-This is part of the Content Editor Internship- Why Random Forest Algorithm is the best Algorithm to start as Data Science Fresher

aravind2010 — 8 November 2020

Introduction:

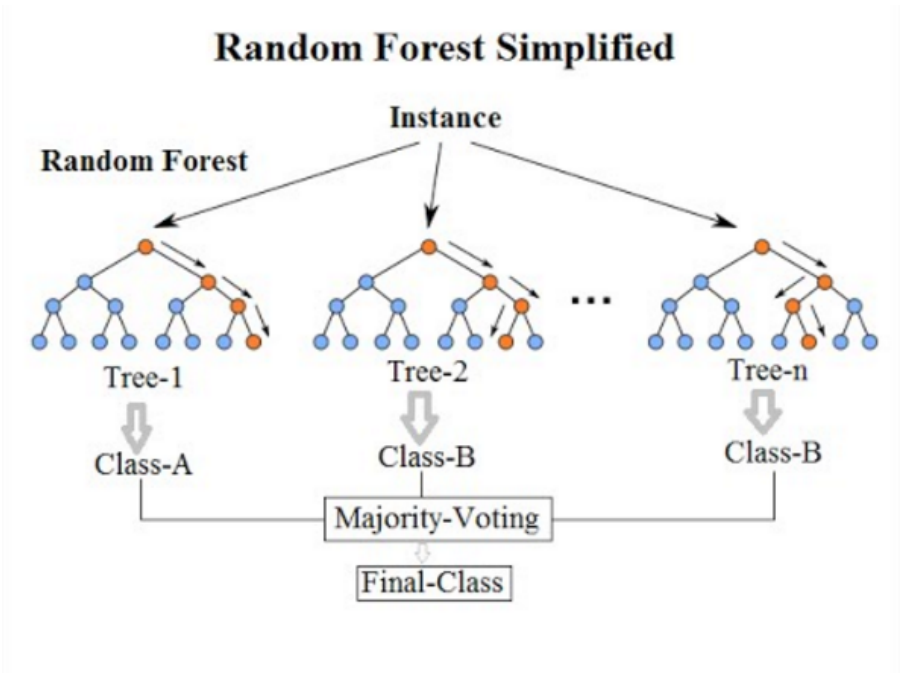
To all those Freshers with basic knowledge in Statistics and who is trying to be a Data Science Aspirant and don't know which algorithm you should start, You came to the right article. The answer to your question is Random Forest, why?.

Whenever you have a Dataset and you have no idea how to begin your analysis on given dataset Random Forest always act as your friend and gives some insight about the datasets.

Anyone with basic knowledge of working of Random Forest can understand the other ML algorithms and even the Neural Networks which are basically consist of tree like structure but called has hidden layers and nodes

What is Random Forest?

Random Forest is a Tree based algorithm and it uses Bagging technique, it is typically a collection of Decision trees and in which each tree contains the part of whole input features which is called has Bootstrapping and the RF model gives the output based on the majority voting from each Decision Tree.

