Random Forest Python

Random Forest Introduction

Random forest is one of the popular algorithms which is used for classification and regression as an ensemble learning. It means random forest includes multiple decision trees. The average of the result of each decision tree would be the final outcome for random forest. There are some drawbacks in decision tree such as over fitting on training set which causes high variance, although it was solved in random forest with the help of Bagging (Bootstrap Aggregating). Now firstly, it is better to pay attention to decision tree algorithm and then study about random forest. Because random forest is divided to multitude decision tree.

Decision Tree

Decision tree uses tree-like graph to take the possible decision by considering all elements of graph. For instance, remember the tennis player who has an agenda to play in different weather conditions. And now, we want to know if the player will play on the 15th day or not?

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No
D15	***Rain***	***High***	***Weak***	***?***

Finding Pure Branch

There are 15 days of which on 9 days he played and on 5 days he did not play; and now we want to know whether he plays in a specific situation or not. You should look at train data carefully, there are different features with different values. We must see if for all of which value of which feature (Play = Yes for all days or Play = No) or in better word, which value is in the same color for the above table.

(Humidity = High □ For all days: (Play = Yes) | (Play = No)) ***No***

(Humidity = Normal □ For all days: (Play = Yes) | (Play = No)) ***No***

(Wind = Weak □ For all days: (Play = Yes) | (Play = No)) ***No***

(Wind = Strong □ For all days: (Play = Yes) | (Play = No)) ***No***

(Outlook = Rain □ For all days: (Play = Yes) | (Play = No)) ***No***

(Outlook = Sunny □ For all days: (Play = Yes) | (Play = No)) ***No***

(Outlook = Overcast □ For all days: (Play = Yes) | (Play = No)) ***Yes*** √

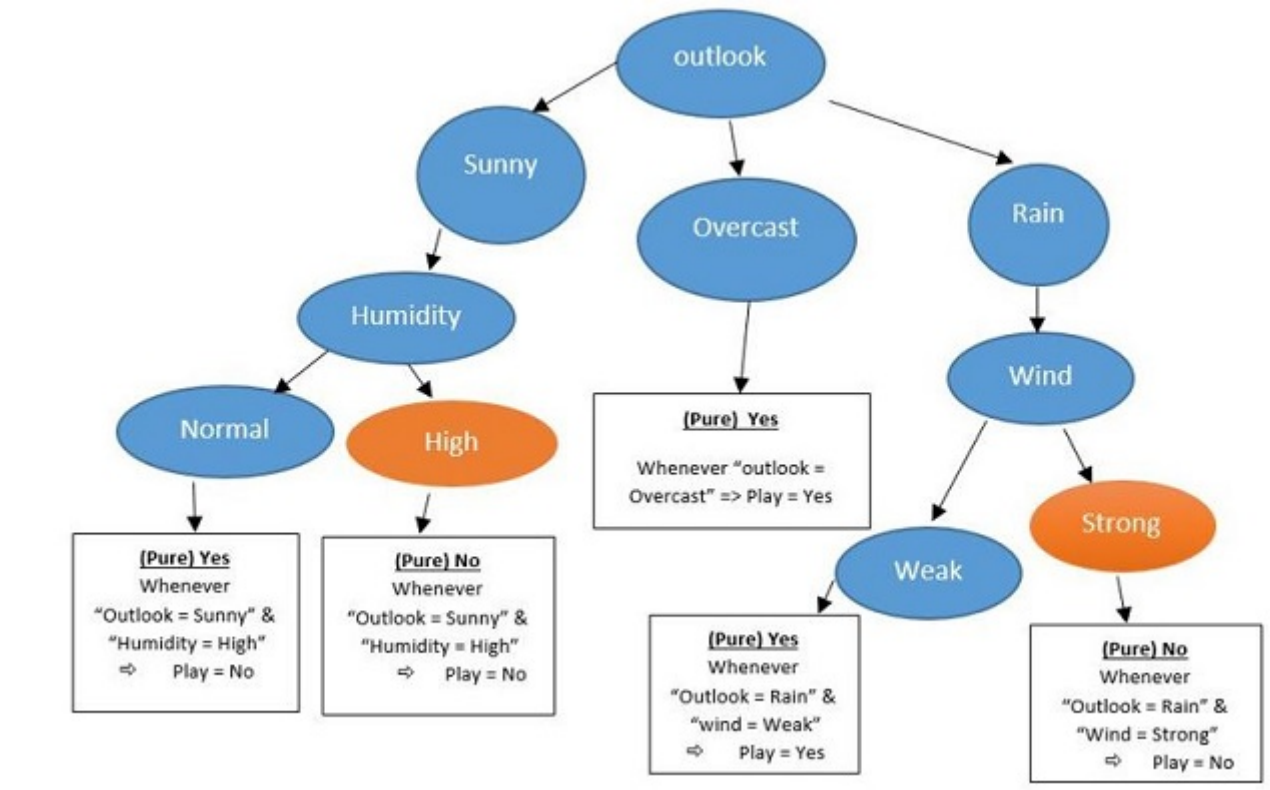
So our player played always when in “outlook =overcast” (D3, D7, D12, D13); we should start from “outlook” feature to make branch hand our root. So the next time, we should care about “Outlook = Sunny” and try to find that which days play situation was complete yes or no, because we want to know if “humidity” or “wind” is the next one. In D1, D2, D8, D9, D11; just in (D1, D2, D8) (Humidity = High) is in the same color or same value which this time (Play = No) Sunny: D1, D2, D8, D9, D11

