STAT 452 Project 2

Section 1: ROC curves

- Wish to display *specificity* and *sensitivity* of a classification model (for example, LDA)
- *Specificity*: percentage of non-defaulters that are correctly identified (true negative rate)
- *Sensitivity*: percentage of true defaulters that are identified (true positive rate)
- ROC curve plots 1-specificity (false positive rate) on the x-axis, and sensitivity on the y-axis. Ideally, we want a high sensitivity and a low 1-specificity
- ROC curve displays the 2 errors over all possible thresholds
- Default classifier (threshold) is 0.5, but can be higher/lower depending on the trade-off of the 2 errors
    - A lower threshold increases both 1-specificity and sensitivity, while a higher threshold decreases both
- Summary table:

specificity	True Negative Rate	$\frac{TN}{N} = \frac{True\ Negatives}{True\ Negatives + False\ Positives}$
1-specificity (x-axis)	False Positive Rate	$1 - \frac{TN}{N} = \frac{FP}{N} = \frac{False\ Positives}{True\ Negatives + False\ Positives}$
Sensitivity (y-axis)	True Positive Rate	$\frac{TP}{P} = \frac{True\ Negatives}{True\ Negatives + False\ Negatives}$

- The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the (ROC) curve (**AUC**)
- The larger the AUC, the better the classifier (an ideal ROC curve will hug the top left corner)
- AUC should not be lower than 0.5, because at 0.5 it is the same as a classifier purely predicting by chance

The ROC curve functions are provided in the ROCR library. First, the *rocplot* function is defined as the following:

```
library(ROCR)
rocplot =function(pred, truth, ...){
  predob = prediction(pred, truth)
  perf = performance (predob, "tpr", "fpr")
  plot(perf,...)}
```

The data used is a water quality dataset from Kaggle (waterQuality1.csv). There are 20 numeric variables, corresponding to the concentrations of 20 different chemicals in water, and the 21st variables *is_safe* is a binary variable: 1 if the water is safe to drink, and 0 if not.

After some data cleaning, 75% of the dataset is put into training set, and the rest 25% is used as validation.

First, a logistic regression model is fit.

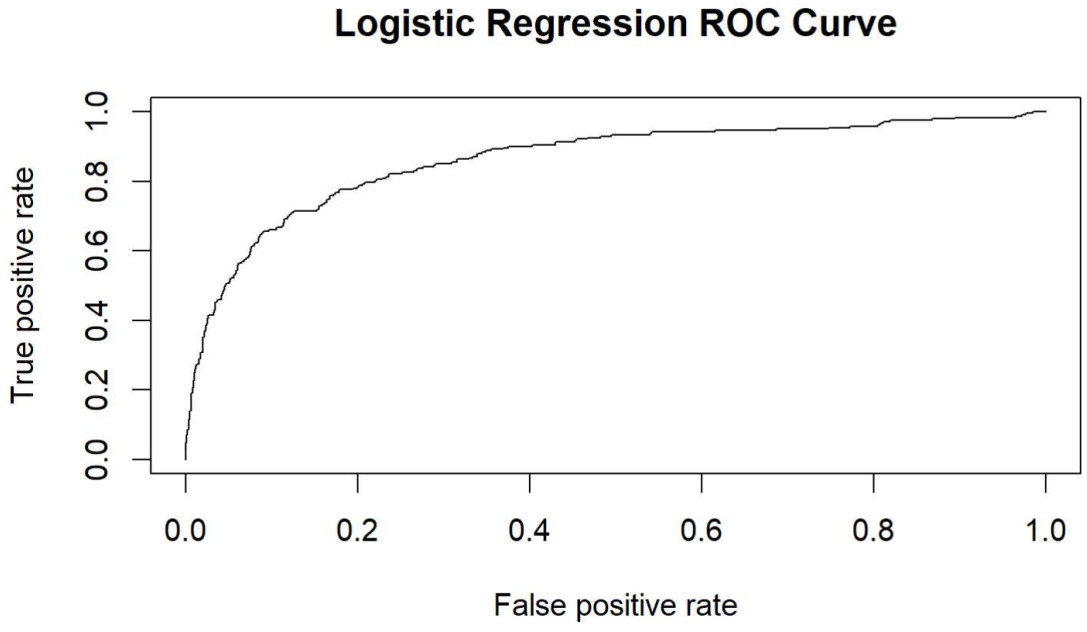
```
fit.log.nnet <- multinom(is_safe ~ ., data = data.train.scale)
## probabilities
pred.log.nnet.probs <- predict(fit.log.nnet, data.valid.scale, type = 'probs')

pred.log.nnet <- predict(fit.log.nnet, data.valid.scale)
table(Y.valid, pred.log.nnet, ### Confusion matrix
  dnn = c("Observed", "Predicted"))

(misclass.log.nnet <- mean(pred.log.nnet != Y.valid))
```

Note that the misclassification rate is around 10%.

