STAT-562 Homework 1

Prashanna Raj Pandit

Problem 1:

(a) Compute the Euclidean distance between each observation and the test point, X1 = X2 = X3 = 0.

Euclidian distance $(d) = \sqrt{(X_1 - 0)^2 + (X_2 - 0)^2 + (X_3 - 0)^2}$.

Obs	X_1	X_2	X_3	Substitution in formula	Distance d
1	0	3	0	$\sqrt{0^2 + 3^2 + 0^2} = \sqrt{9}$	3.000
2	2	0	0	$\sqrt{2^2 + 0^2 + 0^2} = \sqrt{4}$	2.000
3	0	1	3	$\sqrt{0^2 + 1^2 + 3^2} = \sqrt{10}$	3.162
4	0	1	2	$\sqrt{0^2 + 1^2 + 2^2} = \sqrt{5}$	2.236
5	−1	0	1	$\sqrt{(-1)^2 + 0^2 + 1^2} = \sqrt{2}$	1.414
6	1	1	1	$\sqrt{1^2 + 1^2 + 1^2} = \sqrt{3}$	1.732

(b) What is our prediction for Y1 with K = 1? Why?

The prediction for Y1 with K=1 is Green. When using $K = 1$, the algorithm looks at only the single nearest neighbor to the test point $(0,0,0)$. From our distance calculations, the closest observation is Observation 5, which is 1.414 units away and belongs to the Green class.

(c) What is our prediction for Y1 with K = 3? Why?

The predicted class for Y_1 is Red. When $K = 3$, we consider the three nearest neighbors to the test point, which are Observations 5 (Green), 6 (Red), and 2 (Red). Among these, two observations are classified as Red and one as Green. Since the majority class is Red, the model predicts that the new data point also belongs to the Red category.

(d) If we were predicting Y2 instead of Y1, what would the predicted value be with K = 3? Justify your answer.

The predicted value for Y_2 is approximately 6.77.

When predicting a numerical response like Y_2, the k-NN algorithm performs regression by averaging the values of the nearest neighbors. Using the same three nearest points (Observations 5, 6, and 2), their Y_2values are 6.3, 7.2, and 6.8, respectively. Taking the average gives: (6.3+7.2+6.8)/3 = 6.77

Problem 2:

(a) Load the dataset

```
# Load dataset
data("mtcars")
df <- mtcars

# Number of rows and columns
n_row <- nrow(df)
n_col <- ncol(df)

response_var <- c("am")   # Transmission type (0 = Automatic, 1 = Manual)
qualitative_vars <- c("vs")  # Engine type (0 = V-shaped, 1 = Straight)
quantitative_vars <- setdiff(names(df), c(response_var, qualitative_vars))
n_quant <- length(quantitative_vars)
n_qual  <- length(qualitative_vars)
prop<-round(prop.table(table(df$am))*100,digits=1)

cat("Number of rows:", n_row, "\n")

## Number of rows: 32

cat("Number of columns:", n_col, "\n")

## Number of columns: 11

cat("Quantitative variables:", n_quant, "\n")

## Quantitative variables: 9

cat("Qualitative variables:", n_qual, "\n")

## Qualitative variables: 1

cat("Proportion of each category of the response:", prop)

## Proportion of each category of the response: 59.4 40.6
```

(b) Importance of standardizing quantitative predictors:

Answer: k-NN is a distance-based algorithm, so it's sensitive to the scale of features. If one variable has a much larger numeric range than others, it will dominate the distance calculation and bias the results. Standardizing puts all quantitative predictors on a common scale, ensuring each feature contributes equally to the distance and the model's decisions.

(c) Data Preparation:

(i) Converting the qualitative predictors to factors

Answer: The purpose of converting numeric columns to factors is to help R and machine learning models correctly interpret categorical information. Without this step, algorithms may treat the values 0 and 1 as numerical quantities, implying an order or magnitude difference that

doesn't exist. By defining them as factors, R knows these values represent categories or labels, not numerical scales.

```
# Convert the qualitative predictor to a factor variable in df
df$vs <- factor(df$vs, levels = c(0, 1), labels = c("V-shaped", "Straight"))

# Convert factor variable into dummy variables (one-hot encoding)
vs_encoded <- model.matrix(~ vs, data = df)[, -1]

# Convert response to factor with readable labels
df$am <- factor(df$am, levels = c(0, 1), labels = c("Automatic", "Manual"))
```

(ii) Standardize the data

Answer: Before standardization, the variable hp (horsepower) had values ranging from 52 to 335, with a mean of 146.7 and a median of 123. This shows that horsepower values vary widely across cars and exist on a much larger numerical scale compared to other variables.

