DAR Coursework

```r
# renv::install('rmarkdown')
# renv::install('knitr')
# renv::install('tinytex')
# tinytex::install_tinytex()
# renv::install('ISLR')
# renv::install('aod')
# renv::install('ROCR')
library(rmarkdown)
library(knitr)
library(ISLR)
library(aod)
library(ROCR)
library(e1071)
```

1. Statistical learning methods (10%)

For each of parts (a) through (d), indicate whether we would generally expect the performance of a non-parametric statistical learning method to be better or worse than a parametric method. Justify your answer.

(a) The number of predictors p is large, and the number of observations n is small. (2%)

We would generally expect a non-parametric method to perform worse than a parametric method, although it will be a close call. Parametric methods are more suited to small numbers of observations as they try to fit an assumed functional form, which makes it simpler, more explainable, and require less data. Non-parametric methods are more suited to higher numbers of predictors as they are more flexible, allowing them to fit more functional forms, but require more observations to prevent overfitting precisely because they are more powerful. In this case, overfitting from a non-parametric method will likely be a bigger problem than a parametric method not being able to match the actual underlying function. A simple model may still provide a reasonably accurate prediction.

(b) The sample size n is large, and the number of predictors p is also large. (2%)

We would generally expect a non-parametric method to perform better than a parametric method. With a large number of predictors, the underlying function can take many more forms. Parametric methods tend to assume a functional form, so they are constrained to this assumed form, making them a poorer fit for the myriad possible underlying forms. Non-parametric methods are more flexible, where they can fit more functional forms, making them more suitable for larger numbers of predictors. With a large sample size, there is less risk of overfitting, and there will be more sufficient data to provide enough information for all of the predictors.

(c) The sample size n is small, and the relationship between the predictors and response is highly linear. (3%)

We would generally expect a non-parametric method to perform worse than a parametric method. The sample size is small, so a non-parametric method will likely overfit. Even if the sample size was sufficiently large, the model may be overly complex for a relatively simple linear relationship, which results in overfitting again, plus it will be more difficult to interpret. Since we know beforehand that there is an existing relationship between the predictors and response, a parametric method can assume a functional form that is similar to this relationship, which simplifies the problem of estimating the underlying functional form to estimating a set of parameters, making it easier to compute and understand, and likely achieves a good fit. The sample size requirements will also not be as big as a non-parametric method.

(d) The standard deviation of the error terms, i.e. $\sigma = \text{sd}(\varepsilon)$, is extremely high. (3%)

We would generally expect a non-parametric method to perform better than a parametric method. Because of the error term, our parametric model will never perfectly predict the response. If the standard deviation of the error term is low, the error term almost becomes a constant and just another parameter of a consistent population model. But when the error term has a high standard deviation, it means that on average the predicted response will be off by that high amount even if the model is 'correct'. If the standard deviation is extremely high, and on average the predicted response will be very off, then perhaps the assumed functional form of the parametric method is simply not a good fit. The underlying functional form is probably more complex than we thought, or the distribution of the data do not follow the assumptions of a parametric method, so a flexible non-parametric method will work better.

2. Linear regression (20%)

This question involves the `Auto` dataset included in the "ISLR" package.

(a) Use the `lm()` function to perform a simple linear regression with `acceleration` as the response and `cylinders` as the predictor. Use the `summary()` function to print the results. Comment on the output. For example:

```r
# Model the least squares line.
modelLinearReg <- lm(acceleration ~ cylinders, Auto)
print(summary(modelLinearReg))
```

```
##
## Call:
## lm(formula = acceleration ~ cylinders, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4778 -1.7428 -0.2428  1.3897  8.7222
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.0078     0.4052   49.38   <2e-16 ***
## cylinders    -0.8163     0.0707  -11.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.385 on 390 degrees of freedom
## Multiple R-squared:  0.2547, Adjusted R-squared:  0.2528
## F-statistic: 133.3 on 1 and 390 DF,  p-value: < 2.2e-16
```

i. Is there a relationship between the predictor and the response? (3%)

There is a linear relationship between cylinders and acceleration. The p-value for the coefficient of cylinders is very small, at lower than 0.1%, which is below our desired p-value. So we reject the null hypothesis that the coefficient of cylinders is 0. The adjusted R-squared is also not 0, so there is a linear relationship.

ii. How strong is the relationship between the predictor and the response? (3%)

The relationship is not very strong, because although the adjusted R-squared is not 0, it is only at 0.2528. This means that the model only explains 25.28% of the variation in acceleration.

iii. Is the relationship between the predictor and the response positive or negative? (3%)

The relationship is negative because the coefficient of cylinders is negative.

