PS-06 Borui Sun

borui sun

3/7/2020

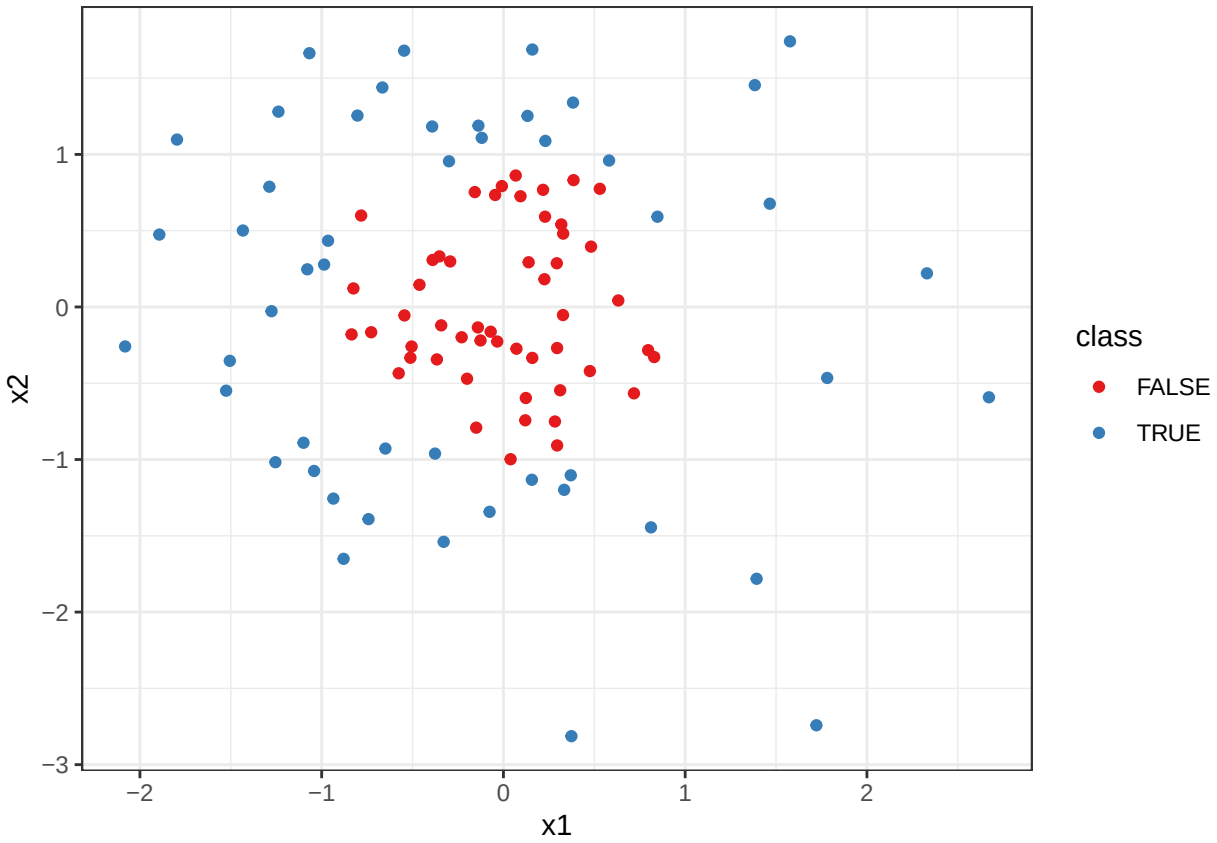
Conceptual exercises

Non-linear separation

1. (15 points) *Generate* a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear separation between the two classes. *Show* that in this setting, a support vector machine with a radial kernel will outperform a support vector classifier (a linear kernel) on the training data. *Which* technique performs best on the test data? Make plots and report training and test error rates in order to support your conclusions.

Based on Fig.1, Fig.2 and table 1, we can see that a SVM with a radial kernel outperform a SVM with a linear kernel on both trainin and test data. Because in my simulation, the decision boundary is a circle that separates the data into a inner-circle class and an outer-circle class. It is impossible to draw a linear hyperplane that seperates the two classes, so the SVM with a linear kernel only has 1 class in its predictions and hence it is expected to be outperformed by the SVM with a radial kernel.

```
set.seed(5904)
two_class <- tibble(x1 = rnorm(100),
                    x2 = rnorm(100),
                    y = x1^2 + x2^2 > 1)
two_class %>%
  ggplot(aes(x=x1, y = x2, color = y)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class")
```

