Machine Learning in R - Classificatin

Chinazom Okoli

2024-04-13

```
# Load required libraries
library(datasets)  # For data sets like Iris
library(dplyr)     # For data manipulation
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(class)     # For k-Nearest Neighbors algorithm
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

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

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

```
# Load Iris dataset from sklearn.datasets
iris_data <- datasets::iris
# Display the first few rows of the dataset
head(iris_data)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
# Split the dataset into features (x) and target variable (y)
x <- iris_data[, -5]  # Features: all columns except the last one
y <- iris_data$Species  # Target variable: Species column
print(x)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
## 17           5.4         3.9          1.3         0.4
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 20           5.1         3.8          1.5         0.3
## 21           5.4         3.4          1.7         0.2
## 22           5.1         3.7          1.5         0.4
## 23           4.6         3.6          1.0         0.2
## 24           5.1         3.3          1.7         0.5
## 25           4.8         3.4          1.9         0.2
## 26           5.0         3.0          1.6         0.2
## 27           5.0         3.4          1.6         0.4
## 28           5.2         3.5          1.5         0.2
## 29           5.2         3.4          1.4         0.2
## 30           4.7         3.2          1.6         0.2
## 31           4.8         3.1          1.6         0.2
## 32           5.4         3.4          1.5         0.4
## 33           5.2         4.1          1.5         0.1
## 34           5.5         4.2          1.4         0.2
## 35           4.9         3.1          1.5         0.2
## 36           5.0         3.2          1.2         0.2
## 37           5.5         3.5          1.3         0.2
## 38           4.9         3.6          1.4         0.1
## 39           4.4         3.0          1.3         0.2
## 40           5.1         3.4          1.5         0.2
## 41           5.0         3.5          1.3         0.3
## 42           4.5         2.3          1.3         0.3
## 43           4.4         3.2          1.3         0.2
## 44           5.0         3.5          1.6         0.6
## 45           5.1         3.8          1.9         0.4
## 46           4.8         3.0          1.4         0.3
## 47           5.1         3.8          1.6         0.2
```

```
## 48          4.6         3.2         1.4         0.2
## 49          5.3         3.7         1.5         0.2
## 50          5.0         3.3         1.4         0.2
## 51          7.0         3.2         4.7         1.4
## 52          6.4         3.2         4.5         1.5
## 53          6.9         3.1         4.9         1.5
## 54          5.5         2.3         4.0         1.3
## 55          6.5         2.8         4.6         1.5
## 56          5.7         2.8         4.5         1.3
## 57          6.3         3.3         4.7         1.6
## 58          4.9         2.4         3.3         1.0
## 59          6.6         2.9         4.6         1.3
## 60          5.2         2.7         3.9         1.4
## 61          5.0         2.0         3.5         1.0
## 62          5.9         3.0         4.2         1.5
## 63          6.0         2.2         4.0         1.0
## 64          6.1         2.9         4.7         1.4
## 65          5.6         2.9         3.6         1.3
## 66          6.7         3.1         4.4         1.4
## 67          5.6         3.0         4.5         1.5
## 68          5.8         2.7         4.1         1.0
## 69          6.2         2.2         4.5         1.5
## 70          5.6         2.5         3.9         1.1
## 71          5.9         3.2         4.8         1.8
## 72          6.1         2.8         4.0         1.3
## 73          6.3         2.5         4.9         1.5
## 74          6.1         2.8         4.7         1.2
## 75          6.4         2.9         4.3         1.3
## 76          6.6         3.0         4.4         1.4
## 77          6.8         2.8         4.8         1.4
## 78          6.7         3.0         5.0         1.7
## 79          6.0         2.9         4.5         1.5
## 80          5.7         2.6         3.5         1.0
## 81          5.5         2.4         3.8         1.1
## 82          5.5         2.4         3.7         1.0
## 83          5.8         2.7         3.9         1.2
## 84          6.0         2.7         5.1         1.6
## 85          5.4         3.0         4.5         1.5
## 86          6.0         3.4         4.5         1.6
## 87          6.7         3.1         4.7         1.5
## 88          6.3         2.3         4.4         1.3
## 89          5.6         3.0         4.1         1.3
## 90          5.5         2.5         4.0         1.3
## 91          5.5         2.6         4.4         1.2
## 92          6.1         3.0         4.6         1.4
## 93          5.8         2.6         4.0         1.2
## 94          5.0         2.3         3.3         1.0
## 95          5.6         2.7         4.2         1.3
## 96          5.7         3.0         4.2         1.2
## 97          5.7         2.9         4.2         1.3
## 98          6.2         2.9         4.3         1.3
## 99          5.1         2.5         3.0         1.1
## 100         5.7         2.8         4.1         1.3
## 101         6.3         3.3         6.0         2.5
```

