Animal Classification

Angeli Cristiano

01/09/2020

Abstract

The data set contains 101 observations (animals) and 18 predictors (animal characteristics) including "class_type" (1 = Mammal, 2 = Bird, 3 = Reptile, 4 = Fish, 5 = Amphibian, 6 = Bug, 7 = Invertebrate). The goal is to find a model that classifies animals as correctly as possible.

Introduction

I start by analyzing the data set checking if there are missing values and if the types of predictors are correct.

```r
library(readr)
zoo <- read.csv("./Animal Classification/zoo.csv")
class <- read.csv("./Animal Classification/class.csv")
sum(is.na(zoo))
```

```
## [1] 0
```

```r
str(zoo)
```

```
## 'data.frame':    101 obs. of  18 variables:
##  $ animal_name: Factor w/ 100 levels "aardvark","antelope",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ hair       : int  1 1 0 1 1 1 1 0 0 1 ...
##  $ feathers   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ eggs       : int  0 0 1 0 0 0 0 1 1 0 ...
##  $ milk       : int  1 1 0 1 1 1 1 0 0 1 ...
##  $ airborne   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ aquatic    : int  0 0 1 0 0 0 0 1 1 0 ...
##  $ predator   : int  1 0 1 1 1 0 0 0 1 0 ...
##  $ toothed    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ backbone   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ breathes   : int  1 1 0 1 1 1 1 0 0 1 ...
##  $ venomous   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ fins       : int  0 0 1 0 0 0 0 1 1 0 ...
##  $ legs       : int  4 4 0 4 4 4 4 0 0 4 ...
##  $ tail       : int  0 1 1 0 1 1 1 1 1 0 ...
##  $ domestic   : int  0 0 0 0 0 0 1 1 0 1 ...
##  $ catsize    : int  1 1 0 1 1 1 1 0 0 0 ...
##  $ class_type : int  1 1 4 1 1 1 1 4 4 1 ...
```

I proceed with modifying the characteristics to make them factors and excluding the "animal name" predictor since it is not useful for this classification.

```r
B <- as.character(class$Class_Type)
for (i in 1:max(class$Class_Number)) {
  zoo$class_type[zoo$class_type == i] <- B[i]
}

zoo[,2:18] <- data.frame(lapply(zoo[,2:18], as.factor))
df.zoo <- data.frame(zoo[, 2:18])
#for (j in colnames(df.zoo)) {
 # df.zoo[,j] <- lapply(df.zoo[,j], as.factor)
#type.convert(df.zoo, as.is = FALSE)
#}
attach(df.zoo)
```

KNN

In this section I will find the number of nearest neighbors k such that it classifies the training set well and gives the best accuracy on the test set.

Since the original data set is small, I use the K-fold repeated cross validation method choosing $K = 10$ and repeat this process three times.

```r
library(caret)
```