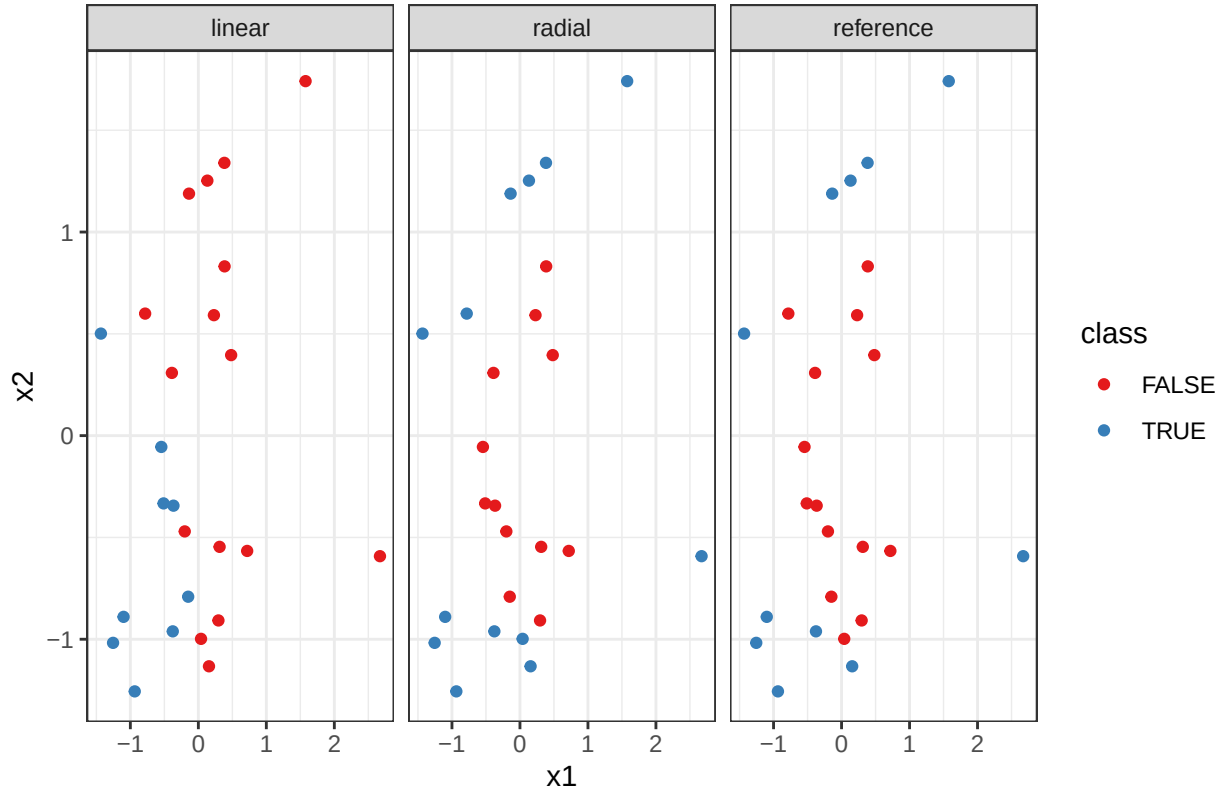



```
set.seed(5904)
split <- initial_split(two_class)
train <- training(split)
test <- testing(split)

cv_ctrl <- trainControl(method = "cv",
                        number = 10,
                        savePredictions = "final")

svm_radial <- train(
  as.factor(y) ~ .,
  data = two_class,
  method = "svmRadial",
  trControl = cv_ctrl,
)

svm_linear <- train(
  as.factor(y) ~ .,
  data = train,
  method = "svmLinear",
  tuneLength = 10,
  trControl = cv_ctrl
)

train %>%
  bind_cols(linear = as.vector(predict(svm_linear, train))) %>%
```

```
  bind_cols(radial = as.vector(predict(svm_radial, train))) %>%
  mutate(reference = as.character(y)) %>%
  pivot_longer(cols = c(linear, radial, reference),
               names_to = "svm",
               values_to = "class") %>%
  ggplot(aes(x = x1, y = x2, color = class)) +
  geom_point() +
  facet_wrap(~svm) +
  scale_color_brewer(palette = "Set1") +
  labs(title = "Fig1. Two-class Non-linear Prediction on Training Data")
```


```
test %>%
  bind_cols(linear = as.vector(predict(svm_linear, test))) %>%
  bind_cols(radial = as.vector(predict(svm_radial, test))) %>%
  mutate(reference = as.character(y))%>%
  pivot_longer(cols = c(linear, radial, reference),
               names_to = "svm",
               values_to = "class") %>%
  ggplot(aes(x = x1, y = x2, color = class)) +
  geom_point() +
  facet_wrap(~svm) +
  scale_color_brewer(palette = "Set1") +
  labs(title = "Fig. 2 Two-class Non-linear Prediction on Test Data")
```

Fig. 2 Two−class Non−linear Prediction on Test Data




```
test_mse.linear <- mean(predict(svm_linear, test) != test$y)
train_mse.linear <- mean(predict(svm_linear, train) != train$y)
test_mse.radial <- mean(predict(svm_radial, test) != test$y)
train_mse.radial <- mean(predict(svm_radial, train) != train$y)
tibble(Model = c("Linear", "Radial"),
       Train_MSE = c(train_mse.linear, train_mse.radial),
       Test_MSE = c(test_mse.linear, test_mse.radial)) %>% kable(caption = "SVM Linear and Radial Train/
```

Table 1: SVM Linear and Radial Train/Test MSE

Model	Train_MSE	Test_MSE
Linear	0.3333333	0.40
Radial	0.0133333	0.08
#### SVM	vs. logistic	regression