1. Sunny: D1, D2, D8 & (Humidity = High) □ (Play = No)
2. Sunny: D9, D11 & (Humidity = Normal) □ (Play = Yes)

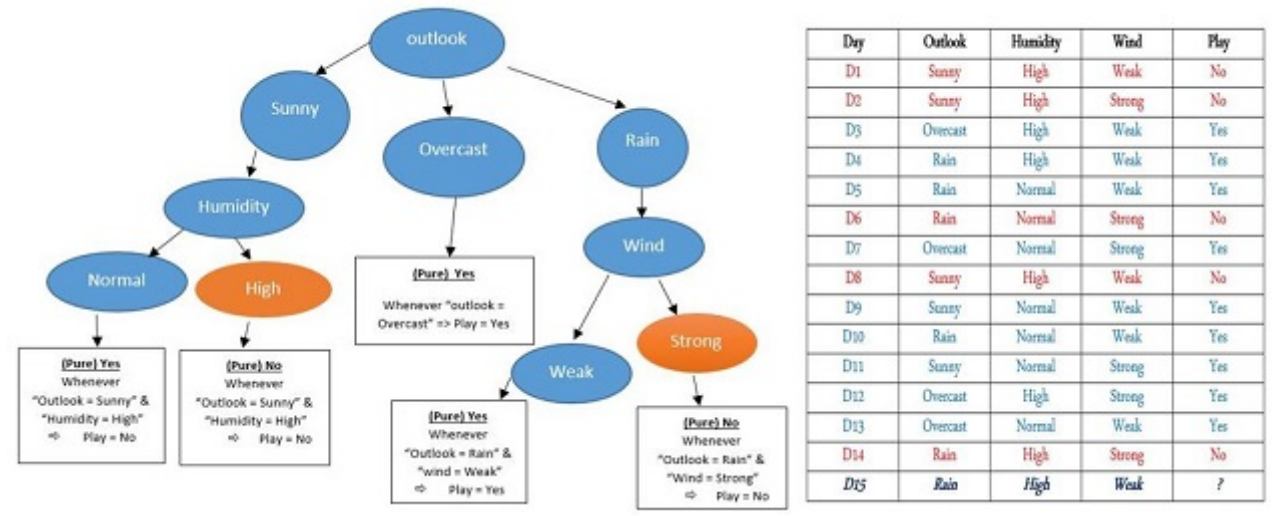
   So our next branch is “Humidity” because of finding “Pure” value. Now, look at when

   “Outlook = Rain” Rain: D4, D5, D6, D10, D14 1. Rain: D4, D5, D10 & (Wind = Weak) □ (Play = Yes)

3. Rain: D6, D14 & (Wind = Strong) □ (Play = no)



Now have a comparison of both table and decision tree in one glance:



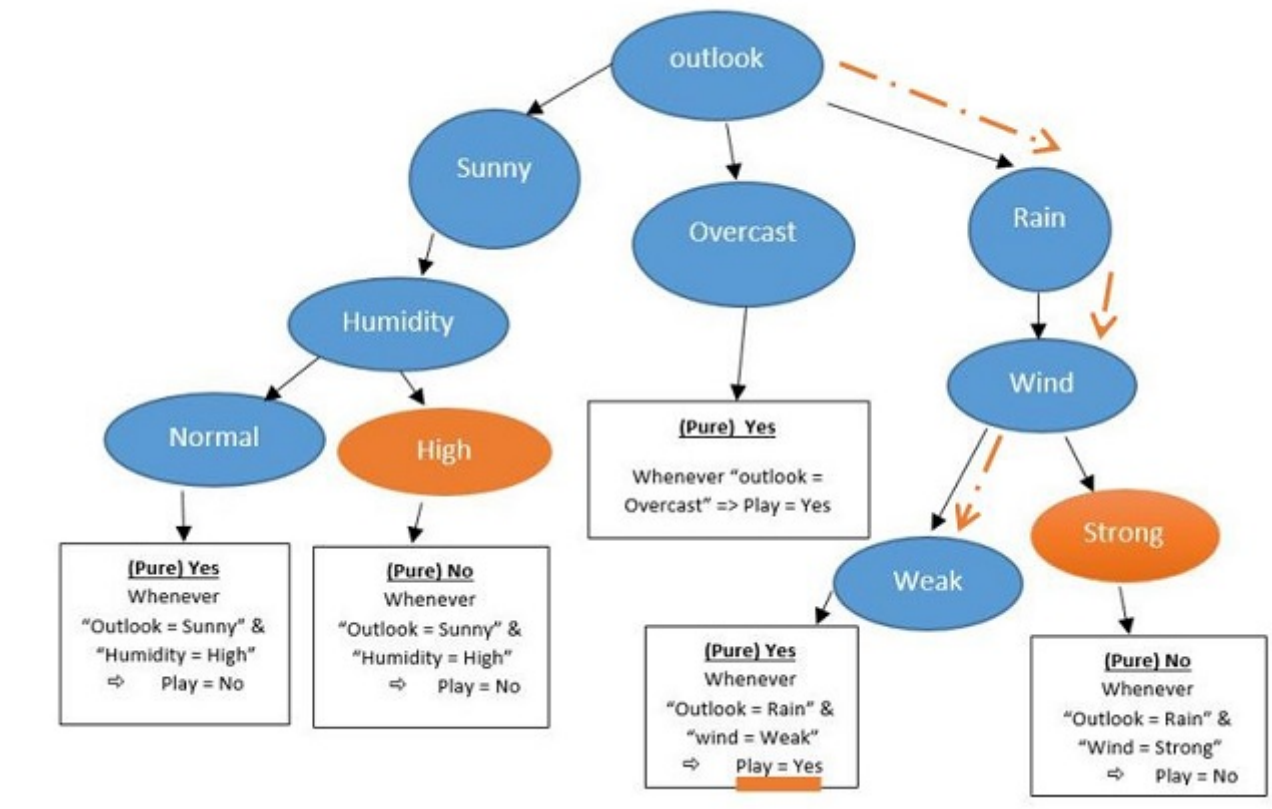
Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No
D15	*Rain*	*High*	*Weak*	*?*