```
## 102          5.8         2.7          5.1         1.9
## 103          7.1         3.0          5.9         2.1
## 104          6.3         2.9          5.6         1.8
## 105          6.5         3.0          5.8         2.2
## 106          7.6         3.0          6.6         2.1
## 107          4.9         2.5          4.5         1.7
## 108          7.3         2.9          6.3         1.8
## 109          6.7         2.5          5.8         1.8
## 110          7.2         3.6          6.1         2.5
## 111          6.5         3.2          5.1         2.0
## 112          6.4         2.7          5.3         1.9
## 113          6.8         3.0          5.5         2.1
## 114          5.7         2.5          5.0         2.0
## 115          5.8         2.8          5.1         2.4
## 116          6.4         3.2          5.3         2.3
## 117          6.5         3.0          5.5         1.8
## 118          7.7         3.8          6.7         2.2
## 119          7.7         2.6          6.9         2.3
## 120          6.0         2.2          5.0         1.5
## 121          6.9         3.2          5.7         2.3
## 122          5.6         2.8          4.9         2.0
## 123          7.7         2.8          6.7         2.0
## 124          6.3         2.7          4.9         1.8
## 125          6.7         3.3          5.7         2.1
## 126          7.2         3.2          6.0         1.8
## 127          6.2         2.8          4.8         1.8
## 128          6.1         3.0          4.9         1.8
## 129          6.4         2.8          5.6         2.1
## 130          7.2         3.0          5.8         1.6
## 131          7.4         2.8          6.1         1.9
## 132          7.9         3.8          6.4         2.0
## 133          6.4         2.8          5.6         2.2
## 134          6.3         2.8          5.1         1.5
## 135          6.1         2.6          5.6         1.4
## 136          7.7         3.0          6.1         2.3
## 137          6.3         3.4          5.6         2.4
## 138          6.4         3.1          5.5         1.8
## 139          6.0         3.0          4.8         1.8
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
```

```
print(y)
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
```

```
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```r
# Encode the target variable from strings to numerical equivalents
# This is bc machine learning deals with numbers
y <- as.integer(factor(y))
print(y)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```r
# Split the dataset into training and testing sets
set.seed(123)  # For reproducibility
train_index <- sample(1:nrow(iris_data), 0.7 * nrow(iris_data))  # 70% for training
x_train <- x[train_index, ]
y_train <- y[train_index]
x_test <- x[-train_index, ]
y_test <- y[-train_index]

print(train_index)
```

```
##   [1]  14  50 118  43 150 148  90  91 143  92 137  99  72  26   7  78  81 147
##  [19] 103 117  76  32 106 109 136   9  41  74  23  27  60  53 126 119 121  96
##  [37]  38  89  34  93  69 138 130  63  13  82  97 142  25 114  21  79 124  47
##  [55] 144 120  16   6 127  86 132  39  31 134 149 112   4 128 110 102  52  22
##  [73] 129  87  35  40  30  12  88 123  64 146  67 122  37   8  51  10 115  42
##  [91]  44  85 107 139  73  20  46  17  54 108  75  80  71  15  24
```

