Homework #3: Support Vector Machines and Tree-based methods

Maximum margin classifiers

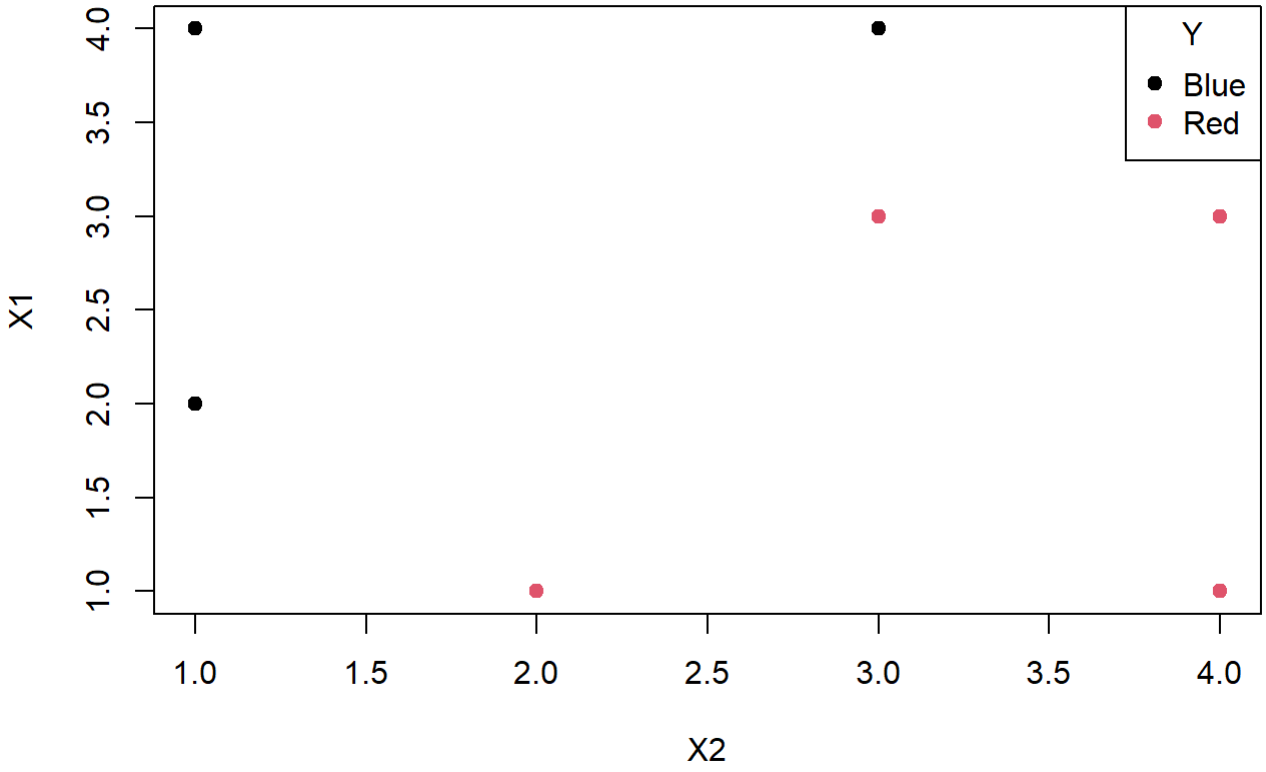
1. Run the following code to create a simple dataset.

```
set.seed(0)
X1 <- c(3,1,3,1,2,4,4)
X2 <- c(4,2,3,4,1,3,1)
Y <- as.factor(c(rep("Red",4),rep("Blue",3)))
dat <- data.frame(X1,X2,Y)
print(dat)
```

```
##   X1 X2    Y
## 1  3  4  Red
## 2  1  2  Red
## 3  3  3  Red
## 4  1  4  Red
## 5  2  1 Blue
## 6  4  3 Blue
## 7  4  1 Blue
```

2. Plot the data. Color-code the points by their outcome value `Y`.

```
plot(X2, X1, col= as.integer(Y),
     xlab='X2', ylab = 'X1', pch=19)
legend('topright', legend = levels(Y), col = 1:2, pch=19, title='Y')
```



3. (3 points) Fit a maximal margin classifier with a linear decision boundary to this dataset using the `svm` function from `e1071`. Use the option `scale=FALSE` so that the model does not do post-processing of the training data. Plot the fitted model.