iv. What is the predicted `acceleration` associated with an `cylinders` of 3.0? What are the associated 99% confidence and prediction intervals? (3%)

```r
# Prepare data.
uniqueCylinders = sort(unique(Auto[, 'cylinders']), decreasing=FALSE)
print(uniqueCylinders)
```

```
## [1] 3 4 5 6 8
```

```r
# Predict on unique cylinder numbers.
prediction <- predict(
  modelLinearReg,
  newdata=data.frame(cylinders=uniqueCylinders)
)
print(prediction)
```

```
##        1        2        3        4        5
## 17.55906 16.74280 15.92655 15.11029 13.47779
```

```r
confidenceInterval <- predict(
  modelLinearReg,
  newdata=data.frame(cylinders=uniqueCylinders),
  interval='confidence',
  level=.99
)
print(confidenceInterval)
```

```
##        fit      lwr      upr
## 1 17.55906 17.00962 18.10849
## 2 16.74280 16.33076 17.15484
## 3 15.92655 15.60302 16.25008
## 4 15.11029 14.78388 15.43671
## 5 13.47779 12.91987 14.03571
```

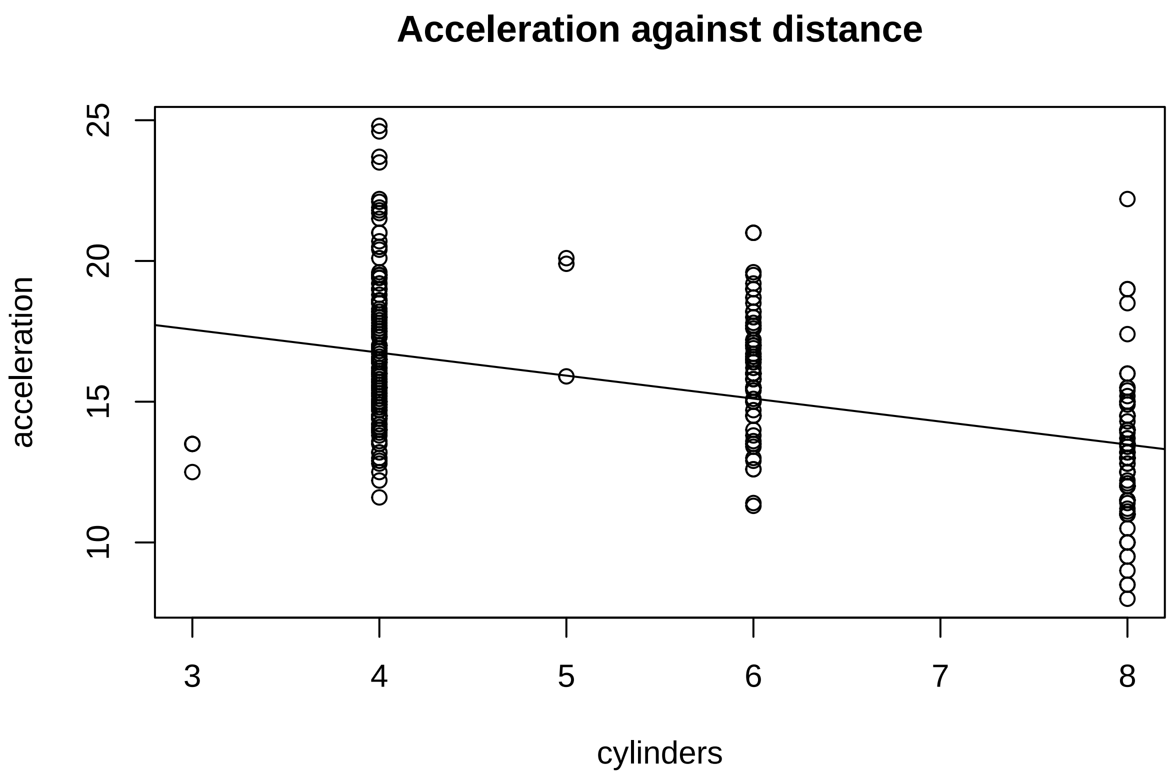
```r
predictionInterval <- predict(
  modelLinearReg,
  newdata=data.frame(cylinders=uniqueCylinders),
  interval='prediction',
  level=.99
)
print(predictionInterval)
```

```
##        fit       lwr      upr
## 1 17.55906 11.361636 23.75648
## 2 16.74280 10.556048 22.92956
## 3 15.92655  9.745058 22.10804
## 4 15.11029  8.928652 21.29194
## 5 13.47779  7.279606 19.67597
```

For 3.0 cylinders, the predicted acceleration is 17.55906, the confidence interval is 17.00962 - 18.10849, and the prediction interval is 11.361636 - 23.75648. The confidence and prediction intervals for the rest of the cylinders are shown above, with the row numbers corresponding to the row numbers in the uniqueCylinders vector representing the number of cylinders.

