Handmade Random Forest

Álvaro Orgaz Expósito

THEORY

The aim of the *Random Forest* algorithm is to predict the variable y (categorical for *classification* problem or continuous for *regression problem*) knowing the explanatory variables $x = \{x_1, \ldots, x_p\}$.

Firstly, you need to understand the concept of *bagging* applied in *Random Forest*. The idea of *bagging* is resampling with replacement a fraction of the training data observations (known as *bootstrapping*) to generate B different bootstrap samples of the training data. Then, it trains a *base learner* on each sample in order to average the predictions of all them as final prediction. The idea is to average the predictions of uncorrelated models with high variance in its predictions. Then, *bagging* reduces prediction variance over a single tree and improves prediction accuracy at the expense of interpretability.

$$Variance(\hat{y}_{bagging}) = Variance\left(\frac{\sum_{b=1}^{B}\hat{y}^b}{B}\right) = \frac{1}{B^2}Variance\left(\sum_{b=1}^{B}\hat{y}^b\right) = \frac{B \cdot Variance(\hat{y})}{B^2} = \frac{Variance(\hat{y})}{B}$$

Note: A *base learner* can be any predictive model appropiate for the type of problem.

Secondly, the *Random Forest* uses the *Decision Tree* as *base learner* and you can find all the necessary information about it in the paper *Handmade_Decision_Tree.pdf*. But apart from resampling the observations with *bagging*, it also resamples the number of explanatory features used to train each B *base learners*. It means that we fix a hyperparameter as the number of features to consider in each iteration and we select them randomly.

It has a clever rationale because if there is one very strong predictor in the dataset, most of the trees would use this strong predictor in the top splits and consequently, all of the bagged trees will be similar and their predictions will be highly correlated. Thus, deleting this feature in some of the *base learner* adds diversity to the final prediction and more reliable results are obtained.

Note: As each *base learner* is a *Decision Tree*, apart from the hyperparameters of the *Random Forest* we also have the hyperparameters of the *Decision Tree* model. Ingeneral, it is better to let the *Decision Tree* grow as much as possible because then we will get single *base learner* models with more variance in its predictions and less correlated with each other.

Thirdly, let's see the main steps of the *Random Forest* algorithm, which is one of the available algorithms that use *bagging*.

1. Define the inputs $x_{[NxP]}$ and $y_{[Nx1]}$.
2. Define the number of iterations or trees B, the fraction of observations and features randomly sampled in each tree, and the hyperparameters of the base learner *Decision Tree* used in each iteration.
3. For each iteration $b = 1, ..., B$:

   3.1 Create the random bootstraped sample of the data.

   3.2 Select randomly as many features as previously defined.

   3.3 Train a *Decision Tree* with the defined hyperparameters and the sampled data to predict the target variable.
4. Once we have the B trees built and the predictions in each tree, we can make the final *Random Forest* prediction $\hat{y}$ as:

- In a regression problem

$$\hat{y} = \frac{\sum_{b=1}^{B} \hat{y}^b}{B}$$

- In a *classification* problem with K categories in y

$$\hat{y} = max_k \left(\sum_{b=1}^{B} \begin{cases} 1 & if \ \ \hat{y}^b = k \\ 0 & if \ \ \hat{y}^b \neq k \end{cases} \right) \quad with \quad k = 1, ..., K$$

Note: The overall classification of each observation in the final *Random Forest* model is the class k more voted after aggregating the votes of the B base learners or trees.

5. Now, predict a x^{test} data by passing each observation through the B trees and predicting as explained.

Note: We will use for each *base learner* the implemented function *rpart* for *Decision Trees* in the R package *rpart*, setting all the hyperparameters by default.

CODE

The data used is the popular dataset *iris*. Let's predict the categorical variable *Species* (then it is a *classification* problem with 3 categories) with the explanatory features: *Petal.Width* and *Sepal.Width*.