The *pred.log.nnet.probs* has the probabilities P(Y=1|X). The default threshold is set at 0.5. This is required as the argument passed into the ROC Curve function defined earlier. The actual predictions of 0s and 1s are in *pred.log.nnet*.

The ROC Curve is obtained with the following line:

```
rocplot(pred.log.nnet.probs, Y.valid, main = "Logistic Regression ROC Curve")
```



The AUC is calculated using this line:

```
(performance(prediction(pred.log.nnet.probs, Y.valid), measure = "auc"))@y.values[[1]]
```

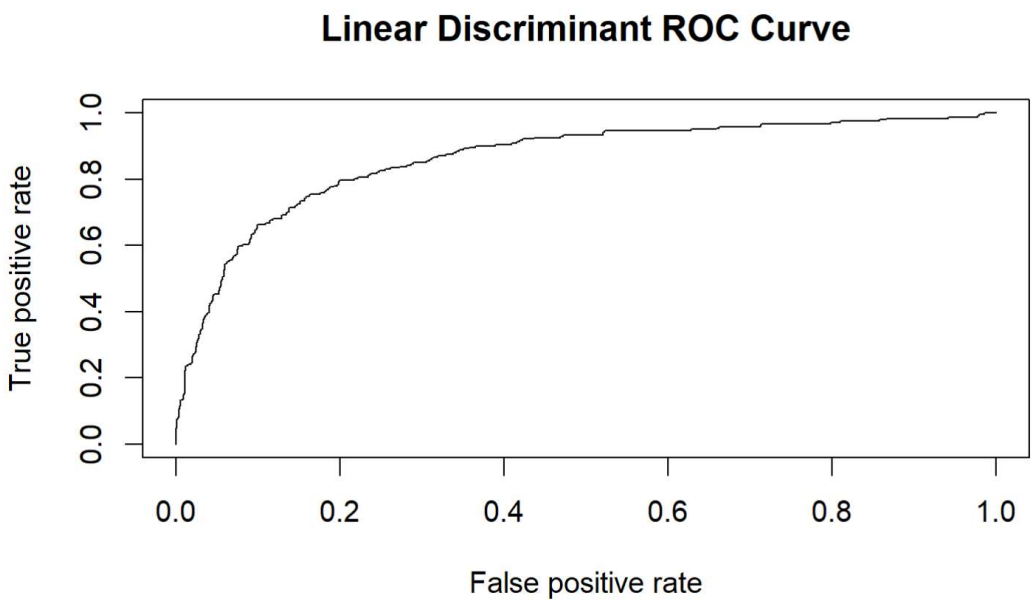
And the AUC for the logistic regression is 0.862, which is decent.

Now, a similar model is fit using linear discriminant analysis.

```
fit.lda <- lda(X.train.DA, Y.train)

## this gets the probability of predicting a 1
pred.lda.probs <- (predict(fit.lda, X.valid.DA)$posterior)[,2]
pred.lda <- predict(fit.lda, X.valid.DA)$class

table(Y.valid, pred.lda, dnn = c("Obs", "Pred"))

(miss.lda <- mean(Y.valid != pred.lda))
```

The misclassicification rate for this model is just under 11%. Here *pred.lda.probs* has the probability of predicting a 1, P(Y=1|X), and *pred.lda* is the actual predictions.

```
rocplot(pred.lda.probs, Y.valid, main = "Linear Discriminant ROC Curve")
```




```
(performance(prediction(pred.lda.probs, Y.valid), measure = "auc"))@y.values[[1]]
```

The AUC is 0.861, so it's very similar to the logistic regression model.

Section 2: Support Vector Machines (SVM)

- Aims for better classification of most of the training observations, when the two classes cannot be separated by a hyperplane
- Allows for some observations to be on the wrong side of the margin/hyperplane for a more robust classification overall

- Maximizes the width of the margin M from the hyperplane, based on *slack variables* ε_i, and tuning parameter C such that $\sum \varepsilon_i \leq C$
- If $\varepsilon_i = 0$ then the ith observation is on the correct side of the margin; $\varepsilon_i > 0$ then the ith observation is on the wrong side of the margin; $\varepsilon_i > 1$ then it is on the wrong side of the hyperplane
- C is the tolerance parameter, or the severity of the violations of the margin
- Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors*, and only those vectors affect the classifier
- SVM is an extension of the support vector classifier such that it enlarges the feature space, using *kernels*, to accommodate non-linear boundary between classes
- Advantages:
    - Can handle non-linear relationships with kernels (logistic regression cannot)
    - Does not recursively split data (as in decision trees)
    - Robust to overfitting
- Disadvantages:
    - Sensitive to outliers (random forest is less sensitive)
    - Computationally expensive
    - Weak intuition

It is hard to show plots with the water quality data because there is just way too many variables.