(b) Plot the response and the predictor. Use the `abline()` function to display the least squares regression line. (3%)

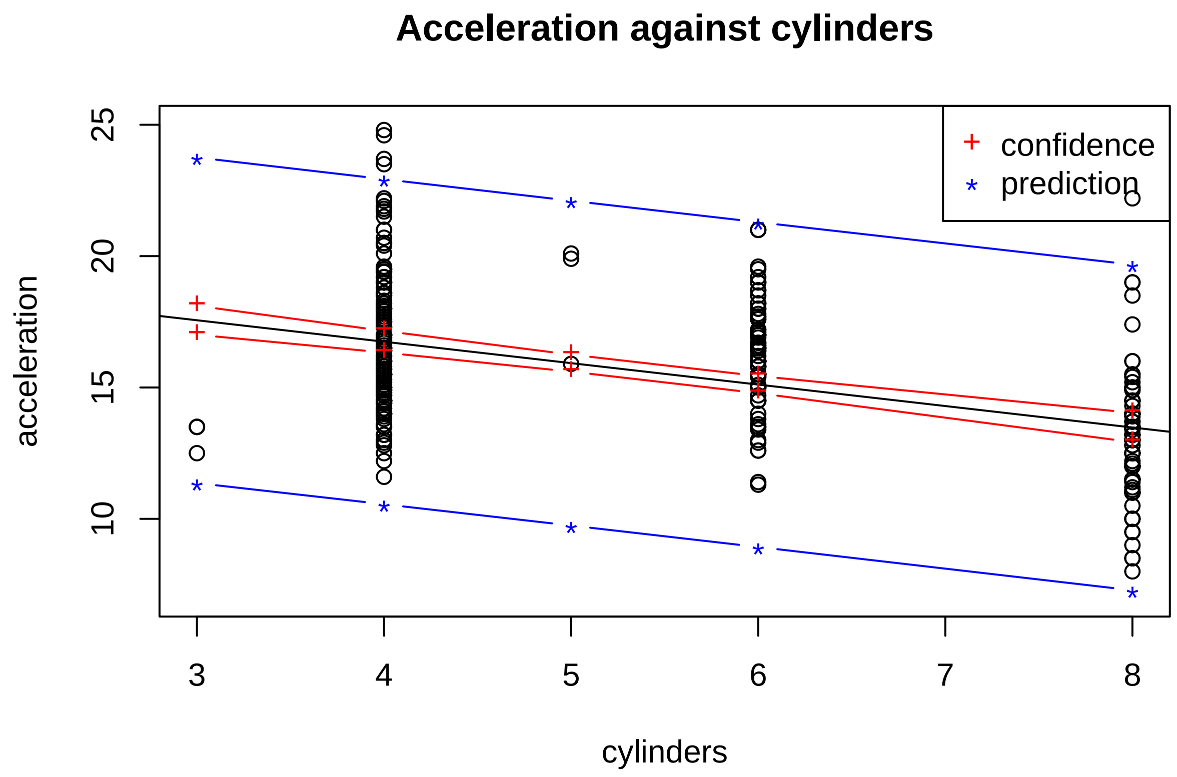
```r
# Plot the dataset.
plot(
  acceleration ~ cylinders,
  Auto,
  xlab='cylinders',
  ylab='acceleration',
  main='Acceleration against distance'
)
# Then plot the least squares line.
abline(modelLinearReg)
```



(c) Plot the 99% confidence interval and prediction interval in the same plot as (b) using different colours and legends. (5%)

```r
# Re-plot the dataset and least squares line.
plot(
  acceleration ~ cylinders,
  Auto,
  xlab='cylinders',
  ylab='acceleration',
  main='Acceleration against cylinders',
  ylim=c(7, 25)
)
abline(modelLinearReg)

# Plot the lines for confidence and prediction intervals.
lines(
  confidenceInterval[, 'lwr'] ~ uniqueCylinders,
  col='red',
  pch='+',
  type='b'
)
lines(
  confidenceInterval[, 'upr'] ~ uniqueCylinders,
  col='red',
  pch='+',
  type='b'
)
lines(
  predictionInterval[, 'lwr'] ~ uniqueCylinders,
  col='blue',
  pch='*',
  type='b'
)
lines(
  predictionInterval[, 'upr'] ~ uniqueCylinders,
  col='blue',
  pch='*',
  type='b'
)
legend(
  'topright',
  pch=c('+', '*'),
  col=c('red', 'blue'),
  legend=c('confidence', 'prediction')
)
```