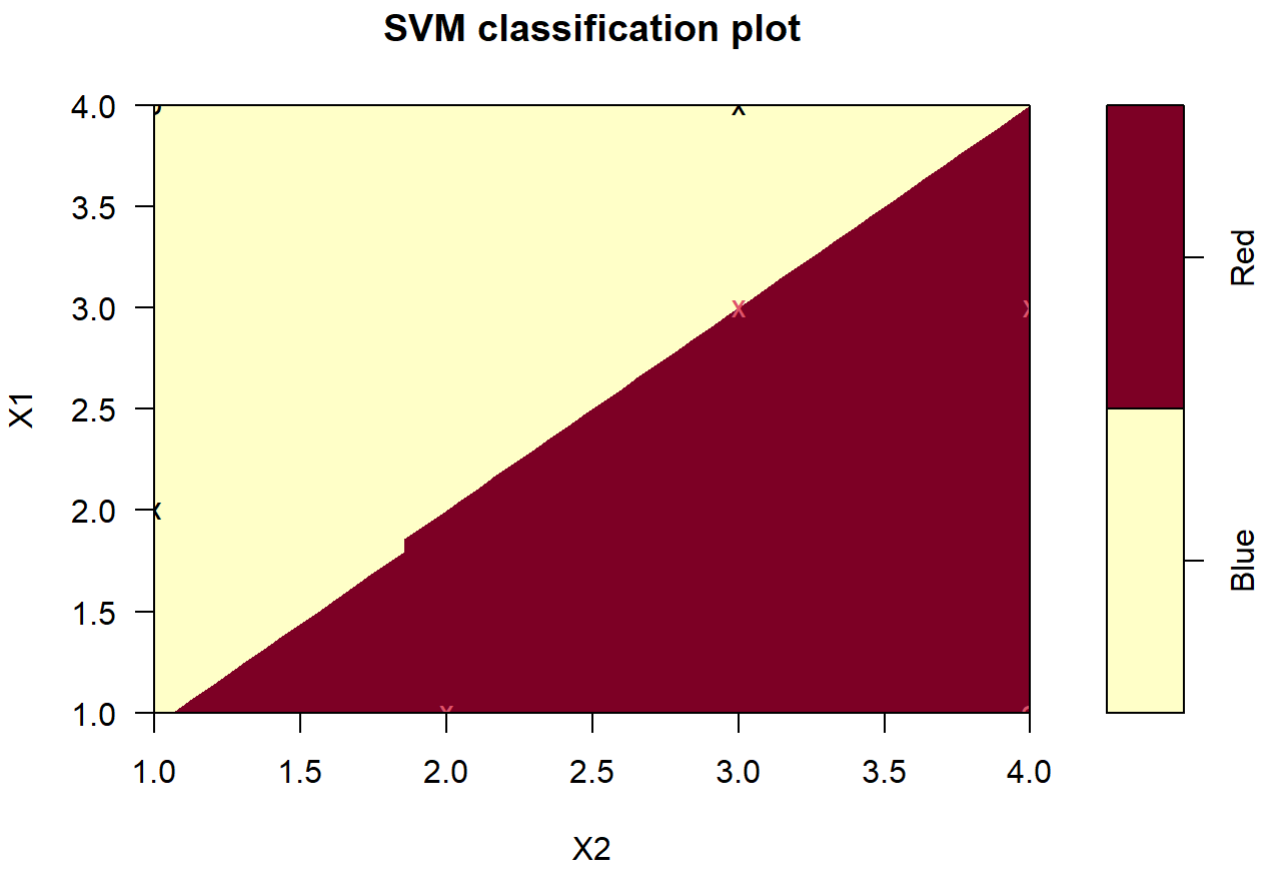
```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.2
```

```
svm1fit = svm(Y ~ ., data = dat, kernel='linear', scale = FALSE)
svm1fit$nsv
```

```
## NULL
```

```
plot(svm1fit, data=dat)
```




```
svm1fit
```

```
##
## Call:
## svm(formula = Y ~ ., data = dat, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  5
```

4. (1 point) Use the `coef` function to extract model coefficients from the maximum margin classifier.

```
coef(svm1fit)
```

```
##   (Intercept)            X1            X2
## -0.0005866667 -0.9995200000  0.9998400000
```

5. (2 points) Suppose we are going to add another observation to the dataset and refit the maximum margin classifier. Provide a new datapoint that, when added to this dataset, will not change the decision boundary.

Refit the model to double check. Call this new datapoint `point1` .

```
point1 <- data.frame(X1= 2.5, X2= 1.5, Y= "Blue")
new_dat1 <- rbind(dat,point1)
svm2fit <- svm(Y ~ . , data = new_dat1, kernel='linear', scale = 'FALSE')
```