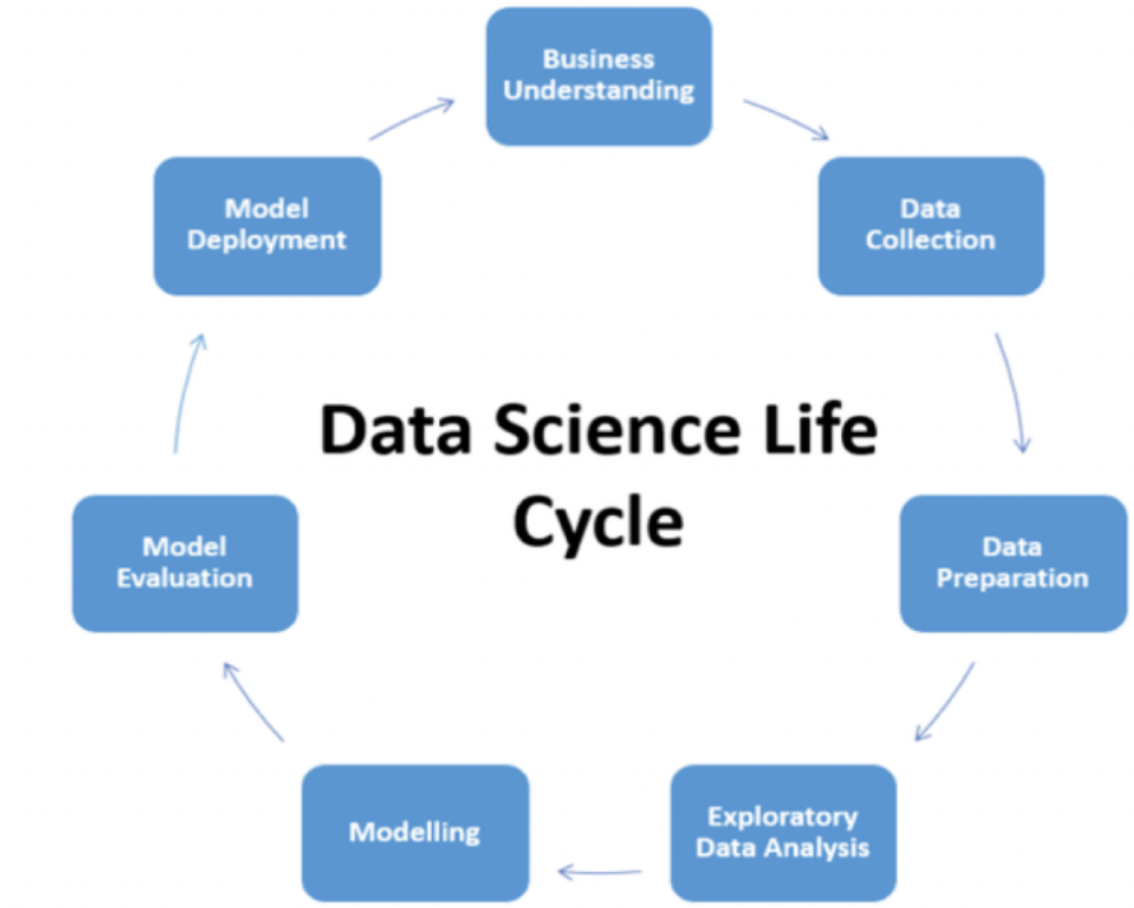


Venkata Jagannath - https://community.tibco.com/wiki/random-forest-template-tibco-spotfirer-wiki-page

Random forest is good because the chances of an ensemble of decision trees are correlated to each other are less due to the method of random subset of feature in each Decision Tree which is called has bootstrapping. This is necessary so that variance can be averaged away.

Why RF Algorithm is good to learn as a fresher?

After learning about Data Prepossessing and handling the data one should work with different algorithms and find, the better algorithm which gives good evaluation results so looking to the below diagram:



NETALI AGRAWAL - https://mk0analyticsindf35n9.kinstacdn.com/wp-content/uploads/2020/04/Screenshot-2020-04-15-at-10.08.12-AM-770x632.png

Except for Modelling which is the type of Algorithm, the other cycle of the DS are going to be similar with slight changes in the evaluation of the used models.

So, Random Forest can be treated as base of other models its because:

We can use The RF model for Both Classification and regression data sets
It has the decision making nodes, which will determine which node to be selected
It has various Hyper Parameter tuning methods
so these are the fundamentals point to be remember for any DS Algorithms so if one can understand these it can be easier to learn other models to.

Types of Random Forest:

Random Forest Classifier:

The Working of the model is similar takes features, create multiple decision trees and majority voting gives the result. The can be done not only for Binary classification one can do Multiple-class Classification with the same technique

Random Forest Regressor:

The Working of the model is similar takes features, create multiple decision trees, but instead of majority voting whichever results are determined the average of all values are taken as the final result in the RF Regressor.

How RF Works?