3. Bayesian networks and naïve Bayes classifiers. (30%)

a) Given a training dataset including 30 observations and a Bayesian network indicating the relationships between 3 features (i.e. Income, Student and Credit Rate) and the class attribute (i.e. Buy Computer), please create the conditional probability tables by hand. (10%)

```r
# Inspect data.
bayesianTrainingData = read.table(
  'question 3.txt',
  header=TRUE,
  sep=' '
)

# Conditional probability table for Credit Rating.
income = c('low', 'low', 'low', 'low', 'high', 'high', 'high', 'high')
student = c('false', 'false', 'true', 'true', 'false', 'false', 'true', 'true')
buyComputer = c('no', 'yes', 'no', 'yes', 'no', 'yes', 'no', 'yes')
fairCreditRating = c(0.667, 0.333, 0.75, 0.5, 0.333, 0.333, 0.75, 0.5)
excellentCreditRating = c(0.333, 0.667, 0.25, 0.5, 0.667, 0.667, 0.25, 0.5)
creditRatingCpt = data.frame(
  cbind(income, student, buyComputer, fairCreditRating, excellentCreditRating)
)
colnames(creditRatingCpt) <- c(
  'Income',
  'Student',
  'Buy Computer',
  'Credit Rating = Fair',
  'Credit Rating = Excellent'
)
print.data.frame(creditRatingCpt, row.names=FALSE)
```

```
##  Income Student Buy Computer Credit Rating = Fair Credit Rating = Excellent
##     low   false           no                0.667                     0.333
##     low   false          yes                0.333                     0.667
##     low    true           no                 0.75                      0.25
##     low    true          yes                  0.5                       0.5
##    high   false           no                0.333                     0.667
##    high   false          yes                0.333                     0.667
##    high    true           no                 0.75                      0.25
##    high    true          yes                  0.5                       0.5
```

```r
# Conditional probability table for Income.
buyComputer <- c('no', 'yes')
lowIncome <- c(0.611, 0.583)
highIncome <- c(0.389, 0.417)
incomeCpt <- data.frame(cbind(buyComputer, lowIncome, highIncome))
colnames(incomeCpt) <- c('Buy Computer', 'Income = Low', 'Income = High')
print.data.frame(incomeCpt, row.names=FALSE)
```

```
##  Buy Computer Income = Low Income = High
##            no        0.611         0.389
##           yes        0.583         0.417
```

```r
# Conditional probability table for Student.
falseStudent <- c(0.333, 0.5)
trueStudent <- c(0.667, 0.5)
studentCpt <- data.frame(cbind(buyComputer, falseStudent, trueStudent))
```

```r
colnames(studentCpt) <- c('Buy Computer', 'Student = False', 'Student = True')
print.data.frame(studentCpt, row.names=FALSE)
```

```
##  Buy Computer Student = False Student = True
##            no           0.333          0.667
##           yes             0.5            0.5
```

```r
# Conditional probability table for Buy Computer.
noBuyComputer <- c(0.6)
yesBuyComputer <- c(0.4)
buyComputerCpt <- data.frame(cbind(noBuyComputer, yesBuyComputer))
colnames(buyComputerCpt) <- c('Buy Computer = No', 'Buy Computer = Yes')
print.data.frame(buyComputerCpt, row.names=FALSE)
```

```
##  Buy Computer = No Buy Computer = Yes
##                0.6                0.4
```

From top to bottom, the code output shows the conditional probability tables for Credit Rating, Income, Student, and Buy Computer respectively.

b) Make predictions for 2 testing observations by using a Bayesian network classifier. (5%)

For observation 31:

P(Buy Computer=No | Income=Low, Student=True, Credit Rating=Excellent)
is proportional to P(Income=Low, Student=True, Credit Rating=Excellent, Buy Computer=No)
= P(Income=Low | Buy Computer=No)
* P(Student=True | Buy Computer=No)
* P(Credit Rating=Excellent | Income=Low, Student=True, Buy Computer=No)
* P(Buy Computer=No)
= 0.611 * 0.667 * 0.25 * 0.6
= 0.06113055

P(Buy Computer=Yes | Income=Low, Student=True, Credit Rating=Excellent)
is proportional to P(Income=Low, Student=True, Credit Rating=Excellent, Buy Computer=Yes)
= P(Income=Low | Buy Computer=Yes)
* P(Student=True | Buy Computer=Yes)
* P(Credit Rating=Excellent | Income=Low, Student=True, Buy Computer=Yes)
* P(Buy Computer=Yes)
= 0.583 * 0.5 * 0.5 * 0.4
= 0.0583

Hence, the prediction for observation 31 is No, because P(Buy Computer=No | Income=Low, Student=True, Credit Rating=Excellent) is greater than P(Buy Computer=Yes | Income=Low, Student=True, Credit Rating=Excellent).

For observation 32:

P(Buy Computer=No | Income=High, Student=True, Credit Rating=Fair)
is proportional to P(Income=High, Student=True, Credit Rating=Fair, Buy Computer=No)
= P(Income=High | Buy Computer=No)
* P(Student=True | Buy Computer=No)
* P(Credit Rating=Fair | Income=High, Student=True, Buy Computer=No)
* P(Buy Computer=No)

= 0.389 * 0.667 * 0.75 * 0.6
= 0.11675835

P(Buy Computer=Yes | Income=High, Student=True, Credit Rating=Fair)
is proportional to P(Income=High, Student=True, Credit Rating=Fair, Buy Computer=Yes)
= P(Income=High | Buy Computer=Yes)
* P(Student=True | Buy Computer=Yes)
* P(Credit Rating=Fair | Income=High, Student=True, Buy Computer=Yes)
* P(Buy Computer=Yes)
= 0.417 * 0.5 * 0.5 * 0.4
= 0.0417

Hence, the prediction for observation 32 is No, because P(Buy Computer=No | Income=High, Student=True, Credit Rating=Fair) is greater than P(Buy Computer=Yes | Income=High, Student=True, Credit Rating=Fair).

