs may not be too big, we notice the superior advantages of the non-linear kernels at identifying the true nature of the data-generating process.

Tuning Cost

```
set.seed(03072020)
n_obs2 <- 2000
```

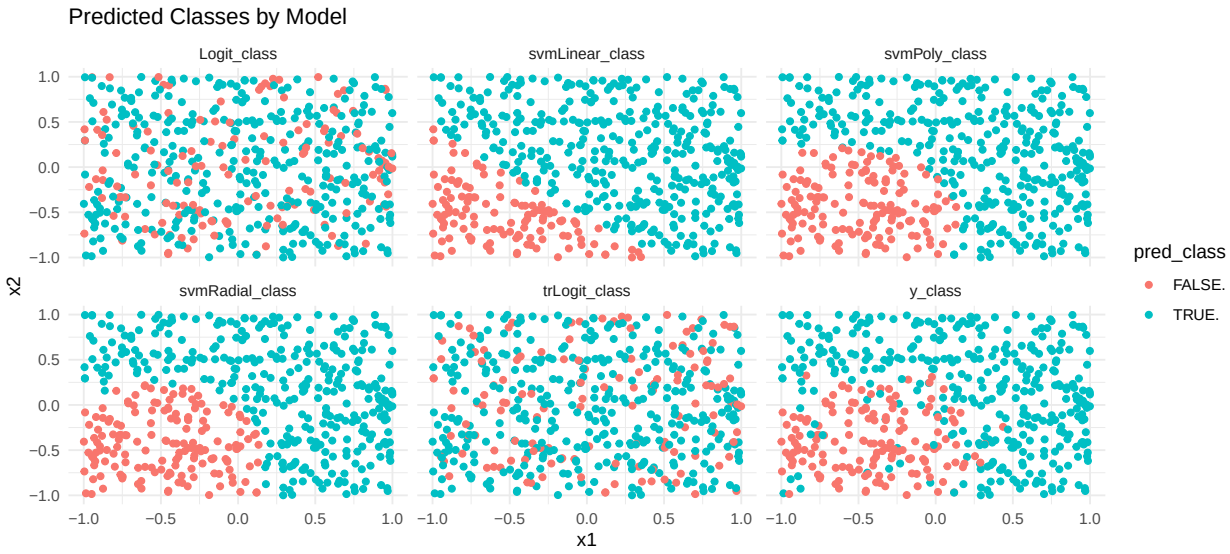


Figure 6: Classification of data by kernel

```
simdata2_df <- tibble(x1 = runif(n_obs2, -1, 1),
                      x2 = runif(n_obs2, -1, 1),
                      err =  rnorm(n_obs2, mean = 0, sd = 0.15)) %>%
  mutate(f_x = x1 +x2 + err, y = exp(f_x)/(1+exp(f_x)),
         y_class = as.factor(y > 0.5)) %>%
  mutate(y_class = factor(y_class, labels = make.names(levels(y_class))))

simdata2_df %>%
  ggplot() + geom_point(aes(x = x1, y = x2, colour = y_class)) +
  theme_minimal() + ggtitle("True Class assignments of Simulated Data")
```

```
simdata2_dfsplit <- initial_split(simdata2_df, prop = 0.5)
simdata2_df_train <- training(simdata2_dfsplit)
simdata2_df_test <- testing(simdata2_dfsplit)

tr_control2 <- trainControl(method = "cv", number = 10,
                            classProbs=T, savePredictions = "all")

simdf2_svmLinear_tr <- train(form = y_class ~ x1+x2, data = simdata2_df_train,
                      method = "svmLinear", preProcess = c("center", "scale"),
                      metric = "Accuracy", trControl = tr_control2,
                      tuneGrid = expand.grid(C = 10^seq(-3, 2, by = 0.25)))
```

```
# simdf2_svmLinear$results
ggplot(simdf2_svmLinear_tr$results, aes(x=C, y=Accuracy)) +
  geom_line() + geom_point()+
  geom_errorbar(aes(ymin=Accuracy-AccuracySD, ymax=Accuracy+AccuracySD, colour = AccuracySD),
                width=.2, position=position_dodge(0.05), alpha = 0.5) +
  scale_x_log10() + theme_minimal() + ggtitle("Training Accuracy", "SVM Linear")
```

```
simdf2_svmLinear_tr$pred %>% group_by(C) %>%
  summarize(mean_accuracy = mean(pred == obs),
            sd_accuracy = sd(pred == obs)) %>%
```