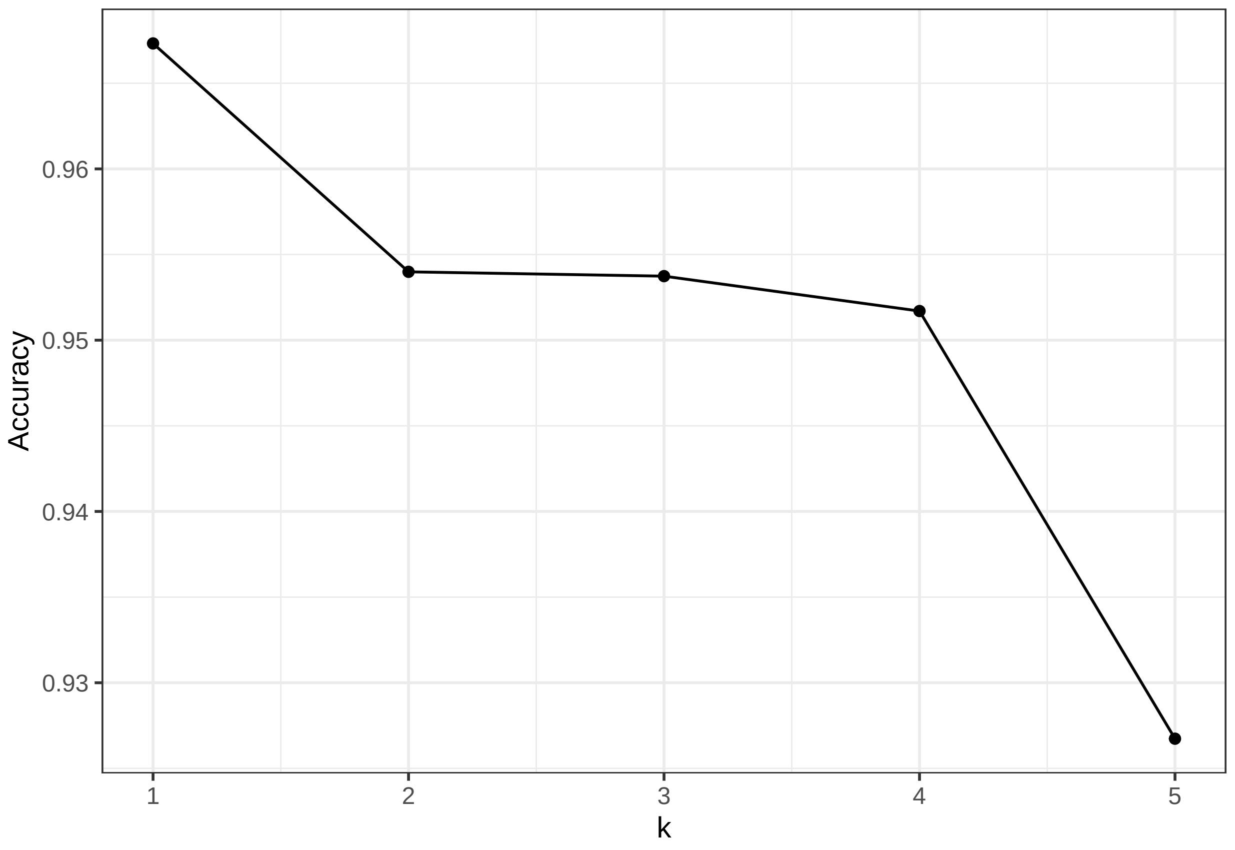
```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
trCont.knn <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

set.seed(123)
knn.fit <- train(class_type ~., method = "knn", tuneGrid = expand.grid(k = 1:5), trControl = trCont.knn
             metric = "Accuracy", data = df.zoo)
knn.fit
```

```
## k-Nearest Neighbors
##
## 101 samples
##  16 predictor
##   7 classes: 'Amphibian', 'Bird', 'Bug', 'Fish', 'Invertebrate', 'Mammal', 'Reptile'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 90, 89, 91, 92, 91, 91, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   1  0.9673232  0.9561952
##   2  0.9539899  0.9388354
##   3  0.9537374  0.9380343
##   4  0.9517003  0.9360357
##   5  0.9267340  0.9033810
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

Graphically, the accuracy trend looks like this:

```r
ggplot(knn.fit) + theme_bw() + labs(x = "k", y = "Accuracy")
```



```r
#plot(varImp(knn.fit, scale = F))
```

which at $k = 1$ it gives an accuracy of 96.7%.

Tree

Decision Trees

In this section I will use the validation set approach, so I split the data set in two subsets, training set and test set, which they contains respectively the 70% and 30% of the original data set observations.

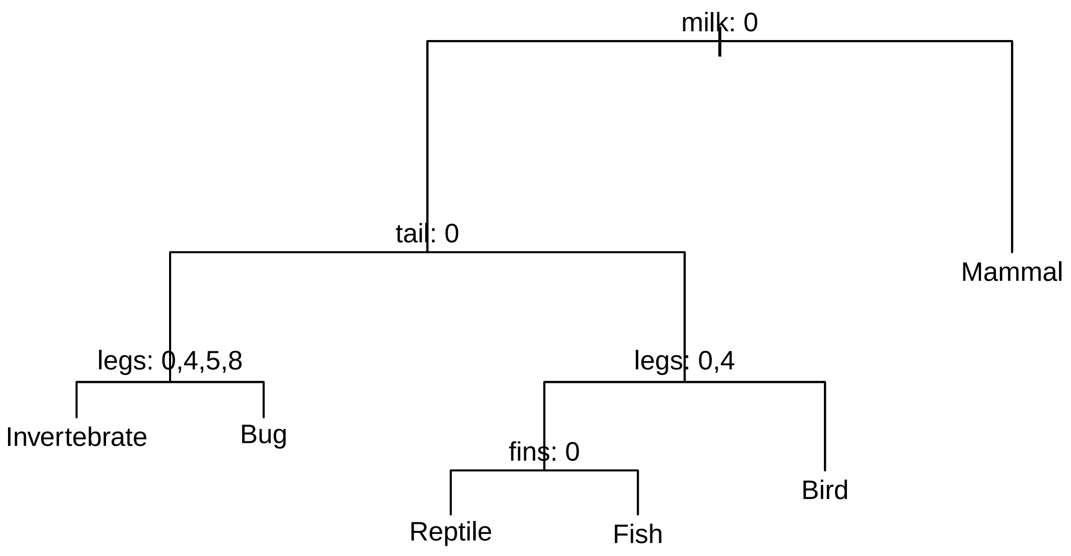
```r
set.seed(123)
id <- sample(2, nrow(df.zoo), prob = c(0.7, 0.3), replace = TRUE)
zoo.train <- df.zoo[id == 1, ]
zoo.test <- df.zoo[id == 2, ]
```

I start to built a tree on the training set:

