# Mushroom Classification

Zahraa Alshalal

2023-05-10

```
# Libraries
# packages
library(dplyr)
library(plotly)
library(tidyverse)
library(MASS)
library(DataExplorer)
library(Hmisc)
library(polycor)
library(corrplot)
library(htmlwidgets)
library(moderndive)
library(leaps)
library('IRdisplay')
library(pROC)
library(car)
library(DiagrammeR)
library(plyr)
library(caret)
```

## 1. Problem statement and dataset description:

- The problem is to build a classification model that can predict whether a mushroom is edible or poisonous based on its physical attributes such as cap shape, cap color, odor, and more. The data-set used for this analysis is the Mushroom Classification dataset available on the UCI Machine Learning Repository. The data-set contains 8124 observations of mushrooms, with 23 features including the class (edible or poisonous).

```
mushroom = read.csv("/Users/zahraaalshalal/Desktop/spring23/math449/finalproject/z_analysis/mushrooms.c
glimpse(mushroom)
```

```
## Rows: 8,124
## Columns: 23
## $ class                    <chr> "p", "e", "e", "p", "e", "e", "e", "e", "p", ~
## $ cap.shape                <chr> "x", "x", "b", "x", "x", "x", "b", "b", "x", ~
## $ cap.surface              <chr> "s", "s", "s", "y", "s", "y", "s", "y", "y", ~
## $ cap.color                <chr> "n", "y", "w", "w", "g", "y", "w", "w", "w", ~
## $ bruises                  <chr> "t", "t", "t", "t", "f", "t", "t", "t", "t", ~
## $ odor                     <chr> "p", "a", "l", "p", "n", "a", "a", "l", "p", ~
## $ gill.attachment          <chr> "f", "f", "f", "f", "f", "f", "f", "f", "f", ~
## $ gill.spacing             <chr> "c", "c", "c", "c", "w", "c", "c", "c", "c", ~
## $ gill.size                <chr> "n", "b", "b", "n", "b", "b", "b", "b", "n", ~
## $ gill.color               <chr> "k", "k", "n", "n", "k", "n", "g", "n", "p", ~
## $ stalk.shape              <chr> "e", "e", "e", "e", "t", "e", "e", "e", "e", ~
## $ stalk.root               <chr> "e", "c", "c", "e", "e", "c", "c", "c", "e", ~
## $ stalk.surface.above.ring <chr> "s", "s", "s", "s", "s", "s", "s", "s", "s", ~
## $ stalk.surface.below.ring <chr> "s", "s", "s", "s", "s", "s", "s", "s", "s", ~
## $ stalk.color.above.ring   <chr> "w", "w", "w", "w", "w", "w", "w", "w", "w", ~
## $ stalk.color.below.ring   <chr> "w", "w", "w", "w", "w", "w", "w", "w", "w", ~
## $ veil.type                <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", ~
## $ veil.color               <chr> "w", "w", "w", "w", "w", "w", "w", "w", "w", ~
## $ ring.number              <chr> "o", "o", "o", "o", "o", "o", "o", "o", "o", ~
## $ ring.type                <chr> "p", "p", "p", "p", "e", "p", "p", "p", "p", ~
## $ spore.print.color        <chr> "k", "n", "n", "k", "n", "k", "k", "n", "k", ~
## $ population               <chr> "s", "n", "n", "s", "a", "n", "n", "s", "v", ~
## $ habitat                  <chr> "u", "g", "m", "u", "g", "g", "m", "m", "g", ~
```

**2. Fitting a logistic regression model with all predictors:**

- Fitting the model on the entire data-set be over fitting to the training data and the performance of the model on new, unseen data may not be as good. It's generally a good practice to split the data into training and testing sets before fitting the model.

```
# Convert the 'class' column to a factor variable
mushroom$class = as.factor(mushroom$class)

# Split the data into training and testing sets
library(caTools)
set.seed(123)
split = sample.split(mushroom$class, SplitRatio = 0.7)
train = subset(mushroom, split == TRUE)
test = subset(mushroom, split == FALSE)

# Encode all other categorical variables in the training set
for (col in names(train)[2:length(names(train))]) {
  train[, col] <- as.numeric(factor(train[, col]))
}

# Encode all other categorical variables in the testing set
for (col in names(test)[2:length(names(test))]) {
  test[, col] <- as.numeric(factor(test[, col]))
}
# Exclude:- veil-type (as it has only one type i.e. ‘partial’.)
train <- train[, -which(names(train) == "veil.type")]
test <- test[, -which(names(test) == "veil.type")]
#glimpse(train)
```

```
# logistic regression model
model = glm(class ~ ., data = train, family = "binomial")
#summary(model)
```