c) Based on the conditional independence assumption between features, please create the conditional probability tables by hand. (10%)

```
# Conditional probability table for Credit Rating.
buyComputer <- c('no', 'yes')
fairCreditRating <-c(0.667, 0.417)
excellentCreditRating <- c(0.333, 0.583)
creditRatingCpt <- data.frame(
  cbind(buyComputer, fairCreditRating, excellentCreditRating)
)
colnames(creditRatingCpt) <- c(
  'Buy Computer',
  'Credit Rating = Fair',
  'Credit Rating = Excellent'
)
print.data.frame(creditRatingCpt, row.names=FALSE)
```

```
##  Buy Computer Credit Rating = Fair Credit Rating = Excellent
##            no                0.667                     0.333
##           yes                0.417                     0.583
```

```
# Conditional probability table for Income.
lowIncome <- c(0.611, 0.583)
highIncome <- c(0.389, 0.417)
incomeCpt <- data.frame(cbind(buyComputer, lowIncome, highIncome))
colnames(incomeCpt) <- c('Buy Computer', 'Income = Low', 'Income = High')
print.data.frame(incomeCpt, row.names=FALSE)
```

```
##  Buy Computer Income = Low Income = High
##            no        0.611         0.389
##           yes        0.583         0.417
```

```
# Conditional probability table for Student.
falseStudent <- c(0.333, 0.5)
trueStudent <- c(0.667, 0.5)
studentCpt <- data.frame(cbind(buyComputer, falseStudent, trueStudent))
colnames(studentCpt) <- c('Buy Computer', 'Student = False', 'Student = True')
print.data.frame(studentCpt, row.names=FALSE)
```

```
##  Buy Computer Student = False Student = True
##            no           0.333          0.667
##           yes             0.5            0.5
```

```
# Conditional probability table for Buy Computer.
noBuyComputer <- c(0.6)
yesBuyComputer <- c(0.4)
buyComputerCpt <- data.frame(cbind(noBuyComputer, yesBuyComputer))
colnames(buyComputerCpt) <- c('Buy Computer = No', 'Buy Computer = Yes')
print.data.frame(buyComputerCpt, row.names=FALSE)
```

```
##  Buy Computer = No Buy Computer = Yes
##                0.6                0.4
```

From top to bottom, the code output shows the conditional probability tables for Credit Rating, Income, Student, and Buy Computer respectively, based on the conditional independence assumption where the predictors are assumed to be independent of each other. Only the table for Credit Rating has changed from question 3b as it is the only predictor dependent on other predictors.

d) Make predictions for 2 testing observations by using a naïve Bayes classifier. (5%)

For observation 31:

P(Buy Computer=No | Income=Low, Student=True, Credit Rating=Excellent)
is proportional to P(Income=Low, Student=True, Credit Rating=Excellent, Buy Computer=No)
= P(Income=Low | Buy Computer=No)
* P(Student=True | Buy Computer=No)
* P(Credit Rating=Excellent | Buy Computer=No)
* P(Buy Computer=No)
= 0.611 * 0.667 * 0.333 * 0.6
= 0.0814258926

P(Buy Computer=Yes | Income=Low, Student=True, Credit Rating=Excellent)
is proportional to P(Income=Low, Student=True, Credit Rating=Excellent, Buy Computer=Yes)
= P(Income=Low | Buy Computer=Yes)
* P(Student=True | Buy Computer=Yes)
* P(Credit Rating=Excellent | Buy Computer=Yes)
* P(Buy Computer=Yes)
= 0.583 * 0.5 * 0.583 * 0.4
= 0.0679778

Hence, the prediction for observation 31 is No, because P(Buy Computer=No | Income=Low, Student=True, Credit Rating=Excellent) is greater than P(Buy Computer=Yes | Income=Low, Student=True, Credit Rating=Excellent).

For observation 32:

P(Buy Computer=No | Income=High, Student=True, Credit Rating=Fair)
is proportional to P(Income=High, Student=True, Credit Rating=Fair, Buy Computer=No)
= P(Income=High | Buy Computer=No)
* P(Student=True | Buy Computer=No)
* P(Credit Rating=Fair | Buy Computer=No)
* P(Buy Computer=No)
= 0.389 * 0.667 * 0.667 * 0.6
= 0.1038370926

P(Buy Computer=Yes | Income=High, Student=True, Credit Rating=Fair)
is proportional to P(Income=High, Student=True, Credit Rating=Fair, Buy Computer=Yes)
= P(Income=High | Buy Computer=Yes)
* P(Student=True | Buy Computer=Yes)
* P(Credit Rating=Fair | Buy Computer=Yes)
* P(Buy Computer=Yes)
= 0.417 * 0.5 * 0.417 * 0.4
= 0.0347778

Hence, the prediction for observation 32 is No, because P(Buy Computer=No | Income=High, Student=True, Credit Rating=Fair) is greater than P(Buy Computer=Yes | Income=High, Student=True, Credit Rating=Fair).

