Multilabel Classification for Flower Species

Om Prabhu

31 July 2023

Introduction

In this project, I have used the *iris* data set to train different classification models with the aim of predicting iris species based on a set of variables.

Load packages & data set

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(kernlab)) install.packages("kernlab", repos = "http://cran.us.r-project.org")

library(caret)
library(kernlab)
```

We now load the data set. The *iris* data set comes built-in with R, hence, there is no need to load it externally.

```
# attach the iris dataset to the environment
data(iris)

# rename the dataset
dataset <- iris
```

Creating a train-test split

We will split the data set into two parts, 80% of which will be used to train our models and 20% that we will reserve for testing & validation. In order to ensure consistent results, we use a fixed seed for splitting the data set.

```
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(2)
validation_index <- createDataPartition(dataset$Species, p=0.80, list=FALSE)

# select 20% of the data for validation
validation <- dataset[-validation_index,]

# use the remaining 80% of data to training and testing the models
dataset <- dataset[validation_index,]
```

Analysis of the Data Set

We can now have a look at the data set that will be used for training the models. We can look at the data in a few different ways, such as: 1. Dimensions of the data set 2. Types of the attributes 3. Take a look at the

data itself 4. Levels of the class attribute 5. Breakdown of the instances in each class 6. Summary of all attributes

A Quick Look at the Data Set

Let us first take a quick look at the data set.

```
# first 6 rows of the data
head(dataset)
```

```
##   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
## 6          5.4         3.9          1.7         0.4  setosa
## 8          5.0         3.4          1.5         0.2  setosa
```

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the `dim` function.

```
# dimensions of dataset
dim(dataset)
```

```
## [1] 120   5
```

As observed, there are 120 instances and 5 attributes (sepal length, sepal width, petal length, petal width and species). We can also get an idea of the types of attributes as follows:

```
# types for each attribute
sapply(dataset, class)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

As observed, all the inputs are doubles and the output class value is a factor. Let's look at the class levels (the class refers to the 'Species' column in our data set).

```
# levels for the class
levels(dataset$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

As we can see, the class has 3 different variables, hence, this is a multi-label classification problem. We can further analyze the data to find out the proportion of instances that belong to each class.

```
# summarize the class distribution
percentage <- prop.table(table(dataset$Species)) * 100
cbind(freq=table(dataset$Species), percentage=percentage)
```

```
##            freq percentage
## setosa       40   33.33333
## versicolor   40   33.33333
## virginica    40   33.33333
```

We can see that the data set is perfectly balanced across the three class labels, since each class has the same proportion of instances.

Overall Summary of the Data Set

Finally, we can also take a look at a statistical summary of each attribute, which includes the minimum, maximum, mean values along with some important percentile values as well. This gives us a clear idea of the distributions of each attribute.

