Cross validation and classification

John Paul Gosling

2024-11-07

In this practical, we will be utilising the `caret` package in R to perform cross-validation and classification tasks. The `caret` package is a powerful tool for machine learning in R and provides a consistent interface for many different models.

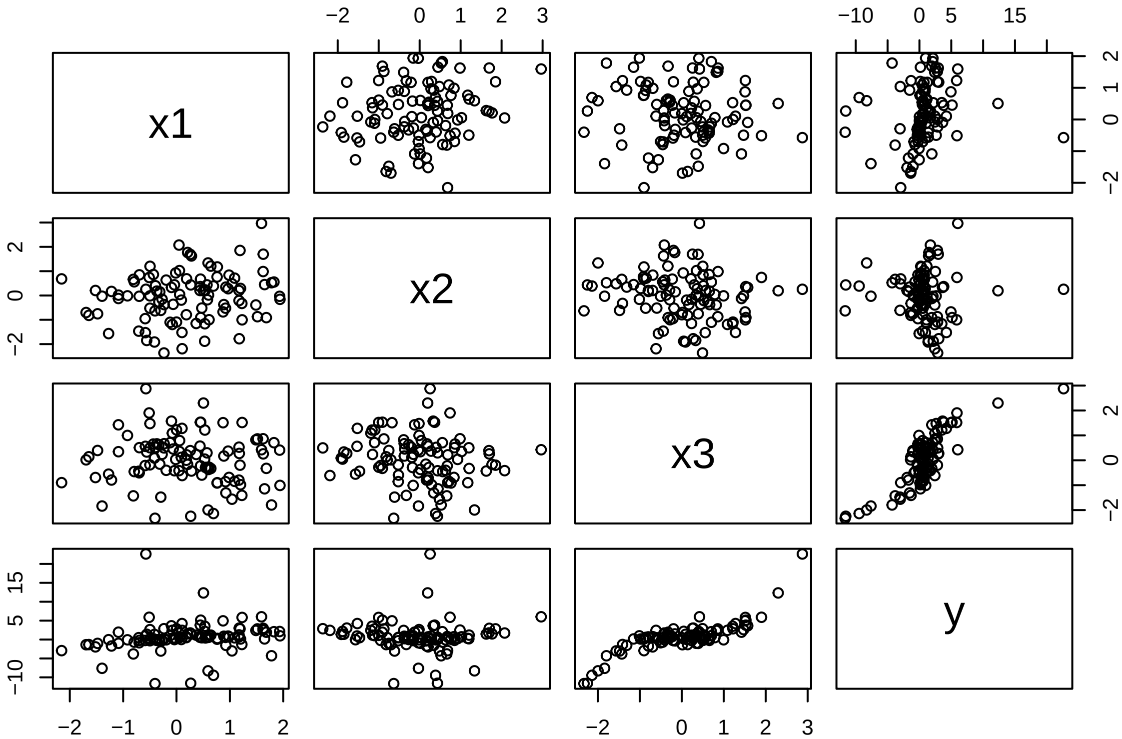
Cross validation

Recall that cross-validation is a technique used to assess the performance of a model. The idea is to split the data into a training set and a test set, and then train the model on the training set and evaluate it on the test set. This process is repeated multiple times, with different splits of the data, and the results are averaged to get an estimate of the model's performance.