After applying standardization using the scale() function, the same variable was transformed to have a mean of approximately 0 and a standard deviation of 1. The new standardized values range from about –1.38 to 2.75, representing how far each car's horsepower is from the average in standard deviation units.

This makes all quantitative predictors comparable in magnitude, ensuring that features like horsepower do not dominate the k-NN distance calculations simply because they were measured on a larger numeric scale.

```
scaled_quant <- scale(df[, quantitative_vars])
# Display summary of 'hp' before and after standardization
summary(df$hp)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    52.0    96.5   123.0   146.7   180.0   335.0

summary(scaled_quant[,"hp"])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.3810 -0.7320 -0.3455  0.0000  0.4859  2.7466

data_ready<-cbind(as.data.frame(scaled_quant),vs_encoded)
```

(d) Train/Test split: (70/30)

(i)

Answer: In the training set, the response variable (am) consists of 59.1% Automatic and 40.9% Manual cars. In the test set, the proportions are 60% Automatic and 40% Manual.

```
library(rsample)
set.seed(123)

# Combine predictors and response temporarily for splitting
split_data <- cbind(data_ready, am = df$am)
```

```
# Perform 70/30 stratified split by 'am'
split_obj <- initial_split(split_data, prop = 0.70, strata = am)

# Extract training and test datasets
train_data <- training(split_obj)
test_data  <- testing(split_obj)

# Check proportions in train/test
prop.table(table(train_data$am))

##
## Automatic    Manual
## 0.5909091 0.4090909

prop.table(table(test_data$am))

##
## Automatic    Manual
##       0.6       0.4

train_labels<-train_data$am
test_labels<-test_data$am
# Remove the label column from the feature matrices
train_data <- subset(train_data, select = -am)
test_data  <- subset(test_data, select = -am)
```

(ii) Importance of tratified sampling:

Answer: Maintaining the same distribution of the response variable in both the training and testing sets is important because it ensures that the model learns and is evaluated on data that truly represent the overall population.

If one class (for example, “Automatic”) is overrepresented in the training data but underrepresented in the test data, the model may become biased learning patterns that favor the majority class and performing poorly on the minority class.

By using stratified sampling, we make sure both sets reflect the real-world balance between Automatic and Manual cars. This leads to a more reliable, fair, and accurate evaluation of the model’s performance.

(e) k-NN with k = √n

```
library(class)
k=sqrt(nrow(train_data))
k

## [1] 4.690416

test_pred<-knn(train = train_data,test = test_data,cl=train_labels,k=5)
```

(f) onfusion matrix and calculate accuracy

```
table(test_labels,test_pred)
```

```
##            test_pred
## test_labels Automatic Manual
##   Automatic         4      2
##   Manual            0      4
```

```
accuracy<-mean(test_labels==test_pred)
accuracy
```

```
## [1] 0.8
```

(g) Perform hyperparameter tuning

```
library(caret)
```

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

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:rsample':
##
##     calibration
```

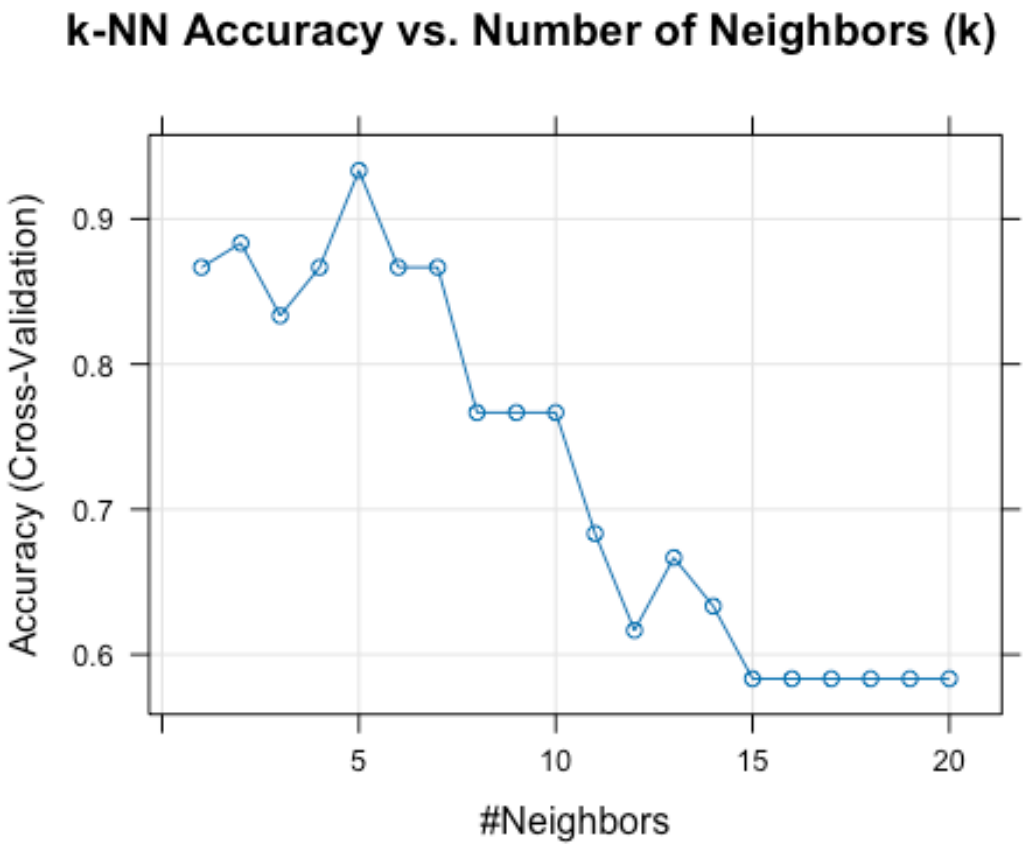
```
set.seed(123)
ctrl<-trainControl(method = "cv",number=10)
knn_grid<-expand.grid(k=1:20)
train_knn<-cbind(train_data,am=train_labels)

