assignement2

Lilach Herzog & Leslie Cohen

13 5 2022

Our whole project and code can be found on github at https://github.com/LeslieLebon/assignment-2.git

Our objective in this project was to analyze the "wheat-seeds" dataset (where the 'Type' of the seed is the column used for labeling and classifying) using two classifier models that we built. We decided on using the KNN algorithm and the Decision Tree (DT) algorithm.

preproccessing

Upload data and libraries

```
library(gmodels)
library(C50)
library(class)
library(rpart)
library(rpart.plot)
library(tidymodels) # for the rsample package, along with the rest of tidymodels
library(modeldata)  # for the cells data
library(vip)        # for variable importance plots
library(pROC)

seeds <- read.csv("seeds.csv")# read data
set.seed(1234)#to initialize a pseudo random number generator.
```

initial look at the data

We know the class 'Type' is the column (label) to classify according to. We took a quick look at the data:

```
str(seeds)
```

```
## 'data.frame':    199 obs. of  8 variables:
##  $ Area           : num  15.3 14.9 14.3 13.8 16.1 ...
##  $ Perimeter      : num  14.8 14.6 14.1 13.9 15 ...
##  $ Compactness    : num  0.871 0.881 0.905 0.895 0.903 ...
##  $ Kernel.Length  : num  5.76 5.55 5.29 5.32 5.66 ...
##  $ Kernel.Width   : num  3.31 3.33 3.34 3.38 3.56 ...
##  $ Asymmetry.Coeff: num  2.22 1.02 2.7 2.26 1.35 ...
##  $ Kernel.Groove  : num  5.22 4.96 4.83 4.8 5.17 ...
##  $ Type           : int  1 1 1 1 1 1 1 1 1 1 ...
```

There are 3 Types of seeds (in the column 'Type'): 1, 2 and 3.

Since the whole table is sorted according to the' Type' column, and we want to work randomly, we will shuffle the whole table

In order to further understand the data and be able to work with it we used the "table" function. this function builds a contingency table of the counts at each combination of factor levels- what interests us is the Type:

```
table(seeds$Type)
```

```
##
##  1  2  3
## 66 68 65
```

We can see the Type has 66 times Type 1, 68 times Type 2, and 65 times Type 3, about a third of each Type.

Before starting on teaching the algorithm, we converted the 'Type' column to a factor, as that is what is required by the C50 package.

```
seeds$Type<-as.factor(seeds$Type)
```

Now- let's get to work!!

split into training and test sets

```
length_of_training_set <-round(0.8*(nrow(seeds)),0) #  0.8% of the observations out of the total 199
seeds_train <- seeds[1:length_of_training_set, ]
seeds_test <- seeds[(length_of_training_set+1):nrow(seeds), ]
seeds_train_labels <-seeds_train[, 8]
seeds_test_labels <-seeds_test[, 8]
```

Since we know that there are about a third of each Type in the data set, we want to be sure that the distribution is as we defined (about 1/3 of each Type in of both training and test sets):

```
##
##         1         2         3
## 0.3081761 0.3333333 0.3584906
```

```
##
##     1     2     3
## 0.425 0.375 0.200
```