So, we all know that RF model is an ensemble of Decision Trees. So to know how the model is created based on given input features one should know how these Decision trees make split. The decision trees makes the split based on the

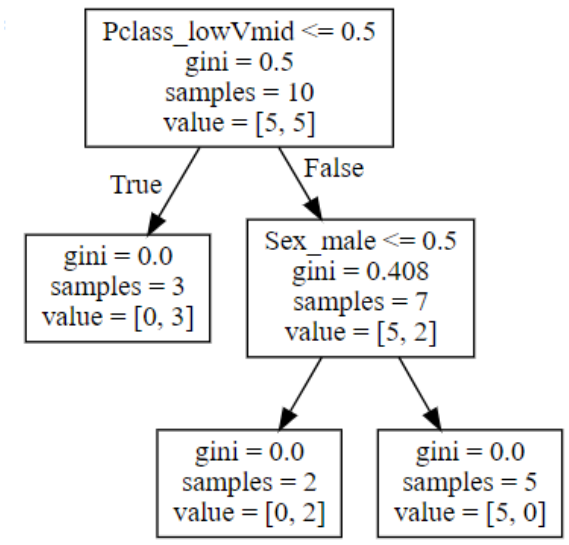
Entropy or Gini Index
Information gain.

Entropy or Gini Index:

Both Entropy and Gini Index plays the sole purpose of finding the homogeneous node which is the split node contains only the desired output and not mixed with other output:

So, In the above diagram see that at start 10 value inputs are given in which 5 inputs belong to say "A" output and another 5 belongs to "B" output. The ultimate goal is to find the node which explains the exact output from each input.

These splits are done based on given below formula:



https://i.stack.imgur.com/iXc11.png

$$GI = 1 - \sum_{i=1}^{n} (p)^2$$

$$GI = 1 - \left[(P_{(+)})^2 + (P_{(-)})^2 \right]$$

https://media.geeksforgeeks.org/wp-content/uploads/20200620175232/Screenshot-2020-06-20-at-5.51.42-PM.png

In the above formula, by determining the probability at each node, for example: in the above diagram after first split we can see that in the second node lets say "A" has 5 inputs and "B" has 2 inputs

After the first split we can determine the Gini index .**Gini Index is always between 0 to 0.5** and getting 0 is the ultimate goal which is called has **homogeneous node**

Coming to Entropy it always lies between 0 to 1. and it posses the same property of Gini Index. The only differentiation is, It has different formula which is:

So, by discussing above two and which should you choose between GI and H(s) **most of the researchers use GI** since it requires less computation power its because Entropy requires logarithmic process and its values ranges from 0 to 1.

Information Gain:

We saw how to determine the homogeneity of the node. Now, lets see how these GI or Entropy helps to calculate Information Gain

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in D_A} \frac{|S_v|}{|S|} Entropy(S_v)$$

https://www.c-sharpcorner.com/article/decision-tree/Images/ig.png

In the formula we can either take entropy value or Gini index

Information Gain([5,5]), A1) = 0.5 - (3/10)*0 - (7/10)*0.408 -(2/10)*0 - (5/10)0 = 0.2144

It is the information gain value of present constructed decision tree. similarly for different input feature tree is constructed and which ever tree gives maximum information gain that tree is considered
This is how GI and IG plays the vital role in splitting of Tree.

Hyperparameter Tuning:

The key feature for any ML or DL model is hyper parameter tuning. If any on wondering what is hyper parameter tuning it is modifying the model such it varies how the model is trained. In other words lets say in RF model we can vary these parameter such a way that constructing of tree differs

Why we want to tune this parameters?

without parameter tuning the growing of tree still takes place which leads to over fitting of the model

The following five hyperparameters are commonly adjusted:

N_estimators

This hyperparameter determines the number of Decision tree to be used, a addition of decision tree can increase accuracy of the prediction but it leads to in crease in computational power

Random forest models are ensembles of decision trees and we can define the number of decision trees in the forest. Additional decision trees typically improve model accuracy because predictions are made based on a larger number of “votes” from diverse trees, however, large numbers of trees are computationally expensive.

Max_features

This hyperparameter helps to resample the input features. A larger value of Max_features can increase the performance of the model but it also leads to overfitting. Some of the common choice of Max_features are as follows:

‘auto’: places no restrictions on the number of features,

‘sqrt’: square root of the total number of features,

‘log2’: base two logarithm of the total number of features,

'0.9': 90% of the input features taken in the account

'0.2': 20% of the input features taken in the account

Max_depth

We all know that in RF Model each tree do multiple splits to determine the homogeneous node. But, these larger fits can leads to overfitting to avoid this Max_depth is used the default value of Max_depth is "None" in which trees will split until homogeneity node is attained or it stops when value less than min_samples_split samples

Min_samples_split

With this parameter we can control the number of samples required to split the node in tree. The values are determined based on the number of rows in the dataset. The selected values should not be to large because tree won't do split and it leads to under fitting

Min_samples_leaf

This parameter is used to stop the growth of tree it helps to determine the number of samples to be in leaf node. The values are determined by the size of the training dataset and it should not be too large which may cause insufficient variation of data.