# train kNN using cross validation
knn_cv<-train(am~.,data=train_knn,
              method="knn",
              trControl=ctrl,
              tuneGrid=knn_grid)
```

```
## Warning in knn3Train(train = structure(c(-0.230734525663942,
## -0.330287399658271, : k = 20 exceeds number 19 of patterns
```

```
## Warning in knn3Train(train = structure(c(-0.230734525663942,
## -0.960788934955696, : k = 20 exceeds number 19 of patterns
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
best_k<-knn_cv$bestTune$k
plot(knn_cv, main = "k-NN Accuracy vs. Number of Neighbors (k)")
```




```
best_k
```

```
## [1] 5
```

(i)

Answer: Based on the accuracy vs. k plot, the optimal value of k for this dataset is 5, where the model achieves the highest cross-validated accuracy. So, it is selected as the best number of neighbors for building the final k-NN model.

(h) final k-NN model with cross validation

```
train_control<-trainControl(method = "cv", number = 10, classProbs = TRUE, savePredictions = "final")
# Fit final k-NN model with cross validation
final_model_cv<-train(am~.,data=train_knn,method="knn",tuneGrid=data.frame(k=best_k),
trControl=train_control)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
#print(final_model_cv)
```

```
# predict on test dataset
final_prediction<-predict(final_model_cv,newdata = test_data)
final_prediction
```

```
##  [1] Automatic Manual    Manual    Automatic Automatic Manual    Manual
##  [8] Automatic Manual    Manual
## Levels: Automatic Manual
```

(i) Performance

```
final_accuracy<-mean(final_prediction==test_labels)
final_accuracy
```

```
## [1] 0.8
```

```
conf_matrix <- confusionMatrix(final_prediction, test_labels)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  Automatic Manual
##   Automatic         4      0
##   Manual            2      4
##
##                Accuracy : 0.8
##                  95% CI : (0.4439, 0.9748)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.1673
##
##                   Kappa : 0.6154
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.6667
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6000
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.4000
##       Balanced Accuracy : 0.8333
##
##        'Positive' Class : Automatic
##
```

```
# Extract performance metrics
overall_accuracy  <- conf_matrix$overall["Accuracy"]
misclassification <- 1 - overall_accuracy
sensitivity       <- conf_matrix$byClass["Sensitivity"]   # True Positive Rate (Recall)
specificity       <- conf_matrix$byClass["Specificity"]   # True Negative Rate
precision         <- conf_matrix$byClass["Precision"]
recall            <- conf_matrix$byClass["Recall"]
f1_score          <- conf_matrix$byClass["F1"]