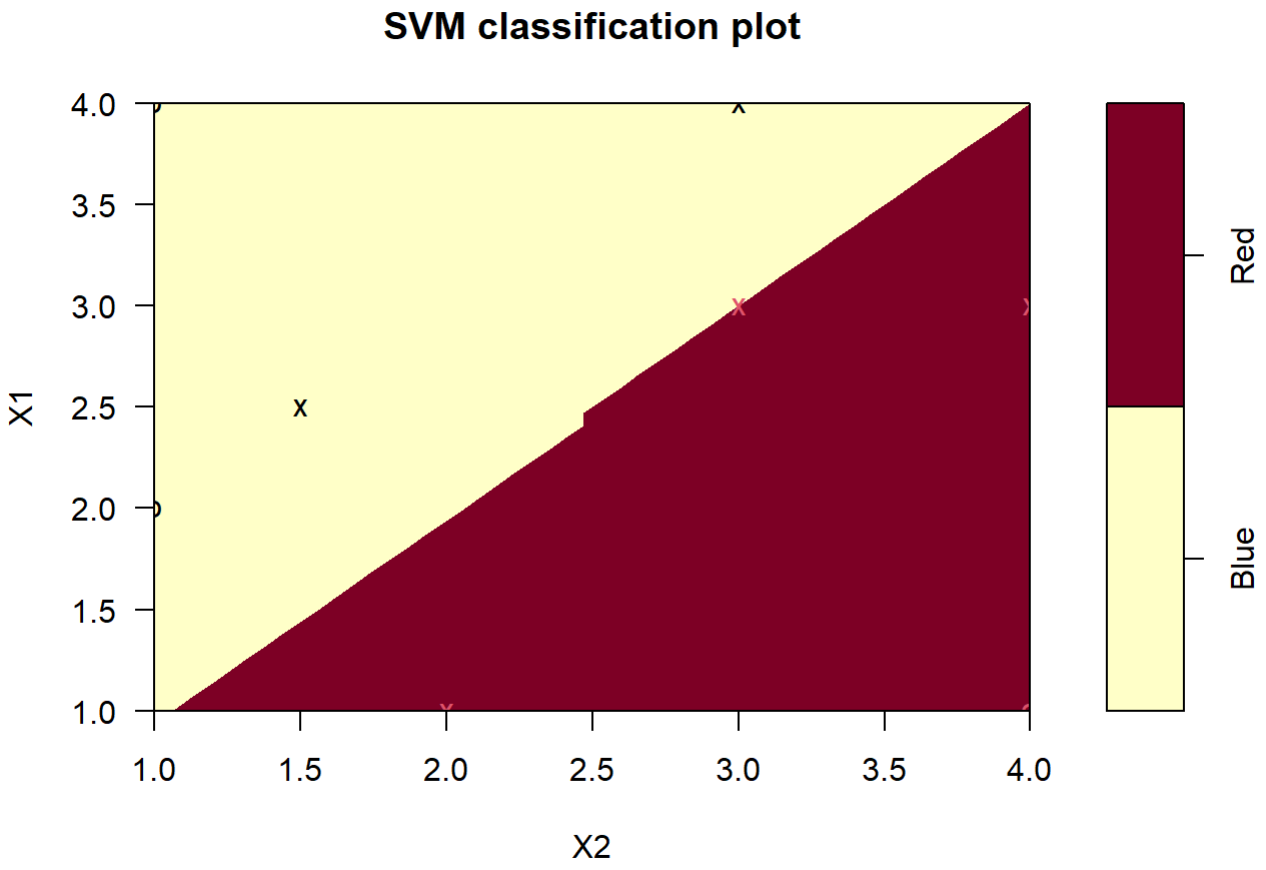
```
## Warning in any(scale): coercing argument of type 'character' to logical

## Warning in any(scale): coercing argument of type 'character' to logical
```

```
## Warning in any(object$scaled): coercing argument of type 'character' to logical
```

```
plot(svm2fit, new_dat1)
```

```
## Warning in any(object$scaled): coercing argument of type 'character' to logical
```



6. (2 points) Now provide a new datapoint that, when added to dataset `dat` , WILL change the decision boundary. Call this new datapoint `point2` . Pick an example point where the new dataset remains linearly separable.

```
point2 <- data.frame(X1= 1.5, X2= 3.0, Y= "Blue")
new_dat2 <- rbind(dat,point2)
svm3fit <- svm(Y ~ . , data = new_dat2, kernel='linear', scale = 'FALSE')
```