Let's define the inputs x and y. Also, let's define x^{test} by creating artificial data with all combinations of both features from the minimum to the maximum values in x by 0.02, it will be interesting for observing the *Random Forest decision boundaries* in the following plots.

```
x <- iris[,c("Sepal.Width","Petal.Width")]
y <- iris["Species"]
x_test <- expand.grid(list(Sepal.Width=seq(min(x[,1]),max(x[,1]),0.02),
                           Petal.Width=seq(min(x[,2]),max(x[,2]),0.02)))
```

Let's create the *base_learner* function which uses the inputs x and y for training a *Classification Decision Tree* with hyperparameters by default, and also computes the predictions for the input x^{test}.

```
base_learner <- function(x,y,x_test){
  library(rpart)
  decision_tree <- rpart(y~.,data=x,method="class")
  predictions <- predict(object=decision_tree,newdata=x_test,type="class")
  return(list(decision_tree=decision_tree,predictions=predictions))
}
```

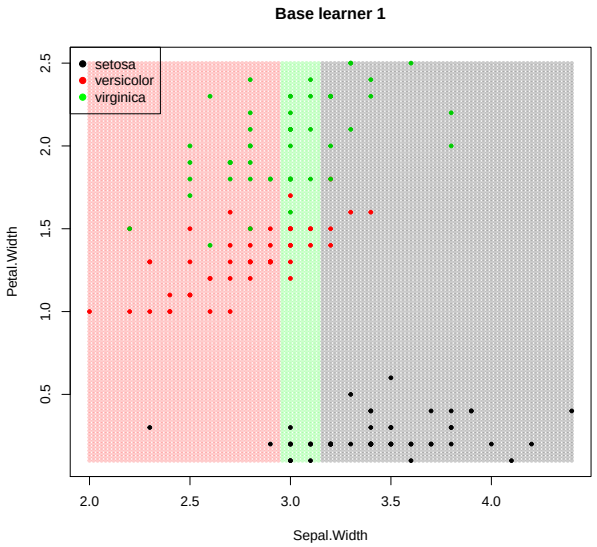
Let's create the *Random Forest* algorithm.

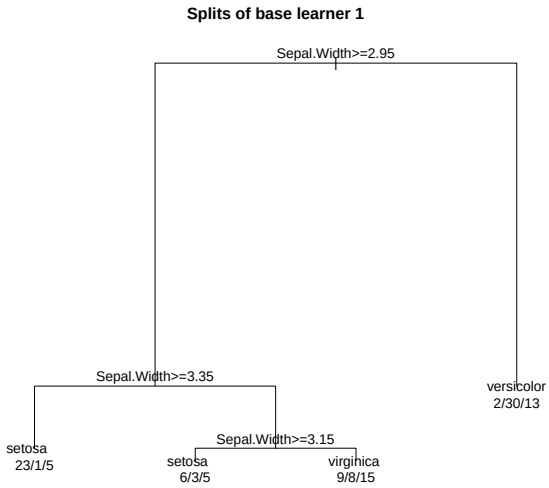
```
RandomForest <- function(x,y,x_test,base_learner,B,bootstrap,number_features){
  N <- nrow(x)
  P <- ncol(x)
  K <- length(unique(y))
  N_test <- nrow(x_test)
  # Create a matrix for the final Random Forest model prediction of x_test
  predictions_test <- matrix(0,nrow=N_test,ncol=K)
  # Iterate the B bagging iterations
  for(b in 1:B){
    # Boostraped observations index
    set.seed(b)
    observations_b <- sample(1:N)[1:round(N*bootstrap)]
    # Sampled features index
    set.seed(b)
```

```
    features_b <- sample(1:P)[1:number_features]
    # Train the base learner b and compute the predictions for x_test
    predictions_test_b <- base_learner(x[observations_b,features_b,drop=FALSE],
                                       y[observations_b],x_test)$predictions
    # For each observation in x_test, sum 1 to the predicted class
    for(i in 1:N_test){
      k <- predictions_test_b[i]
      predictions_test[i,k] <- predictions_test[i,k] + 1
    }
  }
  # Compute the single overall model prediction for x_test
  y_test <- rep(0,times=N_test)
  for(i in 1:N_test){
      y_test[i] <- which.max(predictions_test[i,])
  }
  return(y_test)
}
```