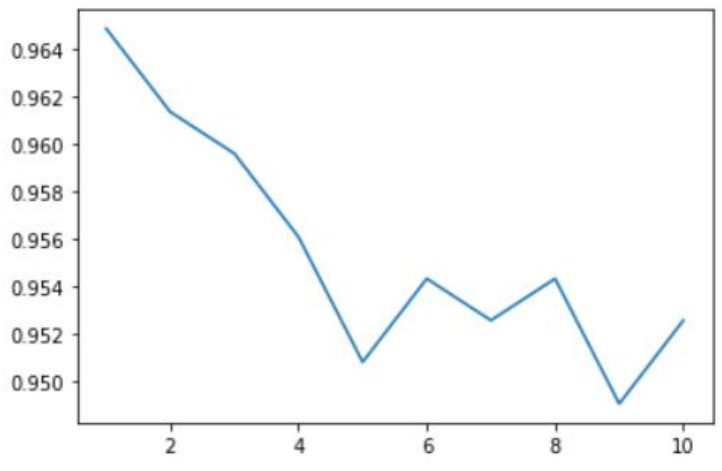
Lets see code for parameter tuning:

```
result=[]
features=[1,2,3,4,5,6,7,8,9,10]
for variables in features:
    model = RandomForestClassifier(n_estimators=50,max_features="log2",random_state=100,oob_score=True,min_samples_leaf=va
    model.fit(x,y)
    print("features_value :",variables)
    oob=model.oob_score_
    print("oob :",oob)
    result.append(oob)

pd.Series(result,features).plot()
```

```
features_value : 1
oob : 0.9648506151142355
features_value : 2
oob : 0.961335676625659
features_value : 3
oob : 0.9595782073813708
features_value : 4
oob : 0.9560632688927944
features_value : 5
oob : 0.9507908611599297
features_value : 6
oob : 0.9543057996485061
features_value : 7
oob : 0.9525483304042179
features_value : 8
oob : 0.9543057996485061
features_value : 9
oob : 0.9490333919156415
features_value : 10
oob : 0.9525483304042179

<matplotlib.axes._subplots.AxesSubplot at 0x1a4f3e7a6c8>
```



In this above code we are using For - loop to determine the best Min_samples_leaf value based on the oob score which is out_of_bag score(features not in tree used for determining the accuracy).

Similar to the above code we can also find the values for other hyper parameters.

Apart from the above method we can also use RandomizedSearchCv from Model_selection of Sklearn library:

```
from sklearn.model_selection import RandomizedSearchCV
RFR=RandomForestRegressor(random_state=1)
RFR_random = RandomizedSearchCV(estimator = RFR,
                                param_distributions = grid_param, n_iter = 500,
                                cv = 5, verbose=2, random_state=42,
                                n_jobs = -1)

grid_param = {'n_estimators': n_estimators,
              'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

# number of trees
n_estimators=[500, 800, 1500, 2500, 5000]
# max number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']
# max number of levels in tree
max_depth = [10, 20, 30, 40, 50]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 20]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10, 15]
```

Performance metrics:

After tuning the model it is necessary to determine the performance of the model which is done by follows

Accuracy Score (for regressor)
Classification report (for classifier)

Accuracy Score:

sklearn.metrics.accuracy_score(*y_true*, *y_pred*, ***, *normalize=True*, *sample_weight=None*)