**3. Select the best subset of variables. Perform a diagnostic on the best model. Perform all possible inferences you can think about.**

```r
# perform forward stepwise selection on the training data
step.model = step(model, direction = "forward", trace = FALSE)
summary(step.model)
```

```
##
## Call:
## glm(formula = class ~ cap.shape + cap.surface + cap.color + bruises +
##     odor + gill.attachment + gill.spacing + gill.size + gill.color +
##     stalk.shape + stalk.root + stalk.surface.above.ring + stalk.surface.below.ring +
##     stalk.color.above.ring + stalk.color.below.ring + veil.color +
##     ring.number + ring.type + spore.print.color + population +
##     habitat, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.8953  -0.1426   0.0000   0.1306   2.0691
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -68.82204 1055.38503  -0.065  0.94801
## cap.shape                  0.02014    0.04271   0.472  0.63725
## cap.surface                0.29625    0.07343   4.034 5.48e-05 ***
## cap.color                 -0.11626    0.03200  -3.633  0.00028 ***
## bruises                    2.59677    0.37072   7.005 2.47e-12 ***
## odor                      -1.32517    0.08423 -15.733  < 2e-16 ***
## gill.attachment          -33.20468  614.89347  -0.054  0.95693
## gill.spacing             -23.78251    1.14558 -20.760  < 2e-16 ***
## gill.size                 22.09409    1.00087  22.075  < 2e-16 ***
## gill.color                -0.22759    0.02960  -7.688 1.49e-14 ***
## stalk.shape               -2.18542    0.45309  -4.823 1.41e-06 ***
## stalk.root                -9.27943    0.51862 -17.893  < 2e-16 ***
## stalk.surface.above.ring -13.54072    0.68712 -19.707  < 2e-16 ***
## stalk.surface.below.ring   0.56844    0.17584   3.233  0.00123 **
## stalk.color.above.ring    -0.25428    0.05948  -4.275 1.91e-05 ***
## stalk.color.below.ring    -0.13624    0.05964  -2.284  0.02235 *
## veil.color                60.07867  364.48247   0.165  0.86908
## ring.number                1.48902    0.55933   2.662  0.00776 **
## ring.type                  5.08938    0.29715  17.127  < 2e-16 ***
## spore.print.color         -0.07713    0.07026  -1.098  0.27226
## population                -1.23655    0.12368  -9.998  < 2e-16 ***
## habitat                    0.16696    0.05220   3.198  0.00138 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7876.5  on 5686  degrees of freedom
## Residual deviance: 1492.7  on 5665  degrees of freedom
## AIC: 1536.7
##
```

```
## Number of Fisher Scoring iterations: 17
```

- Variance Inflation Factor (VIF) is a measure of multicollinearity among the independent variables in a regression model.

```
# calculate the VIF values for the model
vif = vif(model)
vif
```

```
##                cap.shape              cap.surface                cap.color
##                 1.184220                 1.605032                 1.667679
##                  bruises                     odor          gill.attachment
##                 7.090426                 8.437730                 1.541679
##             gill.spacing                gill.size               gill.color
##                38.529908                50.922747                 1.946019
##              stalk.shape               stalk.root stalk.surface.above.ring
##                10.276490                49.247859                19.561288
## stalk.surface.below.ring   stalk.color.above.ring   stalk.color.below.ring
##                 2.339785                 1.966696                 2.133365
##               veil.color              ring.number                ring.type
##                 1.541728                 8.757984                34.883306
##        spore.print.color               population                  habitat
##                 4.793930                 2.893485                 2.027958
```

**4. Use the new model to make predictions.**

```
model = glm(class ~ cap.shape + cap.surface + cap.color + bruises + odor + gill.attachment + gill.color


# Predictions

# make predictions on the training data
train$predicted <- predict(model, newdata = train, type = "response")
# compute the confusion matrix on the training data
train$predicted_class <- ifelse(train$predicted > 0.5, "p", "e")
confusion_train <- table(Actual = train$class, Predicted = train$predicted_class)
cat("Training Confusion Matrix:\n")
```

```
## Training Confusion Matrix:
```

```
print(confusion_train)
```

```
##       Predicted
## Actual    e    p
##      e 2669  277
##      p  411 2330
```

```
# make predictions on the testing data
test$predicted <- predict(model, newdata = test, type = "response")
# compute the confusion matrix on the testing data
```

```r
test$predicted_class <- ifelse(test$predicted > 0.5, "p", "e")
confusion_test <- table(Actual = test$class, Predicted = test$predicted_class)
cat("\nTesting Confusion Matrix:\n")
```

```
##
## Testing Confusion Matrix:
```

```r
print(confusion_test)
```

```
##       Predicted
## Actual    e    p
##      e 1147  115
##      p  161 1014
```