```
# summarize the attribute
summary(dataset)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.700   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.822   Mean   :3.055   Mean   :3.748   Mean   :1.202
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.700   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :40
##  versicolor:40
##  virginica :40
##
##
##
```

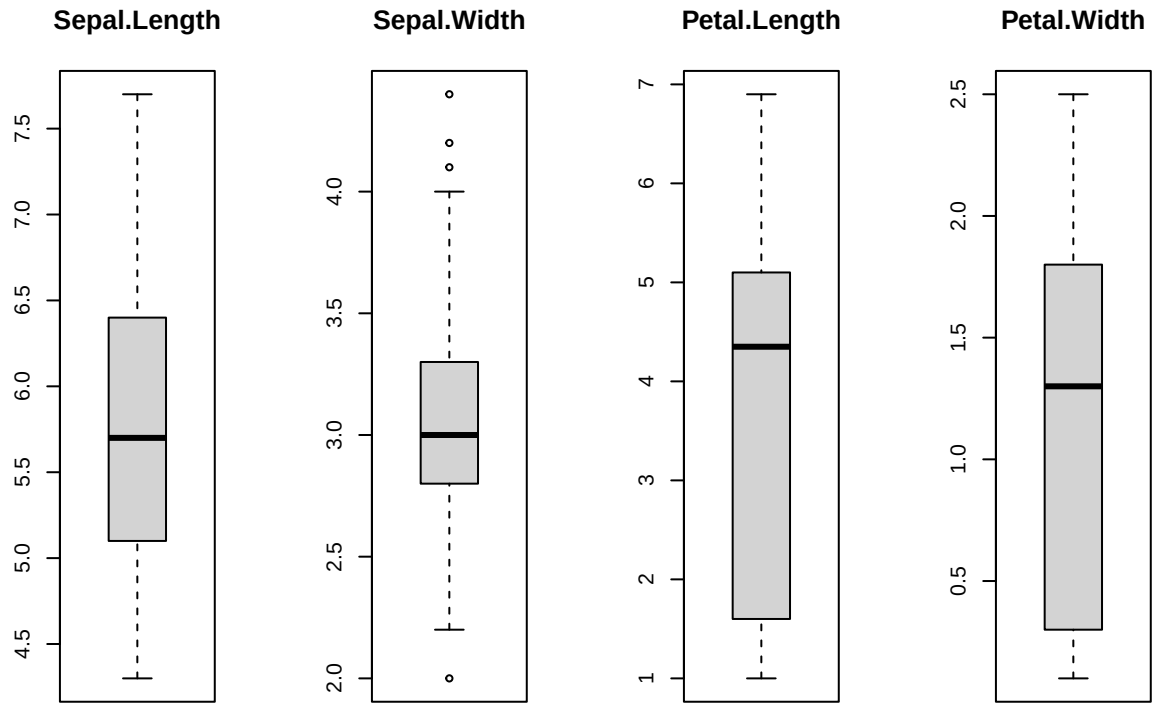
Visualization of the Data Set

Now that we have explored the data and have a basic idea of the types and distributions of each attribute. We can plot the distributions for each attribute as boxplots to better visualize the range of the attributes.

Single Variable Plots

```
# split input and output
x <- dataset[,1:4]
y <- dataset[,5]

# boxplot for each attribute
par(mfrow=c(1,4))
  for(i in 1:4) {
  boxplot(x[,i], main=names(iris)[i])
}
```

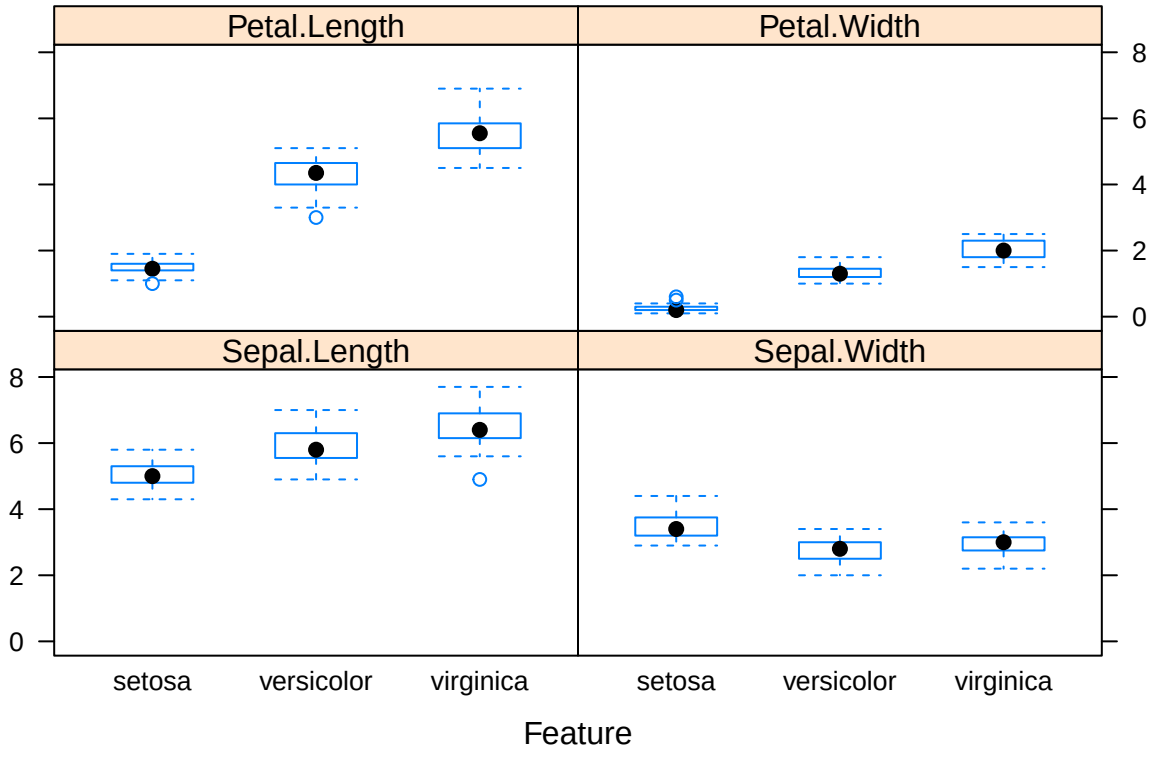


This gives a much better idea of the distributions of each attribute and helps us compare the mean values and range of each attribute. As observed, the two attributes denoting petal sizes have a much bigger range compared to the ones denoting sepal sizes.

Multivariate Plots

Now, let us look at a more detailed comparison between the individual variables. We can again look at boxplots for each attribute, but this time broken down by their class label.