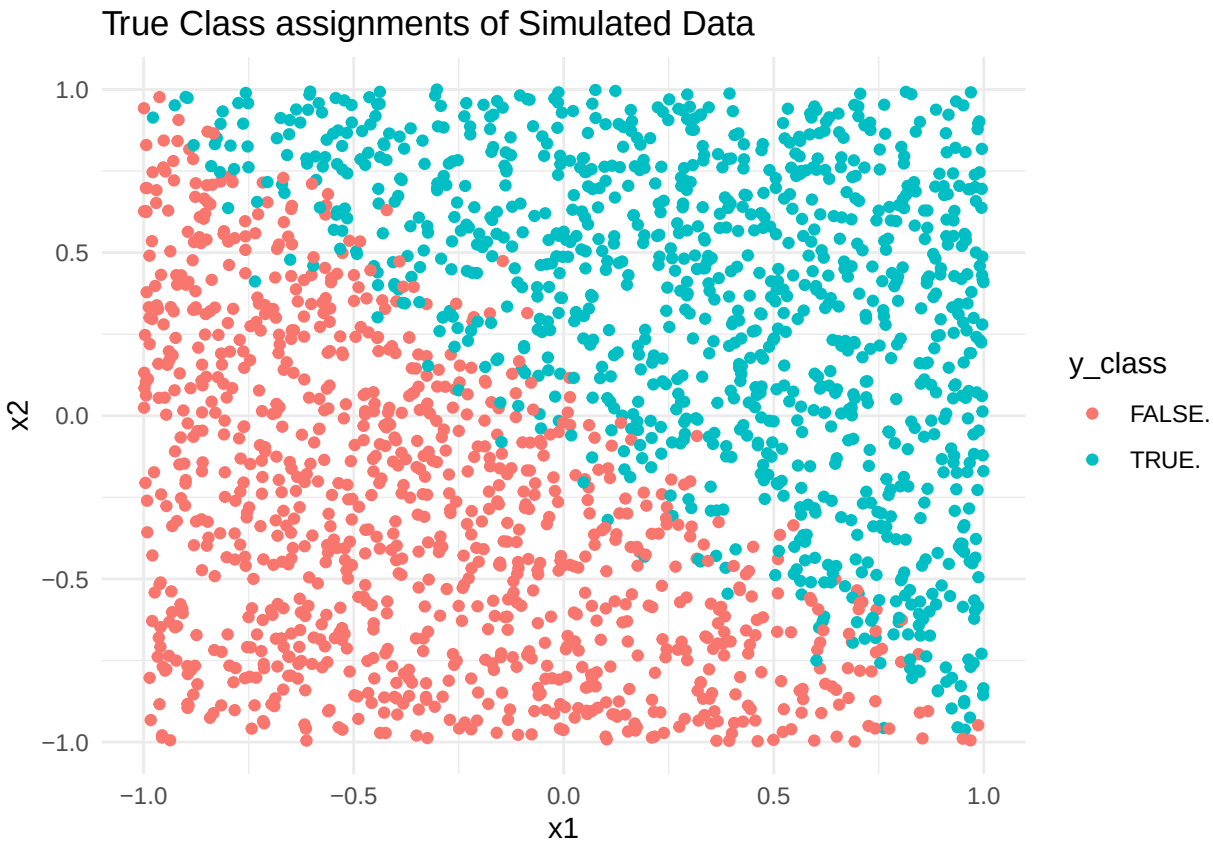


Figure 7: True Class Assignments of Simulated Data