Task 1 - splitting the data

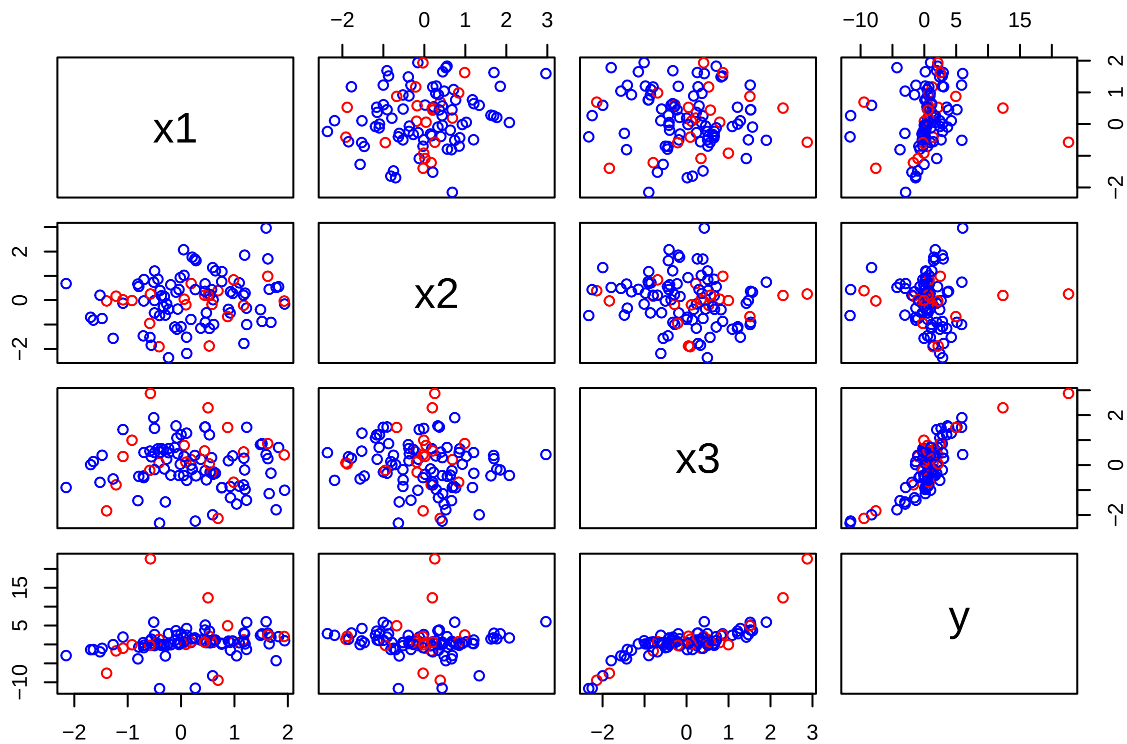
We will start be creating some synthetic data to work with. We will generate a data set with 100 observations that have three features and one continuous response variable.

```r
# Generate synthetic data
data <- data.frame(
  x1 = rnorm(100),
  x2 = rnorm(100),
  x3 = rnorm(100))
data$y = data$x1 + 0.5*data$x2^2 +
  (data$x3-0.1*data$x2)^3 + rnorm(100,0,0.1)

# Pairs plot
pairs(data)
```



Now that we have our data, we can split it into a training set and a test set. We will use 80% of the data for training and 20% for testing.

```
# Split the data
train_indices <- sample(1:nrow(data), 0.8*nrow(data))
data_train <- data[train_indices,]
data_test <- data[-train_indices,]

# Pairs plot colouring by training/test
pairs(data,
      col = ifelse(1:nrow(data) %in% train_indices,
                   "blue", "red"))
```



Task 2 - fitting a polynomial

Fit a polynomial model of degree two to the training data and evaluate its performance on the test data (specifically, MSE, adjusted R-squared and actual vs predictions plot).

```
# Fit a polynomial model
model <- lm(y ~ poly(x1, 2) + poly(x2, 2) + poly(x3, 2),
            data = data_train)

# Predict on the test data
predictions <- predict(model,
                       newdata = data_test)

# Calculate MSE
mse <- mean((data_test$y - predictions)^2)
mse
```