```r
print(x_train)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 14           4.3         3.0          1.1         0.1
## 50           5.0         3.3          1.4         0.2
## 118          7.7         3.8          6.7         2.2
## 43           4.4         3.2          1.3         0.2
## 150          5.9         3.0          5.1         1.8
## 148          6.5         3.0          5.2         2.0
## 90           5.5         2.5          4.0         1.3
## 91           5.5         2.6          4.4         1.2
## 143          5.8         2.7          5.1         1.9
## 92           6.1         3.0          4.6         1.4
## 137          6.3         3.4          5.6         2.4
## 99           5.1         2.5          3.0         1.1
## 72           6.1         2.8          4.0         1.3
## 26           5.0         3.0          1.6         0.2
## 7            4.6         3.4          1.4         0.3
## 78           6.7         3.0          5.0         1.7
## 81           5.5         2.4          3.8         1.1
## 147          6.3         2.5          5.0         1.9
## 103          7.1         3.0          5.9         2.1
## 117          6.5         3.0          5.5         1.8
## 76           6.6         3.0          4.4         1.4
## 32           5.4         3.4          1.5         0.4
## 106          7.6         3.0          6.6         2.1
## 109          6.7         2.5          5.8         1.8
## 136          7.7         3.0          6.1         2.3
## 9            4.4         2.9          1.4         0.2
## 41           5.0         3.5          1.3         0.3
## 74           6.1         2.8          4.7         1.2
## 23           4.6         3.6          1.0         0.2
## 27           5.0         3.4          1.6         0.4
## 60           5.2         2.7          3.9         1.4
## 53           6.9         3.1          4.9         1.5
## 126          7.2         3.2          6.0         1.8
## 119          7.7         2.6          6.9         2.3
## 121          6.9         3.2          5.7         2.3
## 96           5.7         3.0          4.2         1.2
## 38           4.9         3.6          1.4         0.1
## 89           5.6         3.0          4.1         1.3
## 34           5.5         4.2          1.4         0.2
## 93           5.8         2.6          4.0         1.2
## 69           6.2         2.2          4.5         1.5
## 138          6.4         3.1          5.5         1.8
## 130          7.2         3.0          5.8         1.6
## 63           6.0         2.2          4.0         1.0
## 13           4.8         3.0          1.4         0.1
## 82           5.5         2.4          3.7         1.0
## 97           5.7         2.9          4.2         1.3
## 142          6.9         3.1          5.1         2.3
## 25           4.8         3.4          1.9         0.2
## 114          5.7         2.5          5.0         2.0
```

```
## 21          5.4         3.4         1.7         0.2
## 79          6.0         2.9         4.5         1.5
## 124         6.3         2.7         4.9         1.8
## 47          5.1         3.8         1.6         0.2
## 144         6.8         3.2         5.9         2.3
## 120         6.0         2.2         5.0         1.5
## 16          5.7         4.4         1.5         0.4
## 6           5.4         3.9         1.7         0.4
## 127         6.2         2.8         4.8         1.8
## 86          6.0         3.4         4.5         1.6
## 132         7.9         3.8         6.4         2.0
## 39          4.4         3.0         1.3         0.2
## 31          4.8         3.1         1.6         0.2
## 134         6.3         2.8         5.1         1.5
## 149         6.2         3.4         5.4         2.3
## 112         6.4         2.7         5.3         1.9
## 4           4.6         3.1         1.5         0.2
## 128         6.1         3.0         4.9         1.8
## 110         7.2         3.6         6.1         2.5
## 102         5.8         2.7         5.1         1.9
## 52          6.4         3.2         4.5         1.5
## 22          5.1         3.7         1.5         0.4
## 129         6.4         2.8         5.6         2.1
## 87          6.7         3.1         4.7         1.5
## 35          4.9         3.1         1.5         0.2
## 40          5.1         3.4         1.5         0.2
## 30          4.7         3.2         1.6         0.2
## 12          4.8         3.4         1.6         0.2
## 88          6.3         2.3         4.4         1.3
## 123         7.7         2.8         6.7         2.0
## 64          6.1         2.9         4.7         1.4
## 146         6.7         3.0         5.2         2.3
## 67          5.6         3.0         4.5         1.5
## 122         5.6         2.8         4.9         2.0
## 37          5.5         3.5         1.3         0.2
## 8           5.0         3.4         1.5         0.2
## 51          7.0         3.2         4.7         1.4
## 10          4.9         3.1         1.5         0.1
## 115         5.8         2.8         5.1         2.4
## 42          4.5         2.3         1.3         0.3
## 44          5.0         3.5         1.6         0.6
## 85          5.4         3.0         4.5         1.5
## 107         4.9         2.5         4.5         1.7
## 139         6.0         3.0         4.8         1.8
## 73          6.3         2.5         4.9         1.5
## 20          5.1         3.8         1.5         0.3
## 46          4.8         3.0         1.4         0.3
## 17          5.4         3.9         1.3         0.4
## 54          5.5         2.3         4.0         1.3
## 108         7.3         2.9         6.3         1.8
## 75          6.4         2.9         4.3         1.3
## 80          5.7         2.6         3.5         1.0
## 71          5.9         3.2         4.8         1.8
## 15          5.8         4.0         1.2         0.2
```