Now, let's apply the first base learner (iteration 1 of the *Random Forest*) with bootstrap fraction 0.50 and number of features $\sqrt{P}$, and plot the result.

```
par(mfrow=c(1,2),xpd=TRUE)
# Boostraped observations index
bootstrap <- 0.8
N <- nrow(x)
set.seed(1)
observations_1 <- sample(1:N)[1:round(N*bootstrap)]
# Sampled features index
P <- ncol(x)
number_features <- round(sqrt(P))
set.seed(1)
features_1 <- sample(1:P)[1:number_features]
# Train the base learner 1
base_learner_1 <- base_learner(x[observations_1,features_1,drop=FALSE],y[observations_1],x_test)
# Plot decision boundaries
predictions <- base_learner_1$predictions
transparent_colors <- scales::alpha(c("black","red","green"),0.2)
plot(x_test[,1],x_test[,2],col=transparent_colors[as.numeric(predictions)],
     pch=19,cex=0.5,xlab="Sepal.Width",ylab="Petal.Width",main="Base learner 1")
points(x[,1],x[,2],col=y,pch=19,cex=0.6)
legend("topleft",legend=unique(y),col=c("black","red","green"),pch=19)
# Splits
plot(base_learner_1$decision_tree,main="Splits of base learner 1")
text(base_learner_1$decision_tree,use.n=TRUE)
```