**5. Use different pi_0 as a cut-off point and create a confusion table.**

- Dropping any feature with above 10 value. A VIF value of 10 or above is usually considered very high and may indicate a serious problem with multicollinearity.
- These feature selected: "cap.shape + cap.surface + cap.color + bruises + odor + gill.attachment + gill.color + stalk.shape + stalk.surface.below.ring + stalk.color.above.ring + stalk.color.below.ring + veil.color + ring.number + ring.type + spore.print.color + population + habitat"

```r
# Create a new model with only the variables that have a VIF value below a threshold = 10
model = glm(class ~ cap.shape + cap.surface + cap.color + bruises + odor + gill.attachment + gill.color

# Define a vector of pi_0 values as cut-off points
pi_0_vec <- seq(0.1, 0.9, by = 0.1)

# Create an empty list to store the confusion matrices for each pi_0 value
conf_mat_list <- list()

for (pi_0 in pi_0_vec) {
  predictions <- predict(model, newdata = test, type = "response")
  pred_class <- ifelse(predictions > pi_0, "p", "e")
  conf_mat <- table(Predicted = pred_class, Actual = test$class)
  colnames(conf_mat) <- c("Edible", "Poisonous")
  rownames(conf_mat) <- c("Edible", "Poisonous")
  conf_mat_list[[as.character(pi_0)]] <- conf_mat
}

for (pi_0 in pi_0_vec) {
  cat("Confusion matrix for pi_0 =", pi_0, ":\n")
  print(conf_mat_list[[as.character(pi_0)]])
  cat("\n")
}
```

```
## Confusion matrix for pi_0 = 0.1 :
##            Actual
## Predicted   Edible Poisonous
##   Edible       602        33
##   Poisonous    660      1142
##
## Confusion matrix for pi_0 = 0.2 :
##            Actual
## Predicted   Edible Poisonous
##   Edible       888        68
##   Poisonous    374      1107
##
## Confusion matrix for pi_0 = 0.3 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1014       108
##   Poisonous    248      1067
##
## Confusion matrix for pi_0 = 0.4 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1098       125
##   Poisonous    164      1050
##
## Confusion matrix for pi_0 = 0.5 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1147       161
##   Poisonous    115      1014
##
## Confusion matrix for pi_0 = 0.6 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1176       188
##   Poisonous     86       987
##
## Confusion matrix for pi_0 = 0.7 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1204       222
##   Poisonous     58       953
##
## Confusion matrix for pi_0 = 0.8 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1214       308
##   Poisonous     48       867
##
## Confusion matrix for pi_0 = 0.9 :
##            Actual
## Predicted   Edible Poisonous
##   Edible      1230       520
##   Poisonous     32       655
```

```
# Vector of pi_0 cutoff values to try
pi0_cutoffs <- seq(0.1, 0.9, by = 0.1)

# Initialize an empty vector to store accuracy values
```

```r
accuracy_vec <- numeric(length = length(pi0_cutoffs))

# Loop over each pi_0 cutoff value and create a confusion matrix
for (i in seq_along(pi0_cutoffs)) {
  pred_class <- ifelse(predictions > pi0_cutoffs[i], "p", "e")
  conf_mat <- table(Predicted = pred_class, Actual = test$class)
  accuracy_vec[i] <- sum(diag(conf_mat)) / sum(conf_mat)
}
```