```r
library(tree)
tree.fit <- tree(class_type ~., data = zoo.train)
```

and the built tree has the following shape:

```
plot(tree.fit)
text(tree.fit, pretty = 0, cex = 0.8)
```



Next, I evaluate it using the test set,

```
treefit.pred <- predict(tree.fit, newdata = zoo.test, type = "class")
treefit.confmx <- confusionMatrix(treefit.pred, zoo.test$class_type)
treefit.confmx$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   0.8620689655   0.7989601386   0.6833593906   0.9611051687   0.4827586207
## AccuracyPValue  McnemarPValue
##   0.0000244331            NaN
```

and it gives an accuracy of 86.2%.

Since I have a small data set perhaps this approach is not suitable so I try to improve this result by using the Random Forest method that I use below.

Random Forest

In this section I will find the number of predictors *mtry*, randomly chosen at each step, to consider to make a classification and, through the OOB observation, I can estimate the accuracy of the model.

I use OOB method instead of CV because the error rate is calculated simultaneously with the model, so it gives a computational vantage.

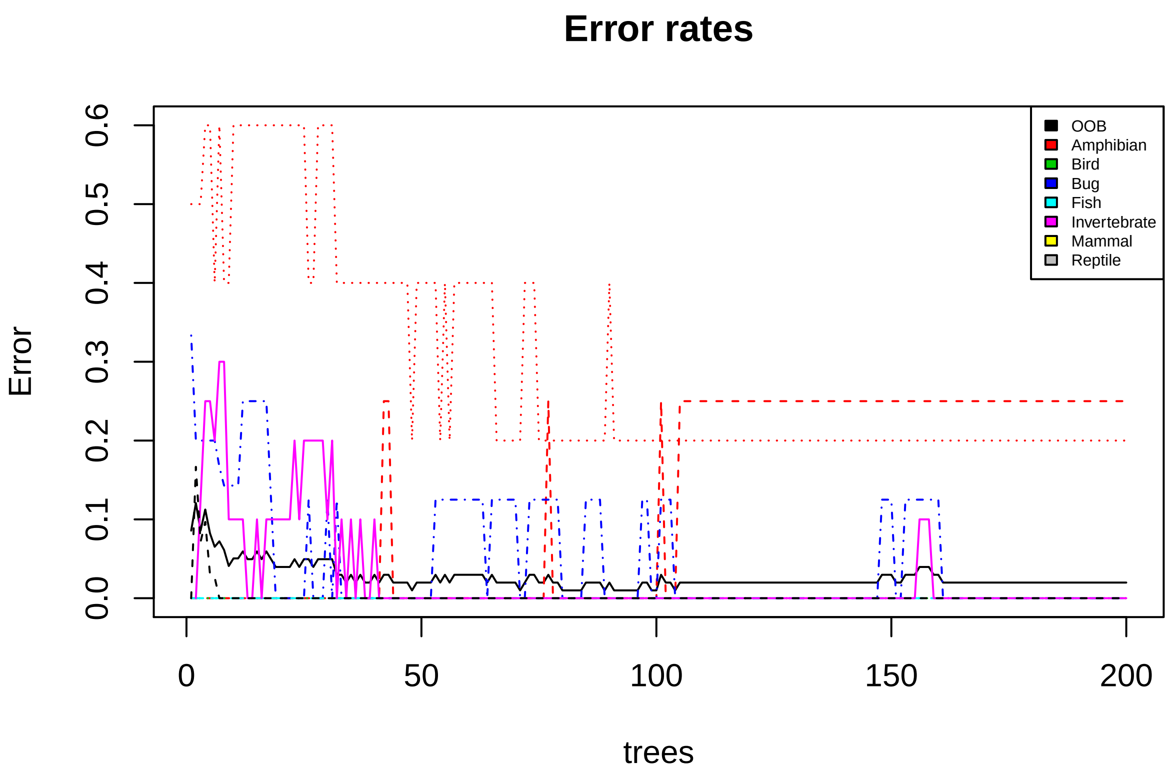
```
trCont.rf <- trainControl(method = "oob")

set.seed(123)
rf.fit <- train(class_type ~ ., data = df.zoo, method = "rf", trControl = trCont.rf, tuneLength = 10,
                    ntree = 200)
```