```
## 24           5.1         3.3          1.7         0.5
```

```
print(y_train)
```

```
##   [1] 1 1 3 1 3 3 2 2 3 2 3 2 2 1 1 2 2 3 3 3 2 1 3 3 3 1 1 2 1 1 2 2 3 3 3 2 1
##  [38] 2 1 2 2 3 3 2 1 2 2 3 1 3 1 2 3 1 3 3 1 1 3 2 3 1 1 3 3 3 1 3 3 3 2 1 3 2
##  [75] 1 1 1 1 2 3 2 3 2 3 1 1 2 1 3 1 1 2 3 3 2 1 1 1 2 3 2 2 2 1 1
```

```
print(x_test)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 5            5.0         3.6          1.4         0.2
## 11           5.4         3.7          1.5         0.2
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 28           5.2         3.5          1.5         0.2
## 29           5.2         3.4          1.4         0.2
## 33           5.2         4.1          1.5         0.1
## 36           5.0         3.2          1.2         0.2
## 45           5.1         3.8          1.9         0.4
## 48           4.6         3.2          1.4         0.2
## 49           5.3         3.7          1.5         0.2
## 55           6.5         2.8          4.6         1.5
## 56           5.7         2.8          4.5         1.3
## 57           6.3         3.3          4.7         1.6
## 58           4.9         2.4          3.3         1.0
## 59           6.6         2.9          4.6         1.3
## 61           5.0         2.0          3.5         1.0
## 62           5.9         3.0          4.2         1.5
## 65           5.6         2.9          3.6         1.3
## 66           6.7         3.1          4.4         1.4
## 68           5.8         2.7          4.1         1.0
## 70           5.6         2.5          3.9         1.1
## 77           6.8         2.8          4.8         1.4
## 83           5.8         2.7          3.9         1.2
## 84           6.0         2.7          5.1         1.6
## 94           5.0         2.3          3.3         1.0
## 95           5.6         2.7          4.2         1.3
## 98           6.2         2.9          4.3         1.3
## 100          5.7         2.8          4.1         1.3
## 101          6.3         3.3          6.0         2.5
## 104          6.3         2.9          5.6         1.8
## 105          6.5         3.0          5.8         2.2
## 111          6.5         3.2          5.1         2.0
## 113          6.8         3.0          5.5         2.1
## 116          6.4         3.2          5.3         2.3
## 125          6.7         3.3          5.7         2.1
## 131          7.4         2.8          6.1         1.9
## 133          6.4         2.8          5.6         2.2
## 135          6.1         2.6          5.6         1.4
```

```
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 145          6.7         3.3          5.7         2.5
```

```r
print(y_test)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
## [39] 3 3 3 3 3 3 3
```

```r
# Define the number of folds for cross-validation
num_folds <- 10

# Define the cross-validation control
ctrl <- trainControl(method = "cv", number = num_folds)

# Define the model
model <- train(x = x_train, y = y_train, method = "kknn", trControl = ctrl)

# Print the cross-validated results
print(model)
```

```
## k-Nearest Neighbors
##
## 105 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 94, 95, 94, 95, 95, 95, ...
## Resampling results across tuning parameters:
##
##   kmax  RMSE       Rsquared   MAE
##   5     0.1554323  0.9548060  0.06463103
##   7     0.1687012  0.9513526  0.07454549
##   9     0.1763667  0.9485440  0.08127454
##
## Tuning parameter 'distance' was held constant at a value of 2
## Tuning
##  parameter 'kernel' was held constant at a value of optimal
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were kmax = 5, distance = 2 and kernel
##  = optimal.
```

```r
model_svm <- train(x = x_train, y = y_train, method = "svmLinear", trControl = ctrl)