Therefore, for computing the below situation, firstly we consider to "Outlook = Rain", the new go through the right branch and check "Wind = Weak" so we go deep on let side which the answer is "Yes". Hint: Over fitting definition; here in the below graph; as you saw, we have split each node until we achieve "Pure" result. For example, all of "Play" result should be the same "Yes" or "No" in the last "leaf (end node of tree)". If splitting continues until the end of pure set perfectly, then the accuracy will be 100%. It means each leaf in the end has just one specific value. The more splitting, we will have more accuracy and big tree, because tree grew up more and more. If tree size is as same as data set, then we have "over fitting". Over fitting causes algorithm would be too specific to data (train) but it cannot generalize test data. How to avoid or stop over fitting is to prune tree, somehow remove branches which are not suitable and fit on future test data, then look at the result whether removing could hurt or improve our analyzing.

[If ("outlook = Sunny" & "Humidity = Normal" □ all of "Play = Yes")]

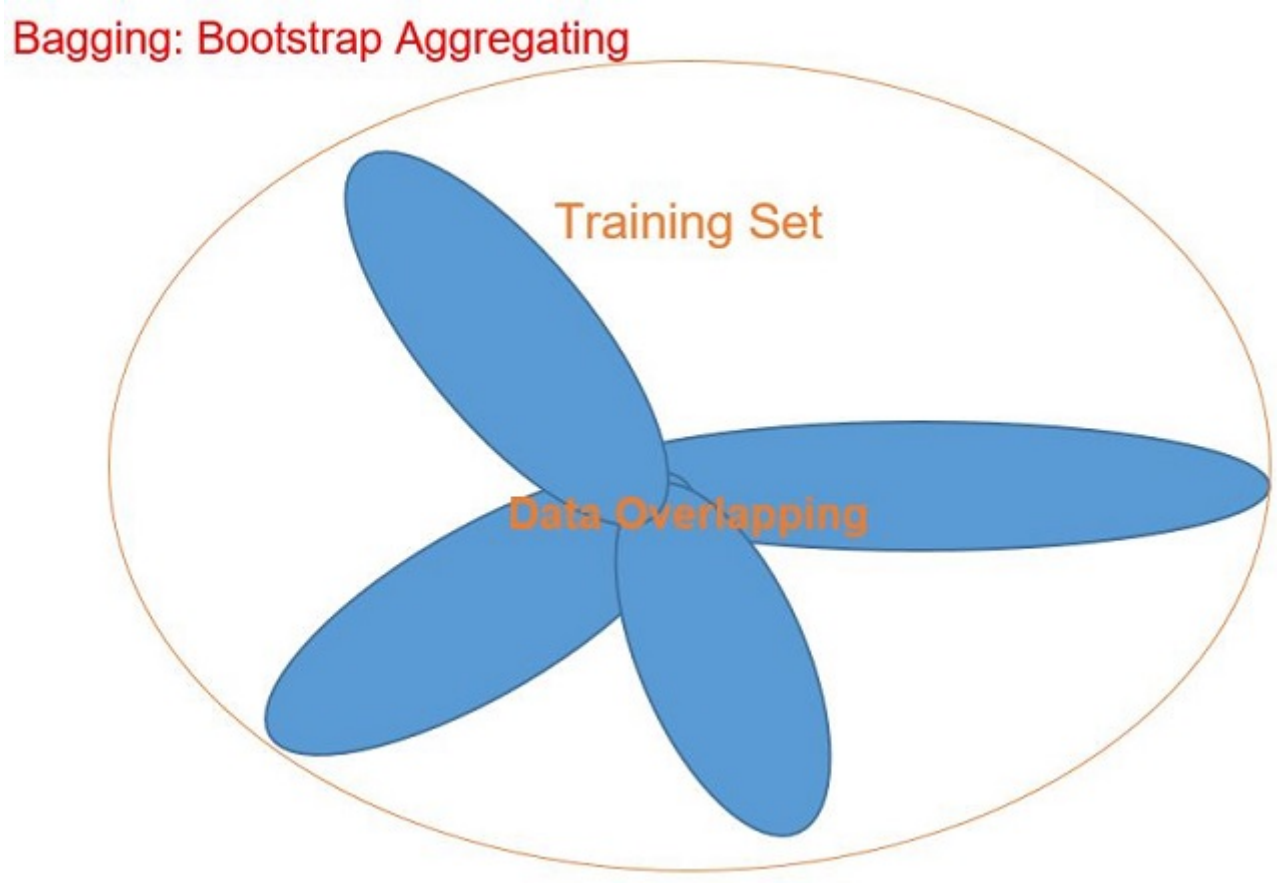
[If ("outlook = Rain" & "wind = weak" □ all of "Play = Yes")]