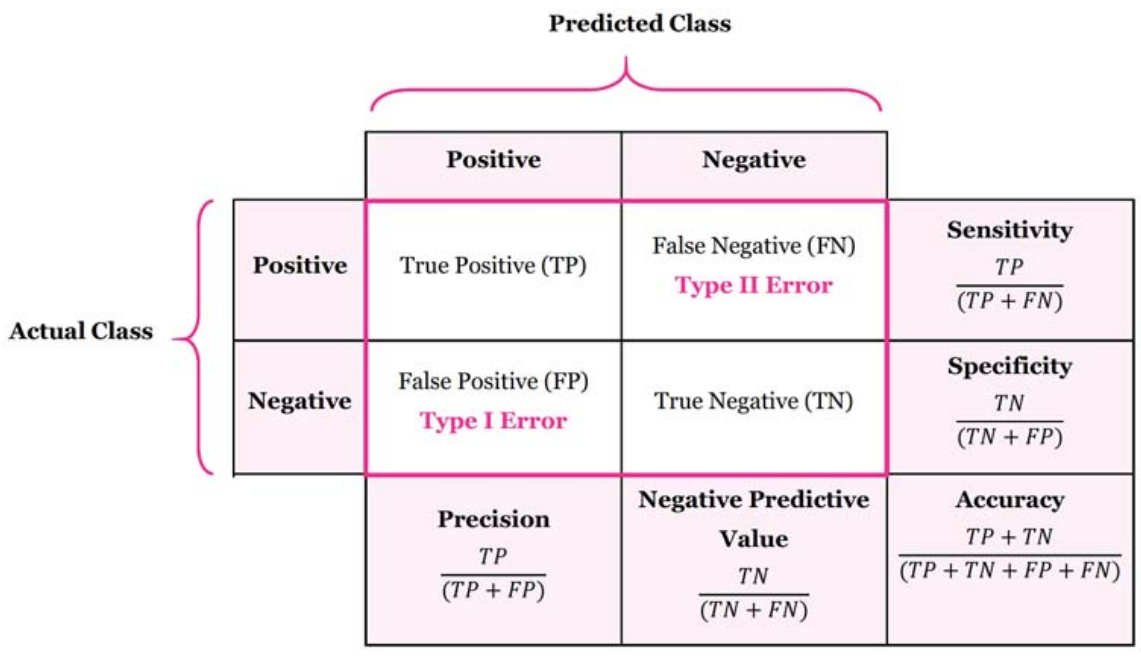
the above code helps to determine the accuracy of the model based on the predicted output and true output

Classification Report :

sklearn.metrics.classification_report(*y_true*, *y_pred*, ***, *labels=None*, *target_names=None*, *sample_weight=None*, *digits=2*, *output_dict=False*, *zero_division='warn'*)

The above code is used to determine the performance of the classifier model.

These compresses of confusion matrix and F1- score



Confusion https://2.bp.blogspot.com/-EvSXDotTOwc/XMfeOGZ-CVI/AAAAAAAAEiE/oePFfvhfOQM11dgRn9FkPxlegCXbgOF4QCLcBGAs/s1600/confusionMatrxiUpdated.jpgmatrix and their formulas

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	50
Iris-versicolor	0.77	0.96	0.86	50
Iris-virginica	0.95	0.72	0.82	50
avg / total	0.91	0.89	0.89	150

Sample Clashttps://muthu.co/wp-content/uploads/2018/07/Snip20180707_108.pngsification report

The general formula for non-negative real β is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (\text{precision} \cdot \text{recall})}{(\beta^2 \cdot \text{precision} + \text{recall})}$$

https://i.stack.imgur.com/swW0x.png

The accuracy form Confusion matrix only take True Positive and True negative in account which is not good to determine the performance of model so to avoid that **F1-score** is used where the False Negatives and False Positives are the important

Score is said to F1 score its because **β²** value is 1 in most of the case,

β values is '1' both False Negative and False Positive are balance
β values is '0.5' both False Positive given more importance than False Negative
β values is '2' both False Negative given more importance than False Positive
This how evaluation of the model is done

Conclusion:

Random forest is a great algorithm for starters who are trying to become a Data science Aspirant.

The Algorithm is a great choice to start as one can develop a model quickly and also it provides a pretty good indicator of the importance which is assigned to the features.

The RF model also has lots of Hyper parameters where we can tune different parameters and build the better ensemble of trees.

Machine Learning

About the Authors

aravind2010

Follow

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Previous Post

8 Thoughts on How to Transition into Data Science from Different Backgrounds

Next Post

16 Key Questions You Should Answer Before Transitioning into Data Science