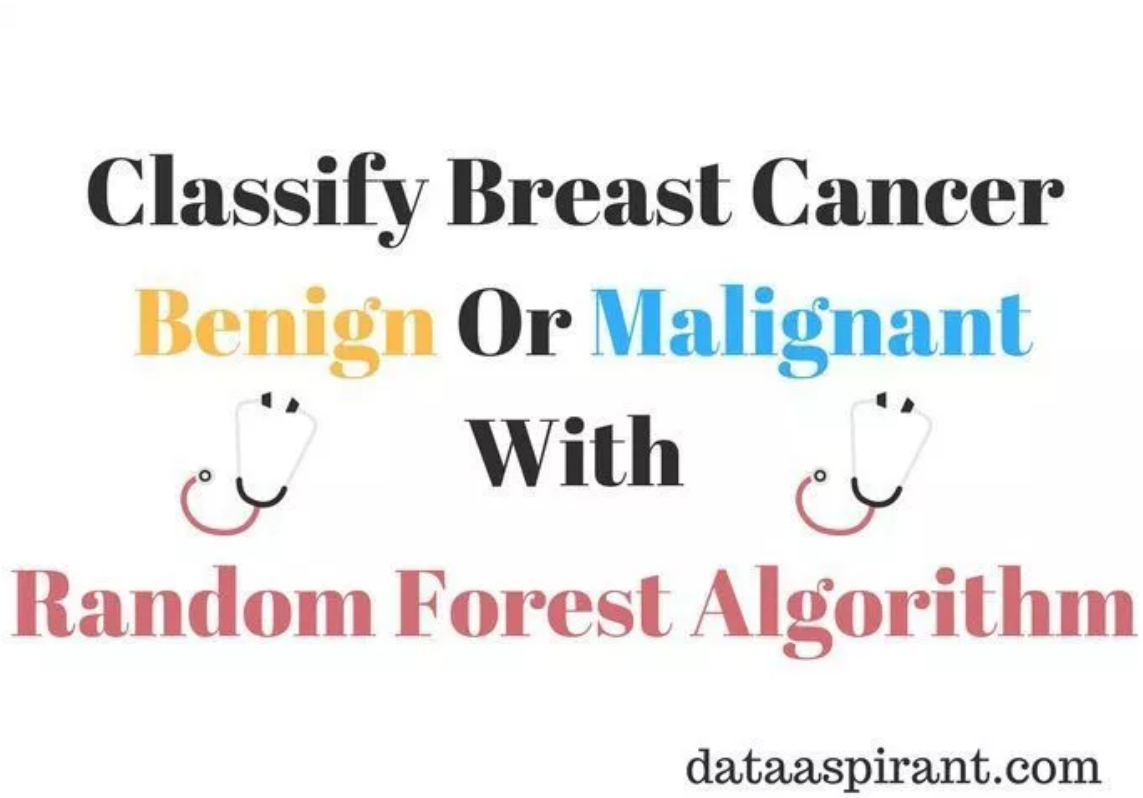
BUILDING RANDOM FOREST CLASSIFIER WITH PYTHON SCIKIT LEARN

June 26, 2017 Saimadhu Polamuri 17 Comments Machine Learning



Random Forest Algorithm in Python

Building Random Forest Algorithm in Python

In the Introductory article about random forest algorithm, we addressed how the random forest algorithm works with real life examples. As continues to that, In this article we are going to build the **random forest algorithm** in python with the help of one of the best Python machine learning library **Scikit-Learn**.

To build the random forest algorithm we are going to use the Breast Cancer dataset. To summarize in this article we are going to build a random forest classifier to predict the Breast cancer type (**Benign or Malignant**).

> Before we begin. Let's quickly look at the **table of contents**.

Table of contents:

- Decision trees and random forest algorithm guide.
- Ensemble machine learning: Random forest and Adaboost.
- Machine learning classification concepts for beginners.
- Applying machine learning classification techniques case studies.

Building Random Forest Algorithm in Python

Overview of Random forest algorithm

Random forest algorithm is an **ensemble classification** algorithm. Ensemble classifier means a group of classifiers. Instead of using only one classifier to predict the target, In ensemble, we use multiple classifiers to predict the target.

In case, of random forest, these ensemble classifiers are the randomly created decision trees. Each decision tree is a single classifier and the target prediction is based on the **majority voting** method.