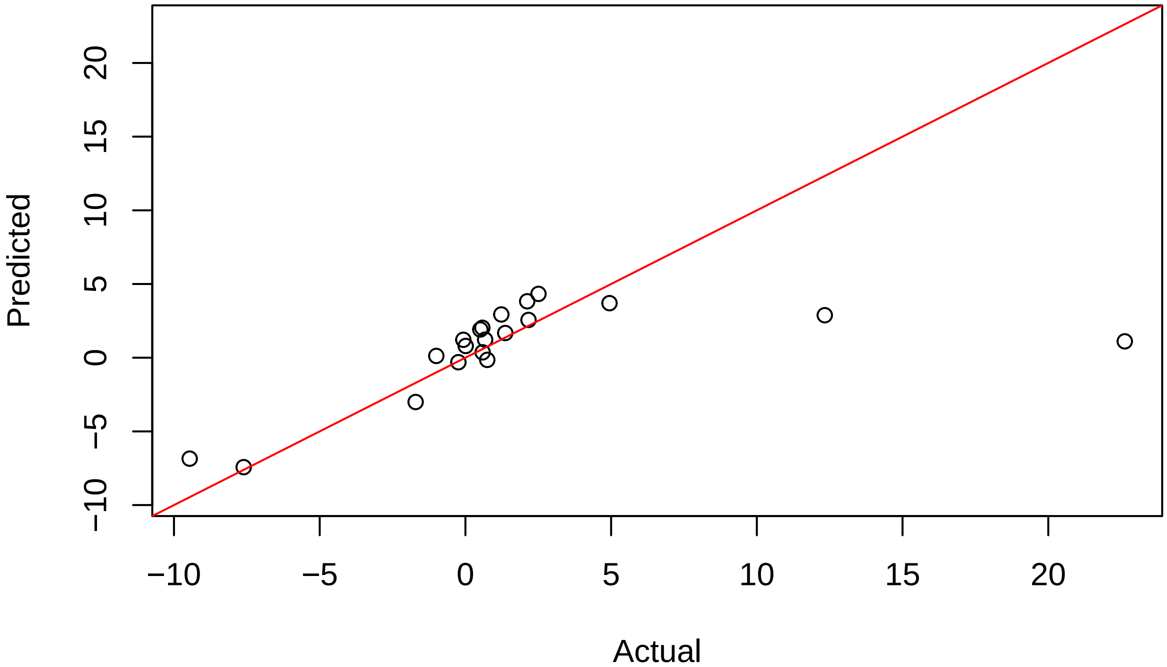
```
## [1] 29.01871
```

```
# Calculate adjusted R-squared
# (Hint it has been calculated in the model summary)
adj_r2 <- summary(model)$adj.r.squared
adj_r2
```

```
## [1] 0.8159031
```

```
# Plot actual vs predictions
plot(data_test$y, predictions,
     xlab = "Actual",
     ylab = "Predicted",
     xlim = c(min(data_test$y, predictions),
              max(data_test$y, predictions)),
```

```r
     ylim = c(min(data_test$y, predictions),
              max(data_test$y, predictions)))
abline(a = 0, b = 1, col = "red")
```



Task 3 - fitting using cross validation

Now, we will use cross-validation to fit a polynomial model of degree two to the data. We will use 5-fold cross-validation.

```r
# Load the caret package
library(caret)
```

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

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

```r
# Set up the cross-validation
control <- trainControl(method = "cv",
                        number = 5)

# Fit the model using cross-validation
model_cv <- train(y ~ poly(x1, 2) + poly(x2, 2) + poly(x3, 2),
                  data = data,
                  method = "lm",
                  trControl = control)
```

Now, let's try to understand what is contained in the `model_cv` object.

```r
# List the elements
names(model_cv)
```

```
##  [1] "method"       "modelInfo"    "modelType"    "results"      "pred"
##  [6] "bestTune"     "call"         "dots"         "metric"       "control"
## [11] "finalModel"   "preProcess"   "trainingData" "ptype"        "resample"
## [16] "resampledCM"  "perfNames"    "maximize"     "yLimits"      "times"
```

```
## [21] "levels"       "terms"        "coefnames"    "xlevels"
```

Typing `?train` in the console will give you more information about the `train` function and these outputs.

We are interested in the performance of the model. Let's extract the results from the cross-validation.