D15: "Outlook = Rain" & "Humidity=High" & "Wind=Weak"

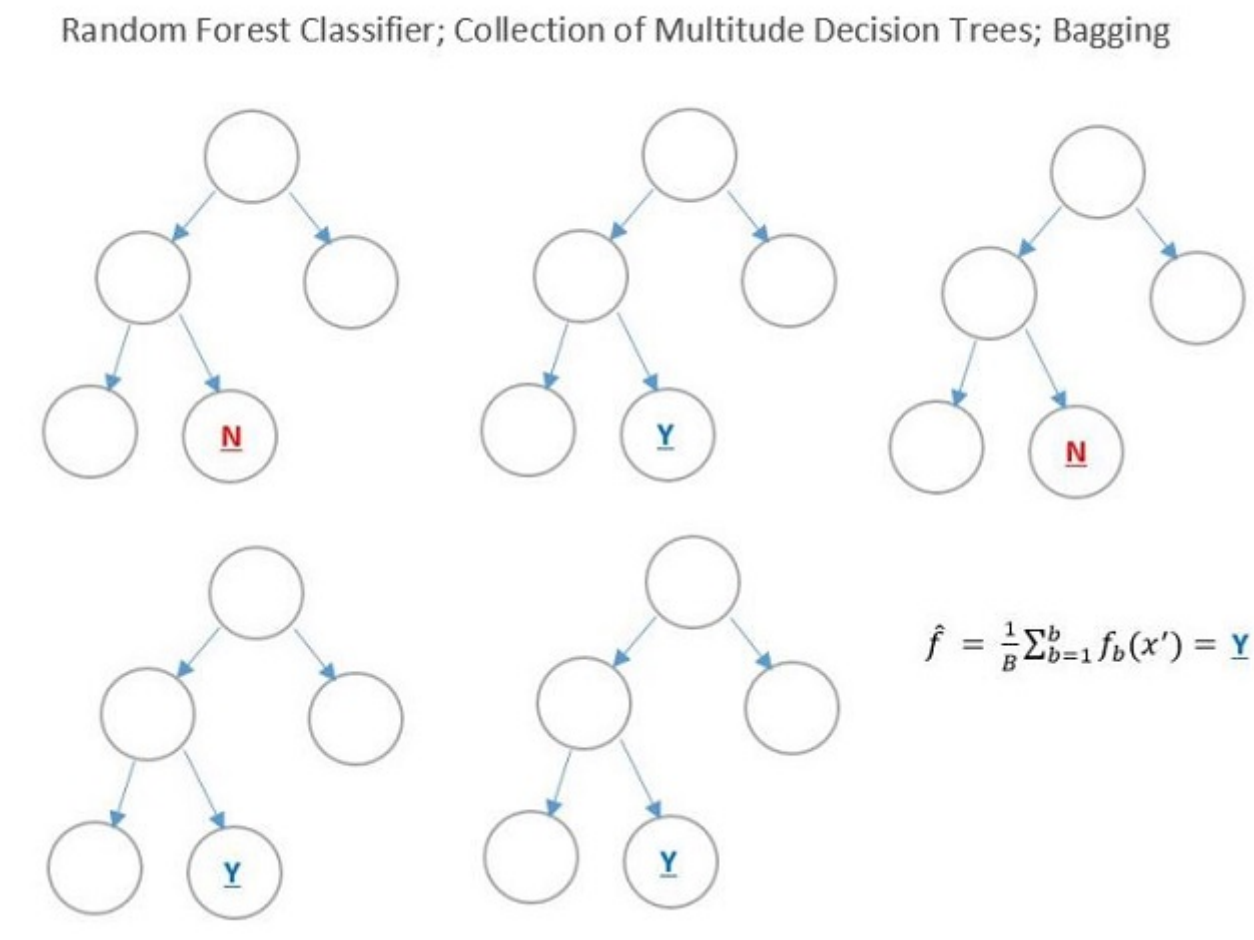


Random Forest

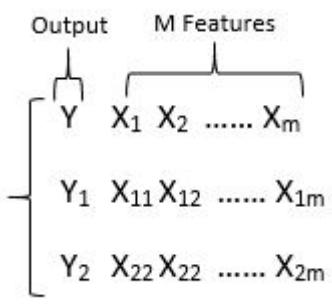
Random forest is an ensemble learning method which is very suitable for supervised learning such as classification and regression. In random forest, we divided train set to smaller part and make each small part as independent tree which its result has no effect on other trees besides them. Random forest gets training set and divided it by "Bagging = Bootstrap Aggregating" which is algorithm to increase accuracy by prevent over fitting and decrease variance. It starts to divide data set to 60% as unique decision tree and 30% as overlapping data.



It divides train set to the “B” different decision tree (which 60% use unique data and 30% use duplicate data), then start to compute result or each decision tree and split them until the appropriate situation (when it is enough for generalization for test data). Below, you see that there are two as “No” answers and three as “Yes” answers, so the average of answers is “Yes”.



Error Computation in Random Forest: There are some solutions to enhance random forest optimization. Cross Validation is to assess how the result of prediction model can generalize to another independent test data. There is train set as blew which has been divided to output as “Y” and features as “X”:



Bootstrapping

Creating T trees randomly; T = { T1 , T2 , T3 , T4 , T5 } with replacement n times for each data. Out of Bag Error: If we consider randomly to specific j = (Xi , Yi) and looking for j in all of trees and find some trees which are without this value, so, they will be out of bag error. Indeed fraction of time when it repeats n times when (Xi , Yi) is not inside trees.