```
# box plots for each attribute separated by class labels
featurePlot(x=x, y=y, plot="box")
```



Using the above multivariate plot, it is easier to appreciate the differences in the distributions of the attributes for each class value.

Algorithms

Now that we have a detailed insight into the data as well as the comparison between individual attributes across all the classes, we can now train some models on the data and use them to estimate the accuracy on unseen data.

Creating a Test Harness

We will first set up a test harness to use 10-fold cross validation to estimate accuracy. This will split the data set into 10 parts, 9 for training and 1 for testing for all the combinations of train-test splits. In order to get more accurate results, I have repeated the process 3 times with different train-test splits.

```
# Set up 10-fold cross validation
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

As observed, we use the accuracy metric for evaluating the models. This is the percentage of results that are predicted correctly by the model.

Building Classification Models

Let us now evaluate 5 different classification models on our test harness: 1. LDA (Linear Discriminant Analysis) 2. CT (Classification Trees) 3. kNN (k-Nearest Neighbours) 4. SVM (Support Vector Machines) 5. RF (Random Forest)

To ensure that the evaluation of each algorithm is performed using exactly the same data splits, we need to use the same seed for fitting each model.

```
# LDA
set.seed(42)
fit.lda <- train(Species~., data=dataset, method="lda", metric=metric, trControl=control)

# Classification Tree
set.seed(42)
fit.cart <- train(Species~., data=dataset, method="rpart", metric=metric, trControl=control)

# kNN
set.seed(42)
fit.knn <- train(Species~., data=dataset, method="knn", metric=metric, trControl=control)

# SVM
set.seed(42)
fit.svm <- train(Species~., data=dataset, method="svmRadial", metric=metric, trControl=control)

# Random Forest
set.seed(42)
fit.rf <- train(Species~., data=dataset, method="rf", metric=metric, trControl=control)
```

Selecting the Best Model

We now have 5 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate. We do this by reporting the accuracy of each model using the summary function.