The majority voting concept is same as the political votings. Each person **votes** per one political party out all the political parties participating in elections. In the same way, **every classifier will votes** to one target class out of all the target classes.

To declare the election results. The votes will calculate and the party which got the **most number of votes** treated as the election winner. In the same way, the target class which got the **most number of votes** considered as the final predicted target class.

Before we go further it's better to spend some time on the below articles to understand how the random forest algorithm works.

- How the random forest algorithm works in machine learning
- Decision tree algorithm concepts for beginners
- Building decision tree classifier in Python

I hope you have a clear understanding of how the random forest algorithm works. Now let's implement the same. As I said earlier, we are going to use the breast cancer dataset to implement the random forest.

Before we begin let's look at some stats and the **impact of breast cancer** in present generation.

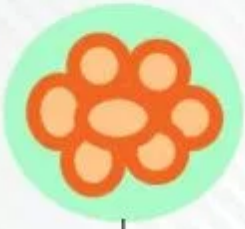
About Breast Cancer

Sadly breast cancer is to **second most** death reason for women's. In the US during the year 2016, almost **246,660** women's breast cancer cases are diagnosed. The myth people believe tumor as cancer but which is not true.

Only the **continuously growing tumor** causes death. Based on this properties the tumors are mainly of 2 kinds.

- Benign Tumor
- Malignant Tumor

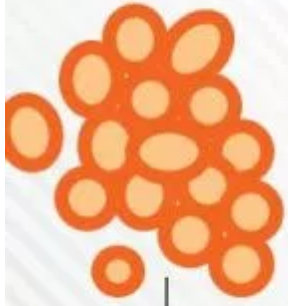
Learn the difference and what you can do to prevent them!



BENIGN (not cancer) tumor cells grow only locally and cannot spread by invasion or metastasis.

They are often surrounded by a protective "sac" – a mechanism performed by your immune system – that segregates it from the rest of your body and enables it to be easily removed.

- Capsulated
- Slow growing
- Non-invasive
- Non-Cancerous



MALIGNANT (cancer) cells invade neighboring tissues, enter blood vessels, and metastasize to different sites.

They may not have symptoms initially and the first indication that something isn't right may be the detection of a painless lump.

- Cancerous
- Invasive & Infiltrate
- Fast growing
- Non-capsulated

malignant benign tumor difference
Image Credit:: thetruthaboutcancer.com

Benign:

A benign tumor is not a cancerous tumor. Which means it's not able to spread through the body like the cancerous tumors. The benign is serious when it's growing in sensitive places. This kind of tumors are will well terminated with proper treatment and with the change in diet habits.

Malignant

The malignant tumor is the cancerous tumor which causes death. These tumors can grow so fast and spread over various parts of the body.

A good read about these tumor and health prevention can be found in the thetruthaboutcancer article.

UCI breast cancer dataset description

We are using the UCI breast cancer dataset to build the random forest classifier in Python. You can download the data from UCI or You can download the code from Dataaspirant Github.

This breast cancer dataset is the most popular classification dataset. Which is having **10 features** and **1 target class.**

Breast Cancer dataset features:

- Sample code number:
  - id number
- Clump Thickness:
  - The values are in the range of 1 – 10
- Uniformity of Cell Size:
  - The values are in the range of 1 – 10
- Uniformity of Cell Shape:
  - The values are in the range of 1 – 10
- Marginal Adhesion:
  - The values are in the range of 1 – 10
- Single Epithelial Cell Size:
  - The values are in the range of 1 – 10
- Bare Nuclei:
  - The values are in the range of 1 – 10
- Bland Chromatin:
  - The values are in the range of 1 – 10
- Normal Nucleoli:
  - The values are in the range of 1 – 10
- Mitoses:
  - The values are in the range of 1 – 10