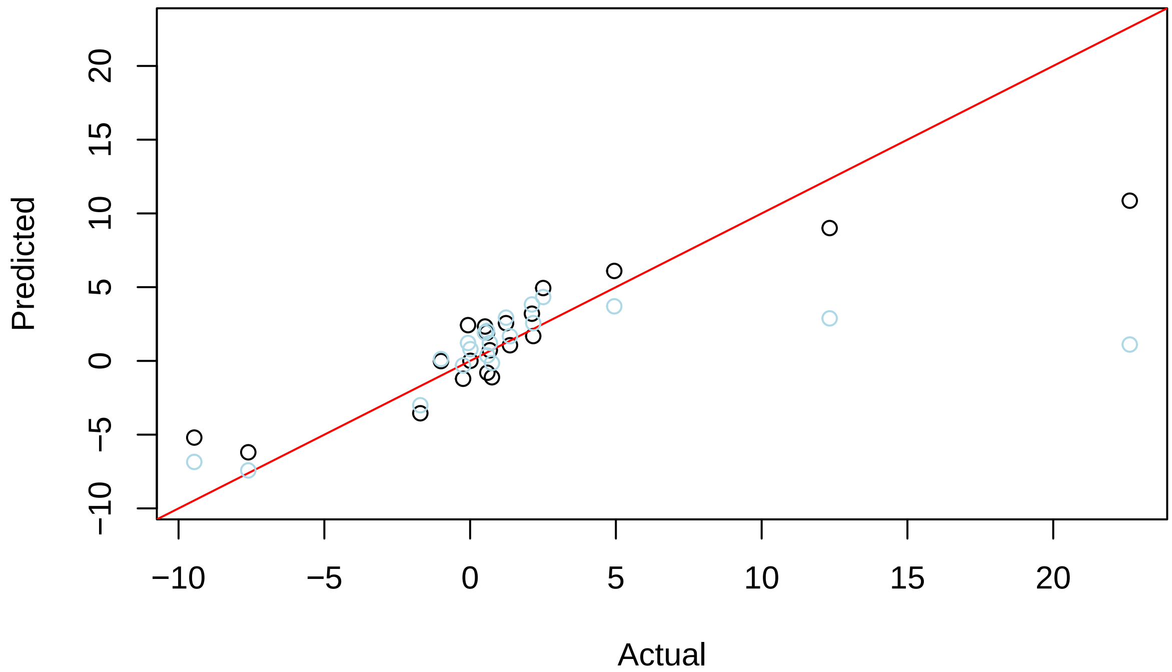
```r
# Extract the results
model_cv$results
```

```
##   intercept     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      TRUE 2.658633 0.5853097 1.832213 1.566867  0.2014804 0.4577722
```

```r
# Hence, calculate the MSE and adjusted R-squared
mse_cv <- model_cv$results$RMSE^2
mse_cv
```

```
## [1] 7.068329
```

```r
#adj_r2_cv <- model_cv$results$Rsquared
#adj_r2_cv

# Make predictions on the "test" data
predictions_cv <- predict(model_cv,
                          newdata = data_test)

# Plot actual vs predictions
plot(data_test$y, predictions_cv,
     xlab = "Actual",
     ylab = "Predicted",
     xlim = c(min(data_test$y, predictions_cv),
              max(data_test$y, predictions_cv)),
     ylim = c(min(data_test$y, predictions_cv),
              max(data_test$y, predictions_cv)))
abline(a = 0, b = 1, col = "red")

# Overlay points from initial model
points(data_test$y, predictions, col = "lightblue")
```



Task 4 - repeat with leave-one-out cross-validation

Now, we will repeat the cross-validation process, but this time we will use leave-one-out cross-validation. This is a special case of cross-validation where each observation is used as the test set once, and the rest of the data is used as the training set.

```
# Set up the cross-validation
control <- trainControl(method = "LOOCV")

# Fit the model using leave-one-out cross-validation
model_loocv <- train(y ~ poly(x1, 2) + poly(x2, 2) + poly(x3, 2),
                     data = data,
                     method = "lm",
                     trControl = control)

# Extract the results
model_loocv$results
```