# Print the cross-validated results
print(model_svm)
```

```
## Support Vector Machines with Linear Kernel
##
## 105 samples
##   4 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 94, 94, 94, 95, 96, 95, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.2292213  0.9197212  0.1817882
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
# Define the SVM with RBF kernel model
model_svmR <- train(x = x_train, y = y_train, method = "svmRadial", trControl = ctrl)

# Print the cross-validated results
print(model_svmR)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 105 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 96, 94, 94, 94, 94, 95, ...
## Resampling results across tuning parameters:
##
##   C     RMSE       Rsquared   MAE
##   0.25  0.2803570  0.8948591  0.2039372
##   0.50  0.2421622  0.9127897  0.1714530
##   1.00  0.2302240  0.9147304  0.1571252
##
## Tuning parameter 'sigma' was held constant at a value of 0.9841231
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.9841231 and C = 1.
```

```
# Train the k-Nearest Neighbors (KNN) classifier - Algorithm
knn_model <- knn(train = x_train, test = x_test, cl = y_train) # Extract the hidden pattern between  x_
print(knn_model)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 3 3 3 3 3
## [39] 3 3 3 3 3 3 3
## Levels: 1 2 3
```

```
# Predict the classes for test data
predictions <- knn_model
print(predictions)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 3 3 3 3 3
## [39] 3 3 3 3 3 3 3
## Levels: 1 2 3
```

```r
# what was predicted
 y_test
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
## [39] 3 3 3 3 3 3 3
```

```r
 print(y_test)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
## [39] 3 3 3 3 3 3 3
```

```r
# Calculate the accuracy of the model
accuracy <- mean(predictions == y_test)
print(accuracy)
```

```
## [1] 0.9777778
```

Evaluate the model

```r
library(caret) # Required package
predictionsf <- as.factor(predictions) # Converting predictions and y_test to factor
y_testf <- as.factor(y_test)

cm <- confusionMatrix(predictionsf, y_testf) #  summarizes the performance of a classification algorith
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 14  0  0
##          2  0 17  0
##          3  0  1 13
##
## Overall Statistics
##
##                Accuracy : 0.9778
##                  95% CI : (0.8823, 0.9994)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9664
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   0.9444   1.0000
```

```
## Specificity            1.0000   1.0000   0.9688
## Pos Pred Value         1.0000   1.0000   0.9286
## Neg Pred Value         1.0000   0.9643   1.0000
## Prevalence             0.3111   0.4000   0.2889
## Detection Rate         0.3111   0.3778   0.2889
## Detection Prevalence   0.3111   0.3778   0.3111
## Balanced Accuracy      1.0000   0.9722   0.9844
```

Explanation of the output

The output provided by the confusion matrix along with the various statistics computed from it. Let's break down each part:

Confusion Matrix: A confusion matrix is a table that summarizes the performance of a classification algorithm. It presents the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class.

- **Reference**: The actual labels or true classes.
- **Prediction**: The predicted labels or classes made by the model.

In our confusion matrix:

	1	2	3
1	14	0	0
2	0	17	0
3	0	1	13

- The diagonal elements represent the number of correctly classified instances (TP).
- Off-diagonal elements represent misclassifications.

Overall Statistics:

- **Accuracy**: The proportion of correctly classified instances out of total instances.
- **95% CI**: 95% confidence interval for the accuracy.
- **No Information Rate (NIR)**: The accuracy achieved by always predicting the majority class.
- **P-Value [Acc > NIR]**: The statistical significance of the difference between the model's accuracy and the no information rate.
- **Kappa**: Cohen's Kappa coefficient, a measure of inter-rater agreement corrected for chance.

Statistics by Class:

- **Sensitivity (True Positive Rate)**: The proportion of actual positive cases correctly identified by the model.
- **Specificity (True Negative Rate)**: The proportion of actual negative cases correctly identified by the model.
- **Positive Predictive Value (Precision)**: The proportion of instances predicted as positive that are truly positive.
- **Negative Predictive Value**: The proportion of instances predicted as negative that are truly negative.
- **Prevalence**: The proportion of instances in each class in the dataset.

- **Detection Rate**: The proportion of instances correctly identified as positive (sensitivity) or negative (specificity).
- **Detection Prevalence**: The proportion of instances predicted as positive (or negative) by the model.
- **Balanced Accuracy**: The average of sensitivity and specificity, which gives a balanced view of the model's performance across classes.

These statistics provide insights into how well the model performs for each class and overall. High values for sensitivity, specificity, and accuracy indicate good performance.

Evaluating classification performance

1. **True Positives (TP)**: True positives are the instances that are correctly classified as positive by the model. In other words, they are the instances where the model correctly predicts the positive class when the true class is indeed positive.
2. **False Positives (FP)**: False positives are the instances that are incorrectly classified as positive by the model. They represent the instances where the model incorrectly predicts the positive class when the true class is actually negative.
3. **True Negatives (TN)**: True negatives are the instances that are correctly classified as negative by the model. They are the instances where the model correctly predicts the negative class when the true class is indeed negative.
4. **False Negatives (FN)**: False negatives are the instances that are incorrectly classified as negative by the model. They represent the instances where the model incorrectly predicts the negative class when the true class is actually positive.

These four metrics are fundamental components of a confusion matrix, which is a table used in binary classification to evaluate the performance of a classification model. They help in understanding the model's ability to correctly classify instances and provide insights into its strengths and weaknesses.