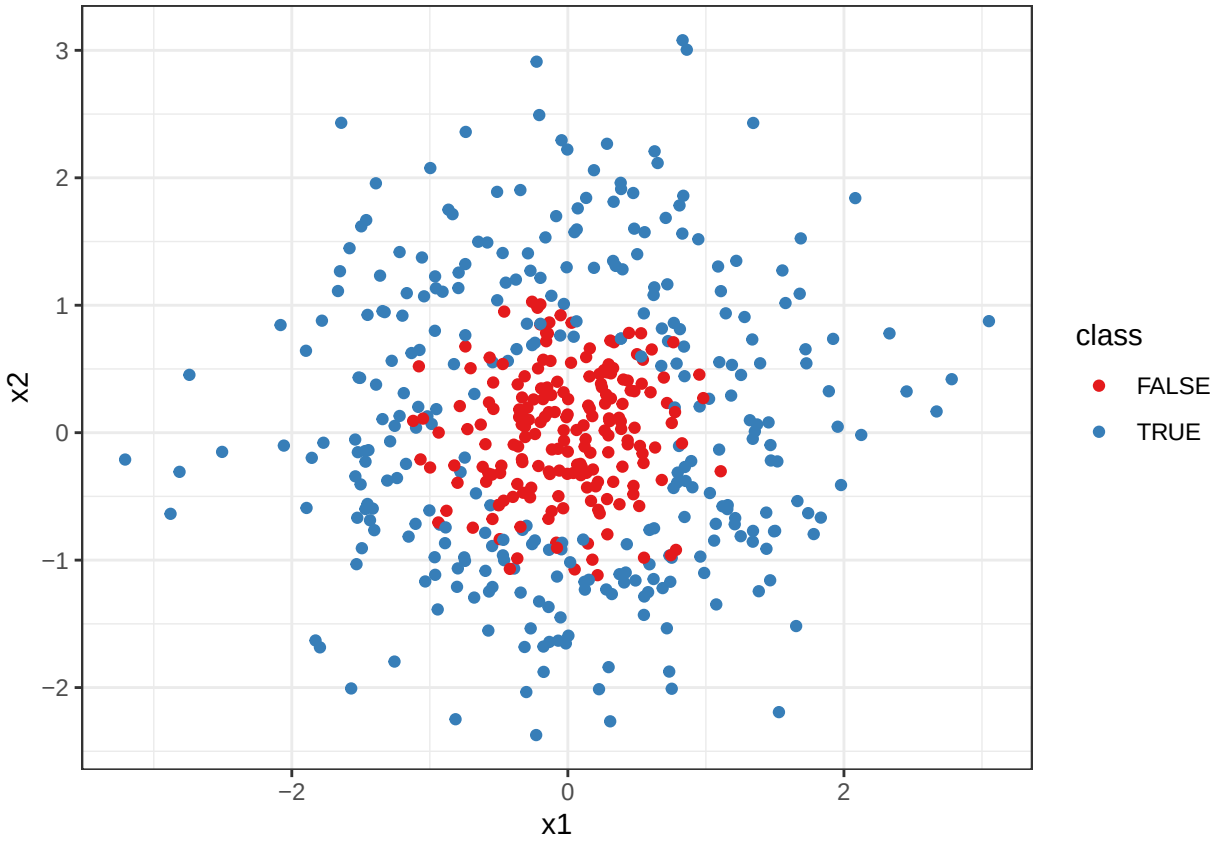
2. (5 points) Generate a data set with $n = 500$ and $p = 2$, such that the observations belong to two classes with some **overlapping, non-linear** boundary between them.

```
set.seed(5904)
two_class <- tibble(x1 = rnorm(500),
                    x2 = rnorm(500),
                    y = x1^2 + x2^2 > sample(c(0.5, 1.5), 500, replace =TRUE))
```

3. (5 points) Plot the observations with colors according to their class labels (y). Your plot should display

X_1 on the x-axis and X_2 on the y-axis.

```
two_class %>%
  ggplot(aes(x=x1, y = x2, color = y)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class")
```



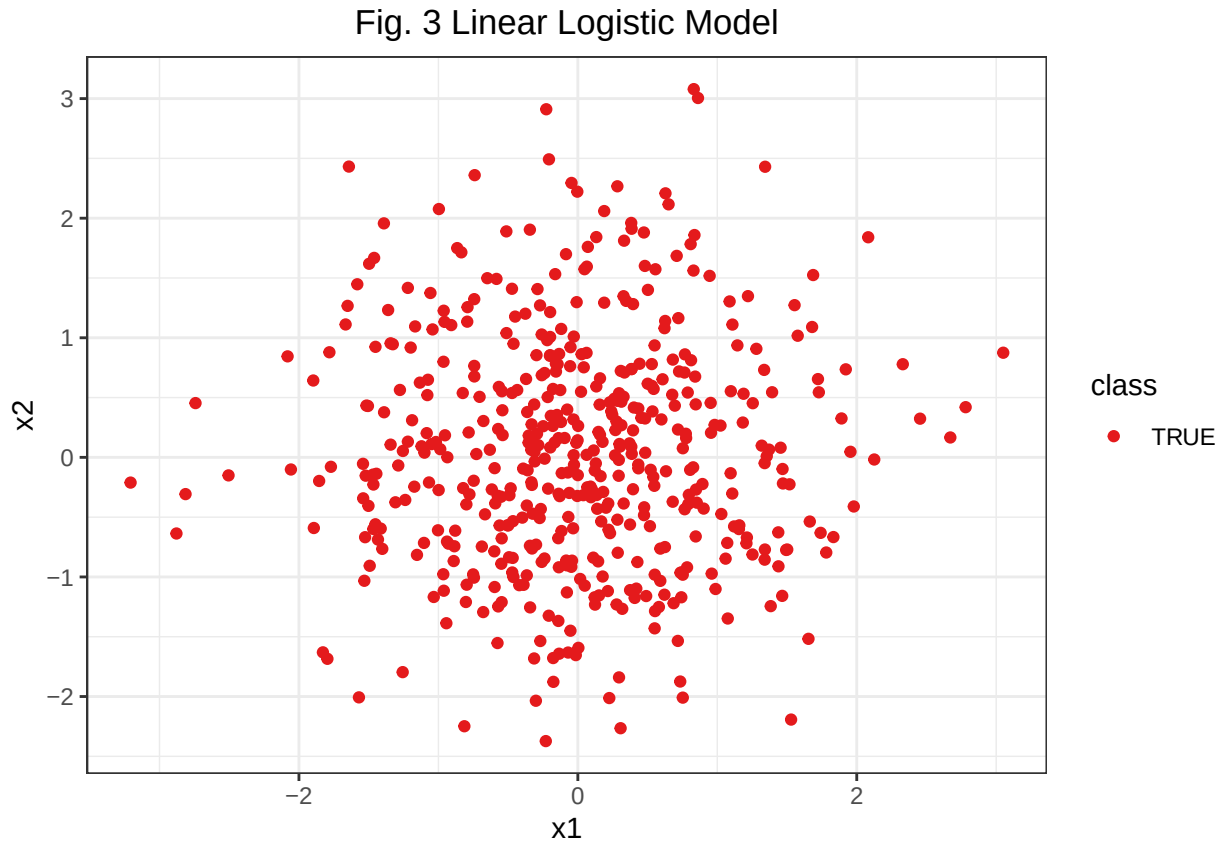
4. (5 points) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

```
logistic.linear <- train(
  as.factor(y)~.,
  data = two_class,
  method = "glm",
  trControl = cv_ctrl
)
```