4. Predicting wine quality by using support vector machine classification algorithm. **(40%)**

a) Download the full wine quality training and testing datasets from Moodle, and use the training dataset to find out the optimal value of hyperparameter C for a linear kernel-based svm. Define the value of the random seed equals 1 and cost = c(0.01, 1, 100). (5%)

```r
# Prepare data.
dataTrain = read.table(
  'WineQuality_training.txt',
  header=TRUE,
  sep=',',
  dec='.'
)
dataTrain$quality <- as.factor(dataTrain$quality)
dataTest = read.table(
  'WineQuality_testing.txt',
  header=TRUE,
  sep=',',
  dec='.'
)
dataTest$quality <- as.factor(dataTest$quality)

# Comparing different costs with 10-fold cross-validation.
set.seed(1)
tunedSvmLinear <- tune(
  svm,
  quality ~ .,
  data=dataTrain,
  kernel='linear',
  ranges=list(cost=c(0.01, 1, 100))
)
print(summary(tunedSvmLinear))
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.238
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.2460000 0.01623896
## 2 1e+00 0.2380000 0.01886927
## 3 1e+02 0.2396667 0.01855589
```

The most optimal value of the cost hyperparameter is 1 as it produced the lowest error of 0.238. This is for tuning with scale set to TRUE. Tuning the SVM with scale set to FALSE takes too much time, so it is not performed.

b) Train a svm classifier by using the linear kernel and the corresponding optimal value of hyperparameter C, then make predictions on the testing dataset, report the classification accuracy. (10%)

```r
# With scaling.
modelSvmLinear <- svm(
  quality ~ .,
  data=dataTrain,
  kernel='linear',
  cost=1,
  scale=TRUE
)
pred <- predict(modelSvmLinear, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.6825
```

```r
# Without scaling.
modelSvmLinear <- svm(
  quality ~ .,
  data=dataTrain,
  kernel='linear',
  cost=1,
  scale=FALSE
)
pred <- predict(modelSvmLinear, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.6975
```

The optimal value of the cost hyperparameter is set to 1. The classification accuracy is 68.25% when trained with scaling, and 69.75% when trained without scaling. Training without scaling resulted in a slightly higher testing accuracy, although SVM training should usually be performed with scaling, it took a noticeably longer time, and there could be a better cost hyperparameter as tuning was only done with scaling.

c) Use the training dataset to find out the optimal values of hyperparameters C and for an RBF kernel-based svm. Define the value of the random seed equals 1, cost = c(0.01, 1, 100) and gamma=c(0.01, 1, 100). (5%)

```r
# Compare different costs and gammas with 10-fold cross-validation.
set.seed(1)
tunedSvmRadial <- tune(
  svm,
  quality ~ .,
  data=dataTrain,
  kernel='radial',
  ranges=list(
    cost=c(0.01, 1, 100),
    gamma=c(0.01, 1, 100)
  )
)
print(summary(tunedSvmRadial))
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100     1
##
## - best performance: 0.1556667
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1 1e-02 1e-02 0.2826667 0.02647151
## 2 1e+00 1e-02 0.2340000 0.01755415
## 3 1e+02 1e-02 0.2003333 0.03233505
## 4 1e-02 1e+00 0.5060000 0.04870673
## 5 1e+00 1e+00 0.1623333 0.03067351
## 6 1e+02 1e+00 0.1556667 0.02923088
## 7 1e-02 1e+02 0.5120000 0.03103960
## 8 1e+00 1e+02 0.3253333 0.03006988
## 9 1e+02 1e+02 0.3253333 0.03006988
```

The most optimal values of the cost and gamma hyperparameters are 100 and 1 respectively as it produced the lowest error of 0.1556667. This is for tuning with scale set to TRUE. Tuning the SVM with scale set to FALSE takes too much time, so it is not performed.

d) Train a svm classifier by using the RBF kernel and the corresponding optimal values of hyperparameters C and gamma, then make predictions on the testing dataset, report the classification accuracy. (10%)

```r
# With scaling.
modelSvmRadial <- svm(
  quality ~ .,
  data=dataTrain,
  kernel='radial',
  cost=100,
  gamma=1,
  scale=TRUE
)
pred <- predict(modelSvmRadial, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.64
```