```
#confusion matrix
conf_matrix <- matrix(c(14, 0, 0, 0, 17, 0, 0, 1, 13), nrow = 3, byrow = TRUE)
colnames(conf_matrix) <- c("1", "2", "3")
rownames(conf_matrix) <- c("1", "2", "3")

# Calculate TP, FP, FN, TN
TP <- diag(conf_matrix)
FP <- colSums(conf_matrix) - TP
FN <- rowSums(conf_matrix) - TP
Total <- sum(conf_matrix)


# Print the counts
print("True Positives (TP):")
```

```
## [1] "True Positives (TP):"
```

```
print(TP)
```

```
##  1  2  3
## 14 17 13
```

```r
print("False Positives (FP):")
```

```
## [1] "False Positives (FP):"
```

```r
print(FP)
```

```
## 1 2 3
## 0 1 0
```

```r
print("False Negatives (FN):")
```

```
## [1] "False Negatives (FN):"
```

```r
print(FN)
```

```
## 1 2 3
## 0 0 1
```

To evaluate the performance of a classification model by providing insights into its ability to correctly classify instances, we compute these three evaluation metrics:

1. **Recall (Sensitivity)**:
   - Recall, also known as sensitivity, measures the ability of a classifier to find all positive instances. It is calculated as the ratio of true positives (TP) to the sum of true positives and false negatives (FN).
   - Explanation: Recall is crucial in scenarios where missing positive instances is costly or undesirable. For example, in medical diagnosis, high recall ensures that all patients with a disease are correctly identified, minimizing false negatives (missed diagnoses).
2. **Precision**:
   - Precision measures the ability of a classifier to correctly identify only the relevant instances among all the instances it has classified as positive. It is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP).
   - Explanation: Precision is important when the cost of false positives is high. For instance, in an email spam detection system, high precision ensures that legitimate emails are not incorrectly classified as spam, minimizing false positives.
3. **F1-score**:
   - The F1-score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.
   - Explanation: The F1-score considers both false positives and false negatives, making it a suitable metric when there is an imbalance between positive and negative classes or when both precision and recall are equally important. It rewards classifiers that have both high precision and high recall.

These metrics play crucial roles in evaluating the performance of classification models and help in understanding how well the model is performing in different aspects. They provide insights into the model's ability to correctly classify instances and balance between minimizing false positives and false negatives, depending on the specific requirements of the problem domain.

```r
# Given values
TP <- c(14, 17, 13)  # True Positives for each class
FP <- c(0, 1, 0)     # False Positives for each class
FN <- c(0, 0, 1)     # False Negatives for each class
TN <- c(31, 27, 31)  # True Negatives for each class

# Calculate precision, recall, and F1-score for each class
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print the results for each class
for (i in 1:length(TP)) {
  cat("Class", i, "\n")
  cat("Precision:", precision[i], "\n")
  cat("Recall:", recall[i], "\n")
  cat("F1-score:", f1_score[i], "\n\n")
}
```

```
## Class 1
## Precision: 1
## Recall: 1
## F1-score: 1
##
## Class 2
## Precision: 0.9444444
## Recall: 1
## F1-score: 0.9714286
##
## Class 3
## Precision: 1
## Recall: 0.9285714
## F1-score: 0.962963
```