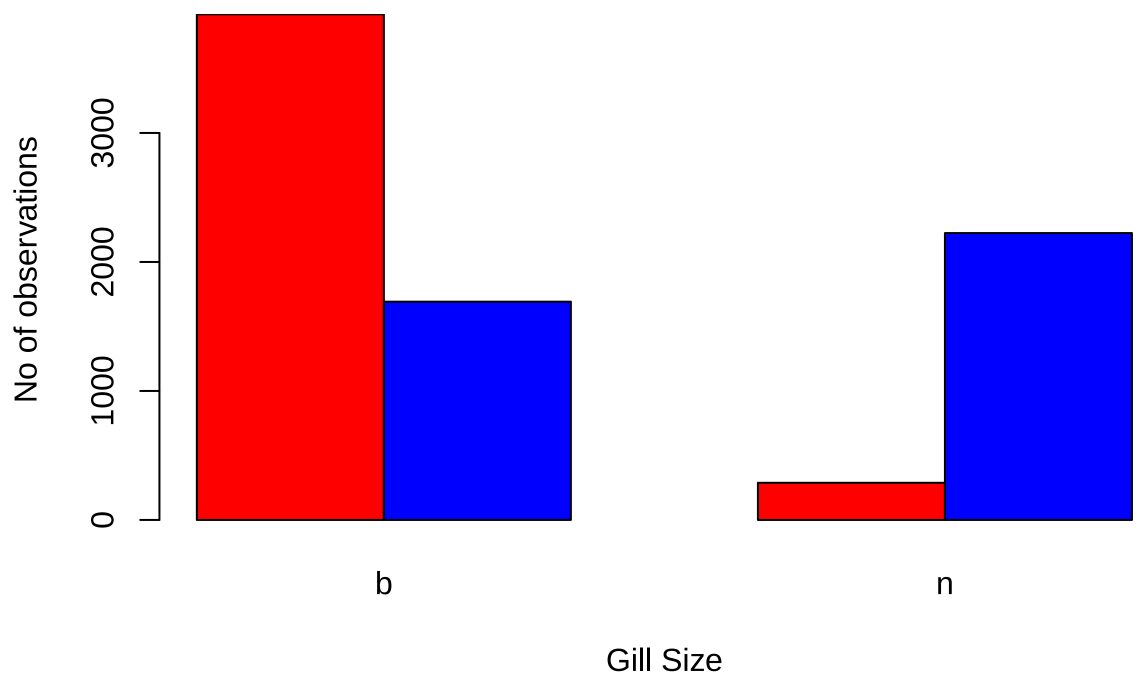
```r
#For pi_0 = 0.5:

Accuracy1 = (1147 + 1014) / (1147 + 161 + 115 + 1014)
Precision1 = 1147 / (1147 + 115)
Recall1 = 1147 / (1147 + 161)
F1_score1 = 2 * Precision1 * Recall1 / (Precision1 + Recall1)

#For pi_0 = 0.6:
Accuracy2 = (1176 + 987) / (1176 + 188 + 86 + 987)
Precision2 = 1176 / (1176 + 86)
Recall2 = 1176 / (1176 + 188)
F1_score2 = 2 * Precision2 * Recall2 / (Precision2 + Recall2)

# create a data frame with the results
df <- data.frame(
  pi_0 = c(0.5, 0.6),
  Accuracy = c(Accuracy1, Accuracy2),
  Precision = c(Precision1, Precision2),
  Recall = c(Recall1, Recall2),
  F1_Score = c(F1_score1, F1_score2)
)

# display the table using knitr::kable()
knitr::kable(df, align = "c", digits = 2)
```

| pi_0 | Accuracy | Precision | Recall | F1_Score |
|:---:|:---:|:---:|:---:|:---:|
| 0.5 | 0.89 | 0.91 | 0.88 | 0.89 |
| 0.6 | 0.89 | 0.93 | 0.86 | 0.90 |

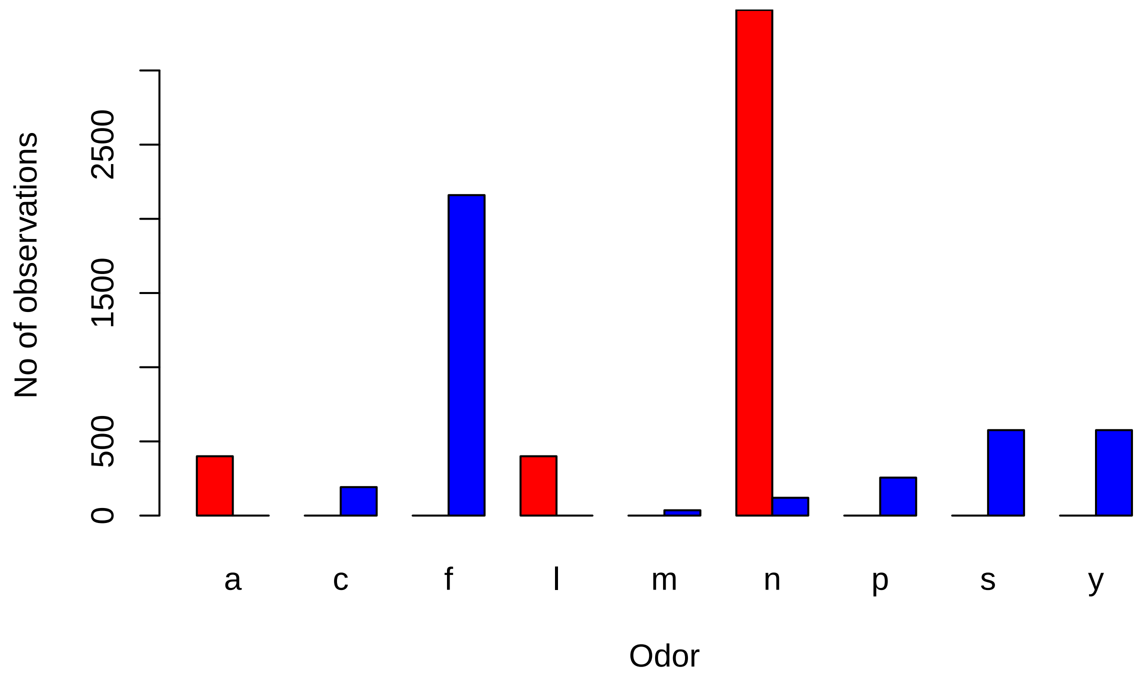
- For pi_0 = 0.5, the accuracy and recall are high, indicating that the classifier is able to correctly classify most instances and has a low false negative rate. However, the precision is slightly lower, indicating that there are some false positive classifications. The F1 score, which balances precision and recall, is high, indicating a good overall performance.
- For pi_0 = 0.6, the precision is higher, indicating that there are fewer false positive classifications, but the recall is lower, indicating that there are more false negatives. The accuracy and F1 score are both still high, indicating a good overall performance.