5. (5 points) Obtain a predicted class label for each observation based on the logistic model previously fit. Plot the observations, colored according to the *predicted* class labels (*the predicted decision boundary should look linear*).

(Some issue here as in Q.1. Because the actual decision boundary is a circle, a linear model can only return 1 class.)

```
two_class %>%
  bind_cols(linear = as.vector(predict(logistic.linear, two_class))) %>%
  ggplot(aes(x=x1, y = x2, color = linear)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class",
       title = "Fig. 3 Linear Logistic Model")
```

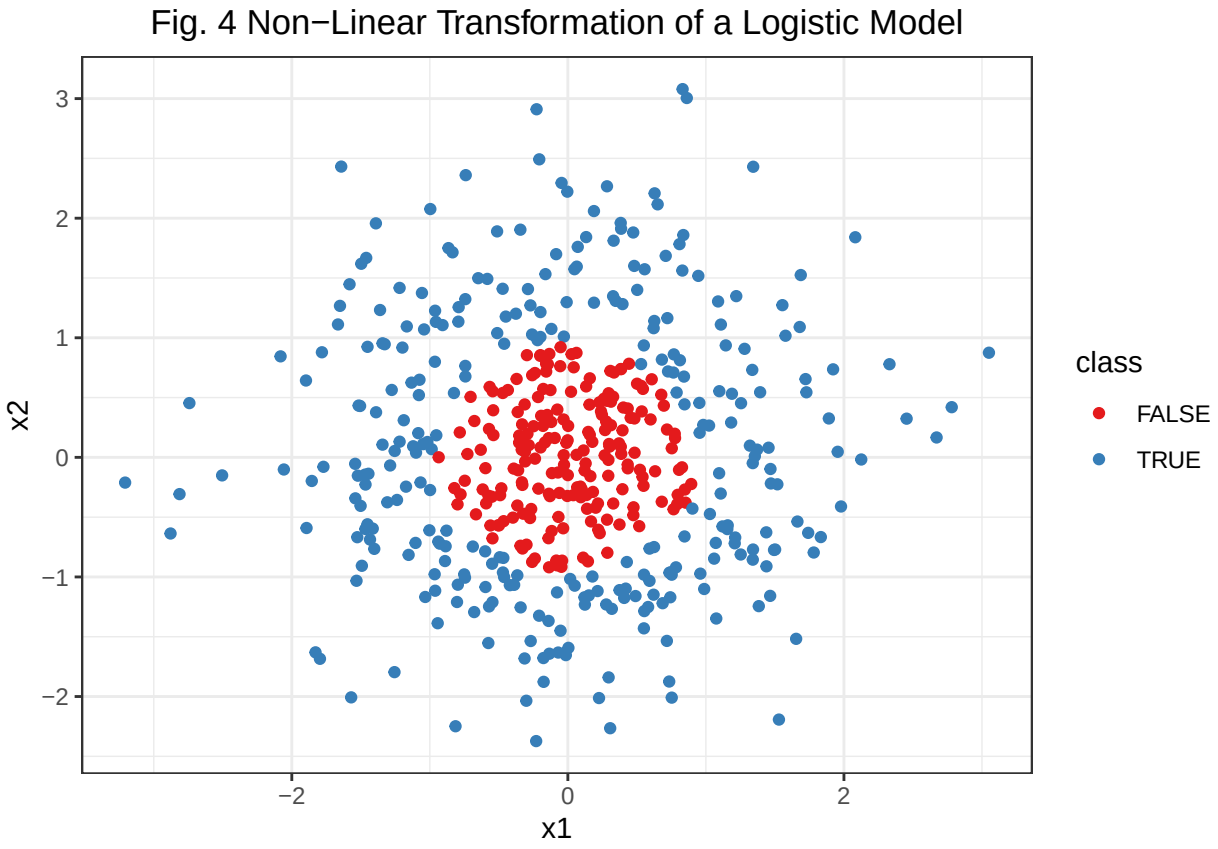


6. (5 points) Now fit a logistic regression model to the data, but this time using some *non-linear function* of both X_1 and X_2 as predictors (e.g. $X_1^2, X_1 \times X_2, \log(X_2)$, and so on).

```
logistic.nonlinear <- train(
  as.factor(y)~cos(x1) + cos(x2),
  data = two_class,
  method = "glm",
  trControl = cv_ctrl
)
```

7. (5 points) Now, obtain a predicted class label for each observation based on the fitted model with non-linear transformations of the X features in the previous question. Plot the observations, colored according to the new *predicted* class labels from the non-linear model (*the decision boundary should now be obviously non-linear*). If it is not, then repeat earlier steps until you come up with an example in which the predicted class labels and the resultant decision boundary are clearly non-linear.

