assignement2

Lilach Herzog & Leslie Cohen

13 5 2022

Table of Contents

[introduction 1](#_Toc105598070)

[the work 2](#_Toc105598071)

[preproccessing 2](#_Toc105598072)

[environment and data: 2](#_Toc105598073)

[find the type 2](#_Toc105598074)

[Normalization 2](#_Toc105598075)

[split into training and test sets 2](#_Toc105598076)

[target factor (label) vector for classification 3](#_Toc105598077)

[Decision tree 3](#_Toc105598078)

[simple facts about the tree 3](#_Toc105598079)

[Explanation of the tree: 3](#_Toc105598080)

[Adaptive Boosting 3](#_Toc105598081)

[DT conclusions 4](#_Toc105598082)

[KNN Algorithm 4](#_Toc105598083)

[KNN conclusions 5](#_Toc105598084)

[total conclusions 5](#_Toc105598085)

# introduction

Classify the seed types using 2 classifying methods:

# the work

## preproccessing

### environment and data:

library(gmodels)  
library(C50)  
  
# read data  
seeds <- read.csv("seeds.csv")  
  
# Consider setting a seed for reproducible results  
  
#to initialize a pseudo random number generator.   
set.seed(1234)

### find the type

The aim: classify the seed types. There are 3 types of seeds: 1,2,3. we took a quick look at the data to try to find which parameter is important :

##### convert type from int variable to factor

we converted the class (“type”) to a factor, as it is required by the C50 package.

The only “repetitive variable” we found is kernel groove.

the kernel groove varies between 4.5 and 6.55, the kernel lenght between 4.89 and 6.67 and the kernel width of seeds varies between 2.63 and 4.033.

How can we associate that to the different types of seeds ??

### Normalization

We’ll have to normalize our data to make sure our classifier will work. Lets create a simple normalization function (min/max normalization):

Use the normalization function on our data, and make sure it worked:

### split into training and test sets

Lets split into training and test sets:

There are 66 type 1, 68 type 2 and 65 type 3 from our table seeds, which means that we have about a third of each type in the data set. For that we should check that we got about 1/3 of each type in each data set (training and test sets):

#### target factor (label) vector for classification

## Decision tree

The 8th column of the dataset is the type class variable, so we need to exclude it from the training data frame, but supply it as the target factor (label) vector for classification:

The tree size is 6 and it contains 7 predictors.

### simple facts about the tree

The preceding text shows some simple facts about the tree, including the function call that generated it, the number of features (labeled predictors), and examples (labeled samples) used to grow the tree. Also listed is the tree size of 7, which indicates that the tree is 7 decisions deep >>>>>>> d969d9d6123bf03d2377ec0523df743bd3513325

Next we’ll look at the summary of the model.

### Explanation of the tree:

There are only 2 possibilities of the kernel groove is greater than 5.528, The kernel groove <=5.528, there is one possibility of Area > 13.37, otherwise, Area<=13.37. In this case, there is only one possibility to Kernel.Groove <= 4.783, otherwise, one possibility of Kernel.Groove > 4.783 , otherwise, Kernel.Groove<=4.783 and one possibility for Asymmetry.Coeff <= 1.502, otherwise, Asymmetry.Coeff> 1.502 and3 possibilities to kernel.Groove>4.914, otherwise, kernel.groove<=13.914 and 3 possibilities for perimeter<=13.47 and 1 possibility foe Perimeter>13.47.

the percentage of error is relatively low: 1.3% Kernel.Groove and area seem to be the better attribute. To construct the tree, the model used 4 attributes: Kernel.Groove, area, Asymmetry.Coeff and Kernel.Width.

We want to evaluate our prediction

Type seed 2 and 3 are predicted correctly at 100%, there is no false positive or false negative. The type seed 1 has some false negatives,but few.

We will try to improve our results, using adaptive boosting to have a lower percentage of error. This is a process in which many decision trees are built and the trees vote on the best class for each example.

### Adaptive Boosting

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:

The classifier made 1 mistake for an error rate of 0.6% percent. This is an improvement over the previous training error rate before adding boosting! However, it remains to be seen whether we see a similar improvement on the test data. Let’s take a look:

boosting on test data:

The model made 2 errors instead of 3 before the boost, on seeds of type 1. We will try to ameliorate predictions. We will fine-tune our algorithm, using a cost matrix. The C5.0 algorithm allows us to assign a penalty to different types of errors, in order to discourage a tree from making more costly mistakes. The penalties are designated in a cost matrix, which specifies how much costlier each error is, relative to any other prediction.

First, we’ll create a default 2x2 matrix, to later be filled with our cost values:

if the model get a mistake it put on it a big penalty. it is to have a better prediction. the penalty is based on the cost matrix Suppose we believe that a loan default costs the bank four times as much as a missed opportunity. Our penalty values could then be defined as:

Now lets train again and see if the cost matrix made any difference:

### DT conclusions

then we should write our conclusion about the different predictions obtained. Compare these results to the boosted model; this version makes more mistakes overall, but the types of mistakes are very different. Where the previous models incorrectly classified a small number of defaults correctly, our weighted model has does much better in this regard. This trade resulting in a reduction of false negatives at the expense of increasing false positives may be acceptable if our cost estimates were accurate.

To create our decision trees in this practice we used the C5.0 package. There is another package called “party” which has the ‘ctree’ function which also generates decision trees. You can read about it here: <https://ademos.people.uic.edu/Chapter24.html>.

## KNN Algorithm

First we will work with a new table of data. Then,we will transform the values of the Type variable to a factor variable:

Since we want to use KNN, we have to make sure the range of values of each parameter is similar.

The next step is to create a training set and a test set, and to store the labels in a different vector to be used later. We usually want to divide our data into 80%, 20% test, but we can always try a ratio of 85-15 or even 90-10 and see if results improved.

Now, we will run the KNN algorithm and We will start with K of 3, the square root of 159 and also an odd number, reducing the chance of a tie vote.

So, how well did we do? 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 we’ll load:

??? I did not understand this table where is the error????

In the above table we see that all of the benign were predicted correctly, and 36 out of the 39 malignant were too. 3 malignant labels were mistaken for being benign.

Lets see if we can improve our results. First, lets try z-score standardization instead of normalization:

We’ll quickly repeat the previous steps (divide the data into train and test, classify, and compare the results to the true values):

It seems to be worth, the model predicted only 2 types on 3. What about a different value of K?

### KNN conclusions

# total conclusions