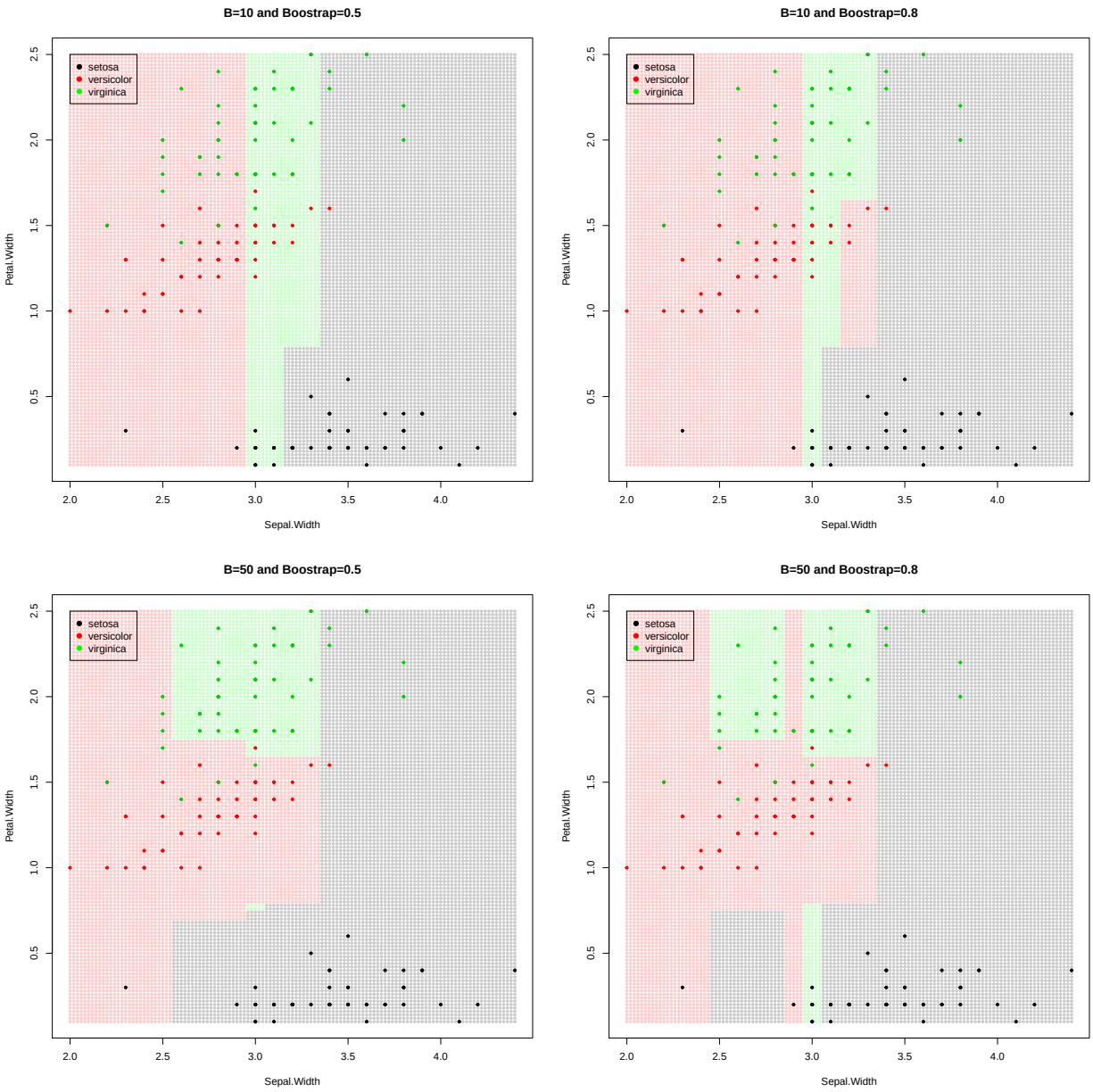
Now, let's apply the *Random Forest* with bootstrap fractions 0.50 or 0.80, number of features $\sqrt{P}$, and number B of trees 10 or 50.

```
par(mfrow=c(2,2))
# Ranfom Forest with b=10 and bootstrap fraction 0.50
predictions <- RandomForest(x,y,x_test,base_learner,B=10,bootstrap=0.50,number_features)
transparent_colors <- scales::alpha(c("black","red","green"),0.2)
plot(x_test[,1],x_test[,2],col=transparent_colors[as.numeric(predictions)],
     pch=19,cex=0.5,xlab="Sepal.Width",ylab="Petal.Width",main="B=10 and Boostrap=0.5")
points(x[,1],x[,2],col=y,pch=19,cex=0.6)
legend(2,2.5,legend=unique(y),col=c("black","red","green"),pch=19)

# Ranfom Forest with B=10 and bootstrap fraction 0.80
predictions <- RandomForest(x,y,x_test,base_learner,B=10,bootstrap=0.8,number_features)
transparent_colors <- scales::alpha(c("black","red","green"),0.2)
plot(x_test[,1],x_test[,2],col=transparent_colors[as.numeric(predictions)],
     pch=19,cex=0.5,xlab="Sepal.Width",ylab="Petal.Width",main="B=10 and Boostrap=0.8")
points(x[,1],x[,2],col=y,pch=19,cex=0.6)
legend(2,2.5,legend=unique(y),col=c("black","red","green"),pch=19)

# Ranfom Forest with B=50 and bootstrap fraction 0.50
predictions <- RandomForest(x,y,x_test,base_learner,B=50,bootstrap=0.50,number_features)
transparent_colors <- scales::alpha(c("black","red","green"),0.2)
plot(x_test[,1],x_test[,2],col=transparent_colors[as.numeric(predictions)],
     pch=19,cex=0.5,xlab="Sepal.Width",ylab="Petal.Width",main="B=50 and Boostrap=0.5")
points(x[,1],x[,2],col=y,pch=19,cex=0.6)
legend(2,2.5,legend=unique(y),col=c("black","red","green"),pch=19)

# Ranfom Forest with B=50 and bootstrap fraction 0.80
predictions <- RandomForest(x,y,x_test,base_learner,B=50,bootstrap=0.8,number_features)
transparent_colors <- scales::alpha(c("black","red","green"),0.2)
plot(x_test[,1],x_test[,2],col=transparent_colors[as.numeric(predictions)],
     pch=19,cex=0.5,xlab="Sepal.Width",ylab="Petal.Width",main="B=50 and Boostrap=0.8")
points(x[,1],x[,2],col=y,pch=19,cex=0.6)
legend(2,2.5,legend=unique(y),col=c("black","red","green"),pch=19)
```



In conclusion, in this plots, we can observe the predictions for x^{test} (transparent coloured points) and the real values y (solid coloured points). Also, we can observe the *decision boundary* for each class in the target variable *Species* in the *iris* data.

Comparing the plots of four models (first base learner and four versions of *Random Forest*), the decision boundaries are more complex and smooth when increasing the number B of *bagging* iterations and the when increasing the *bootstrap* fraction. This is because each iteration improves the accuracy of the final overall models, and getting more fraction of the training data tends to overfit. Then, the model that better generalizes is the *Random Forest* with 50 trees but *bootstrap* fraction 0.5, which is a very interesting example of how avoiding overfitting.

Note: Remember that we are using always the same base learner (*Decision Tree* with the same hyperparameter) and we get smoother decision boundaries and higher accuracy in the single overall model just *bagging* it and selecting different features. It is really useful because it enables us to improve our model without changing it for another more complex.