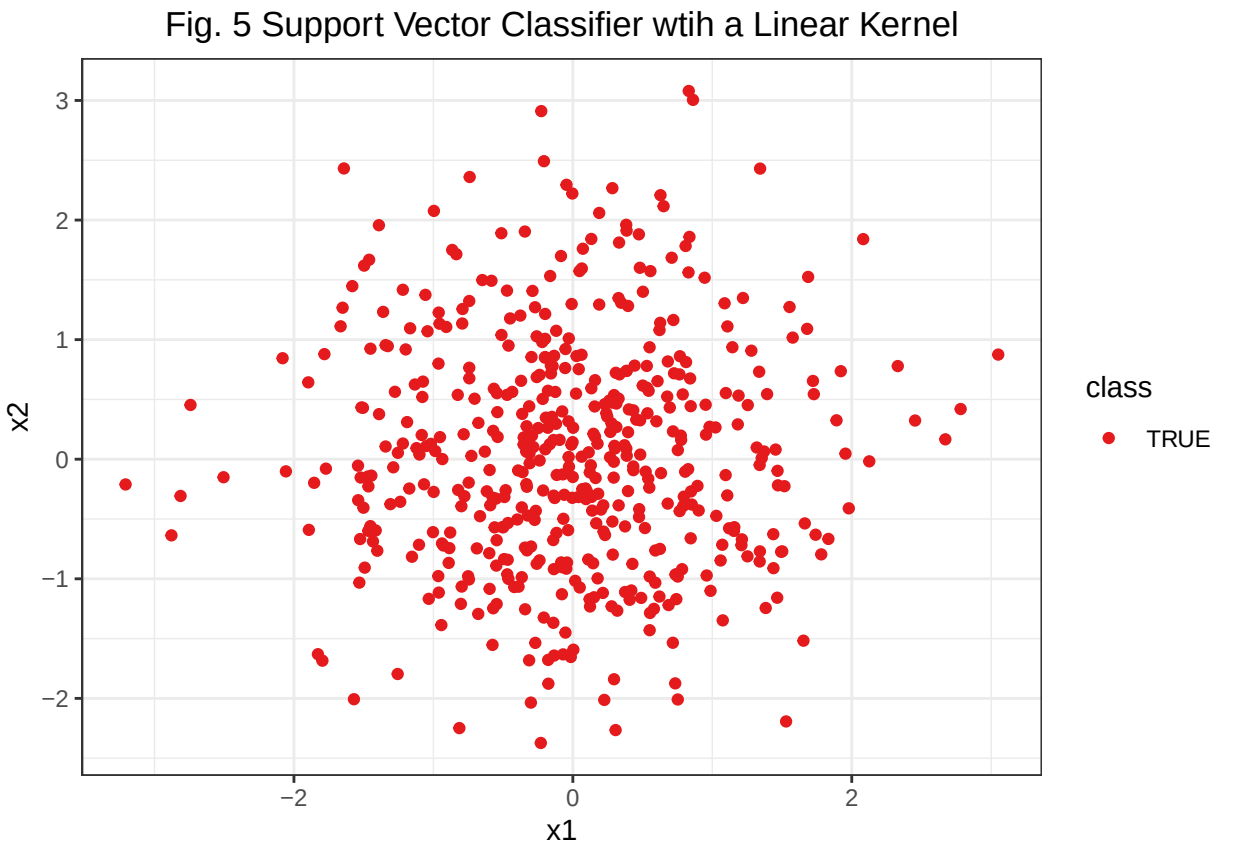
```
two_class %>%
  bind_cols(nonlinear = as.vector(predict(logistic.nonlinear, two_class))) %>%
  ggplot(aes(x=x1, y = x2, color = nonlinear)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class",
       title = "Fig. 4 Non-Linear Transformation of a Logistic Model")
```



8. (5 points) Now, fit a support vector classifier (linear kernel) to the data with *original* X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the *predicted* class labels.

```
svm.linear <- train(
  as.factor(y)~x1 + x2,
  data = two_class,
  method = "svmLinear",
  trControl = cv_ctrl
)

two_class %>%
  bind_cols(linear = as.vector(predict(svm.linear, two_class))) %>%
  ggplot(aes(x=x1, y = x2, color = linear)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
```