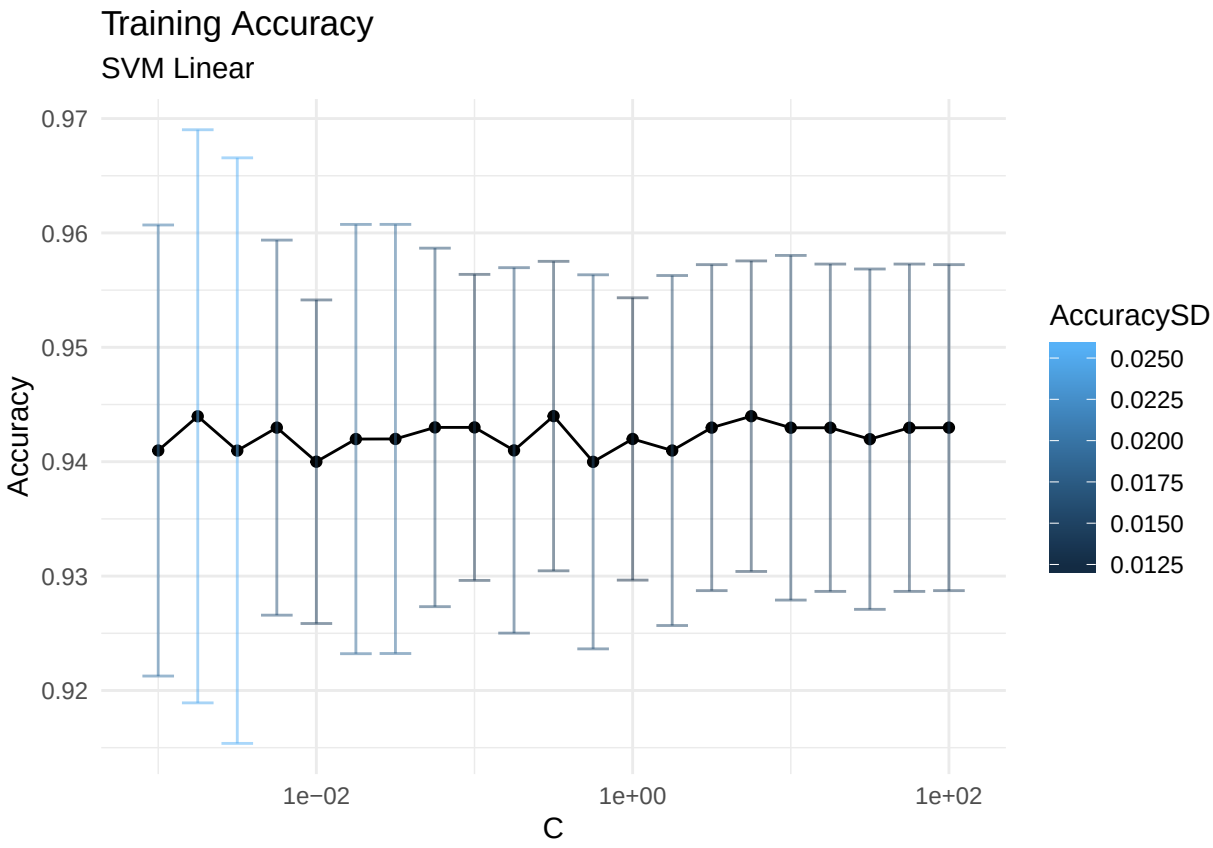
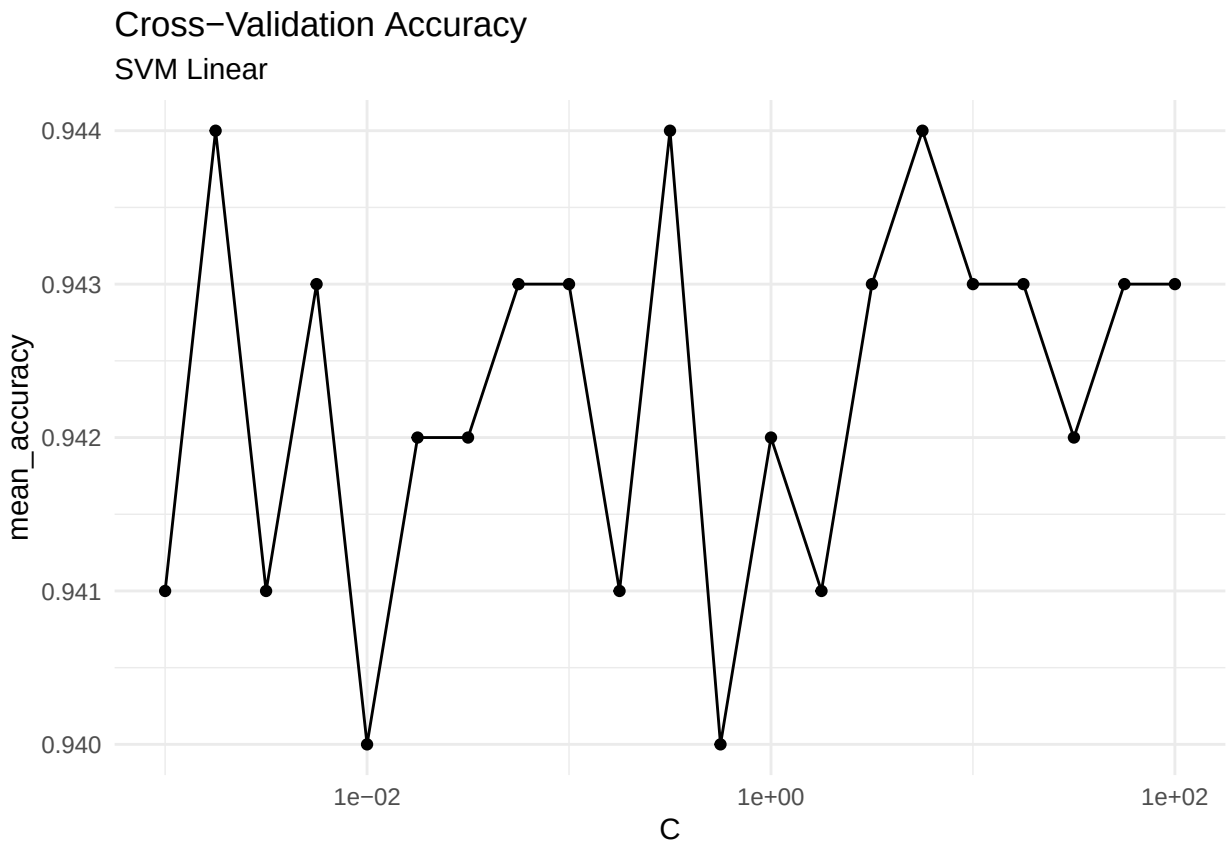