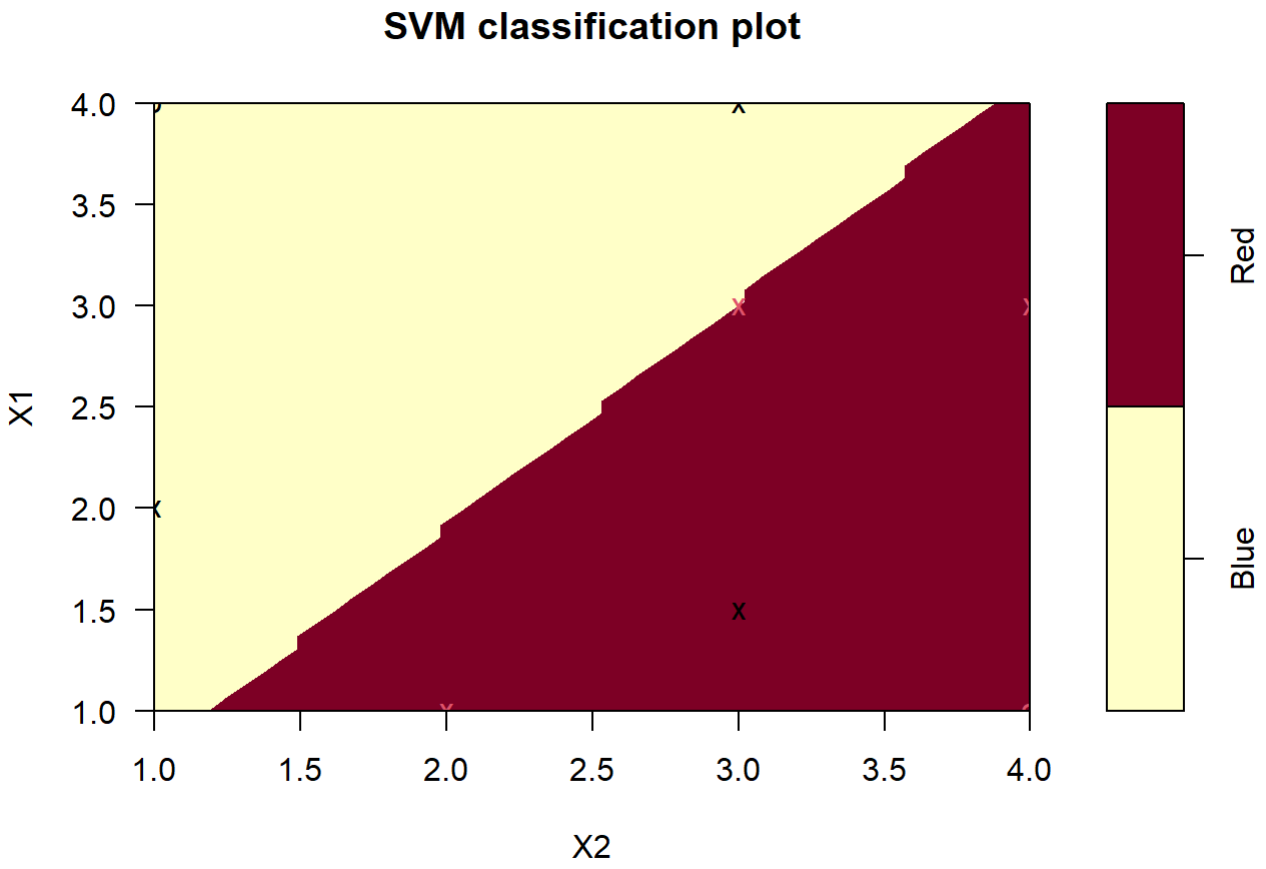
```
## Warning in any(scale): coercing argument of type 'character' to logical

## Warning in any(scale): coercing argument of type 'character' to logical
```

```
## Warning in any(object$scaled): coercing argument of type 'character' to logical
```

```
plot(svm3fit, new_dat2)
```

```
## Warning in any(object$scaled): coercing argument of type 'character' to logical
```



Support vector machines

7. Read in the dementia data “dementia_full.csv” into a data frame called `dementia_dat` . This data is from the UCSF Memory and Aging Center. The goal was to predict the type of dementia based on patterns of brain loss as measured through structural MRI.

```
dementia_dat <- read.csv('dementia_full.csv')
```

8. (1 point) We will now predict the dementia diagnosis based on the available predictors. The diagnosis is given in the multiclass outcome of `MacCohort_kr`. To start, generate a table of the elements in the `MacCohort_kr` variable.

```
table(dementia_dat$MacCohort_kr)
```

```
##
##         AD      bvFTD    CONTROL nfPPAunspc        PSP      svPPA
##        151         68        315         38         44         44
```

9. (1 point) Set the random seed to 7 and then split the data into 2 sets (440 train, and 220 test)

```
set.seed(7)
row_train <- sample(nrow(dementia_dat),440,replace = FALSE)
dementia_train <- dementia_dat[row_train,]
dementia_test <- dementia_dat[-row_train,]
```

10. (3 points) Fit a support vector machine to predict `MacCohort_kr` using predictors `Left_MTG_middle_temporal_gyrus`, `Left_Amygdala` and `Left_AIns_anterior_insula`, with a radial basis kernel. Use 3-fold CV to tune the value of the `C` and `sigma` hyperparameters. Use the `caret` package with `method=svmRadial` and tune over the values `C = 1e-2,1e-1,1,10,100,1000,10000` and `sigma=1e-4,1e-3,1e-2,1e-1, 1,10`. (You may need the `kernlab` package to run this.)