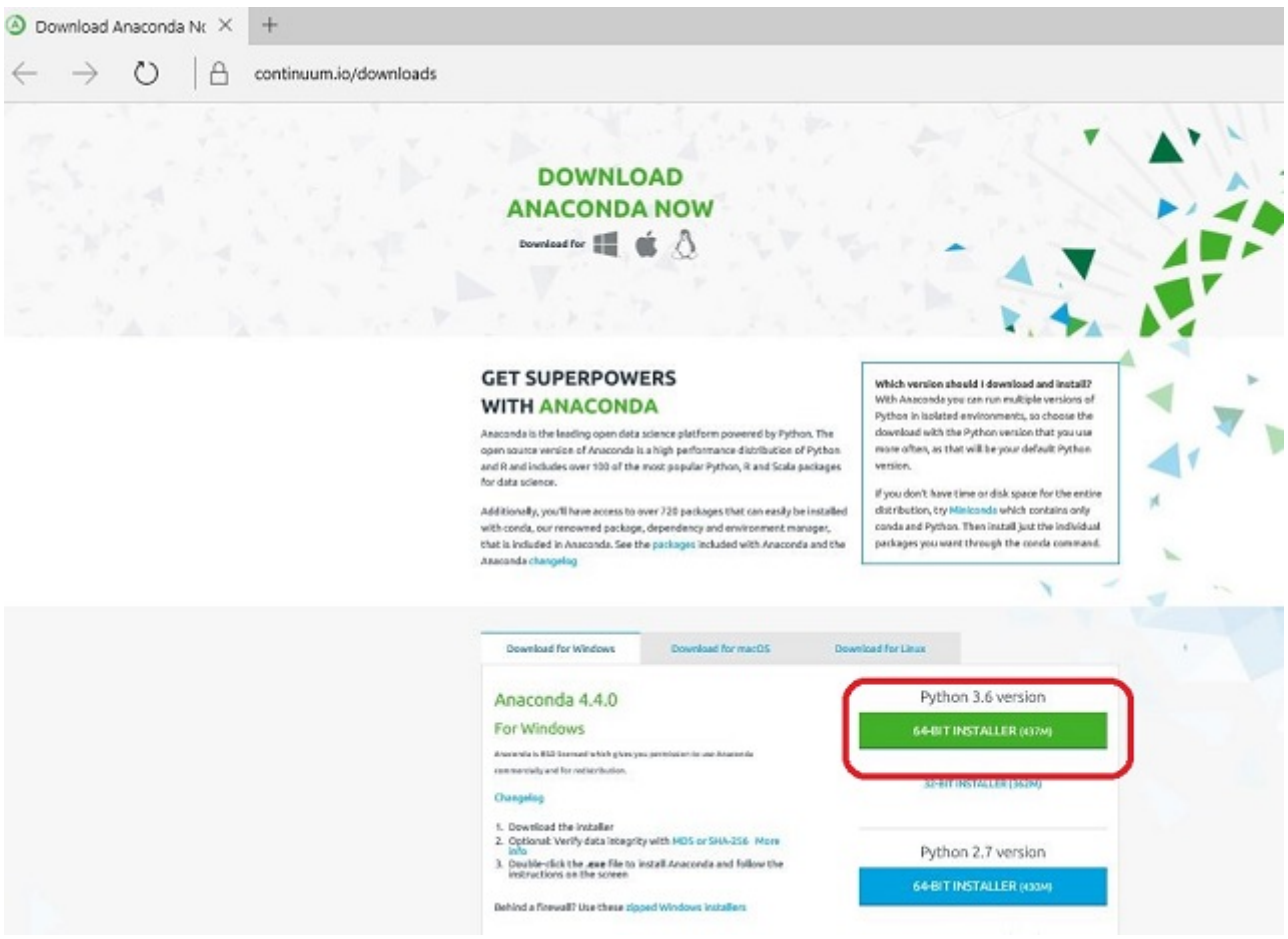
Code

I have prepared data set as train and test set to practice random forest by python. Train set is data which is related to some patients, and in train set we know who will heal based on their age, gender and hospital class. Test set is data for testing and we do not know who heal, we should check it by random forest. I have an empty Excel file as *result.csv* for writing result inside that. I have selected all of their extensions as "*csv*" because it is comfortable to work with python. train test result Random forest is supervised learning. To have more knowledge about it, read this article.

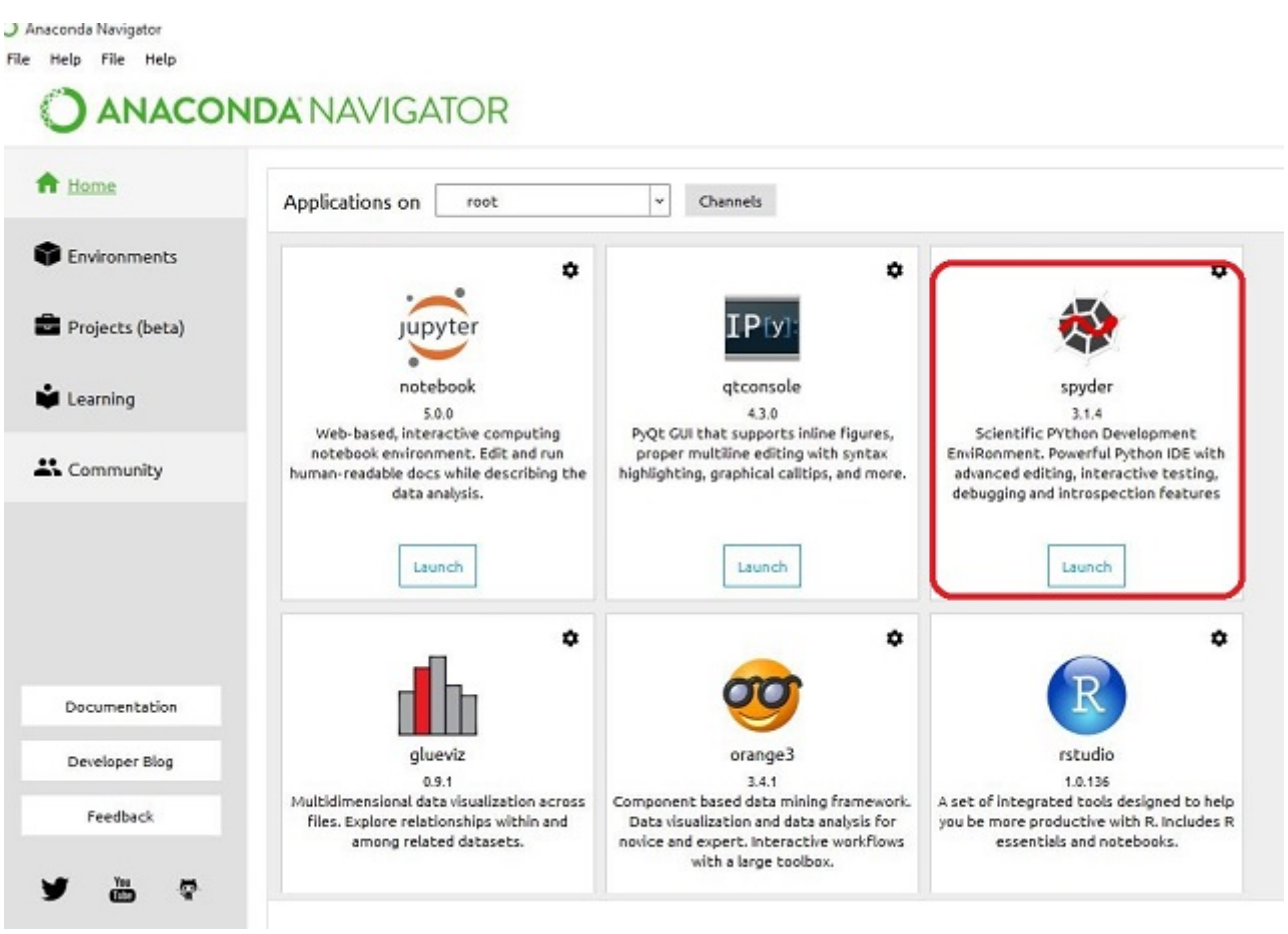
	A	B	C	D	E	F
1	PatientId	healed	Hospitalclass	Name	Sex	Age
2	1	0	3	Alfred, Mr. Abraham	male	22
3	2	1	1	Addison, Mrs. Abramsor	female	38
4	3	1	3	Alma, Miss. Adam	female	26
5	4	1	1	Alvina, Mrs. Albertson	female	35
6	5	0	3	Almond, Mr. Arrington	male	35
7	6	0	3	Alston, Mr. Aston	male	
8	7	0	1	Alvin, Mr. Barton	male	54
9	8	0	3	Arden, Master. Chester	male	2
10	9	1	3	Amanda, Mrs. Norman	female	27
11	10	1	2	Angela, Mrs. Norris	female	14
12	11	1	3	Antonia, Miss. Oliver	female	4
13	12	1	1	Barbara, Miss. Palmer	female	58
14	13	0	3	Benton, Mr. Patterson	male	20
15	14	0	3	Brand, Mr. Peny	male	39
16	15	0	3	Camelia, Miss. Odel	female	14
17	16	1	2	Clara, Mrs. Park	female	55
18	17	0	3	Colvin, Master. Simmons	male	2
19	18	1	2	Cooper, Mr. Philip	male	
20	19	0	3	Darla, Mrs. Spence	female	31
21	20	1	3	Diana, Mrs. Steffen	female	
22	21	0	2	Denver, Mr. Tifft	male	35

