Assaf Bitton

We will be predicting whether the housing price is expensive or not using the `sahp` dataset in the **r02pro** package.

You can run the following code to prepare the analysis.

```
library(r02pro)     #INSTALL IF NECESSARY
```

```
## Warning: package 'r02pro' was built under R version 4.4.3
```

```
library(tidyverse)  #INSTALL IF NECESSARY
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

```
## Warning: package 'tidyr' was built under R version 4.4.2
```

```
## Warning: package 'dplyr' was built under R version 4.4.2
```

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.4.3
```

```
my_sahp <- sahp %>%
  na.omit() %>%
  mutate(expensive = sale_price > median(sale_price)) %>%
  mutate(expensive = as.factor(expensive)) %>%
  dplyr::select(gar_car, liv_area, oa_qual, expensive)
```

Please answer the following questions.

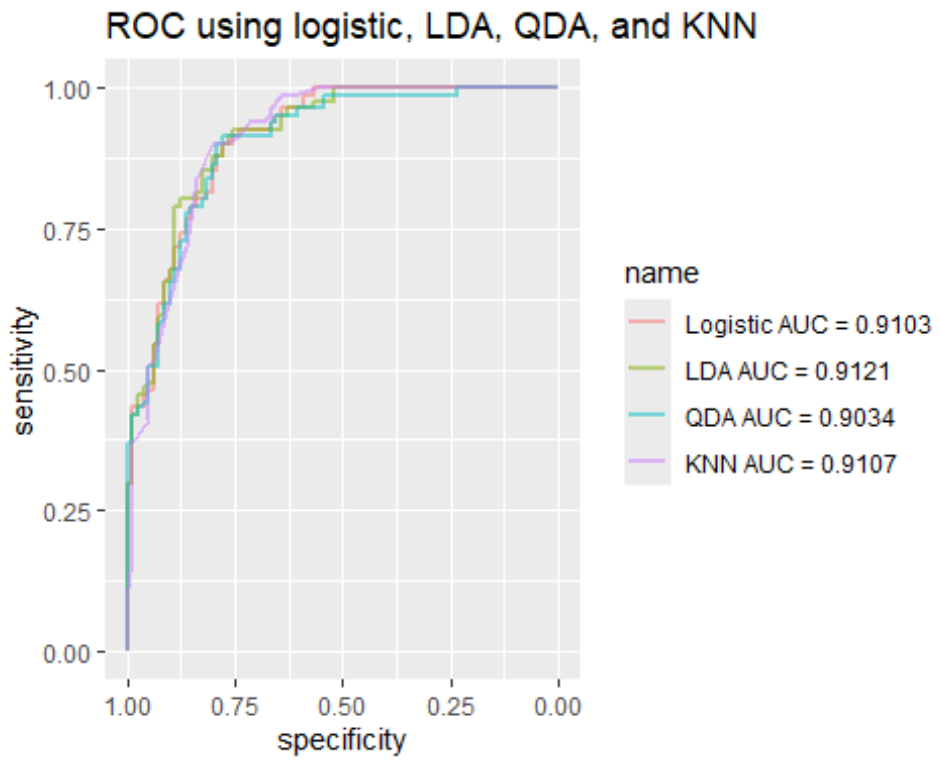
Q1

Use the data `my_sahp` to fit four models — Logistic regression, LDA, QDA, and **standardized** KNN (with $K = 39$) — of `expensive` on all other variables. Visualize the ROC curves for them and add the AUC values to the legend. Discuss your findings. (Note: You can use the entire dataset without splitting for this analysis.)