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.2
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
## Loading required package: lattice
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
dementia_train_building <- dementia_train[,c('Left_MTG_middle_temporal_gyrus','Left_Amygdala',
'Left_AIns_anterior_insula','MacCohort_kr')]

train_control <- trainControl(method="cv", number=3)
caret_grid <- expand.grid(
  C = c(1e-2,1e-1,1,10,100,1000,10000),
  sigma = c(1e-4,1e-3,1e-2,1e-1, 1,10)
)

svm_model <- train(as.factor(MacCohort_kr) ~., data=dementia_train_building, trControl=train_con
trol, method="svmRadial", tuneGrid=caret_grid)
svm_model$results
```

```
##        C sigma  Accuracy       Kappa  AccuracySD     KappaSD
## 1  1e-02 1e-04 0.4795452 0.000000000 0.003402318 0.000000000
## 7  1e-01 1e-04 0.4795452 0.000000000 0.003402318 0.000000000
## 13 1e+00 1e-04 0.4795452 0.000000000 0.003402318 0.000000000
## 19 1e+01 1e-04 0.4795452 0.006115987 0.003402318 0.006189380
## 25 1e+02 1e-04 0.6454664 0.411705298 0.011160109 0.022936298
## 31 1e+03 1e-04 0.6977293 0.516097795 0.003489679 0.007927498
## 37 1e+04 1e-04 0.6931942 0.516883446 0.017646349 0.029426353
## 2  1e-02 1e-03 0.4795452 0.000000000 0.003402318 0.000000000
## 8  1e-01 1e-03 0.4795452 0.000000000 0.003402318 0.000000000
## 14 1e+00 1e-03 0.4795452 0.005094236 0.003402318 0.004717108
## 20 1e+01 1e-03 0.6454664 0.411705298 0.011160109 0.022936298
## 26 1e+02 1e-03 0.6931786 0.508322008 0.001210547 0.002114039
## 32 1e+03 1e-03 0.6931942 0.516488435 0.017646349 0.029006677
## 38 1e+04 1e-03 0.6999969 0.525218414 0.013656833 0.021273004
## 3  1e-02 1e-02 0.4795452 0.000000000 0.003402318 0.000000000
## 9  1e-01 1e-02 0.4795452 0.000000000 0.003402318 0.000000000
## 15 1e+00 1e-02 0.6431833 0.407411658 0.010219707 0.021038686
## 21 1e+01 1e-02 0.6908955 0.503505087 0.008541326 0.015469051
## 27 1e+02 1e-02 0.7000124 0.523497279 0.011248981 0.017317876
## 33 1e+03 1e-02 0.6976827 0.523419720 0.032071295 0.054711151
## 39 1e+04 1e-02 0.6863293 0.503956716 0.030188728 0.051243682
## 4  1e-02 1e-01 0.4795452 0.000000000 0.003402318 0.000000000
## 10 1e-01 1e-01 0.6295468 0.380695113 0.003439549 0.004746125
## 16 1e+00 1e-01 0.6840928 0.491186178 0.010229899 0.020577447
## 22 1e+01 1e-01 0.6931942 0.512172064 0.017646349 0.028760657
## 28 1e+02 1e-01 0.6908645 0.510011781 0.038025979 0.067235000
## 34 1e+03 1e-01 0.6772435 0.489138251 0.026285489 0.048799500
## 40 1e+04 1e-01 0.6681421 0.480458058 0.028203884 0.051296536
## 5  1e-02 1e+00 0.4795452 0.000000000 0.003402318 0.000000000
## 11 1e-01 1e+00 0.6386482 0.396275903 0.005634854 0.017788220
## 17 1e+00 1e+00 0.6886435 0.503599825 0.034121950 0.057480234
## 23 1e+01 1e+00 0.6977138 0.523716378 0.021045028 0.038409627
## 29 1e+02 1e+00 0.6135961 0.413724343 0.018418536 0.021166477
## 35 1e+03 1e+00 0.5614109 0.353906519 0.027209887 0.045770215
## 41 1e+04 1e+00 0.5386419 0.328841099 0.013770420 0.025823772
## 6  1e-02 1e+01 0.4795452 0.000000000 0.003402318 0.000000000
## 12 1e-01 1e+01 0.4795452 0.000000000 0.003402318 0.000000000
## 18 1e+00 1e+01 0.6068089 0.334008690 0.004897764 0.011592802
## 24 1e+01 1e+01 0.5182648 0.244377995 0.035917189 0.057534755
## 30 1e+02 1e+01 0.4863324 0.189486161 0.032840339 0.041432265
## 36 1e+03 1e+01 0.4863324 0.189486161 0.032840339 0.041432265
## 42 1e+04 1e+01 0.4863324 0.189486161 0.032840339 0.041432265
```

11. (1 points) Determine the predictions of the SVM fit on the test dataset. What is the test accuracy?