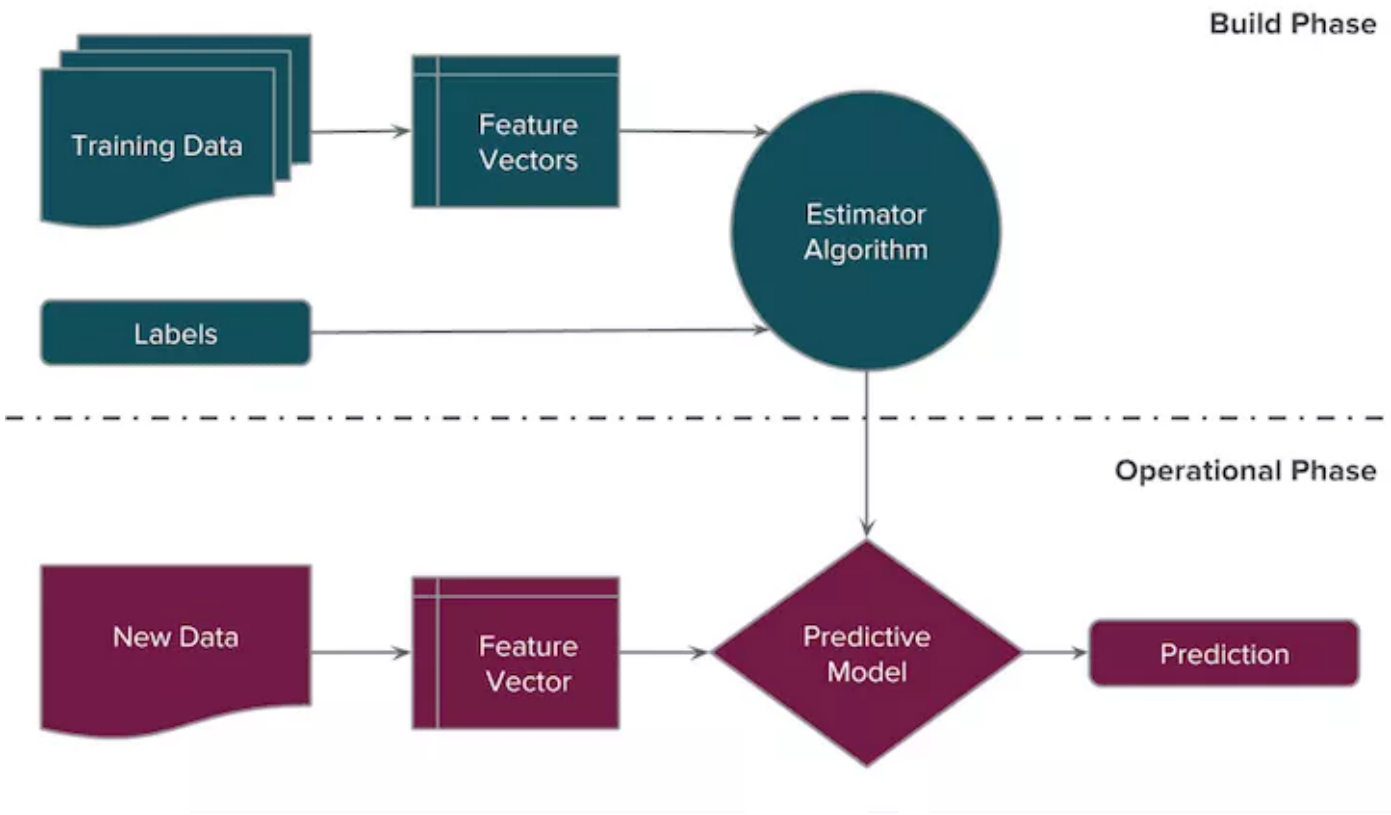
Breast Cancer dataset Target:

The target class having two target classes

- Bening
  - The value will be 2
- Malignant
  - The value will be 4

This dataset also having **missing values**. In the coding section of this article, we are to going deal with the missing values before we model the random forest algorithm.

Machine learning workflow



machine learning workflow

Implementing random forest algorithm in Python

To implement the random forest algorithm we are going follow the below **two phase** with step by step workflow.

- **Build Phase**
  - Creating dataset
  - Handling missing values

    - Splitting data into train and test datasets
    - Training random forest classifier with Python scikit learn
- **Operational Phase**
    - Perform predictions
    - Accuracy calculations
        - Train Accuracy
        - Test Accuracy

Let's begin the journey of building the random forest classifier with importing the required Python machine learning packages.