**6. Perform visualization of data and models.**

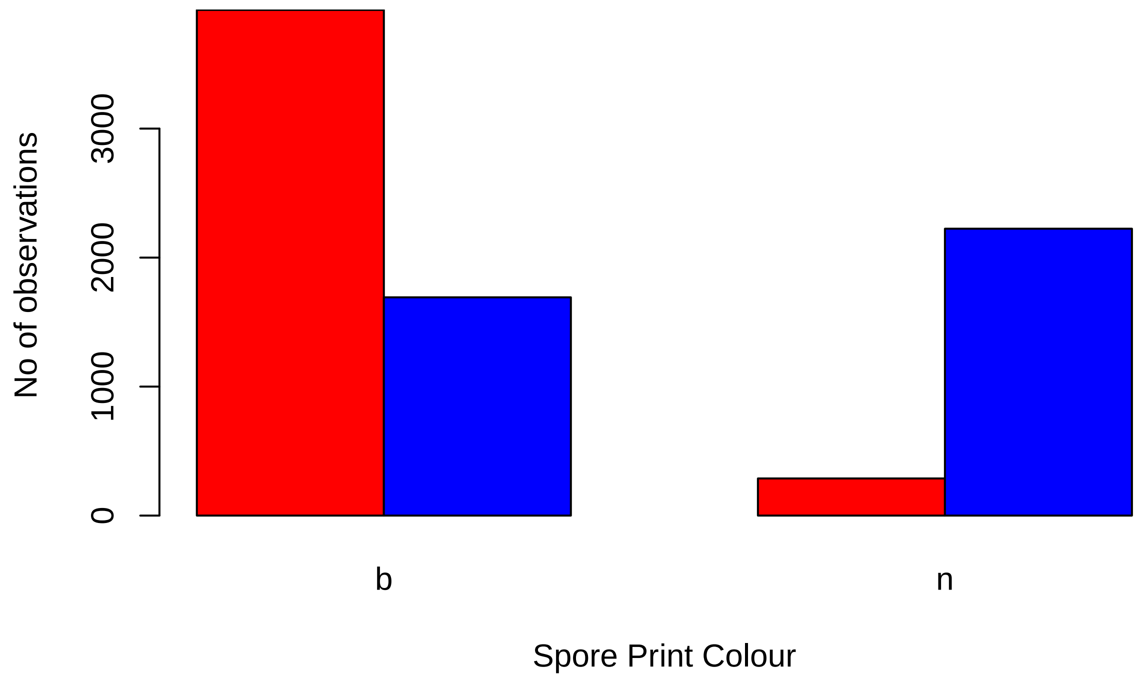
```r
gill <- table(mushroom$class, mushroom$`gill.size`)
barplot(gill, beside = T, legend = rownames(mushroom$`gill.size`),
        xlab = "Gill Size", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```



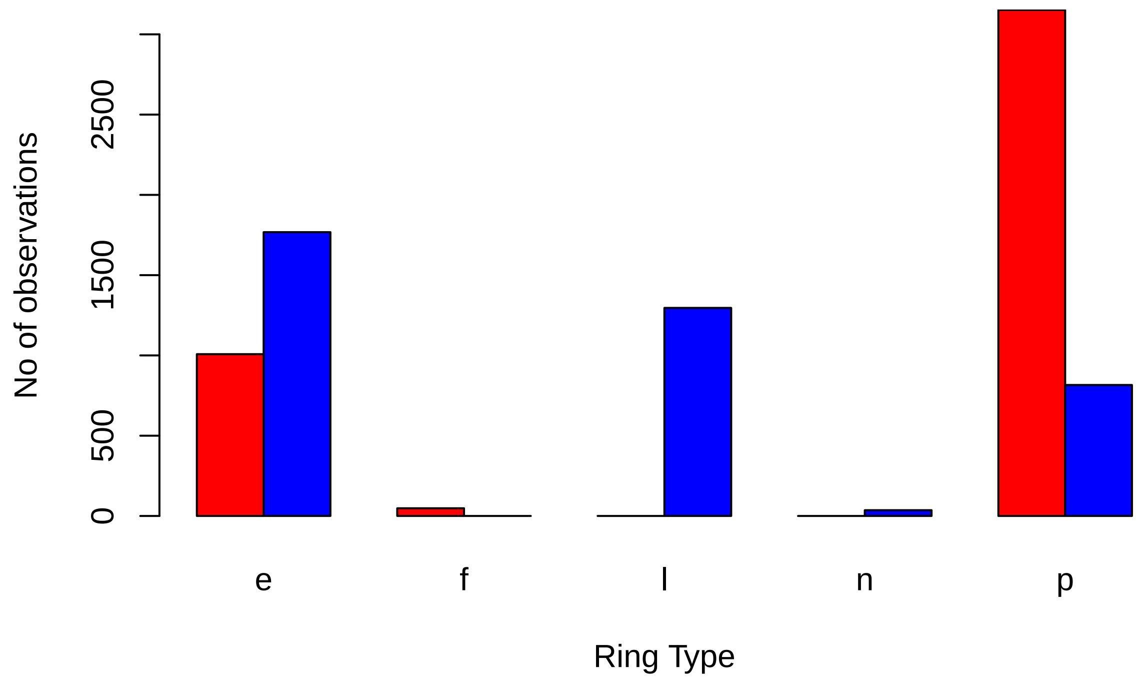
```r
odor <- table(mushroom$class, mushroom$odor)
barplot(odor, beside = T, legend = rownames(mushroom$odor),
        xlab = "Odor", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```