Figure 8: Training Error of Linear Kernel over Cost parameters

```
  ggplot(aes(x=C, y=mean_accuracy)) +
  geom_line() + geom_point()+
  scale_x_log10() + theme_minimal() + theme(legend.position = "bottom") +
  ggtitle("Cross-Validation Accuracy", "SVM Linear")
```




```
simdf2_svmLinear_ts <- train(form = y_class ~ x1+x2, data = simdata2_df_test,
                     method = "svmLinear", preProcess = c("center", "scale"),
                     metric = "Accuracy",
                     tuneGrid = expand.grid(C = 10^seq(-3, 2, by = 0.25)))

ggplot(simdf2_svmLinear_ts$results, aes(x=C, y=Accuracy)) +
  geom_line() + geom_point() +
  scale_x_log10() + theme_minimal() + ggtitle("Testing Accuracy", "SVM Linear")
```

- Best Training Accuracy: 0.9439891
- Optimal Training `Cost`: 0.3162278
- Best Testing Accuracy: 0.9454933
- Optimal Testing `Cost`: 5.6234133

We see CV-error is tightly correlated with training error (as is to be expected). There's no real point in examining CV error if you're doing 10-fold CV and have training error to examine against.

We also notice that the optimal range of *Cost* in the training iteration underperforms for the test data. This is to be expected – higher *Cost* has higher bias but lower variance, so it makes sense that a moderately high value is more sensitive to errors for optimality. A lot of it may be driven by the barely linearly separable nature of the classes.