Import required Python machine learning packages

```
# Required Python Packages
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

The above python machine learning packages we are going to use to build the random forest classifier. Let's talk about the need for these packages in random forest classifier implementation.

- **Pandas:**
    - Pandas package is the best choice for tabular data analysis.
    - All the data manipulation tasks in this article are going to use the Pandas methods.
- **train_test_split:**
    - We imported scikit-learn **train_test_split** method to split the breast cancer dataset into test and train dataset.
    - Train dataset will be used in the training phase and the test dataset will be used in the validation phase.
- **RandomForestClassifier:**

    - We imported scikit-learn **RandomForestClassifier** method to model the training dataset with random forest classifier.
    - Later the modeled random forest classifier used to perform the predictions.
- **accuracy_score:**
    - We imported scikit-learn **accuracy_score** method to calculate the accuracy of the trained classifier.
- **confusion_matrix:**
    - We imported scikit-learn **confusion_matrix** to understand the trained classifier behavior over the test dataset or validate dataset.

Copy the above code in any text file (or you favorite txt editor) and save the file with the python extension (**.py**). Let say **random_forest.py**

Then call the random_forest.py file from the terminal using the below command.

```
python random_forest.py
```

If you install the python machine learning packages properly, you won't face any issues. Even though you install the packages properly and you facing the issue **ImportError: No module named model_selection**. This means the scikit learn package you are using not updated to the new version.

I hope you are using scikit learn **0.17 or lesser version**. You can copy and paste the below code to know your scikit learn version.

```
import sklearn
print (sklearn.__version__)
```

If the version your are using is **0.17** or lesser than that, you need to update your scikit learn version to **0.18**

You can use the below commands to update your scikit learn to the new version (0.18)

Using Pip

```
pip install -U scikit-learn
```

Using Anaconda

```
conda install scikit-learn=0.18
```

Once you upgraded your scikit-learn package. Run the above code and you won't face any issues. If you still face any issue to run the above code do please let me know in the **comments section.**

Now let's create the dataset to model the random forest classifier.

Creating dataset

The downloaded dataset is in the **data** format. So we are going to convert into the **CSV** format. To do that we are going to write a simple function which first loads the **data** format into the pandas dataframe and later the loaded dataframe will save into the **CSV** file format.

```
# File Paths
INPUT_PATH = "../inputs/breast-cancer-wisconsin.data"
OUTPUT_PATH = "../inputs/breast-cancer-wisconsin.csv"


def data_file_to_csv():
    """

    :return:
    """

    # Headers
    headers = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",
               "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses",
               "CancerType"]
    # Load the dataset into Pandas data frame
    dataset = read_data(INPUT_PATH)
```

```
    # Add the headers to the loaded dataset
    dataset = add_headers(dataset, headers)
    # Save the loaded dataset into csv format
    dataset.to_csv(OUTPUT_PATH, index=False)
    print "File saved ...!"
```

The **INPUT_PATH** is having the path for the downloaded data format file and the **OUTPUT_PATH** is having the output where the CSV format file is going to save.

Using the pandas **read_csv** method we loaded the data format file into pandas dataframe.

The loaded dataset doesn't have the header names. So we need to add the header names to the loaded dataframe. To do the same we have written a function with takes the **dataset** and **header names** as input and add the header names to the dataset.

```
def add_headers(dataset, headers):
    """
    Add the headers to the dataset
    :param dataset:
    :param headers:
    :return:
    """
    dataset.columns = headers
    return dataset
```

After adding the header names to dataset we are saving the dataset into CSV format. While saving the file we parameterized the **index=False**. When we save the loaded dataframe without this the saved file will have an extra column with the indexes. So to eliminate this we are parameterized the **index=False.**

Now we are ready with the dataset. The next biggest thing is the **preprocessing** the data.

Sometimes, if it's our day, we don't need to do much work on the preprocessing stage. But, if not.

> **We need to spend a lot of time in the preprocessing stage.**

Handling of missing values once such task in preprocessing the data.

Handling missing values

The process of handling missing values will differ from dataset to dataset. For the cancer dataset, we are using simple tasks to handle the missing values in the loaded dataset.

Before reviling what those missing values, I want to show the ways to identify the missing values. So when you are working on a different dataset. You can identify the missing values by yourself.

The best idea to start with is, calculating basic statistics for each column (features and target) of the dataset. You may be wondering what the use of calculating basic statistics of the dataset and how it gonna helps to find the missing values.

Yes, finding the basic statistics will helps us to find the missing values in the dataset. The idea is we can use pandas describe method on the loaded dataset to calculate the basic statistics. This outputs the stats about only the columns which are not having any missing values or categorical values.

Feeling missed somewhere, No issue let's implement a function to calculate the basic statistics then you get the clear idea of what I talking about.