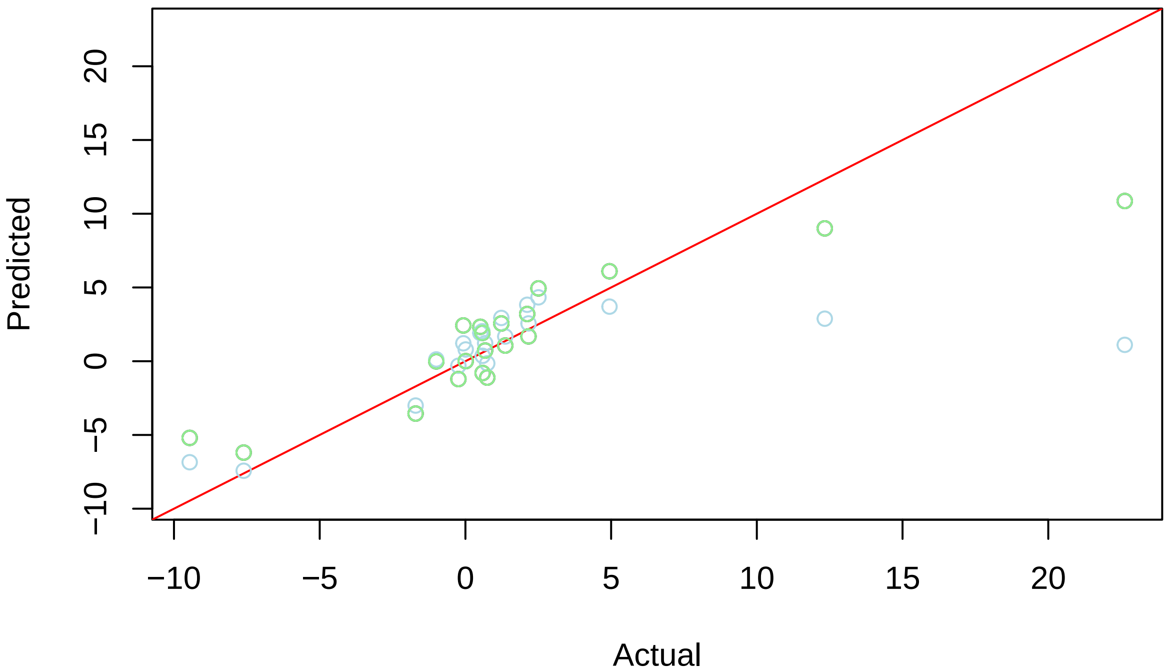
```
##   intercept     RMSE  Rsquared      MAE
## 1      TRUE 2.654419 0.5438271 1.735235
```

```
# Calculate the MSE and adjusted R-squared
mse_loocv <- model_loocv$results$RMSE^2
mse_loocv
```

```
## [1] 7.045942
```

```
#adj_r2_loocv <- model_loocv$results$Rsquared
#adj_r2_loocv

# Make predictions on the "test" data
predictions_loocv <- predict(model_loocv,
                             newdata = data_test)
```

```r
# Plot actual vs predictions
plot(data_test$y, predictions_loocv,
     xlab = "Actual",
     ylab = "Predicted",
     xlim = c(min(data_test$y, predictions_loocv),
              max(data_test$y, predictions_loocv)),
     ylim = c(min(data_test$y, predictions_loocv),
              max(data_test$y, predictions_loocv)))
abline(a = 0, b = 1, col = "red")

# Overlay points from initial model
points(data_test$y, predictions, col = "lightblue")

# Overlay points from 5-fold cross-validation
points(data_test$y, predictions_cv, col = "lightgreen")
```



Classification

Task 5 - k-nearest neighbours

We will utilise part of a weather classification data set to demonstrate the k-nearest neighbours algorithm. We have 13,200 observations in the data set with 11 features. We will attempt to predict the season based on five continuous features.

```r
# Load the data
weather_full <- read.csv("https://www.maths.dur.ac.uk/users/john.p.gosling/MATH3431_practicals/weather_c

# Display the first few rows
head(weather_full)
```

```
##   Temperature Humidity Wind.Speed Precipitation....    Cloud.Cover
## 1          14       73        9.5                82 partly cloudy
## 2          39       96        8.5                71 partly cloudy
```

```
## 3          30       64       7.0                16         clear
## 4          38       83       1.5                82         clear
## 5          27       74      17.0                66      overcast
## 6          32       55       3.5                26      overcast
##   Atmospheric.Pressure UV.Index Season Visibility..km. Location Weather.Type
## 1              1010.82        2 Winter             3.5   inland        Rainy
## 2              1011.43        7 Spring            10.0   inland       Cloudy
## 3              1018.72        5 Spring             5.5 mountain        Sunny
## 4              1026.25        7 Spring             1.0  coastal        Sunny
## 5               990.67        1 Winter             2.5 mountain        Rainy
## 6              1010.03        2 Summer             5.0   inland       Cloudy
```

```r
# Select the features of interest
weather <- weather_full[,c(8,1,3,4,6)]

# Convert the season to a factor
weather$Season <- as.factor(weather$Season)

# Summarise the data
summary(weather)
```

```
##     Season      Temperature       Wind.Speed     Precipitation....
##  Autumn:2500   Min.   :-25.00   Min.   : 0.000   Min.   :  0.00
##  Spring:2598   1st Qu.:  4.00   1st Qu.: 5.000   1st Qu.: 19.00
##  Summer:2492   Median : 21.00   Median : 9.000   Median : 58.00
##  Winter:5610   Mean   : 19.13   Mean   : 9.832   Mean   : 53.64
##                3rd Qu.: 31.00   3rd Qu.:13.500   3rd Qu.: 82.00
##                Max.   :109.00   Max.   :48.500   Max.   :109.00
##  Atmospheric.Pressure
##  Min.   : 800.1
##  1st Qu.: 994.8
##  Median :1007.6
##  Mean   :1005.8
##  3rd Qu.:1016.8
##  Max.   :1199.2
```

Now, we will split the data into a training and validation set (80/20 split).

```r
set.seed(123)

# Split the data
train_indices <- sample(1:nrow(weather), 0.8*nrow(weather))
weather_train <- weather[train_indices,]
weather_test <- weather[-train_indices,]
```