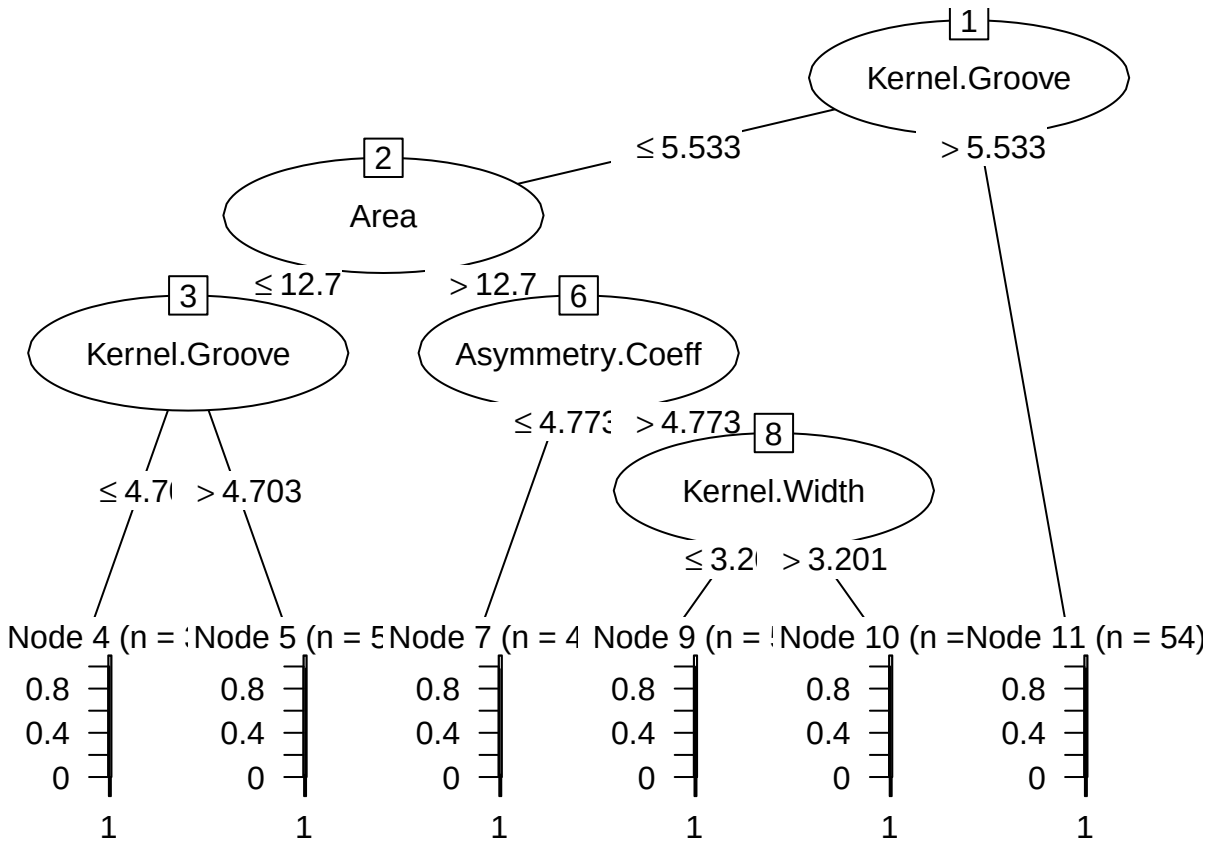
target factor (label) vector for classification

Decision Tree

A Decision Tree uses a serie of questions that lead to a decision- a class. The goal is to improve purity, to try and have a higher and higher percentage of items in the node are similar, until all items in a node belong to the same class. It does so by taking the items in a node and splitting them in such a manner that they become two groups with improved purity in each (more samples in the group belong to the same class).

We will use the C5.0 method. The 8th column of the dataset is the Type class variable, so we need to exclude it from the training data frame, and supply it as the target factor (label) vector for classification:

```
DT_model <- C5.0(seeds_train[-8], seeds_train$Type)
plot(DT_model)
```



Explanation of initial tree:

```
summary(DT_model)
```

The algorithm did six iterations. In the first iteration it divided all the seeds into two subgroups (nodes) according to whether their Groove was higher or lower than 5.533. One node was pure (all seeds in the node were Class 2), and the other node went through another iteration according to the Area, where both nodes were not pure and needed to go through another iteration. One node was split into two pure nodes-again according to Groove-which yielded 2 pure nodes (Class 3 or class 1). The other node was split according to the asymmetry coefficient which resulted in one pure node (of Class 1) whereas the other node went into another iteration according to the width which yielded 2 pure nodes (classes 3 and 1).

The classifier made 3 mistake for an error rate of 1.9%, which is relatively low.

After teaching the model on the training set, we want to use it to predict the class is on the test set, and then compare the test set to what we predicted for the test said using our predictive model.

```
DT_pred <- predict(DT_model, seeds_test)
DT_table_mat<-CrossTable(seeds_test$Type, DT_pred, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  40
```

```
##
##
##               | predicted Type
##  actual Type |         1 |         2 |         3 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
##            1 |        17 |         0 |         0 |        17 |
##              |     0.425 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            2 |         1 |        14 |         0 |        15 |
##              |     0.025 |     0.350 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            3 |         2 |         0 |         6 |         8 |
##              |     0.050 |     0.000 |     0.150 |           |
## -------------|-----------|-----------|-----------|-----------|
## Column Total |        20 |        14 |         6 |        40 |
## -------------|-----------|-----------|-----------|-----------|
##
##
```

Type seeds 2 and 3 both have no false positives (all predictions of Type 2 or three were actually Types 2 or 3) with a few false negatives (one Type 2 and 2 Type 3s that were predicted as Type one). Type 1 on the other hand has no false negative (all Type 1 seeds were predicted as Type 1) but it had a few false positives (as mentioned before- one that was actually Type 2 and two were actually Type 3).