```
prediction = predict(svm_model, newdata= dementia_test)
table(dementia_test$MacCohort_kr, prediction)
```

```
##             prediction
##              AD bvFTD CONTROL nfPPAunspc PSP svPPA
##   AD         26     1      13          0   0     3
##   bvFTD       2    13       5          0   0     2
##   CONTROL     7     0      97          0   0     0
##   nfPPAunspc  2     2       8          0   0     1
##   PSP         3     1      13          0   0     0
##   svPPA       8     1       1          0   0    11
```

```
# test accuracy
mean(prediction == dementia_test$MacCohort_kr)
```

```
## [1] 0.6681818
```

The test accuracy is 0.67.

12. (3 points) What is the accuracy of a model that randomly guesses the class label simply based on the fraction of observations in each class? Is the SVM doing better than random guessing?

```
proportions_observed <- table(dementia_train_building$MacCohort_kr) / nrow(dementia_train_buildi
ng)

expected_accuracy <- sum(proportions_observed^2)

print(expected_accuracy)
```

```
## [1] 0.3108678
```

The SVM's accuracy is better than the random guess because the accuracy is higher than 0.31.

Random Forests

13. (2 points) Read in the prostate cancer dataset "Prostate_GSE6919_U95C.csv". Split 70% of the data for training and 30% for testing.

```
set.seed(7)
prostate <- read.csv("Prostate_GSE6919_U95C.csv")
number_row <- round(nrow(prostate) * 0.7)
train_row <- sample(nrow(prostate), number_row, replace = FALSE)
prostate_train <- prostate[train_row,]
prostate_test <- prostate[-train_row,]
```

14. (2 points) Fit a random forest with cancer outcome (`type`) as outcome and all other gene expression values as candidate predictors. Only consider 10 candidate predictors per node. (You may need to change the variable names to get this to work.) Use the "impurity" option to measure variable importance.

```
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.3.2
```

```
rfCancer <- ranger(as.factor(type)~.,data=prostate_train, mtry = 10, importance="impurity", prob
ability = TRUE, seed = 42)
```

```
## Warning in ranger(as.factor(type) ~ ., data = prostate_train, mtry = 10, :
## Avoid the formula interface for high-dimensional data. If ranger is slow or you
## get a 'protection stack overflow' error, consider the x/y or
## dependent.variable.name interface (see examples).
```

```
rfCancer
```

```
## Ranger result
##
## Call:
##  ranger(as.factor(type) ~ ., data = prostate_train, mtry = 10,      importance = "impurity",
probability = TRUE, seed = 42)
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      80
## Number of independent variables:  12646
## Mtry:                             10
## Target node size:                 10
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.2272575
```

15. (1 point) Determine the AUC of the fitted model on the test dataset.

```
library(ROCit)
```

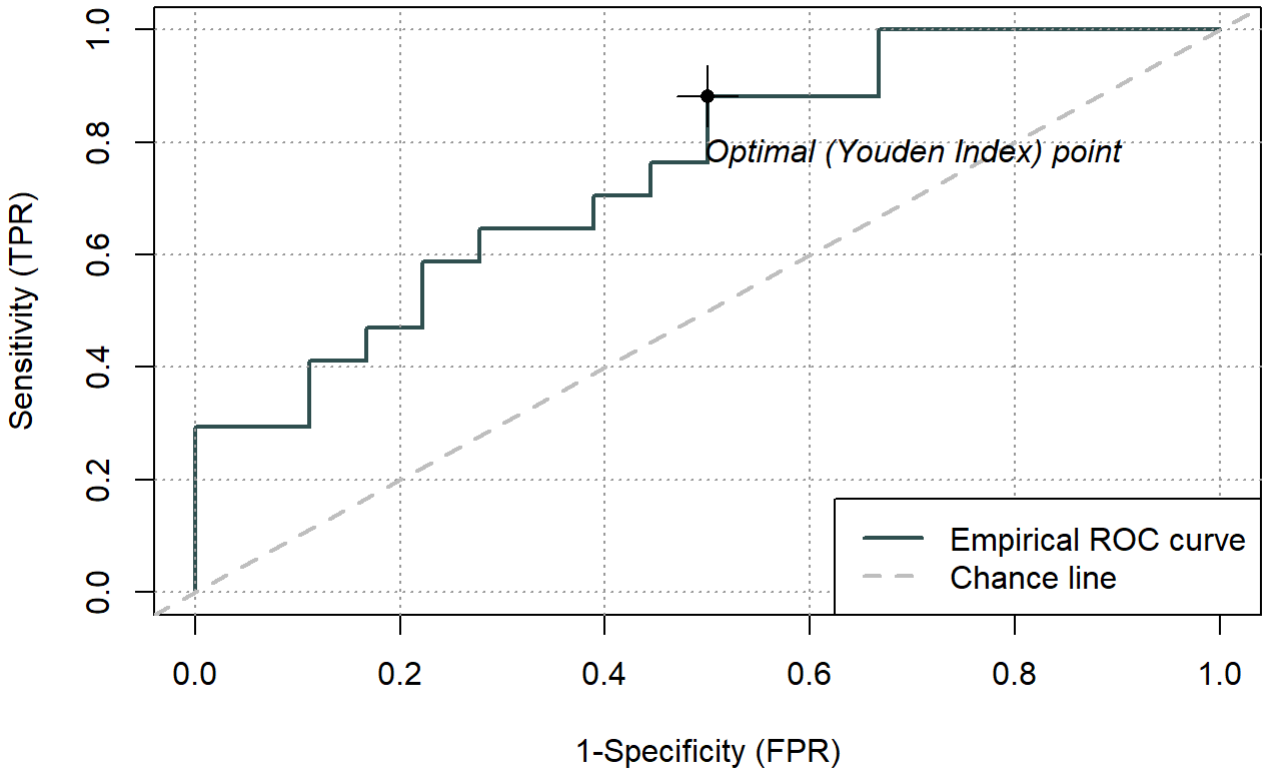
```
## Warning: package 'ROCit' was built under R version 4.3.2
```