Utilise the `caret` package to fit a k-nearest neighbours model to the training data. We will use 10-fold cross-validation to select the optimal value of k.

```r
# Fit a k-nearest neighbours model
model_knn <- train(Season ~ .,
                   data = weather_train,
                   method = "knn",
                   trControl = trainControl(method = "cv",
                                            number = 10))


# Extract the results
```

```r
model_knn$results
```

```
##   k  Accuracy     Kappa  AccuracySD    KappaSD
## 1 5 0.4190361 0.1764632 0.013756183 0.01864285
## 2 7 0.4135425 0.1673106 0.012058863 0.01566726
## 3 9 0.4150555 0.1692315 0.009676837 0.01307511
```

```r
# Make predictions on the test data
predictions_knn <- predict(model_knn,
                           newdata = weather_test)

# Calculate the confusion matrix
confusionMatrix(predictions_knn,
                weather_test$Season)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Autumn Spring Summer Winter
##     Autumn    121    121    123    123
##     Spring    117    148    136    115
##     Summer    109    143    101     98
##     Winter    150    155    128    752
##
## Overall Statistics
##
##                Accuracy : 0.425
##                  95% CI : (0.406, 0.4441)
##     No Information Rate : 0.4121
##     P-Value [Acc > NIR] : 0.09279
##
##                   Kappa : 0.1863
##
##  Mcnemar's Test P-Value : 0.03359
##
## Statistics by Class:
##
##                      Class: Autumn Class: Spring Class: Summer Class: Winter
## Sensitivity                0.24346       0.26102       0.20697        0.6912
## Specificity                0.82874       0.82248       0.83736        0.7210
## Pos Pred Value             0.24795       0.28682       0.22395        0.6346
## Neg Pred Value             0.82528       0.80273       0.82321        0.7691
## Prevalence                 0.18826       0.21477       0.18485        0.4121
## Detection Rate             0.04583       0.05606       0.03826        0.2848
## Detection Prevalence       0.18485       0.19545       0.17083        0.4489
## Balanced Accuracy          0.53610       0.54175       0.52216        0.7061
```

What are your thoughts on the model performance? How many nearest neighbours is it utilising?

Task 6 - naive Bayes

We will now use the naive Bayes algorithm to classify the weather data set. We will use the same features as before.

```r
# Fit a naive Bayes model
model_nb <- train(Season ~ .,
```

```r
                  data = weather_train,
                  method = "naive_bayes",
                  trControl = trainControl(method = "cv",
                                           number = 10))

# Extract the results
model_nb$results
```

