Implementing Random Forest in R

A Practical Application of Random Forest in Classifying Breast Cancer Patients

Joe Tran

Sep 10, 2019 · 6 min read ★

Photo by Rural Explorer on Unsplash

What is Random Forest (RF)?

In order to understand RF, we need to first understand about decision trees. Rajesh S. Brid wrote a detailed article about decision trees. We will not go too much in details about the definition of decision trees since that is not the purpose of this article. I just want to quickly summarise a few points. A decision tree is series of Yes/No questions. For each level of the tree, if your answer is Yes, you fall into a category, otherwise, you will fall into another category. You will answer this series of Yes/No questions until you reach the final category. You will be classified into that group.

Taken from here

Trees work well with the data we use to train, but they are not performing well when it comes to new data samples. Fortunately, we have Random Forest, which is a combination of many decision trees with flexibility, hence resulting in an improvement in accuracy.

Here I will not go too much into details about RF because there are various sources outside we can get to understand what is the mathematics behind it. Here is one of them.

This article is more about practical application of RF in classifying cancer patients, so I will jump straight into the coding part. Now let's open Rstudio and get our hands dirty :)

Implementing RF in R

First of all, we need to load the following packages. If you cannot load them, chances are you have not installed them yet. So please do so first before loading the following packages.

```
library(ggplot2)
library(corrplot)
library(reshape2)
library(ggthemes)
library(dplyr)
library(randomForest)
Wisconsin =
read.table(url(paste0("https://archive.ics.uci.edu/ml
/machine-learning-databases/",
"breast-cancer-
wisconsin/wdbc.data")),header=FALSE,sep=",",nrows=570
)
```

I read the data directly from the web link and name the dataset as Wisconsin. Let's inspect the data a little bit

```
head(Wisconsin)
```

V1 is the ID so it is not relevant in our analysis here. V2 is the classification result, with ‘M’ standing for ‘Malignant’ and ‘B’ standing for ‘Benign’. The rest is just the variables of information about the cancer diagnosis.

Now I would like to change M and B to TRUE and FALSE for easier interpretation.

```
Wisconsin$V2 <- Wisconsin$V2 == “M”
```

Pre-processing data

First, we shuffle the data and split it into train and test. We decide a 70–30 split for this.

```
set.seed(2019)

test_size = floor(0.3 * nrow(Wisconsin))
samp = sample(nrow(Wisconsin), test_size,replace =
FALSE)

y_train = Wisconsin[-samp,2]
x_train = Wisconsin[-samp,-c(1,2)] #since the first
column is just ID
y_test= Wisconsin[samp,2]
x_test = Wisconsin[samp,-c(1,2)] #since the first
column is just ID

#convert labels to categorical
y_train = factor(y_train)
y_test = factor(y_test)
```

We should note that RF only works when the response variable is a *factor*. **Just now when we convert 'M' and 'B' into TRUE and FALSE, the type of this variable is logical. Hence we need to convert it into factor by using factor() function**.

Now let's combine x and y to form training and test set.

```
#Create training set and testing set
train = cbind(y_train,x_train)
test = cbind(y_test,x_test)
```

The training set will be used to train the RF model and the test set will be used to test the performance of the model. Now let's give the name to our response variable. Here I named it as 'label'

```
colnames(train)[1] = ‘label’
colnames(test)[1] = ‘label’
```

It now looks like this

Fit a Random Forest model

Now everything is ready. We can start fitting the model. This step is easy.

The ‘randomForest()’ function in the package fits a random forest model to the data. Besides including the dataset and specifying the formula and labels, some key parameters of this function includes:

1. **ntree**: Number of trees to grow. The default value is 500.

2. **mtry**: Number of randomly selected variables for each split. In this

example we use the square root of p (p denotes the number of predictors). Note that for regression analysis the general rule is to use mtry = p/3 and is also the default value of this parameter for regression.

3. **importance**: If True, the model will calculate the feature importance for further analysis. (default = False)

4. **proximity**: If True, the model will contain a N*N matrix which represents the proximity measure.

5. **maxnodes**: The maximum number of terminal nodes the trees can have.

6. **na.action**: A function to specify how the missing data should be treated.

Since there are 30 independent variables, we set **mtry** to be square root of 30 and then fit the model

```
mtry = sqrt(30)
model_1 = randomForest(label~., data = train,
importance = TRUE)
```

That is it. Simple isn't it? Now we already have a RF model

```
print(model_1)
```

The Out-of-bag OOB error estimate rate is 3%, which is very good, i.e. 97% accuracy. If we look at the Confusion Matrix, we can see that classification error is quite low. This shows that our RF model is performing well in classifying the train set.

Let’s test the model with our test set.

```
pred_1 = predict(model_1, x_test)
table(y_test, pred_1)

accuracy_m1 = mean(y_test == pred_1)
```

Looks like our model is performing well on the test set too, with the accuracy of 95%.

Variable Importance

```
varImpPlot(model_1)
```

Importance of variables in the model. **The higher the rank, the more important the variables**

Another way for us to visualise the plot is to use **ggplot package.** Do note that the code below is to visualize the 'Mean Decrease Accuracy'. To get the 'Mean Decrease Gini', simply change the line in bold below to 'MeanDecreaseAccuracy' (no spacing).

```
importance = importance(model_1)
varImportance = data.frame(Variables =
row.names(importance),
 Importance =round(importance[,
"MeanDecreaseAccuracy"],2))

rankImportance=varImportance%>%mutate(Rank=paste('#',
dense_rank(desc(Importance))))

ggplot(rankImportance,aes(x=reorder(Variables,Importa
nce),
 y=Importance,fill=Importance))+
 geom_bar(stat='identity') +
 geom_text(aes(x = Variables, y = 0.5, label = Rank),
 hjust=0, vjust=0.55, size = 4, colour = 'white') +
 labs(x = 'Variables') +
 coord_flip() +
 theme_classic()
```

Mean Decrease Accuracy

The result is similar to the plot we obtained earlier. The result shows that variables V25, V30, V26 and V23 are the most important.

Mean Decrease Gini

Using Mean Decrease Gini, we get V25, V23, and V26 as the most important variables.

Conclusion

This article shows how to implement a simple Random Forest model in solving classification problems. I did not go too deep into how to tune the parameters in order to optimize the model because with such a high accuracy in classification, I think that a simple model would be enough. HOWEVER, in real life, there are other much more complicated classification problems that require us to tune the parameters to obtain the best model, which I will write a separate article on next time. But it is important to remember to ALWAYS start with a simple model and then from there build up the model to get a better prediction.

Thank you for your time. I hope this article helps you guys, especially those who have never tried implementing RF in R before, gain a better idea of how to do it. Let me know if you have any comments or questions.

Have a great day and happy programming :)

Machine Learning Random Forest Data Science Towards Data Science Classification