```
pred = predict(rfCancer, data = prostate_test)
prob <- pred$predictions[,2]

roc_rf <- rocit(score =prob, class = prostate_test$type)
ciAUC(roc_rf)
```

```
##
##    estimated AUC : 0.748366013071895
##    AUC estimation method : empirical
##
##    CI of AUC
##    confidence level = 95%
##    lower = 0.58352992854533     upper = 0.913202097598461
```

```
plot(roc_rf)
```



16. (2 points) Based on the variable importance measures in the random forest, which gene was most important? What proportion of genes have a nonzero variable importance?

```
var_import <- rfCancer$variable.importance
head(var_import[order(var_import, decreasing = TRUE)])
```

```
##  X55971_at  X64138_at  X55964_at  X62542_at  X60570_at  X63112_at
## 0.07889288 0.06673035 0.06025478 0.05609009 0.05296220 0.05224310
```

```
length(var_import[var_import!=0])/length(var_import)
```

```
## [1] 0.2661711
```

X54701_at is most important in this model because the variable importance is highest.

17. (1 point) If you were going to fit a bagged model, how would you modify the call to the random forest code? Hint: How many candidate variables would you consider at each split? Note that you don't have to fit the model.

```
# Candidate variables: Exclude the target variable
num_variables <- ncol(prostate_train) - 1

# Build a bagged model
bagging_model <- ranger(as.factor(type)~.,
                        data=prostate_train,
                        mtry=num_variables,
                        importance="impurity",
                        seed = 42
)
```

```
## Warning in ranger(as.factor(type) ~ ., data = prostate_train, mtry =
## num_variables, : Avoid the formula interface for high-dimensional data. If
## ranger is slow or you get a 'protection stack overflow' error, consider the x/y
## or dependent.variable.name interface (see examples).
```

18. (3 points) Tune `mtry` for the random forest model using 3-fold CV on all the prostate cancer data over the values `10,20,40,80,160,320`. Set `min.node.size` to 1. Which mtry leads to the best cross-validated AUC? What is the cross-validated AUC of the selected model?

```
train_control <- trainControl(method = "cv", number = 3, search = "grid", classProbs = TRUE, sum
maryFunction = twoClassSummary, savePredictions = "final")
caret_grid <- expand.grid(mtry = c(10, 20, 40, 80, 160, 320), min.node.size = 1, splitrule = "gi
ni")

cv_mdl <- train(as.factor(type) ~ ., data=prostate_train,
                trControl=train_control, method="ranger", tuneGrid = caret_grid,
                metric = "ROC")

# Parameters of the best model
print(cv_mdl$bestTune)
```

```
##   mtry splitrule min.node.size
## 5  160      gini             1
```

```
# Cross validated AUC
best_mdl <- cv_mdl$results[cv_mdl$results$mtry == cv_mdl$bestTune$mtry, ]
print(best_mdl$ROC)
```

```
## [1] 0.7110806
```

The best cross-validated AUC is 0.74 and the best model's mtry is 160.

Gradient boosted trees

Let's now fit a gradient boosted tree to see how variable importance can change.

19. (2 points) Fit a gradient boosted tree for this data using `xgboost` . Use 1000 trees, 0.1 eta, and max depth of 1. Make sure to use an appropriate loss function for the binary prediction task.

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.3.2
```

```
# Convert the data
data_matrix <- as.matrix(prostate_train[, -1])
label_vector <- ifelse(prostate_train$type == "primary_prostate_tumor", 1, 0)
# Build XGBoost model
params <- list(
  objective = "binary:logistic",
  eta = 0.1,
  max_depth = 1
)

xgb_model <- xgboost(
  data = data_matrix,
  label = label_vector,
  nrounds = 1000,
  params = params,
  verbose = FALSE
  )
```

20. (1 point) What is the AUC of this fitted model?