First, a standard SVM model is fit using cost = 1 and γ = 1. The kernel chosen is "radial" because it's most likely not linear or polynomial, given the number of variables.

A cost argument allows us to specify the cost of a violation to the margin. When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin.

Gamma (γ) is the radial basis kernel.

```
library(e1071)

## cost = 1
svmfit = svm(is_safe ~., data=data.train, kernel = "radial", gamma = 1, cost=1)
summary(svmfit)
```

```
Call:
svm(formula = is_safe ~ ., data = data.train, kernel = "radial",
    gamma = 1, cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  5986

 ( 669 5317 )


Number of Classes:  2

Levels:
 0 1
```

Next, a second SVM model is fit using cost = 10000

```
## cost = 1e5
svmfit2 = svm(is_safe ~., data=data.train, kernel = "radial", gamma = 1, cost=1e5)
summary(svmfit2)
```

```
Call:
svm(formula = is_safe ~ ., data = data.train, kernel = "radial",
    gamma = 1, cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1e+05

Number of Support Vectors:  5986

 ( 669 5317 )


Number of Classes:  2

Levels:
 0 1
```

What is the ideal cost parameter? This can be determined with a bit of tuning. Let cost parameters range from 0.1, 1, 10, 100, and 1000, and let the gammas range from 0.5, 1, 2, 3, and 4. The tuning function takes a long time to execute.

```
tune.out=tune(svm, is_safe ~., data=data.train, kernel = "radial",
              ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),
                           gamma=c(0.5,1,2,3,4) ))
 (tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  0.1   0.5

- best performance: 0.1167115
```

Looks like the best model is cost = 0.1 and γ = 0.5, the lowest parameter in each set of parameters, thus if even lower parameter values were given then those might be chosen instead.

```
## select best SVM model and use it for prediction
pred.svm = predict(tune.out$best.model, newdata = data.valid)
table(true=Y.valid, pred=pred.svm)
```

```
     pred
true    0    1
   0 1786    0
   1  211    0
```

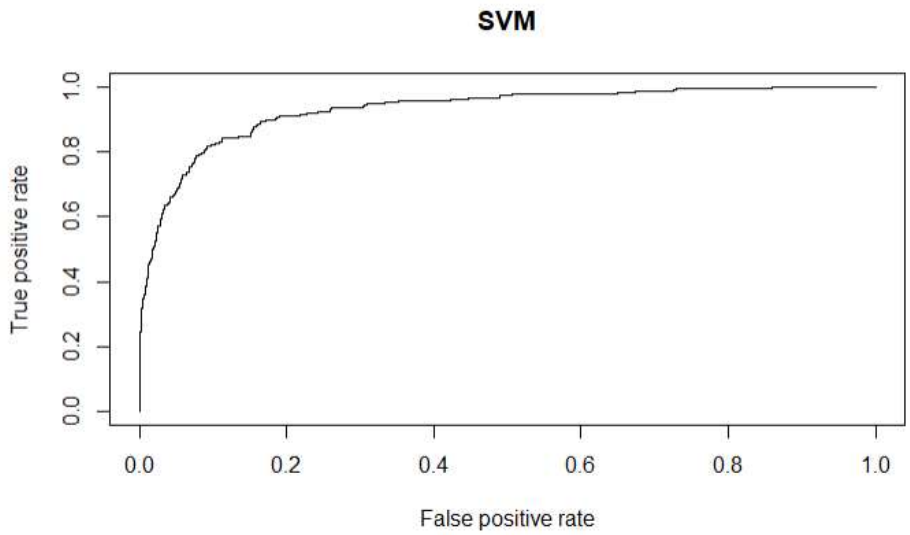
```
(miss.svm <- mean(Y.valid != pred.svm))
```

For some reason, the optimal SVM is predicting 0 for every single observation. This can be due to the sparse number of observations with 1.

The misclassification rate is about 10.6%.

Finally, for completion, an ROC Curve is presented as well.

```
svmfit.opt= svm(is_safe ~., data=data.train, kernel = "radial",
gamma = 0.5, cost = 0.1, decision.values = T)
fitted =attributes(predict(svmfit.opt, data.valid, decision.values=TRUE))$decision.values
rocplot(fitted, Y.valid, main = "SVM")
```




```
(performance(prediction(fitted, Y.valid), measure = "auc"))@y.values[[1]]
```

The AUC for the SVM model is 0.745, which is lower than both the logistic regression and the LDA models actually.