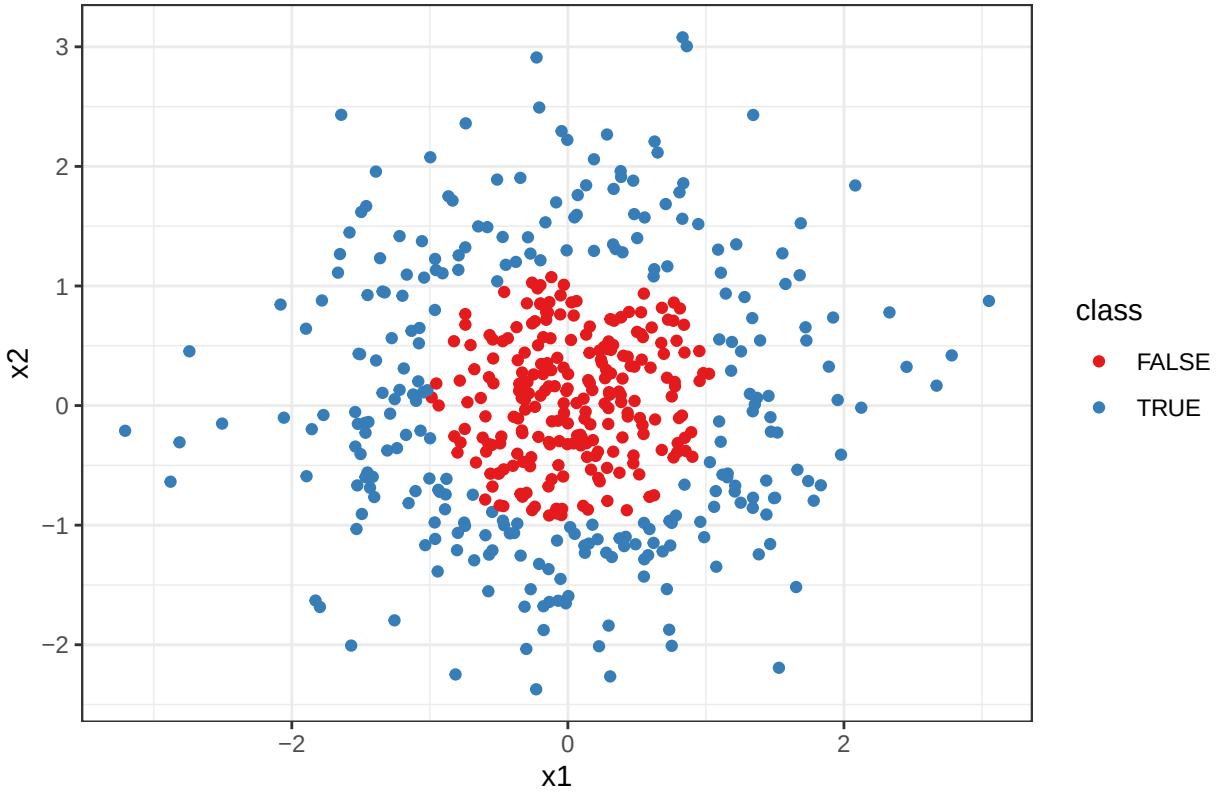
```
  labs(color = "class",
       title = "Fig. 5 Support Vector Classifier wtih a Linear Kernel")
```



9. (5 points) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each observation. Plot the observations, colored according to the *predicted* class labels.

```
svm.nonlinear <- train(
  as.factor(y)~x1 + x2,
  data = two_class,
  method = "svmPoly",
  trControl = cv_ctrl
)

two_class %>%
  bind_cols(nonlinear = as.vector(predict(svm.nonlinear, two_class))) %>%
  ggplot(aes(x=x1, y = x2, color = nonlinear)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class",
       title = "Fig. 6 Support Vector Machine wtih a Polynomial Kernel")
```

Fig. 6 Support Vector Machine wtih a Polynomial Kernel



10. (5 points) Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

```
tibble(Model = c("Linear Logistic", "NonLinear Logistic", "SVM Linear Kernel", "SVM Poly Kernel"),
       Accuracy = c(mean(predict(logistic.linear, two_class) == two_class$y),
                    mean(predict(logistic.nonlinear, two_class) == two_class$y),
                    mean(predict(svm.linear, two_class) == two_class$y),
                    mean(predict(svm.nonlinear, two_class) == two_class$y))) %>%
  kable(caption = "SVM and Logistic Linear vs NonLinear Prediction Accuracy")
```

Table 2: SVM and Logistic Linear vs NonLinear Prediction Accuracy

Model	Accuracy
Linear Logistic	0.616
NonLinear Logistic	0.876
SVM Linear Kernel	0.616
SVM Poly Kernel	0.852

From the four figures and above, we are able to see that the linear logistical model and support vector classifer with a linear kernel yield identical results. Because the underlying decision boundary is a circle, a linear model can only return 1 class. The 0.616 accuracy comes from the fact that the majority class is 61.6% of the whole sample. Therefore, there is no substantial differences between using a logistical or a

linear support vector classifier in my case. The two models are very comparable.