We computed the accuracy test from the confusion matrix (the proportion of true positive and true negative over the sum of the matrix).

```
DT_accuracy_Test <- sum(diag(DT_table_mat[["t"]])) / sum(DT_table_mat[["t"]])
```

Accuracy of 92.5% is pretty good, but we will try to improve our results using adaptive boosting (a process in which many Decision Trees are built and the trees vote on the best class for each example).

Adaptive Boosting

10 trials

We'll start with 10 trials, a number that has become the de facto standard, as research suggests that this reduces error rates on test data by about 25 percent:

```
# boosting with 10 trials (on training)
DT_boost10 <- C5.0(seeds_train[-8], seeds_train$Type, trials = 10)
summary(DT_boost10 )
```

```
DT_boost10[["boostResults"]]
```

```
##    Trial Size Errors Percent         Data
## 1      1    6      3     1.9 Training Set
## 2      2    5     13     8.2 Training Set
## 3      3    5     15     9.4 Training Set
## 4      4    5      6     3.8 Training Set
## 5      5    4     32    20.1 Training Set
## 6      6    7     12     7.5 Training Set
## 7      7    7     20    12.6 Training Set
## 8      8    4      8     5.0 Training Set
## 9      9    6      4     2.5 Training Set
## 10    10    6      4     2.5 Training Set
```

The classifier made 1 mistake for an error rate of 0.6% percent. This is an improvement over the previous

training error rate. However, it remains to be seen whether we see a similar improvement on the test data. Let's take a look at how good the prediction is:

```
DT_boost_pred10 <- predict(DT_boost10, seeds_test)
DT_table_mat_10<-CrossTable(seeds_test$Type, DT_boost_pred10,
prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
dnn = c('actual Type', 'predicted Type'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  40
##
##
##              | predicted Type
##  actual Type |         1 |         2 |         3 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
##            1 |        17 |         0 |         0 |        17 |
##              |     0.425 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            2 |         1 |        14 |         0 |        15 |
##              |     0.025 |     0.350 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            3 |         1 |         0 |         7 |         8 |
##              |     0.025 |     0.000 |     0.175 |           |
## -------------|-----------|-----------|-----------|-----------|
## Column Total |        19 |        14 |         7 |        40 |
## -------------|-----------|-----------|-----------|-----------|
##
##
```

```
DT_accuracy_Test10 <- sum(diag(DT_table_mat_10[["t"]])) / sum(DT_table_mat_10[["t"]])
```

The accuracy improved from 92.5% to 95%: it made 1 false negative (predicted Type 1) in Type 3 instead of 2 (which also lowered the false positive of Type 1 to 2 instead of 3) before the boost.

20 trials

WE will try to improve the result, with more trials than 10, for example 20.

```
DT_boost20 <- C5.0(seeds_train[-8], seeds_train$Type, trials = 20)
summary(DT_boost20 )
```

```
DT_boost20[["boostResults"]]
```

```
##    Trial Size Errors Percent         Data
## 1      1    6      3     1.9 Training Set
## 2      2    5     13     8.2 Training Set
## 3      3    5     15     9.4 Training Set
## 4      4    5      6     3.8 Training Set
## 5      5    4     32    20.1 Training Set
```

```
## 6      6    7     12     7.5 Training Set
## 7      7    7     20    12.6 Training Set
## 8      8    4      8     5.0 Training Set
## 9      9    6      4     2.5 Training Set
## 10    10    7     10     6.3 Training Set
## 11    11    6     20    12.6 Training Set
## 12    12    6      2     1.3 Training Set
## 13    13    7      5     3.1 Training Set
## 14    14    5     35    22.0 Training Set
## 15    15    6     16    10.1 Training Set
## 16    16    7      5     3.1 Training Set
## 17    17    6     14     8.8 Training Set
## 18    18    6      3     1.9 Training Set
## 19    19    6     19    11.9 Training Set
## 20    20    7     13     8.2 Training Set
```