```
rf.fit
```

```
## Random Forest
##
## 101 samples
##  16 predictor
##   7 classes: 'Amphibian', 'Bird', 'Bug', 'Fish', 'Invertebrate', 'Mammal', 'Reptile'
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.950495  0.9346532
##    4    0.970297  0.9607615
##    6    0.980198  0.9738917
##    8    0.960396  0.9478440
##   10    0.980198  0.9737969
##   12    0.980198  0.9739220
##   14    0.980198  0.9739220
##   16    0.970297  0.9608578
##   18    0.980198  0.9739220
##   20    0.980198  0.9739220
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

It gives an accuracy value of 98% at $mtry = 6$.

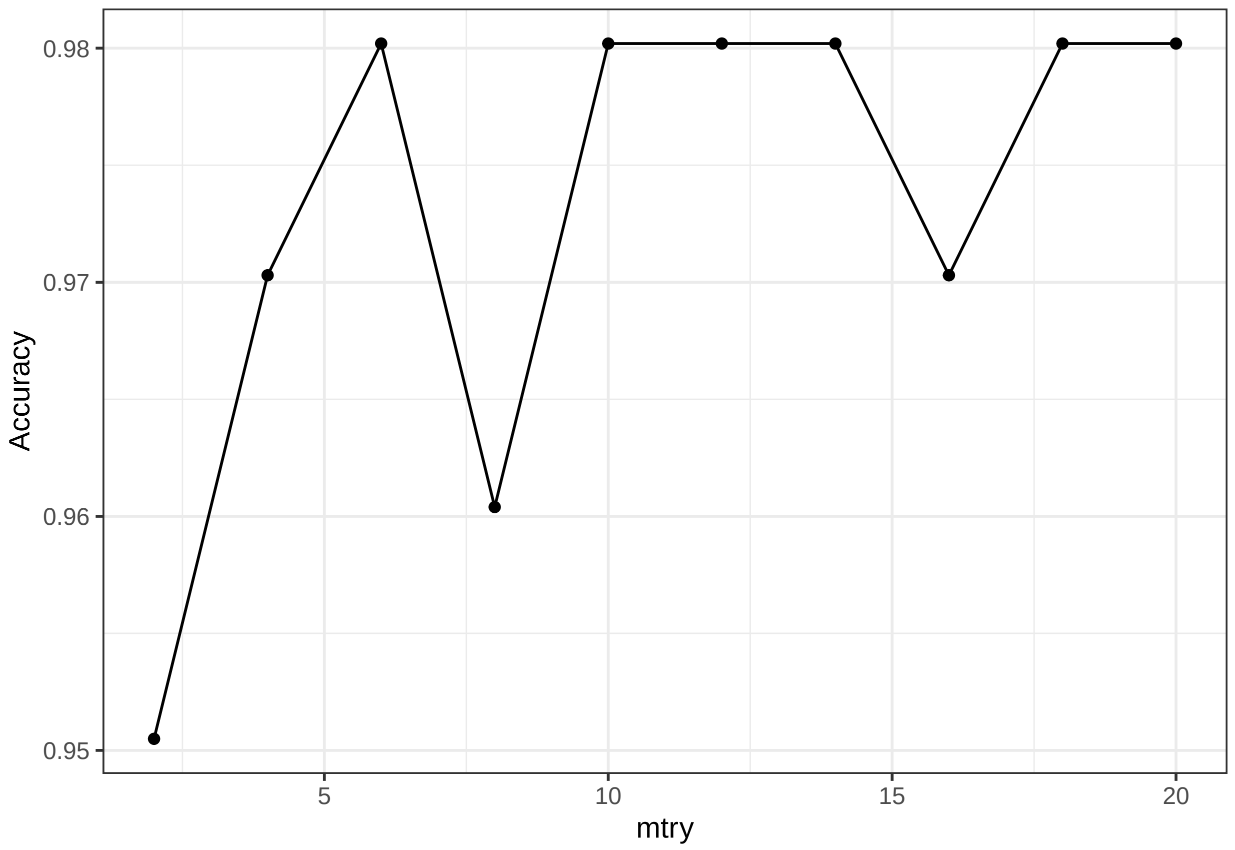
```
plot(rf.fit$finalModel, main = "Error rates")
legend("topright", legend = colnames(rf.fit$finalModel$err.rate),col=1:8,cex=0.5,fill=1:8)
```



In the beginning I considered 400 trees but, since the error rate do not decrease from ~200 onward, I set this last value in order to decrease the computational cost.