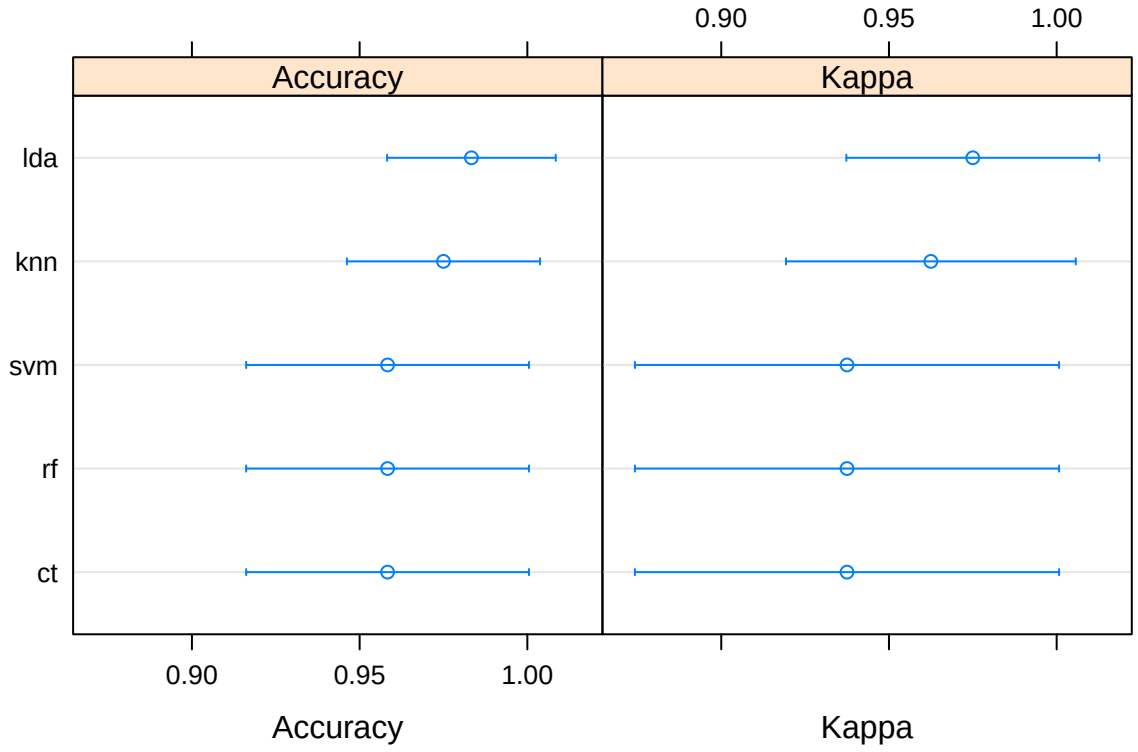
```
# summarize accuracy of models
results <- resamples(list(lda=fit.lda, ct=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.rf))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, ct, knn, svm, rf
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu. Median      Mean 3rd Qu. Max. NA's
## lda 0.9166667 1.0000000      1 0.9833333       1    1    0
## ct  0.8333333 0.9166667      1 0.9583333       1    1    0
## knn 0.9166667 0.9375000      1 0.9750000       1    1    0
## svm 0.8333333 0.9166667      1 0.9583333       1    1    0
## rf  0.8333333 0.9166667      1 0.9583333       1    1    0
##
## Kappa
##      Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## lda 0.875 1.00000      1 0.9750       1    1    0
## ct  0.750 0.87500      1 0.9375       1    1    0
## knn 0.875 0.90625      1 0.9625       1    1    0
## svm 0.750 0.87500      1 0.9375       1    1    0
## rf  0.750 0.87500      1 0.9375       1    1    0
```

We can also better visualize these results by plotting the results and comparing the mean and spread of the

accuracy.

```
# compare accuracy of models
dotplot(results)
```



As observed, the LDA model displays the best performance in terms of accuracy. The results for just the LDA model can be summarized as shown below:

```
# summarize best model
print(fit.lda)
```

```
## Linear Discriminant Analysis
##
## 120 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9833333  0.975
```

Making Predictions

Since LDA was the most accurate model, we can now get an idea of the accuracy of this model on the validation set that we created at the very beginning of the project. This gives us an independent final check on the accuracy of the model.

We can run the LDA model directly on the validation data set and summarize the results in form of a confusion matrix as follows:

```
# accuracy of LDA on the validation dataset
predictions <- predict(fit.lda, validation)
confusionMatrix(predictions, validation$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         1
##   virginica       0          0         9
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.8278, 0.9992)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 2.963e-13
##
##                   Kappa : 0.95
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9000
## Specificity                 1.0000            0.9500           1.0000
## Pos Pred Value              1.0000            0.9091           1.0000
## Neg Pred Value              1.0000            1.0000           0.9524
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.3000
## Detection Prevalence        0.3333            0.3667           0.3000
## Balanced Accuracy           1.0000            0.9750           0.9500
```

We can see that the accuracy is 96.67%, suggesting that the model is reasonably accurate and reliable.

Conclusion

In summary, we have succeeded in building multiple classification models for predicting iris species from the dimensions of the petals and sepals of the flower. We first carried out an in-depth exploratory data analysis, which familiarized us with the data and helped observe the comparisons between the attributes across various classes. We then set up a test harness using 10-fold cross validation in order to select the best model out of the 5 classification models that we trained. Finally, we selected the best performing model and calculated the accuracy on the validation data set. The accuracy of the best model (LDA) on the validation data set was 96.67%, which suggests that the model is reasonable accurate and reliable.

References

Since the data set comes built-in with R, there are no particular references for the data. However, the idea for this project stemmed from a Kaggle project that I came across and which is linked below. Although the idea of the project was borrowed, the final code implementation is entirely done by me.

1. Link to Kaggle Page