```
def dataset_statistics(dataset):
    """
    Basic statistics of the dataset
    :param dataset: Pandas dataframe
    :return: None, print the basic statistics of the dataset
    """
    print dataset.describe()
```

As I said before, We are using pandas **describe** method to get the basic statistics of the dataset.

Now let's call this function and check the what it's outputting.

```
def main():
    """
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)

if __name__ == "__main__":
    main()
```

Script Output

```
CodeNumber  ClumpThickness  UniformityCellSize  UniformityCellShape
count  6.980000e+02      698.000000          698.000000           698.000000
mean   1.071807e+06        4.416905            3.137536             3.210602
std    6.175323e+05        2.817673            3.052575             2.972867
min    6.163400e+04        1.000000            1.000000             1.000000
25%    8.702582e+05        2.000000            1.000000             1.000000
50%    1.171710e+06        4.000000            1.000000             1.000000
75%    1.238354e+06        6.000000            5.000000             5.000000
max    1.345435e+07       10.000000           10.000000            10.000000

       MarginalAdhesion  SingleEpithelialCellSize  BlandChromatin
count        698.000000                698.000000      698.000000
mean           2.809456                  3.217765        3.438395
std            2.856606                  2.215408        2.440056
min            1.000000                  1.000000        1.000000
25%            1.000000                  2.000000        2.000000
50%            1.000000                  2.000000        3.000000
75%            4.000000                  4.000000        5.000000
max           10.000000                 10.000000       10.000000

       NormalNucleoli     Mitoses  CancerType
count      698.000000  698.000000  698.000000
mean         2.869628    1.590258    2.690544
```

```
std          3.055004    1.716162    0.951596
min          1.000000    1.000000    2.000000
25%          1.000000    1.000000    2.000000
50%          1.000000    1.000000    2.000000
75%          4.000000    1.000000    4.000000
max         10.000000   10.000000    4.000000
```

If you observe the above statistics clearly you can identify that we are missing one column details. The column header we are missing is **BareNuclei**.

Now open the CSV file and check the column details of Bare Nuclei. You will find the missing values replaced with ? Which means we need to treat those values as missing values.

As we know the missing values column and the character used to represent the missing values. Now let's write a simple function will take the dataset, header_name and missing value representing the character as input handles the missing values.

```
def handel_missing_values(dataset, missing_values_header, missing_label):
    """
    Filter missing values from the dataset
    :param dataset:
    :param missing_values_header:
    :param missing_label:
    :return:
    """

    return dataset[dataset[missing_values_header] != missing_label]
```

Now let's call this function inside the main function

```
def main():
    """
    Main function
    :return:
    """
    # Headers
    headers = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",
```

```
                "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses",
                "CancerType"]
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
if __name__ == "__main__":
    main()
```

Now our dataset is **missing values free**. Now let's split the data into train and test dataset. The training dataset will use to train the random forest classifier and the test dataset used the validate the model random forest classifier.

Split data into train and test datasets

To split the data into train and test dataset, Let's write a function which takes the dataset, train percentage, feature header names and target header name as inputs and returns the **train_x, test_x, train_y and test_y** as outputs.

```
def split_dataset(dataset, train_percentage, feature_headers, target_header):
    """
    Split the dataset with train_percentage
    :param dataset:
    :param train_percentage:
    :param feature_headers:
    :param target_header:
    :return: train_x, test_x, train_y, test_y
    """

    # Split dataset into train and test dataset
    train_x, test_x, train_y, test_y = train_test_split(dataset[feature_headers], dataset[target_header],
                                                        train_size=train_percentage)
    return train_x, test_x, train_y, test_y
```

Now let's call the above function inside the **main function.**

```
def main():
    """
```

```
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])


if __name__ == "__main__":
    main()
```

Where the **HEADERS[1:-1]** contains all the features header names. We eliminated the first and last header names. The first header name is **id** and the last header the **target** header. The **HEADERS[-1]** contains the target header name.

We can print the shape of the **train_x, test_x, train_y and test_y** to check whether the split proper or not.

```
def main():
    """
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Train and Test dataset size details
    print "Train_x Shape :: ", train_x.shape
    print "Train_y Shape :: ", train_y.shape
    print "Test_x Shape :: ", test_x.shape
    print "Test_y Shape :: ", test_y.shape

if __name__ == "__main__":
```

```
22      main()
```

Script output:

```
1 Train_x Shape ::  (477, 9)
2 Train_y Shape ::  (477,)
3 Test_x Shape ::  (205, 9)
4 Test_y Shape ::  (205,)
```

From the above result, it's clear that the train and test split was proper. Now let's build the random forest classifier using the **train_x and train_y** datasets.

Training random forest classifier with scikit learn

To train the random forest classifier we are going to use the below **random_forest_classifier** function. Which requires the features **(train_x)** and target **(train_y)** data as inputs and returns the train random forest classifier as output.