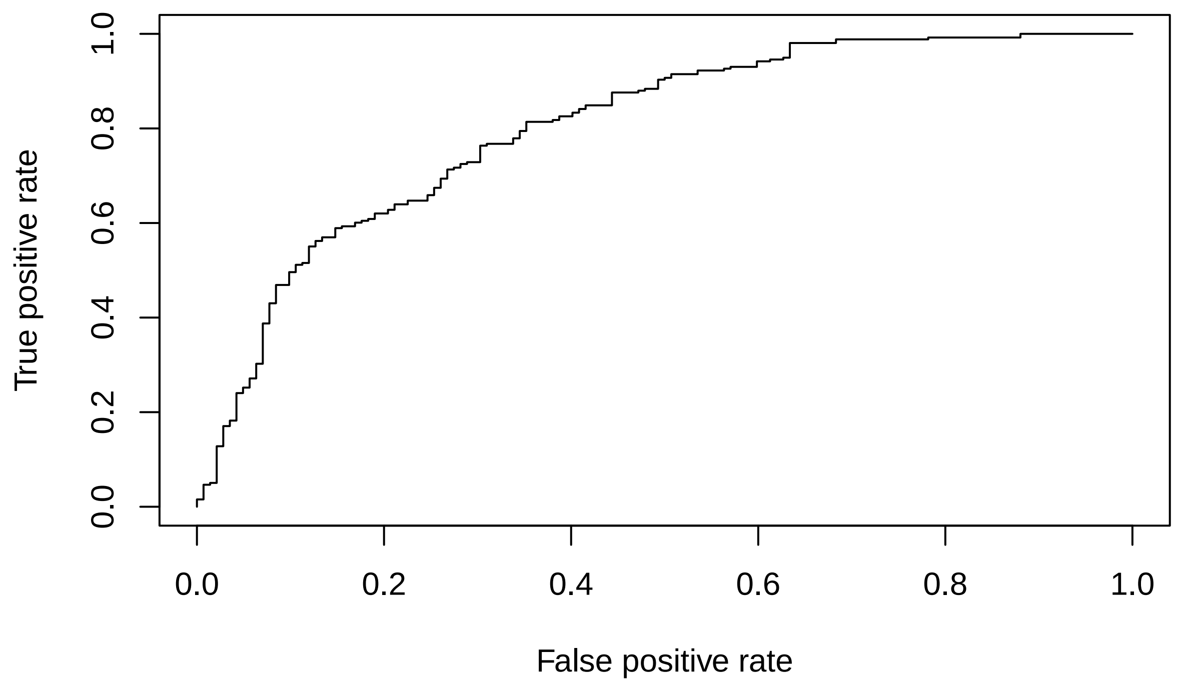
```r
# Without scaling.
modelSvmRadial <- svm(
  quality ~ .,
  data=dataTrain,
  kernel='radial',
  cost=100,
  gamma=1,
  scale=FALSE
)
pred <- predict(modelSvmRadial, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.4875
```

The optimal value of the cost and gamma hyperparameters are set to 100 and 1 respectively. The classification accuracy is 64% when trained with scaling, and 48.75% when trained without scaling. This time, scaling resulted in better testing accuracy, which is what is expected.

e) Train a logistic regression model. Then use the testing dataset to conduct an ROC curve analysis to compare the predictive performance of the trained logistic regression model and those two svm classifiers trained by using linear and RBF kernels respectively. (10%)

```r
# Model the logistic regression.
modelLogisticReg <- glm(quality ~ ., data=dataTrain, family='binomial')
# Obtain predictions in the form of probabilities.
probabilities <- predict(modelLogisticReg, newdata=dataTest, type='response')
# Convert probabilities to performance metrics.
predictions <- prediction(probabilities, dataTest$quality)
tprfpr <- performance(predictions, measure='tpr', x.measure='fpr')
plot(tprfpr)
```



```r
# Select the 'top left' corner of the plot, which is row 215 of tprfpr with
# threshold=0.4332222525, x=0.302816901, and y=0.763565891
classPredictions <- rep('Good', 400)
classPredictions[probabilities < 0.4332222525] <- 'Bad'
print(mean(classPredictions == dataTest$quality))
```

```
## [1] 0.7375
```