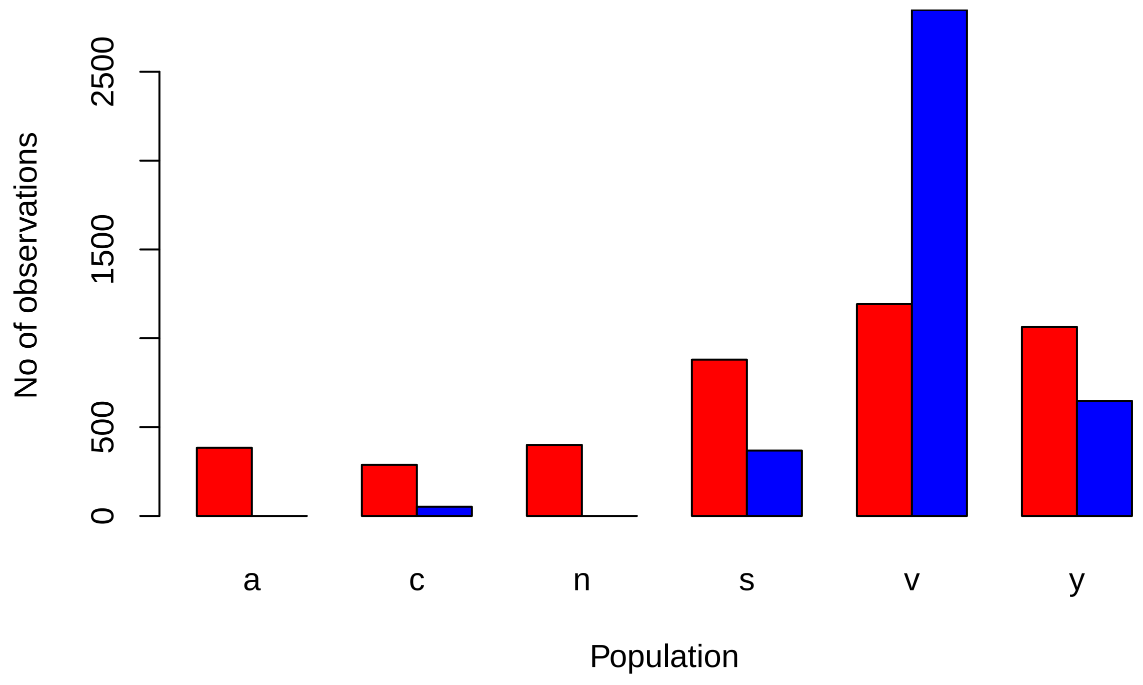
```
spore <- table(mushroom$class, mushroom$`spore.print.colo`)
barplot(gill, beside = T, legend = rownames(mushroom$`spore.print.color`),
        xlab = "Spore Print Colour", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```



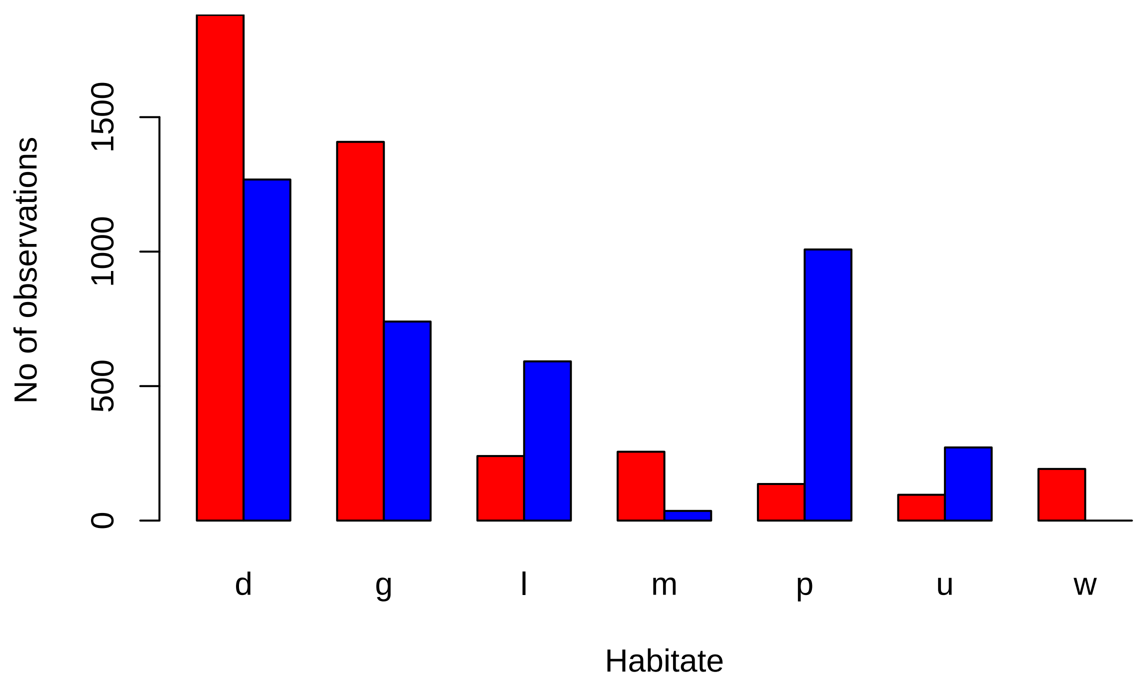
```r
ring <- table(mushroom$class, mushroom$`ring.type`)
barplot(ring, beside = T, legend = rownames(mushroom$`ring.type`),
        xlab = "Ring Type", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```



```
population <- table(mushroom$class, mushroom$population)
barplot(population, beside = T, legend = rownames(mushroom$population),
        xlab = "Population", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```