Solution:

```
#Q1
#Logistic Regression
log_f <- glm(expensive ~.,
             data = my_sahp,
             family = "binomial")
log_p <- predict (log_f,
                  type="response")
#LDA
lda_f <- lda(expensive ~.,
             data = my_sahp)
lda_p <- predict(lda_f)$posterior[,2]
#QDA
```

```r
qda_f <- qda(expensive~.,
             data = my_sahp)
qda_p <- predict(qda_f)$posterior[,2]
#Standardized KNN (K=39)
library(caret)

## Warning: package 'caret' was built under R version 4.4.3

fit_std <- preProcess(my_sahp, method = "scale")
my_sahp_std<-predict(fit_std,newdata=my_sahp)

knn_f <-knn3(expensive~.,
             data=my_sahp_std,
             k=39)
knn_p <- predict(knn_f, newdata=my_sahp_std, type="prob")[,2]
#Visualize ROC curves and add AUC to the legend
library(pROC)

## Warning: package 'pROC' was built under R version 4.4.3

log_roc <- roc(my_sahp$expensive, log_p)
log_auc <- auc(log_roc)

lda_roc <- roc(my_sahp$expensive, lda_p)
lda_auc <- auc(lda_roc)

qda_roc <- roc(my_sahp$expensive, qda_p)
qda_auc <- auc(qda_roc)

knn_roc <- roc(my_sahp$expensive, knn_p)
knn_auc <- auc(knn_roc)

models_roc <- list(Logistic = log_roc,
                   LDA = lda_roc,
                   QDA = qda_roc,
                   kNN = knn_roc)
methods_auc <- paste(c("Logistic",
                       "LDA","QDA","KNN"),
        "AUC =",
        round(c(log_auc, lda_auc, qda_auc, knn_auc), 4))
ggroc(models_roc, linewidth = 1, alpha = 0.5)+
  scale_color_discrete(labels = methods_auc)+
  ggtitle(paste("ROC using logistic, LDA, QDA, and KNN"))
```



- **All four models** have *similar predictive capabilities* **based on the ROC curves**. Models **logistic regression, LDA, QDA, and standardized KNN (K=39)** have AUC values around **0.90 to 0.91**. This indicates **strong predictive capability** for differentiating between **expensive and non expensive homes**. **LDA** has the **highest AUC at ~0.9121**, followed by **KNN and logistic regression** at **~0.9107 and ~0.9103** *respectively*, and finally **QDA** with the **lowest AUC value** of **~0.9034**. This suggests that the **relationship between predictors and response** is modeled well by *linear decision boundaries* and that *quadratic boundaries* (QDA) does **not improve performance**. Overall, the models result in **almost identical AUC values** with *no single model drastically outperforming the others*.

Q2

For questions (a)–(c), use the data `my_sahp` to fit the following five models of `expensive` on all other variables:

- Model 1: logistic regression.
- Model 2: LDA.
- Model 3: QDA.
- Model 4: KNN with $K = 39$.
- Model 5: **standardized** KNN with $K = 39$. (use `caret::preProcess` function on the **training** sets for data standardization; see Lecture 4 slides)

For each question below, **you should**

- Manually split the dataset (instead of using functions like `cv.glm`). This is to ensure that each of the above models uses the same training and validation sets, so that it makes a fair comparison.
- Whenever you split the dataset (e.g., using the `sample` function as we learned in class), you should put `set.seed(1)` right above the data-splitting code to ensure reproducibility.

a. Use the validation set approach to divide the data into training (50%) and validation. Compute the validation classification error for each model and decide which model is the best.

Solution:

```
#Q2.a
#splitting data
n <- nrow(my_sahp)

set.seed(1)

tr_ind <- sample(n, round(n/2))
sahp_tr <- my_sahp[tr_ind,]
sahp_val <- my_sahp[-tr_ind,]

# Logistic
class_labels <- levels(my_sahp$expensive)
c0 <- class_labels[1]
c1 <- class_labels[2]

log_fit <- glm(expensive~., data = sahp_tr, family = binomial)
log_prob <- predict(log_fit, newdata = sahp_val, type = "response")
log_pred <- ifelse(log_prob > 0.5, c1, c0)

# LDA
lda_fit <- lda(expensive~., data = sahp_tr)
lda_pred <- predict(lda_fit, newdata = sahp_val)$class

# QDA
qda_fit <- qda(expensive~., data = sahp_tr)
qda_pred <- predict(qda_fit, newdata = sahp_val)$class

# KNN (K=39)
knn_fit <- knn3(expensive~., data = sahp_tr, k = 39)
knn_pred <- predict(knn_fit, newdata = sahp_val, type = "class")

# KNN (Standardized, K=39)
fit_std <- preProcess(sahp_tr, method = "scale")
sahp_tr_std <- predict(fit_std, newdata = sahp_tr)
sahp_val_std <- predict(fit_std, newdata = sahp_val)
```

```r
knn_std_fit <- knn3(expensive~., data = sahp_tr_std, k = 39)
knn_std_pred <- predict(knn_std_fit,
                        newdata = sahp_val_std,
                        type = "class")

# Validation errors
cv_error_seq <- c(
  Logistic = mean(log_pred != sahp_val$expensive),
  LDA = mean(lda_pred != sahp_val$expensive),
  QDA = mean(qda_pred != sahp_val$expensive),
  KNN = mean(knn_pred != sahp_val$expensive),
  KNN_std = mean(knn_std_pred != sahp_val$expensive)
)
cv_error_seq
```

```
##  Logistic       LDA       QDA       KNN   KNN_std
## 0.2345679 0.2222222 0.2222222 0.2839506 0.2098765
```

```r
best_model <- names(which.min(cv_error_seq))
best_model
```

```
## [1] "KNN_std"
```