The classifier with 20 trials made 0 mistake for an error rate of 0.0% percent(!).

```
DT_boost_pred20 <- predict(DT_boost20, seeds_test)
DT_table_mat_20<-CrossTable(seeds_test$Type, DT_boost_pred20,
prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
dnn = c('actual Type', 'predicted Type'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  40
##
##
##              | predicted Type
##  actual Type |         1 |         2 |         3 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
##            1 |        17 |         0 |         0 |        17 |
##              |     0.425 |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            2 |         1 |        14 |         0 |        15 |
##              |     0.025 |     0.350 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            3 |         1 |         0 |         7 |         8 |
##              |     0.025 |     0.000 |     0.175 |           |
## -------------|-----------|-----------|-----------|-----------|
## Column Total |        19 |        14 |         7 |        40 |
## -------------|-----------|-----------|-----------|-----------|
##
##
```

```
DT_accuracy_Test20 <- sum(diag(DT_table_mat_20[["t"]])) / sum(DT_table_mat_20[["t"]])
```

```
print(paste('Accuracy for prediction with boosting of 20 is:', DT_accuracy_Test10))
```

```
## [1] "Accuracy for prediction with boosting of 20 is: 0.95"
```

The training was better, but the accuracy stayed exactly the same (95%) Indeed, choosing 20 trials in the boost does not improve the result. The error rate stay similar than with 10 trials, so there is no reason to do 20 trials.

DT conclusion

The Decision Tree algorithm gave us accuracy of 92.5% to 95% .we increased the accuracy by boosting it with 10 or 20 trials. Our dataset is relatively small so we didn't see a change in accuracy between 10 trials and 20 trials, but because the error rate in the 20 trial run managed to get to 0% as opposed to 0.6%, we assume that with a bigger dataset there will be a bigger difference between the amount of runs but in our case there is no difference so 10 trials is enough.

In addition to the classification itself, we were able to look at the process and the tree itself and learn some information about our data.

These is the attribute usages :

feature	first run	10 trials	20 trials
Kernel.Groove	100.00%	100.00%	100.00%
Area	66.04%	66.04%	100.00%
Asymmetry.Coeff	31.45%	100.00%	100.00%
Kernel.Width	4.40%	100.00%	100.00%
Perimeter	0.00%	100.00%	100.00%
Compactness	0.00%	0.00%	27.67%
Kernel.Length	0.00%	0.00%	18.87%

The algorithm uses the feature that provides the most Information Gain in every split. Since it used 100.00% of the information from the Kernel.Groove column in all runs, we can assume that that column provided the most information gain (at least in the first iteration). The algorithm also used the Area column, the Asymmetry.Coeff and the Kernel.Width columns in all 3 runs, so that also gives us some information about how much these features influenced the Information Gain function.

KNN Algorithm

In the KNN Algorithm, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. ## Normalization In order to compare the distances of different features with different scales, we made sure the range of values of each parameter is similar by using a simple normalization function (min/max normalization).

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x)))
  }
normalize(c(1, 2, 3, 4, 5)) # test our function 1
normalize(c(10, 20, 30, 40, 50)) # test our function 2
```

We used the normalization function on our training and test sets, and looked at it to make sure it worked:

```
normalized_seeds_train <- as.data.frame(lapply(seeds_train[1:7], normalize))
normalized_seeds_test <- as.data.frame(lapply(seeds_test[1:7], normalize))
summary(normalized_seeds_train$Area)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1331  0.3228  0.3889  0.6470  1.0000
```

run the KNN algorithm

And finally, we are ready to run our algorithm. We'll start with K of 21, the square root of 455 and also an odd number, reducing the change of a tie vote.