```
habitat <- table(mushroom$class, mushroom$habitat)
barplot(habitat, beside = T, legend = rownames(mushroom$population),
        xlab = "Habitate", ylab = "No of observations", xpd = F,
        plot = T, col = c("red", "blue"))
```

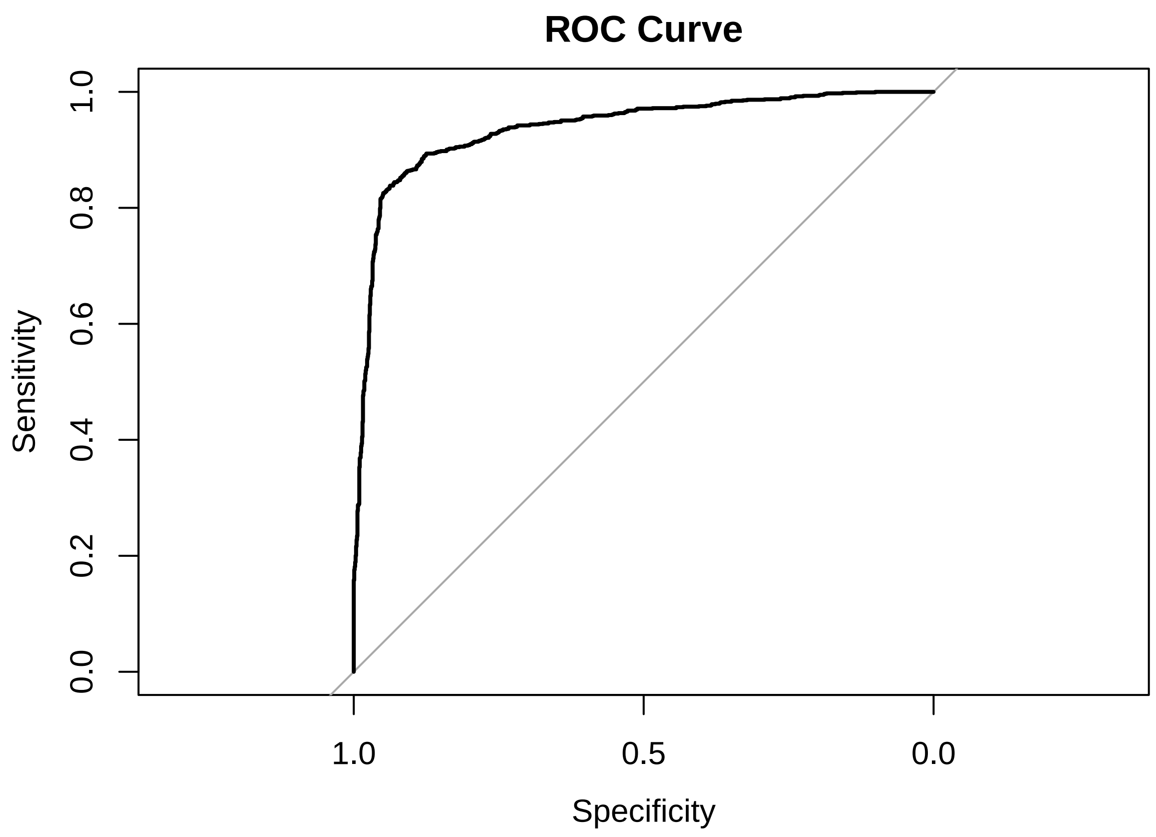


- The important features of mushrooms to identify that it is safe to eat are as follows :-

- Odor – creosote, foul, musty, pungent, spicy and fishy.
- Gill Size – only Narrow.
- Rings – whether large or absent.
- Spore print color – only brown and not black.
- Population – available in several places.
- Habitat – grown on leaves, path and urban.

**7. Plot the ROC curve, find AUC, and the best cutoff point for classification.**