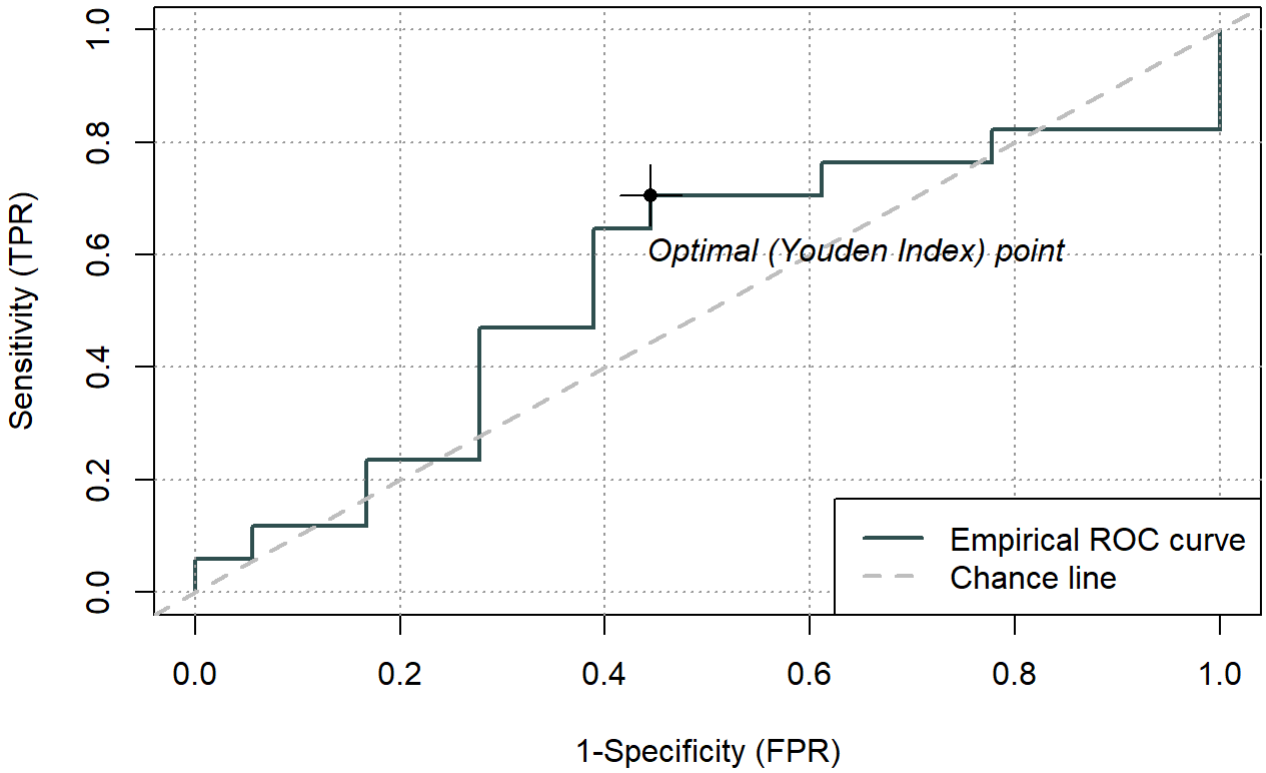
```
# Convert the data
data_matrix_test <- as.matrix(prostate_test[, -1])

# Predict the probabilities
pred <- predict(xgb_model, data_matrix_test)

roc_rf <- rocit(score =pred, class = prostate_test$type )
ciAUC(roc_rf)
```

```
##
##    estimated AUC : 0.558823529411765
##    AUC estimation method : empirical
##
##    CI of AUC
##    confidence level = 95%
##    lower = 0.366167719172639     upper = 0.751479339650891
```

```
plot(roc_rf)
```



###

Estimated AUC is 0.56.

21. (3 points) What is the most important gene selected by xgb? To get this, take the absolute SHAP value of the test observations and find the gene with the highest magnitude SHAP value on average. Is it the same gene selected by the random forest? What is the rank for the top gene selected by the random forest?

```
# Get the most important gene name based on SHAP value
pred_shap <- predict(xgb_model, data_matrix_test, predcontrib = TRUE)
mean_shap <- colMeans(abs(pred_shap))
best_index <- which.max(mean_shap)
names(mean_shap)[best_index]
```

```
## [1] "X65072_r_at"
```

```
# The most important gene in the random forest model is X54701_at.
```

The top gene is different between the random forest model and the xgboost model. There might be two reasons. First, the algorithms of the two models are different. The process of random forest is parallel but the xgboost model is built sequentially. The second reason is that the metrics are different between the variable importance and SHAP. Variable importance depends on the model but SHAP does not.