```
seeds_test_pred <- knn(train = normalized_seeds_train, test = normalized_seeds_test, cl = seeds_train_la
```

Since the knn function returns a factor vector of the predicted values, we'll compare that vector with the true labels we saved in advance. We'll do the comparison with the CrossTable function, from the gmodels package loaded:

```
knn_table_mat<-CrossTable(x = seeds_test_labels, y = seeds_test_pred, prop.chisq=FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  40
##
##
##                    | seeds_test_pred
## seeds_test_labels |         1 |         2 |         3 | Row Total |
## ------------------|-----------|-----------|-----------|-----------|
##                 1 |        17 |         0 |         0 |        17 |
##                   |     1.000 |     0.000 |     0.000 |     0.425 |
##                   |     0.850 |     0.000 |     0.000 |           |
##                   |     0.425 |     0.000 |     0.000 |           |
## ------------------|-----------|-----------|-----------|-----------|
##                 2 |         1 |        14 |         0 |        15 |
##                   |     0.067 |     0.933 |     0.000 |     0.375 |
##                   |     0.050 |     1.000 |     0.000 |           |
##                   |     0.025 |     0.350 |     0.000 |           |
## ------------------|-----------|-----------|-----------|-----------|
##                 3 |         2 |         0 |         6 |         8 |
##                   |     0.250 |     0.000 |     0.750 |     0.200 |
##                   |     0.100 |     0.000 |     1.000 |           |
##                   |     0.050 |     0.000 |     0.150 |           |
## ------------------|-----------|-----------|-----------|-----------|
##      Column Total |        20 |        14 |         6 |        40 |
##                   |     0.500 |     0.350 |     0.150 |           |
## ------------------|-----------|-----------|-----------|-----------|
##
##
```

```
knn21_accuracy_Test <- sum(diag(knn_table_mat[["t"]])) / sum(knn_table_mat[["t"]])
```

In the above table we see that all 17 seeds(100%) of Type 1 were predicted correctly, but it has a few false positives (3 seeds that were wrongly predicted as Type one: one seed was actually Type 2 and 2 were Type 3). on the other hand- Types two and three which had no false positives (all seeds that were predicted as two or three were actually two or 3 respectively) and only a few false negatives (14/15 Type 2 seeds (93.3%) and

6/8 Type 3 seeds (75%) were identified correctly). these are really good results!! Lets see if we can improve our results. First, lets try z-score standardization instead of normalization:

```
seeds_train_z <- as.data.frame(scale(seeds_train[1:7]))
seeds_test_z <- as.data.frame(scale(seeds_test[1:7]))
seeds_test_pred_z <- knn(train = seeds_train_z, test = seeds_test_z, cl = seeds_train_labels, k=3)
knn_z_table_mat<-CrossTable(x = seeds_test_labels, y = seeds_test_pred_z, prop.chisq=FALSE)
knnz_accuracy_Test <- sum(diag(knn_z_table_mat[["t"]])) / sum(knn_z_table_mat[["t"]])
```

Not much of an improvement, maybe even worse. We tried several more times with different K's (3, 9 and 15)

It looks like the results are exactly the same (accuracy went of 92.5%), except for K = 9 we got 7/8 Type 3 seeds (87.5%) that were predicted correctly and only once predicted falsely as one (which improved both the true positive on Type 3 and the false negative on Type one) so that our accuracy went up from 92.5% to 95%.

feature selection

The DT algorithm uses the feature that provides the most Information Gain in every split. Using that logic, the feature that is used the least provides the least information gain, and if it doesn't add much information it might disturb as noise. Thus, since we know that the accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, we tried the KNN algorithm once more, with K=9 and without the "Kernel.Length" column (which attributed the least to the building of the DT).

```
feature_selected_train <- seeds_train[-4]
feature_selected_test <- seeds_test[-4]
feature_selected_test_pred <- knn(train = seeds_train_z, test = seeds_test_z, cl = seeds_train_labels,
knn_9_no4_table_mat<-CrossTable(x = seeds_test_labels, y = feature_selected_test_pred, prop.chisq=FALSE
knn9_no4_accuracy_Test <- sum(diag(knn_9_no4_table_mat[["t"]])) / sum(knn_9_no4_table_mat[["t"]])
```

We managed to improve the true positive of Type three to 100%! We have only one mistake left (a Type two seed that was predicted as Type 1) with accuracy of 97.5%. We tried again without Compactness and again without both Compactness and Kernel.Length, and we still got to 97.5% accuracy

KNN conclusions

We managed to classify our data using the KNN algorithm. We saw that the best results were with K=9 and without the "Kernel.Length" (or "Compactness") column, and we managed to classify correctly all seeds except for one.

total conclusions

By using boosting we increased our accuracy to 95% in the Decision Tree algorithm. We ran the KNN algorithm four times, and found that we get the best results using K = 9. Using the information from the Decision Tree algorithm we managed to increase the accuracy of our KNN algorithm.