- The **standardized KNN model** has the **lowest validation error** at around *21%*, making it the **best** model under the validation set approach.

b. Use LOOCV approach to compute the CV classification error for each model and decide which model is the best.

```r
**Solution:**

# Q2.b
#LOOCV approach (CV classification error) for each model
log_seq <- factor(levels = class_labels)
lda_seq <- factor(levels = class_labels)
qda_seq <- factor(levels = class_labels)
knn_seq <- factor(levels = class_labels)
knnstd_seq <- factor(levels = class_labels)

for (j in 1:n){
  sahp_tr <- my_sahp[-j,]
  sahp_te <- my_sahp[j,]

# Logistic
log_fit <- glm(expensive~., data = sahp_tr, family = "binomial")
log_prob <- predict(log_fit, newdata = sahp_te, type = "response")
log_seq[j] <- ifelse(log_prob>0.5,c1,c0)

#LDA
```

```r
lda_fit <- lda(expensive~., data = sahp_tr)
lda_seq[j] <- predict(lda_fit, sahp_te)$class

#QDA
qda_fit <- qda(expensive~., data = sahp_tr)
qda_seq[j] <- predict(qda_fit, sahp_te)$class

#KNN
knn_fit <- knn3(expensive~., data = sahp_tr, k=39)
knn_seq[j] <- predict(knn_fit, sahp_te, type="class")

#standardized KNN
fit_std <- preProcess(sahp_tr, method = "scale")
sahp_tr_std <- predict(fit_std, sahp_tr)
sahp_te_std <- predict(fit_std, sahp_te)

knnstd_fit <- knn3(expensive ~., data = sahp_tr_std, k=39)
knnstd_seq[j] <- predict(knnstd_fit, sahp_te_std, type="class")
}

#LOOCV classification error for each model
cv_err_loocv <- c(
  logistic = mean(log_seq != my_sahp$expensive),
  LDA = mean(lda_seq != my_sahp$expensive),
  QDA = mean(qda_seq != my_sahp$expensive),
  KNN = mean(knn_seq != my_sahp$expensive),
  KNN_std = mean(knnstd_seq != my_sahp$expensive)
)

cv_err_loocv

##  logistic       LDA       QDA       KNN   KNN_std
## 0.1790123 0.1790123 0.1851852 0.2654321 0.1913580

best_model_loocv <- names(which.min(cv_err_loocv))
best_model_loocv

## [1] "logistic"
```

- The **classification errors** for the five models are: - **Logistic regression = 0.1790123** - **LDA = 0.1790123** - **QDA = 0.1851852** - **KNN (K=39) = 0.2654321** - **Standardized KNN (K=39) = 0.1913580** - **Logistic regression** and **LDA** *tied* for **lowest LOOCV classification error** *(~17.9%)*. Therefore, the **logistic regression and LDA** perform the **best** under *LOOCV validation method*. QDA and standardized KNN perform slightly worse, while un-standardized KNN performs substantially worse.

c. Use 5-fold CV approach to compute the CV classification error for each model and decide which model is the best.