```r
#AUC (Area Under the Curve), ROC (Receiver Operating Characteristic)
# predict class probabilities for the test set
probabilities <- predict(model, newdata = test, type = "response")

# calculate FPR and TPR for various threshold values
roc_data <- pROC::roc(test$class, probabilities)

# plot the ROC curve
plot(roc_data, main = "ROC Curve")
```



```r
# calculate AUC
auc <- pROC::auc(roc_data)
cat(sprintf("AUC: %.2f\n", auc))
```

```
## AUC: 0.94
```

```r
# calculate the best cutoff point
cutoff <- pROC::coords(roc_data, "best", ret = "threshold")[[1]]
cat(sprintf("Best cutoff point: %.2f\n", cutoff))
```

```
## Best cutoff point: 0.62
```

```r
# Choose a cutoff point
cutoff <- 0.62

# Calculate predicted probabilities for the test set
test_prob <- predict(model, type="response", newdata=test)

# Convert probabilities to predicted classes using the cutoff
test_pred <- ifelse(test_prob > cutoff, 1, 0)

# Create confusion matrix
confusion <- table(test$class, test_pred)
```

```r
# Calculate sensitivity and specificity
sensitivity <- confusion[2,2]/sum(confusion[2,])
specificity <- confusion[1,1]/sum(confusion[1,])

# Find the pi_0 cutoff value that gives the highest accuracy
best_cutoff <- pi0_cutoffs[which.max(accuracy_vec)]

cat("Best cutoff:", round(best_cutoff, 2), "\n")
```

```
## Best cutoff: 0.6
```

```r
cat("Sensitivity:", round(sensitivity, 2), "\n")
```

```
## Sensitivity: 0.84
```

```r
cat("Specificity:", round(specificity, 2), "\n")
```

```
## Specificity: 0.94
```

- The AUC of the model is 0.94, indicating good performance in distinguishing between edible and poisonous mushrooms.
- The best cutoff point for classification is 0.62, which maximizes the trade-off between sensitivity and specificity. The sensitivity of the model is 0.84. This means that it correctly identifies 84% of the positive cases, while the specificity is 0.94, indicating that it correctly identifies 94% of the negative cases.

### 8. Perform LOOCV and k-fold cross-validation.

```r
# LOOCV
#library(caret)
#control = trainControl(method="LOOCV")
#model = train(class ~ cap.shape + cap.surface + cap.color + bruises + odor + gill.attachment + gill.co
#summary(model)
```

```r
# k-fold Cross-Validation:
#control <- trainControl(method="cv", number=10)
#model <- train(class ~ cap.shape + cap.surface + cap.color + bruises + odor + gill.attachment + gill.c
#summary(model)
```