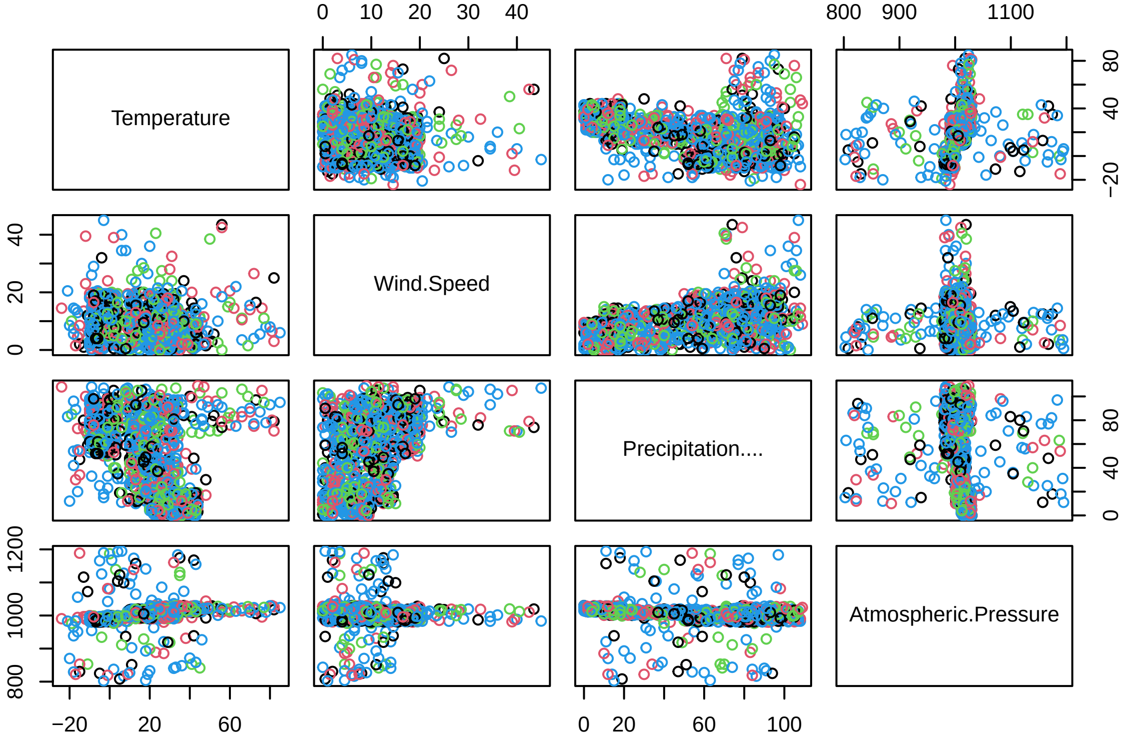
```
##   usekernel laplace adjust  Accuracy     Kappa  AccuracySD    KappaSD
## 1     FALSE       0      1 0.4261357 0.1233466 0.009247048 0.01448711
## 2      TRUE       0      1 0.4170444 0.1782037 0.014726616 0.02180993
```

```r
# Make predictions on the test data
predictions_nb <- predict(model_nb,
                          newdata = weather_test)

# Calculate the confusion matrix
confusionMatrix(predictions_nb,
                weather_test$Season)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Autumn Spring Summer Winter
##     Autumn      1      0      0      0
##     Spring     32     49     43     41
##     Summer    193    194    180    170
##     Winter    271    324    265    877
##
## Overall Statistics
##
##                Accuracy : 0.4193
##                  95% CI : (0.4004, 0.4384)
##     No Information Rate : 0.4121
##     P-Value [Acc > NIR] : 0.2321
##
##                   Kappa : 0.1251
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Autumn Class: Spring Class: Summer Class: Winter
## Sensitivity              0.0020121       0.08642       0.36885        0.8061
## Specificity              1.0000000       0.94404       0.74117        0.4459
## Pos Pred Value           1.0000000       0.29697       0.24423        0.5049
## Neg Pred Value           0.8120500       0.79071       0.83815        0.7663
## Prevalence               0.1882576       0.21477       0.18485        0.4121
## Detection Rate           0.0003788       0.01856       0.06818        0.3322
## Detection Prevalence     0.0003788       0.06250       0.27917        0.6580
## Balanced Accuracy        0.5010060       0.51523       0.55501        0.6260
```

Better or worse than the k-nearest neighbours model? Try looking at a pairs plot of the data to see if you can understand why.

```r
# Pairs plot coloured by season
pairs(weather[sample(1:nrow(weather),1000),2:5],
      col = weather_train$Season)
```




```r
# Also consider the names of the full set of variables
names(weather_full)
```

```
##  [1] "Temperature"        "Humidity"           "Wind.Speed"
##  [4] "Precipitation...."  "Cloud.Cover"        "Atmospheric.Pressure"
##  [7] "UV.Index"           "Season"             "Visibility..km."
## [10] "Location"           "Weather.Type"
```

Task 7 - decision trees

We now move on to a classification task with categorical variables. For this, we will be using a data set that considers whether a mushroom is edible or poisonous based on various features.

```r
# Load the data
mushroom <- read.csv("https://www.maths.dur.ac.uk/users/john.p.gosling/MATH3431_practicals/mushrooms.cs

# Display the first few rows
head(mushroom)
```

```
##   class cap.shape cap.surface cap.color bruises odor gill.attachment
## 1     p         x           s         n       t    p               f
## 2     e         x           s         y       t    a               f
## 3     e         b           s         w       t    l               f
## 4     p         x           y         w       t    p               f
## 5     e         x           s         g       f    n               f
## 6     e         x           y         y       t    a               f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         n          k           e          e
```

```
## 2           c         b          k          e          c
## 3           c         b          n          e          c
## 4           c         n          n          e          e
## 5           w         b          k          t          e
## 6           c         b          n          e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                      w
## 2                        s                        s                      w
## 3                        s                        s                      w
## 4                        s                        s                      w
## 5                        s                        s                      w
## 6                        s                        s                      w
##   stalk.color.below.ring veil.type veil.color ring.number ring.type
## 1                      w         p          w           o         p
## 2                      w         p          w           o         p
## 3                      w         p          w           o         p
## 4                      w         p          w           o         p
## 5                      w         p          w           o         e
## 6                      w         p          w           o         p
##   spore.print.color population habitat
## 1                 k          s       u
## 2                 n          n       g
## 3                 n          n       m
## 4                 k          s       u
## 5                 n          a       g
## 6                 k          n       g
```

```r
# Convert all variables to factors and put back into the data frame
mushroom <- lapply(mushroom, as.factor)
mushroom <- as.data.frame(mushroom)

# Remove all but the first four predictors
# to aid interpretation of the tree
mushroom <- mushroom[,1:5]

# Summarise the data
summary(mushroom)
```

```
##  class     cap.shape cap.surface   cap.color    bruises
##  e:4208    b: 452    f:2320      n      :2284   f:4748
##  p:3916    c:   4    g:   4      g      :1840   t:3376
##            f:3152    s:2556      e      :1500
##            k: 828    y:3244      y      :1072
##            s:  32                w      :1040
##            x:3656                b      : 168
##                                  (Other): 220
```