Download Python

If you want to fee easy with a comfortable IDE and professional editor, without needing to install libraries. You can use Anaconda & Spider.



Then open Anaconda Navigator from star and select "**Spider**".



There are some points:

1. Python is object oriented

2. Dynamic Typing
3. Rich Libraries
4. Simple to Read
5. Python is case sensitive
6. Indent is important for Python

Python Implementation For Random Forest

Step 1: Import Important Libraries such as numpy, csv for I/O, sklearn

Hide Copy Code

```
import numpy as np
import csv as csv
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.cross_validation import StratifiedKFold # Add important libs
```

Step 2: Train and Test Preparation: Reading their Data and Fill it to Array

Please consider your path and change it according to how you save. For example, if you save train and test in D root, replace it by `path1 = r'D:\train.csv'`.

Hide Copy Code

```
train=[]
test=[]         #Array Definition
path1 =  r'D:\random forest\data set\train.csv'     #Address Definition
path2 =  r'D:\random forest\data set\test.csv'
with open(path1, 'r') as f1:    #Open File as read by 'r'
    reader = csv.reader(f1)
    next(reader, None)          #Skip header because file header is not needed
    for row in reader:          #fill array by file info by for loop
        train.append(row)
    train = np.array(train)

with open(path2, 'r') as f2:
    reader2 = csv.reader(f2)
    next(reader2, None)
    for row2 in  reader2:
        test.append(row2)
    test = np.array(test)

train = np.delete(train,[0],1)  #delete first column of which is patientid
test = np.delete(test,[0],1)
```

Step 3: Delete the First Column which is Related to "PatientID" and There is no Need to Know patientid in Our Algorithm

Hide Copy Code

```
train = np.delete(train,[0],1)  #delete first column of which is patientid
test = np.delete(test,[0],1)
```

Step 4: Refine Data Manually by Replacing Gender to Integer Value

Hide Copy Code

```
train[train[0::,3] == 'male', 3] = 1        #replacement gender with number
train[train[0::,3] == 'female', 3] = 0
test[test[0::,2] == 'male',2] = 1
test[test[0::,2] == 'female',2] = 0
```

Step 5: Refine Data

Because healing patient has a direct correlation to their age and there are some miss values in age column, so by understanding the situation of patient, [how old is he or she?] or [what is his or her title?] or [which kind of hospital class, they have used?]. Therefore, firstly extract their title and then according to those, arrange the missed age value.