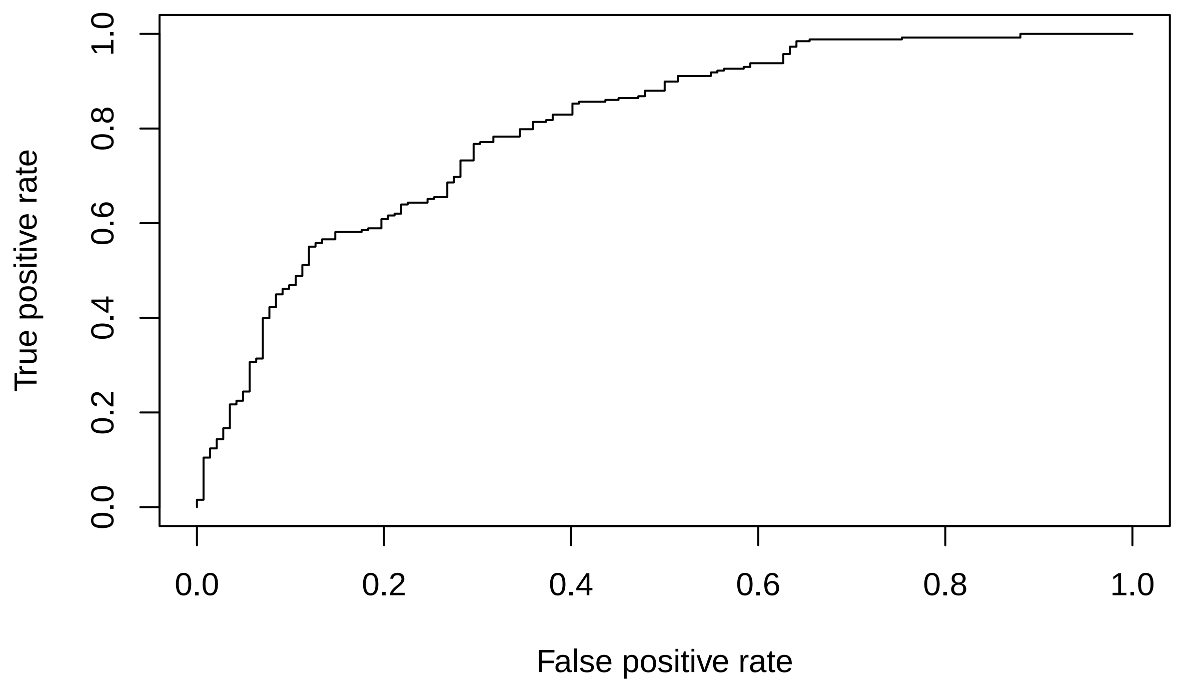
The classification accuracy of the logistic regression model is 73.75%. This is the highest, followed by the linear kernel SVM classifier (68.25%/69.75%), then the RBF kernel SVM classifier (64%/48.75%). However, many factors can affect the evaluated performance. A more extreme threshold selected from the ROC curve may produce a better classification accuracy for this specific test dataset but may not generalise better to new test datasets. The three models were trained with all the possible predictors, but changing the selected predictors may result in better performances for each model. For the logistic regression model, dropping predictors whose p-values do not meet our desired thresholds may improve performance, as well as looking at pairwise correlations between the predictors.

```r
summary(modelLogisticReg)
```

```
##
## Call:
## glm(formula = quality ~ ., family = "binomial", data = dataTrain)
##
## Coefficients:
```

```
##                         Estimate Std. Error z value Pr(>|z|)    
## (Intercept)            8.672e+02  1.036e+02   8.369  < 2e-16 ***
## fixed.acidity          6.078e-01  1.042e-01   5.832 5.46e-09 ***
## volatile.acidity      -6.641e+00  5.637e-01 -11.781  < 2e-16 ***
## citric.acid           -2.670e-01  4.163e-01  -0.641   0.5213    
## residual.sugar         3.707e-01  3.884e-02   9.543  < 2e-16 ***
## chlorides             -1.383e-01  2.623e+00  -0.053   0.9580    
## free.sulfur.dioxide    4.465e-03  3.729e-03   1.197   0.2312    
## total.sulfur.dioxide   3.381e-03  1.610e-03   2.101   0.0357 *  
## density               -8.914e+02  1.051e+02  -8.482  < 2e-16 ***
## pH                     3.469e+00  4.952e-01   7.005 2.47e-12 ***
## sulphates              2.845e+00  4.504e-01   6.317 2.66e-10 ***
## alcohol                1.254e-01  1.283e-01   0.977   0.3284    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4158.9  on 2999  degrees of freedom
## Residual deviance: 3033.6  on 2988  degrees of freedom
## AIC: 3057.6
##
## Number of Fisher Scoring iterations: 4
```

```
# Remove predictors with p-values above 0.05.
modelLogisticReg <- glm(
  quality ~ .-citric.acid-chlorides-free.sulfur.dioxide-alcohol,
  data=dataTrain,
  family='binomial')
probabilities <- predict(modelLogisticReg, newdata=dataTest, type='response')
predictions <- prediction(probabilities, dataTest$quality)
tprfpr <- performance(predictions, measure='tpr', x.measure='fpr')
plot(tprfpr)
```



```r
# Select the 'top left' corner of the plot, which is row 214 of tprfpr with
# threshold=0.4376569330, x=0.309859155, and y=0.763565891
classPredictions <- rep('Good', 400)
classPredictions[probabilities < 0.4376569330] <- 'Bad'
print(mean(classPredictions == dataTest$quality))
```

```
## [1] 0.7425
```

```r
# Experiment with removing the same predictors as above.
set.seed(1)
tunedSvmLinear <- tune(
  svm,
  quality ~ .-citric.acid-chlorides-free.sulfur.dioxide-alcohol,
  data=dataTrain,
  kernel='linear',
  ranges=list(cost=c(0.01, 1, 100))
)
pred <- predict(tunedSvmLinear$best.model, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.685
```

```r
set.seed(1)
tunedSvmRadial <- tune(
  svm,
  quality ~ .-citric.acid-chlorides-free.sulfur.dioxide-alcohol,
  data=dataTrain,
  kernel='radial',
  ranges=list(
    cost=c(0.01, 1, 100),
    gamma=c(0.01, 1, 100)
  )
)
pred <- predict(tunedSvmRadial$best.model, newdata=dataTest[, -12])
mean(pred == dataTest[, 12])
```

```
## [1] 0.6725
```

After removing some predictors for the logistic regression model, the classification accuracy was slightly improved (74.25% vs 73.75%), and another benefit is that the model is more explainable. The same result was observed for the linear (68.5% vs 68.25%) and RBF kernel (67.25% vs 64%) SVM classifiers with scaling, with a bigger performance improvement for the RBF kernel SVM classifier.