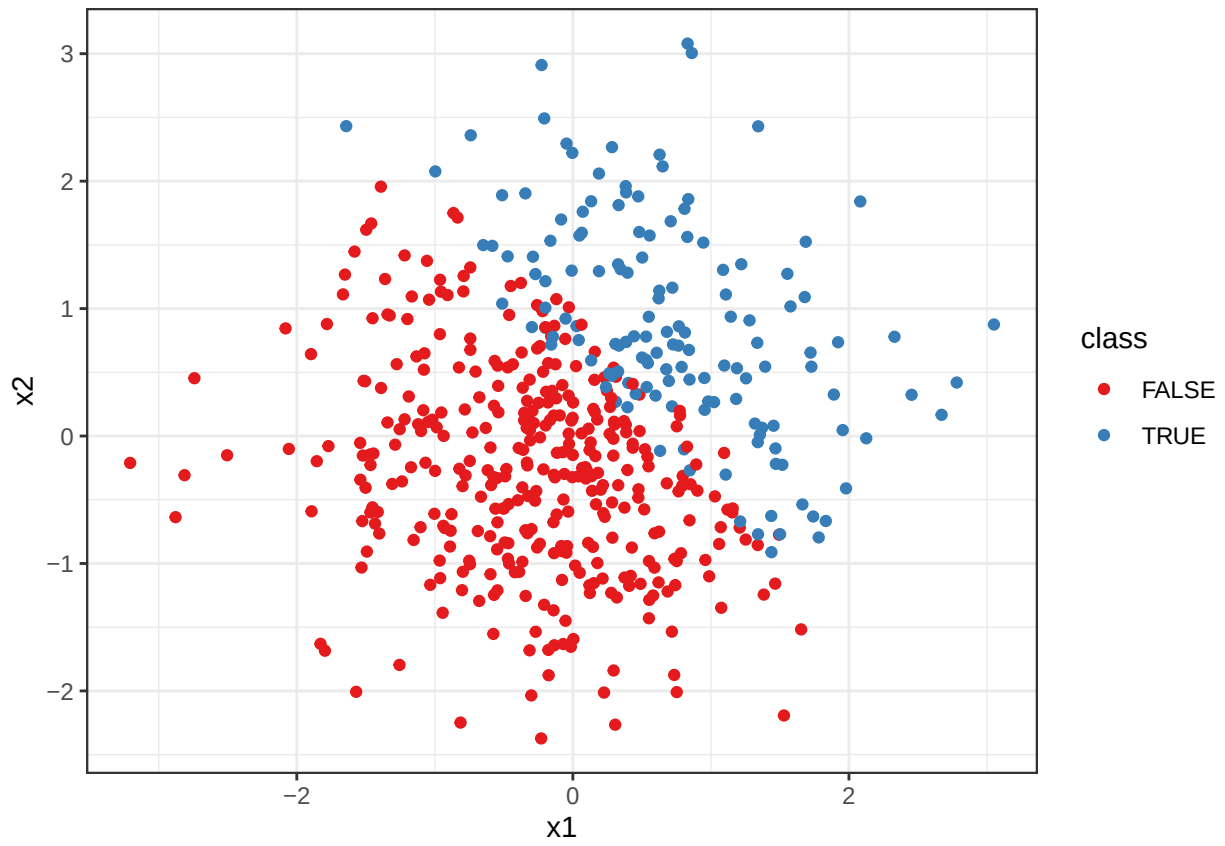
Between the two nonlinear models, logistic model with non-linear transformation outperforms the SVM with a polynomial kernel by about 0.02 in prediction accuracy. Becasue of the "overlapping" data created in the simulation process, logistic model in general is more sensitive to outliers (the overlapping points in this case), while the SVM attempts to maximize the distance between the two (less sensitive to ouliers). This might be reason why SVM is outperformed by logistic model.

Overall, nonlinear model performs better than linear ones.

Tuning cost

11. (5 points) Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

```
set.seed(5904)
two_class <- tibble(x1 = rnorm(500),
                    x2 = rnorm(500),
                    y = x1 + x2 > sample(c(1, 0.5), replace = TRUE))

two_class %>%
  ggplot(aes(x=x1, y = x2, color = y)) +
  geom_point() +
  scale_color_brewer(palette = "Set1") +
  labs(color = "class")
```



12. (5 points) Compute the cross-validation error rates for support vector classifiers with a range of `cost` values. How many training errors are made for each value of `cost` considered, and how does this relate to the cross-validation errors obtained?

Overall, as cost increases, both cv error rate and training error rate decrease. The cv error rate and training error rate seem to be related because the cross-validation data comes from the training data. When cost is equal to 10, we have the lowest cv and training error rate. However, after cost is increased to 0.1, marginal reduction of error rate starts to decline. In other words, when cost is increased from 0.001 to 0.01 and from 0.01 to 0.1, the marginal improvement of error rate is close to around 0.14~0.8. If we further increase the cost, the cv error rates are only reduced by 0.01 or even less. Also, there is no substanial difference between using a cost value of 0.0001 and 0.001. And we have the largest reduction of error rate when cost value is increased from 0.01 to 0.1.

```
split <- initial_split(two_class)
train <- training(split)
test <- testing(split)

costs <- c(0.0001, 0.001, 0.01, 0.1, 1, 10)

svm.cost <- train(
  as.factor(y)~x1 + x2,
  data = train,
  method = "svmLinear",
  trControl = cv_ctrl,
  tuneGrid = expand.grid(C = costs)
)

costs %>% map_dbl(function(x) mean(predict(
  train(as.factor(y)~x1 + x2,
        data = train,
        method = "svmLinear",
        trControl = cv_ctrl,
        tuneGrid = expand.grid(C = x)),
  train) != train$y)) %>%
  as.data.frame() %>%
  `colnames<-`("train_error") %>%
  {bind_cols(svm.cost$results, .)} %>%
  mutate(cv_error = 1- Accuracy) %>%
  select(C, cv_error, train_error) %>% kable()
```

C	cv_error	train_error
1e-04	0.2825747	0.2826667
1e-03	0.2825747	0.2826667
1e-02	0.1438834	0.1253333
1e-01	0.0665718	0.0666667
1e+00	0.0665007	0.0533333
1e+01	0.0638691	0.0506667

13. (5 points) Generate an appropriate test data set, and compute the test errors corresponding to each of the values of `cost` considered. Which value of `cost` leads to the fewest test errors, and how does this compare to the values of `cost` that yield the fewest training errors and the fewest cross-validation errors?

As cost increases, test error rate decreases but as the cost value reaches a certain point, the test error rate starts to rise. Using a cost value of 0.1 yields the lowest test error rate but not the lowest training or cv error rate. We still see the huge decrease of error rate when cost value is increased from 0.01 to 0.1.

```
costs %>% map_dbl(function(x) mean(predict(
  train(as.factor(y)~x1 + x2,
        data = test,
        method = "svmLinear",
        trControl = cv_ctrl,
        tuneGrid = expand.grid(C = x)),
  test) != test$y)) %>%
  as.data.frame() %>%
  `colnames<-`("test_error") %>%
  {bind_cols(svm.cost$results, .)} %>%
  select(C, test_error) %>% kable()
```

C	test_error
1e-04	0.296
1e-03	0.296
1e-02	0.280
1e-01	0.080
1e+00	0.088
1e+01	0.096

14. (5 points) Discuss your results.