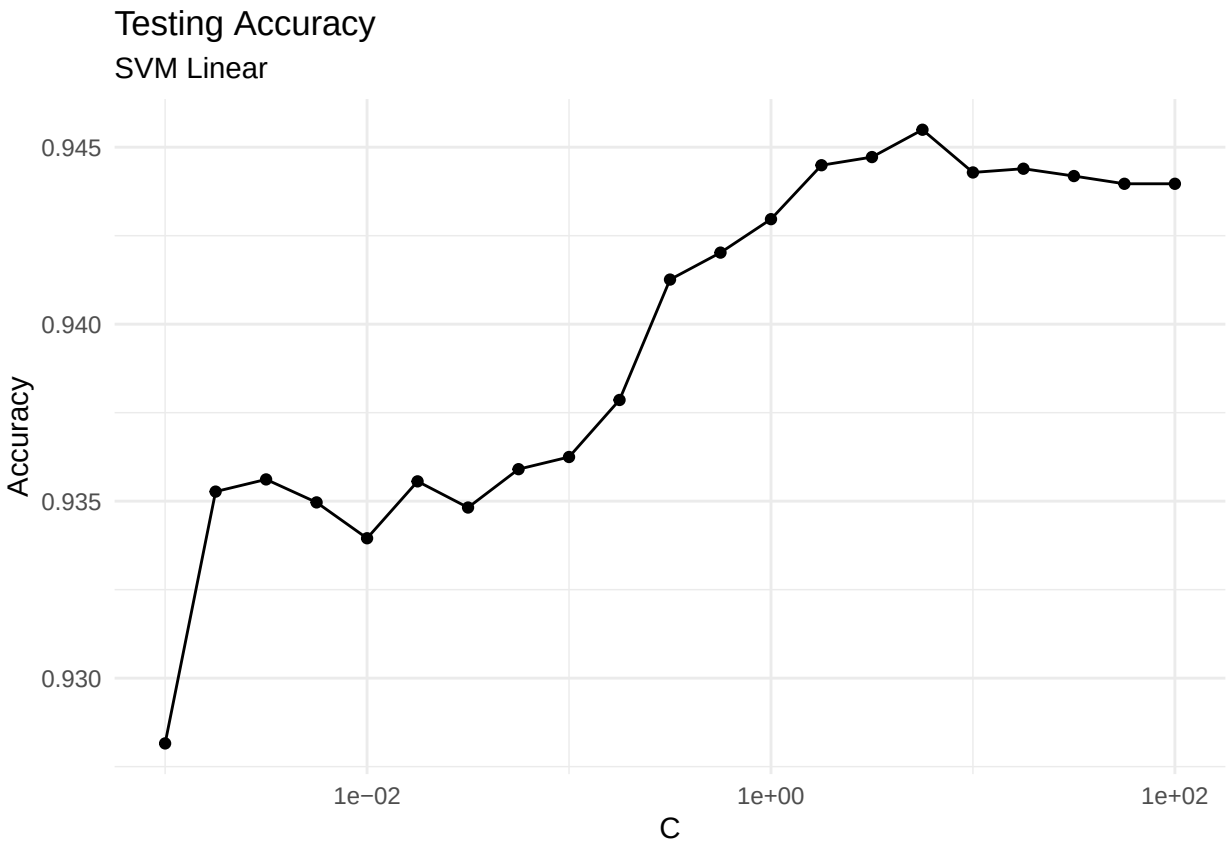


Figure 9: Ttesting Error of Linear Kernel over Cost parameters

Application Exercises – Comparing SVM kernels

```
gss_train_df <- read_csv("../data/gss_train.csv",
                         col_types = cols(colrac = col_factor(levels = c("1", "0")))) %>%
   mutate(colrac = factor(colrac,
                          labels = make.names(levels(colrac))))

gss_test_df <- read_csv("../data/gss_test.csv",
                         col_types = cols(colrac = col_factor(levels = c("1", "0")))) %>%
   mutate(colrac = factor(colrac,
                          labels = make.names(levels(colrac))))
```

Model Estimation

```
# Helper function to feed into `map`

model_gen <- function(model_type){
  set.seed(03052020)

  if (model_type == "svmLinear") {
  tunegrid_in = expand_grid(C = 10^seq(-3, 3, by = 0.5))
} else {
  if (model_type == "svmRadial") {
    tunegrid_in = expand_grid(C = 10^seq(-3, 3, by = 0.5),
                              sigma = 10^seq(-2, 2, by = 0.5))
  } else {
    tunegrid_in = expand_grid(C = 10^seq(-3, 3, by = 0.5),
                              degree = seq(1, 4, by = 1),
                              scale = 1)
    }
}

  tr_control <- trainControl(method = "cv", number = 10,
                             # summaryFunction=postResample,
                             classProbs=T,
                             savePredictions = T)

  gss_model <- train(form = colrac ~ ., data = gss_train_df,
                     method = model_type, preProcess = c("center", "scale"),
                     metric = "Accuracy", trControl = tr_control, tuneGrid = data.frame(tunegrid_in))

  return(gss_model)
}
```

```
plan(multiprocess)
model_df <- c("Linear kernel" = "svmLinear",
              "Radial kernel" = "svmRadial",
              "Polynomial kernel" = "svmPoly") %>%
  enframe(name = "model_name", value = "model_io") %>%
  mutate(model_obj = future_map(model_io, ~model_gen(.x)))
```

```
## maximum number of iterations reached 0.01402076 0.01481618maximum number of iterations reached 7.430
```

```r
cverr_p1 <- model_df$model_obj[[1]]$results %>%
  ggplot(aes(x = C, y = Accuracy)) +
  geom_point() + geom_line() + scale_x_log10() +
  theme_minimal() +
  ggtitle("Training Error for SVM Linear", "Cost values")

cverr_p2 <- model_df$model_obj[[2]]$results %>%
  ggplot(aes(x = C, y = Accuracy, colour = as.factor(log10(sigma)))) +
  geom_point() + geom_line(alpha = 0.5) + scale_x_log10() +
  theme_minimal() +
  theme(legend.position = "bottom") +
  ggtitle("Training Error for SVM Radial", "Sigma and Cost values")

cverr_p3 <- model_df$model_obj[[3]]$results %>%
  ggplot(aes(x = C, y = Accuracy, colour = as.factor(degree))) +
  geom_point() + geom_line() + scale_x_log10() +
  theme_minimal() +
  theme(legend.position = "bottom") +
  ggtitle("Training Error for SVM Polynomial", "Degree and Cost values")

cverr_p1 + cverr_p2 + cverr_p3
```