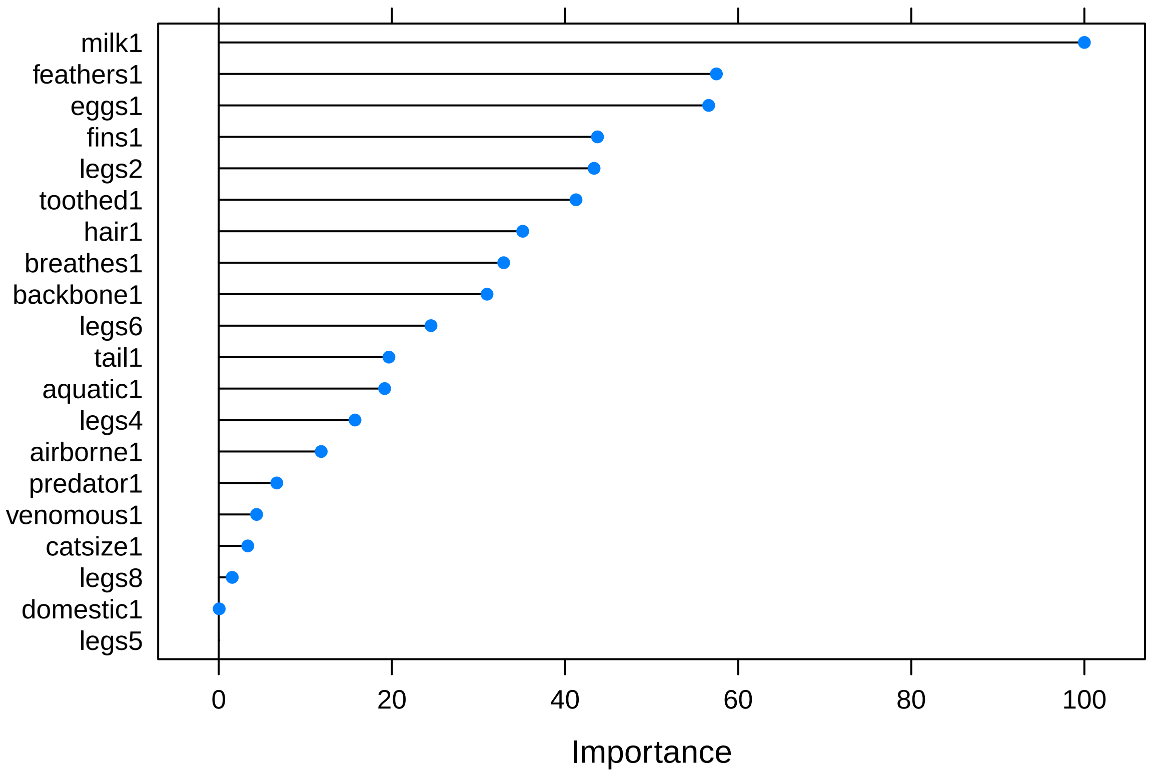
Graphically, the accuracy trend looks like this:

```
ggplot(rf.fit) + theme_bw() + labs(x = "mtry", y = "Accuracy")
```



In the following, I also check the importance of the predictors:

```r
plot(varImp(rf.fit, scale = T))
```



Support Vector Machine

In this section I will find the most appropriate kernel that best separates the various classes.

Here I use the the repeated CV method as in the first case and I chose the linear kernel since the other options do not give a substantial improvement and are computationally more expensive.

```
trCont.svm <- trainControl(method = "repeatedcv", number = 8, repeats = 3)

set.seed(123)
svm.fit <- train(class_type ~., data = df.zoo, method = "svmLinear", trControl = trCont.svm,
                     tuneLength = 5, scale = FALSE)
```

```
svm.fit$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 45
##
## Objective Function Value : -0.4615 -0.4 -0.6667 -0.75 -0.7273 -2.0333 -0.4 -0.4 -0.5151 -0.5116 -0.8
## Training error : 0
```

```
svm.fit$results
```

```
##   C  Accuracy     Kappa AccuracySD    KappaSD
## 1 1 0.9709145 0.9617317 0.05038883 0.06655542
```

It gives an accuracy value of 97.1% with the parameter $C = 1$ so this model permits at most one observation to be in the wrong side of each hyperplane.

Conclusion

S.V.M. and K.N.N. they have an accuracy value very close to that of Random Forest. This may be due to the fact that animals, which belong to the same class, have very similar characteristics. In addiction, changing the set.seed, the accuracy value of each methods change a bit. This probably happens because the data set is small.