According to the results above, we are able to conclude that increasing cost value reduces train, cv and test error rate. However, the marginal reduction of training and cv error rate becomes small and the test error rate starts to increase when cost value reaches a certain point. This phenomenon can be explained by the fact that the two classes are created as "barely linearly separable" in our simulation. It is impossible to obstain a perfect linear decision boundary. Therefore, when cost value is too large, our model is prone to overfit.

Application: Predicting attitudes towards racist college professors

15. (5 points) Fit a support vector classifier to predict `colrac` as a function of all available predictors, using 10-fold cross-validation to find an optimal value for `cost`. Report the CV errors associated with different values of cost, and discuss your results.

The cv error rate decreases at first when we increase our cost value but starts to increase after cost value is equal to 0.001. Similar to our discussion and results in the previous question, it is very likely that because the underlying relationship is not linear, if we set the cost value too high, we end up with overfitting our model (even in the training set). Overall, an accuracy of around 80% (using the optimal cost value) is relatively a good prediction result.

```
gss_train <- read.csv("./data/gss_train.csv")
gss_test <- read.csv("./data/gss_test.csv")

train.y <- gss_train$colrac
train.x <- gss_train %>% select(- colrac) %>% mutate_all(scale)
```

```
test.y <- gss_test$colrac
test.x <- gss_test %>% select(- colrac) %>% mutate_all(scale)


svm.linear <- train(
  y = as.factor(train.y),
  x = train.x,
  method = "svmLinear",
  trControl = cv_ctrl,
  tuneGrid = expand.grid(C = costs))

svm.linear$results %>%
  mutate(CV.error = 1 - Accuracy) %>%
  select(C, CV.error) %>% kable()
```

C	CV.error
1e-04	0.4726555
1e-03	0.2228279
1e-02	0.1991565
1e-01	0.2005124
1e+00	0.2059269
1e+01	0.2045755

16. (15 points) Repeat the previous question, but this time using SVMs with **radial** *and* **polynomial** basis kernels, with different values for `gamma` and `degree` and `cost`. **Present and discuss your results** (e.g., fit, compare kernels, cost, substantive conclusions across fits, etc.).

```
svm.radial <- train(
  y = as.factor(train.y),
  x = train.x,
  method = "svmRadial",
  trControl = cv_ctrl,
  tuneGrid = expand.grid(C = costs, sigma = costs))

svm.poly <- train(
  y = as.factor(train.y),
  x = train.x,
  method = "svmPoly",
  trControl = cv_ctrl,
  tuneGrid = expand.grid(C = costs, degree = c(2:10), scale = 1))
```

SVM with a Radial Kernel

The table below shows 10 model with the lowest CV.error rate. We can see that using a cost value of 10 and sigma of 0.001 yields the lowest cv error rate. However, the difference in cv error rate is very minimal between the best-tuned results and the second optimal results. And the accuracy is very close the results of SVM with a linear kernel.

```
svm.radial$results %>%
  mutate(CV.error = 1 - Accuracy) %>%
  select(C, sigma ,CV.error) %>%
```

```
  arrange(CV.error)%>%
  {.[1:10, ]} %>%
  kable()
```

C	sigma	CV.error
10.0	1e-03	0.1991809
1.0	1e-02	0.2005369
10.0	1e-02	0.2059469
1.0	1e-03	0.2120236
10.0	1e-04	0.2140643
0.1	1e-02	0.2208439
0.1	1e-03	0.2539161
1.0	1e-04	0.2552810
10.0	1e-01	0.2937039
1.0	1e-01	0.3166725

SVM with a Poly Kernel

The table below shows 10 model with the lowest CV.error rate. We can see that using a polynomial degree of 5 yields the lowest cv error rate, regardless of what the cost value is. This similar pattern is also seen with a polynomial degree of 3, where the cv error is the same across different cost values. However, the overall performance of the SVM with polynomial kernels is less favorable than the results from SVM with linear or radial kernel. The cv error rate even with the best-tuned model is about 0.02 higher than the cv error rate of the previous two models.

```
svm.poly$results %>%
  mutate(CV.error = 1 - Accuracy) %>%
  select(C, degree, CV.error) %>%
  arrange(CV.error)%>%
  {.[1:10, ]} %>%
  kable()
```

C	degree	CV.error
1e-04	5	0.2166961
1e-03	5	0.2166961
1e-02	5	0.2166961
1e-01	5	0.2166961
1e+00	5	0.2166961
1e+01	5	0.2166961
1e-03	3	0.2207775
1e-02	3	0.2207775
1e-01	3	0.2207775
1e+00	3	0.2207775

We are also interested in how the three best-tuned model compare on the test set. Similar to the cv error rate, the SVM model with a polynominal kernal has the highest error rate, while SVM with linear and radial kernel yield similar results. However, the SVM radial slightly outperforms the linear one. It is possible that the underlying relationship is not linear and hence the dymanics are really unlikely to be captured by a model.

```r
tibble(Kernel = c("Linear", "Radial", "Poly"),
       Test_MSE =c(
         mean(predict(svm.linear, test.x) != test.y),
         mean(predict(svm.radial, test.x) != test.y),
         mean(predict(svm.poly, test.x) != test.y))) %>% kable()
```

Kernel	Test_MSE
Linear	0.2129817
Radial	0.2048682
Poly	0.2434077