Hide Shrink Copy Code

```
for row in  train:
    Title = row[2].split(',')[1].split('.')[0].strip() #Extracting Name in order to gain title
    row[2] = Title

for row in train:       #Fill empty cell o rage column by the below logic
    if (row[4]==''):
        if (row[1]=='1' and row[2]=='Miss' and row[3]=='0'):
            row[4] =30
        if (row[1]=='1' and row[2]=='Mrs' and row[3]=='0'):
            row[4] =40
        if (row[1]=='2' and row[2]=='Miss' and row[3]=='0'):
            row[4] =24
        if (row[1]=='2' and row[2]=='Mrs' and row[3]=='0'):
            row[4] =31.5
        if (row[1]=='3' and row[2]=='Miss' and row[3]=='0'):
            row[4] =18
        if (row[1]=='3' and row[2]=='Mrs' and row[3]=='0'):
            row[4] =31
        if (row[1]=='1' and row[2]=='Master' and row[3]=='1'):
            row[4] =4
        if (row[1]=='1' and row[2]=='Mr' and row[3]=='1'):
            row[4] =40
        if (row[1]=='2' and row[2]=='Master' and row[3]=='1'):
            row[4] =1
```

```
        if (row[1]=='2' and row[2]=='Mr' and row[3]=='1'):
            row[4] =31
        if (row[1]=='3' and row[2]=='Master' and row[3]=='1'):
            row[4] =4
        if (row[1]=='3' and row[2]=='Mr' and row[3]=='1'):
            row[4] =26

for row in  test:
    Title = row[1].split(',')[1].split('.')[0].strip()
    row[1] = Title

for row in test:
    if (row[3]==''):
        if (row[0]=='1' and row[1]=='Miss' and row[2]=='0'):
            row[3] =32
        if (row[0]=='1' and row[1]=='Mrs' and row[2]=='0'):
            row[3] =48
        if (row[0]=='2' and row[1]=='Miss' and row[2]=='0'):
            row[3] =19.5
        if (row[0]=='2' and row[1]=='Mrs' and row[2]=='0'):
            row[3] =29
        if (row[0]=='3' and row[1]=='Miss' and row[2]=='0'):
            row[3] =22
        if (row[0]=='3' and row[1]=='Mrs' and row[2]=='0'):
            row[3] =28
        if (row[0]=='3' and row[1]=='Ms' and row[2]=='0'):
            row[3] =22
        if (row[0]=='1' and row[1]=='Master' and row[2]=='1'):
            row[3] =9.5
        if (row[0]=='1' and row[1]=='Mr' and row[2]=='1'):
            row[3] =42
        if (row[0]=='2' and row[1]=='Master' and row[2]=='1'):
            row[3] =5
        if (row[0]=='2' and row[1]=='Mr' and row[2]=='1'):
            row[3] =28
        if (row[0]=='3' and row[1]=='Master' and row[2]=='1'):
            row[3] =7
        if (row[0]=='3' and row[1]=='Mr' and row[2]=='1'):
            row[3] =25
```

Step 6: Delete Unnecessary Column

Hide Copy Code

```
train = np.delete(train,[2],1) #Delete name column because it is not needed
test = np.delete(test,[1],1)
```

Step 7: Algorithm Optimization by Grid Search

- `max_depth`: It says to RF to how much should be the depth of each decision tree, it gets array of different value, due to hesitation of the best number, I added two values [3,4].
- `n_estimators`: It says to RF to how many decision tree should be created to have better precision.
- `max_features`: In this example of training set, there are four features:

    1. Hospital Class
    2. name
    3. Sex
    4. Age

    When we want to split the node, we select some of these features or all of them, which has an influence on accurate answer. If it is equal to auto, it means that all of the features have been selected.

- `min_samples_split`: It says to RF to how many splitting should we have for each creating new node. `min_samples_leaf`: Leaf is end node which has no child, at the bottom of each decision tree.
- `bootstrap`:

Hide Copy Code

```
parameter_gridsearch = {
                  'max_depth' : [3, 4],  #depth of each decision tree
                  'n_estimators': [50, 20],  #count of decision tree
                  'max_features': ['sqrt', 'auto', 'log2'],
                  'min_samples_split': [2],
                  'min_samples_leaf': [1, 3, 4],
                  'bootstrap': [True, False],
                  }
```

Step 8: Random Forrest Classifier

Hide Copy Code

```
randomforest = RandomForestClassifier()
crossvalidation = StratifiedKFold(train[0::,0] , n_folds=5)

gridsearch = GridSearchCV(randomforest,             #grid search for algorithm optimization
                                scoring='accuracy',
                                param_grid=parameter_gridsearch,
                                cv=crossvalidation)


gridsearch.fit(train[0::,1::], train[0::,0])    #train[0::,0] is as target
model = gridsearch
parameters = gridsearch.best_params_
```

Step 9: Score Computation

Hide Copy Code

```
print('Best Score: {}'.format(gridsearch.best_score_))
Best Score: 0.8571428571428571
```

Step 10: Writing Answer into result.csv File

Hide Copy Code

```
path3 =  r'D:\random forest\data set\result.csv'

output = gridsearch.predict(test)

print(output)
with open(path3, 'w',  newline='') as f3,
    open(path2, 'r') as f4: # write output and other column from test
    forest_Csv = csv.writer(f3)
    forest_Csv.writerow(["PatientId", "healed", "Hospitalclass", "Name", "Sex", "Age"])
    test_file_object = csv.reader(f4)
    next(test_file_object, None)
    i = 0
    for row in  test_file_object:
        row.insert(1,output[i].astype(np.uint8))
        forest_Csv.writerow(row)
        i += 1
```

Conclusion

Nowadays, random forest is very popular for data scientists because of its precision and optimization facilities. It includes multitude decision trees where we decide how many trees are enough for data navigation. RF has solved over fitting problem in decision tree. Over fitting was training train data set as complete as possible which is 100% fit or all off train set, but generalization for all of test is difficult. Random forest uses bagging or bootstrap aggregating to divide train set to different independent decision tree and compute their result without interfering another tree, and in the end average all of result. Finally, its ability as meta estimator for number of decision trees in different of subset of train set in order to enhance accurate prediction and good manager for controlling over fitting.