cat("Overall Accuracy:", round(overall_accuracy, 4), "\n")
```

```
## Overall Accuracy: 0.8
```

```
cat("Misclassification Rate:", round(misclassification, 4), "\n")
```

```
## Misclassification Rate: 0.2
```

```
cat("Sensitivity:", round(sensitivity, 4), "\n")
```

```
## Sensitivity: 0.6667
```

```
cat("Specificity:", round(specificity, 4), "\n")
```

```
## Specificity: 1
```

```
cat("Precision:", round(precision, 4), "\n")
```

```
## Precision: 1
```

```
cat("Recall:", round(recall, 4), "\n")
```

```
## Recall: 0.6667
```

```
cat("F1 Score:", round(f1_score, 4), "\n")
```

```
## F1 Score: 0.8
```

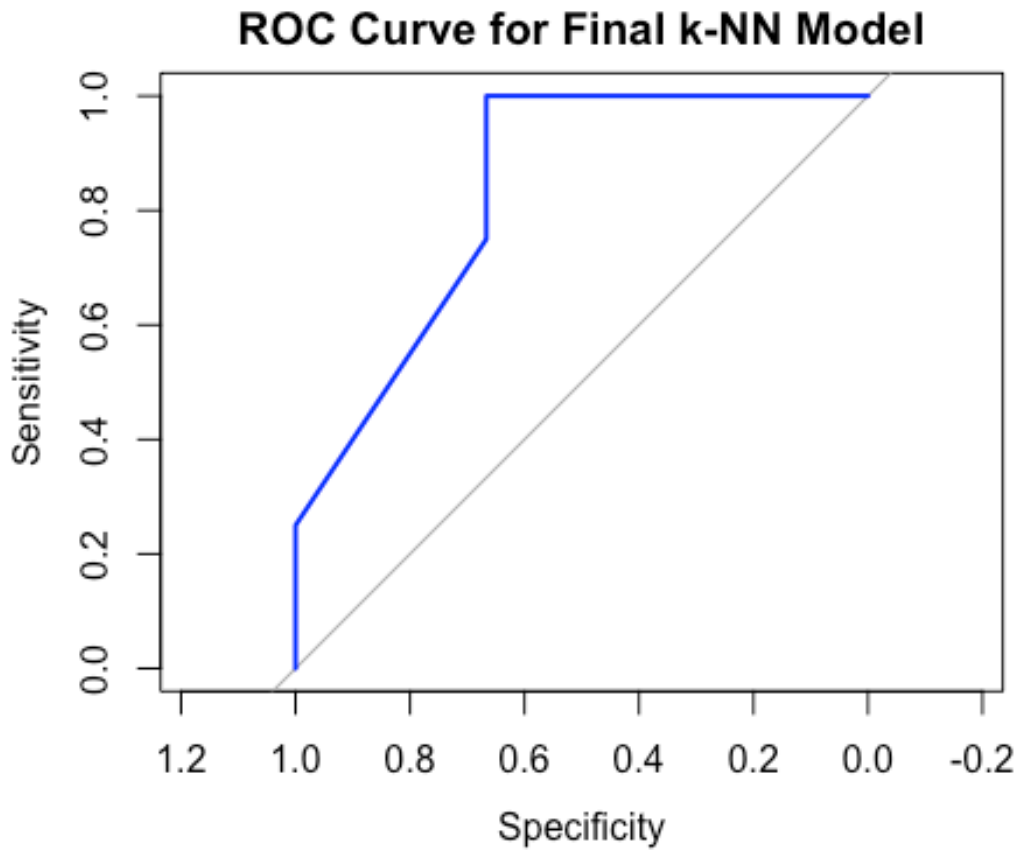
```
# --- ROC Curve and AUC evaluation ---
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Get predicted probabilities for the positive class ("Manual")
prob_predictions <- predict(final_model_cv, newdata = test_data, type = "prob")

# Build ROC curve
roc_curve <- roc(
  response = test_labels,
  predictor = prob_predictions$Manual,
  levels = levels(test_labels)
)
```

```
## Setting direction: controls < cases
```

```
# Plot ROC curve
plot(roc_curve, col = "blue", main = "ROC Curve for Final k-NN Model")
```




```
auc_value <- auc(roc_curve)
auc_value

## Area under the curve: 0.8333
```

(ii)

Sensitivity: The sensitivity of 0.67 means that the model correctly identified about 67% of the cars that actually have automatic transmission. In simple terms, out of all the automatic cars, the model missed roughly one-third; it occasionally misclassified them as manual.

Specificity: The specificity of 1.00 means the model correctly recognized all manual cars - none of them were wrongly labeled as automatic. So, when the model says a car is manual, it's always right.

Precision: The precision of 1.00 indicates that every car predicted as automatic was truly automatic. In other words, the model never made a false claim that a manual car was automatic, it's very cautious when predicting automatics.

Recall: The recall of 0.67 indicates that the model successfully identified about two-thirds of all actual automatic cars. This means it was able to detect most of the automatic vehicles, but it still missed a few, labeling them as manual instead. In essence, recall measures how completely the model captures the "automatic" class - a higher recall would mean fewer missed detections. In

this case, the model demonstrates good but not perfect coverage, prioritizing precision (being certain) over recall (catching every automatic).