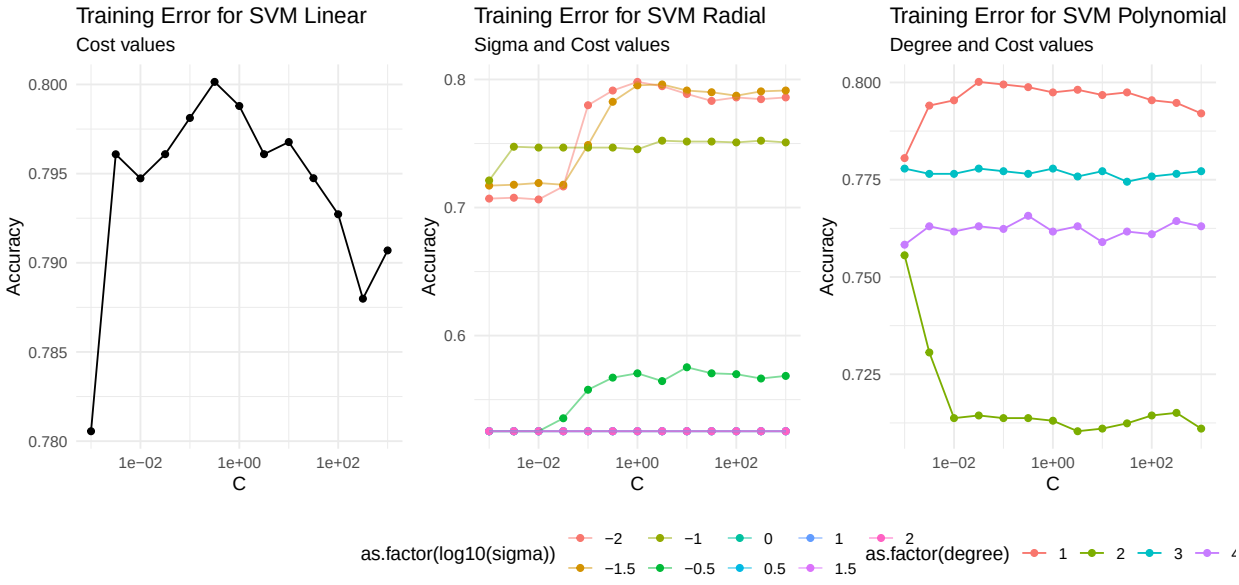


Figure 10: Training Errors for various SVM kernels

SVM Linear: - The best performance is for $Cost = 0.01$, with accuracy as 0.796. - All `cost` parameters generate fairly similar CV errors.

SVM Radial: - The best performing models have an accuracy of 0.8: - $C = 1, \sigma = 10^{-2}$ - $C = 10^{0.5,1,1.5}, \sigma = 10^{-1.5}$ - We see vastly lower (and equal) accuracy for all values of $\sigma \geq 1$

SVM Polynomial: - We get the best performance of 0.79 for $C = 10^{-2}$, $degree = 1$ - The quadratic kernel performs worst, while $degree = 3, 4$ don't show much variation with $Cost$.

We see the Radial Basis Function with the best training performance. I'd prefer variants with higher costs

for more stable generalizations.