Solution:

```
log_seq_fold <- factor(levels = class_labels)
lda_seq_fold <- factor(levels = class_labels)
qda_seq_fold <- factor(levels = class_labels)
knn_seq_fold <- factor(levels = class_labels)
knnstd_seq_fold <- factor(levels = class_labels)

K <- 5
set.seed(1)
fold_ind <- sample(rep_len(1:K, n))


for(k in 1:K){
  sahp_tr <- my_sahp[fold_ind != k,]
  sahp_te <- my_sahp[fold_ind == k,]

#logistic
log_fit <- glm(expensive ~., data = sahp_tr, family=binomial)
log_prob <- predict(log_fit, newdata=sahp_te, type="response")
log_seq_fold[fold_ind==k] <-ifelse(log_prob>0.5,c1,c0)

#LDA
lda_fit <- lda(expensive~., data = sahp_tr)
lda_seq_fold[fold_ind == k] <- predict(lda_fit, sahp_te)$class

#QDA
qda_fit <- qda(expensive~., data = sahp_tr)
qda_seq_fold[fold_ind==k] <- predict(qda_fit, sahp_te)$class

#KNN
knn_fit <- knn3(expensive~., data = sahp_tr, k=39)
knn_seq_fold[fold_ind==k] <- predict(knn_fit,sahp_te,type="class")

#standardized KNN
fit_std <- preProcess(sahp_tr, method="scale")
sahp_tr_std <- predict(fit_std, sahp_tr)
sahp_te_std <- predict(fit_std, sahp_te)

knnstd_fit <- knn3(expensive~., data = sahp_tr_std,k=39)
knnstd_seq_fold[fold_ind==k] <- predict(knnstd_fit, sahp_te_std, type="class"
)
}

cv_err_fold <- c(
  Logistic = mean(log_seq_fold != my_sahp$expensive),
  LDA = mean(lda_seq_fold != my_sahp$expensive),
  QDA = mean(qda_seq_fold != my_sahp$expensive),
  KNN = mean(knn_seq_fold != my_sahp$expensive),
```

```
  KNN_std = mean(knnstd_seq_fold != my_sahp$expensive)
)

cv_err_fold

##  Logistic       LDA       QDA       KNN   KNN_std
## 0.1975309 0.2037037 0.1851852 0.2592593 0.2037037

best_model_fold <- names(which.min(cv_err_fold))
best_model_fold

## [1] "QDA"
```

The **classification errors** for the five models are: - **Logistic regression = 0.1975309** - **LDA = 0.2037037** - **QDA = 0.1851852** - **KNN (K=39) = 0.2592593** - **Standardized KNN (K=39) = 0.2037037** - The **QDA model** has the **lowest** 5 fold CV error at **0.1851852**. Therefore, **QDA** performs the **best** among the five models under this evaluation.
Q3

ISLRv2 Chapter 4 Q4 (Page 189). To receive full credits, you need to show your work.

Solution: Q4. (A) Given we have *1 predictor variable* (p = 1) that is uniform [0,1]: If we use observations within 10% of the range, **we expect to use 10% of the observations for each prediction**. So on average, we will use **10% of observations for each prediction**.

(B) Given we have **2 predictor variables** (p = 2) that is uniform [0,1]: If we use observations within 10% of the range for each predictor, **we form a square that has features**: *[x1 +- 0.05] X [x2 +- 0.05]* whose area is **0.1 X 0.1 = 0.01**, therefore the **fraction of observations used is: 1%**.

(C) Given we have **100 predictor variables** (p = 100) that is uniform [0,1]: If we use observations within 10% of the range for each predictor, **we form a 100 dimensional hyper cube with features**: Each side has a length of 0.10 and therefore **(0.1)^100 represents the 100 dimensional hyper volume of the hyper cube**. Therefore, the value is incredible small *(1 X 100^-100)*. This means **the fraction of observations used for the prediction is 10^-100**.

(D) As the dimension **p increases**, the **volume** of the hyper shape becomes **(0.1)^p**. This **shrinks exponentially** toward 0. Therefore, in *low dimensions* there **are nearby points**, but in *high dimensions* there are **almost no points nearby**. In high dimensional space, KNN uses extremely distant (non-local) points which leads to bad predictions.

(E) Given we want to create a p-dimensional hypercube centered around the test observation that contains 10% of the training observations on average: Let the side of the length be l:

- The **volume of the hypercube is l^p**
- The **fraction of points** it contains must be: **l^p = 0.10**
- Thefefore, the **side length is: l = 0.10^1/p**
- Now, we plug in for **p = 1, 2, and 100**.
- **For p = 1, the length = 0.10^1/1 = 0.10**
- **For p = 2, the length = 0.10^1/2 = 0.316**
- **For p = 100, the length = 0.10^1/100 = 0.97725** This means that as **p dimensions increase**, the hyper cube must **extend closer to the full range of data** and therefore the predictions using KNN **no longer becomes local** and **essentially useless**.