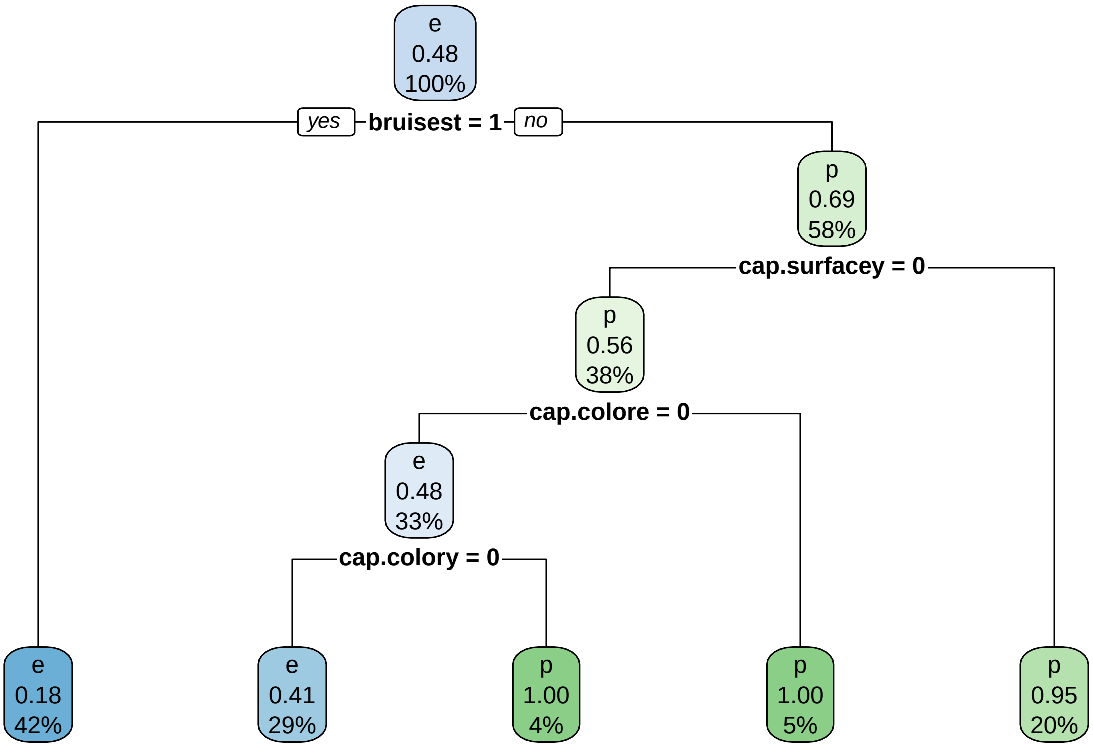
The first column of the data set contains the class label (edible or poisonous), and the remaining columns contain the features. We will use a decision tree to classify the mushrooms.

```r
# Load the `rpart` package
library(rpart)

# Fit a decision tree using 5-fold cross-validation
model_tree <- train(class ~ .,
                    data = mushroom,
```

```r
                        method = "rpart",
                        trControl = trainControl(method = "cv",
                                                 number = 5))
```

It can be useful to visualise the fitted tree.

```r
# Load the `rpart.plot` package
library(rpart.plot)

# Plot the tree
rpart.plot(model_tree$finalModel)
```



What do the numbers relate to on the tree? Could you use the tree to classify a new mushroom?