```
1  def random_forest_classifier(features, target):
2      """
3      To train the random forest classifier with features and target data
4      :param features:
5      :param target:
6      :return: trained random forest classifier
7      """
8      clf = RandomForestClassifier()
9      clf.fit(features, target)
10     return clf
```

Now let's call the above function inside the **main function** and print the trained classifier.

```
1  def main():
2      """
3      Main function
4      :return:
5      """
```

```
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    print "Trained model :: ", trained_model
if __name__ == "__main__":
    main()
```

Script Output:

```
Trained model ::  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

Perform predictions

As we model the classifier. Now let's take few observations from test dataset and print what our model is predicting and what the actual target.

To do that first let's predict target for all the test features **(test_x)** using the trained classifier. Later we will see what our trained model is predicting and what the actual output could be.

```
def main():
    """
    Main function
```

```
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    print "Trained model :: ", trained_model
    predictions = trained_model.predict(test_x)

    for i in xrange(0, 5):
        print "Actual outcome :: {} and Predicted outcome :: {}".format(list(test_y)[i], predictions[i])
if __name__ == "__main__":
    main()
```

First I converted the test_y into list object from pandas dataframe. The reason is as we randomly split the train and test dataset the **indexes** of the test_y won't be **in order**. If we convert the dataframe in to list object the indexes will be in order.

From the above code, we are printing the first 5 values of test_y and the predict results target.

```
Actual outcome :: 2 and Predicted outcome :: 2
Actual outcome :: 2 and Predicted outcome :: 2
Actual outcome :: 2 and Predicted outcome :: 2
Actual outcome :: 2 and Predicted outcome :: 2
Actual outcome :: 4 and Predicted outcome :: 4
```

Seems like the trained classifier predicted the first 5 target classes correctly. To know more about the model let's check the train and test accuracy information.

Accuracy calculations

```
def main():
```

```
    """
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    print "Trained model :: ", trained_model
    predictions = trained_model.predict(test_x)

    # Train and Test Accuracy
    print "Train Accuracy :: ", accuracy_score(train_y, trained_model.predict(train_x))
    print "Test Accuracy  :: ", accuracy_score(test_y, predictions)

if __name__ == "__main__":
    main()
```

To calculate the accuracy we are using scikit learn the **accuracy_score** method.

Script output:

```
Train Accuracy ::  0.991614255765
Test Accuracy  ::  0.970731707317
```

Our trained classifier model giving **99%** accuracy for train dataset and **97%** accuracy for test dataset.

Confusion matrix

To know the ture_postive and true_negative details. Let's print the confusion matrix of our trained classifier.

```
def main():
    """
    Main function
```

```
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    predictions = trained_model.predict(test_x)

    print "Train Accuracy :: ", accuracy_score(train_y, trained_model.predict(train_x))
    print "Test Accuracy  :: ", accuracy_score(test_y, predictions)
    print " Confusion matrix ", confusion_matrix(test_y, predictions)


if __name__ == "__main__":
    main()
```

Script output:

```
Train Accuracy ::  0.991614255765
Test Accuracy  ::  0.970731707317
 Confusion matrix  [[123   5]
 [  1  76]]
```

You can get the complete code below. If you know GitHub, you can clone the complete code from our Github account.

Complete Code:

```
# Required Python Packages
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
7
8  import pdb
9
10 # File Paths
11 INPUT_PATH = "../inputs/breast-cancer-wisconsin.data"
12 OUTPUT_PATH = "../inputs/breast-cancer-wisconsin.csv"
13
14 # Headers
15 HEADERS = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",
16            "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses", "CancerType"]
17
18
19 def read_data(path):
20     """
21     Read the data into pandas dataframe
22     :param path:
23     :return:
24     """
25     data = pd.read_csv(path)
26     return data
27
28
29 def get_headers(dataset):
30     """
31     dataset headers
32     :param dataset:
33     :return:
34     """
35     return dataset.columns.values
36
37
38 def add_headers(dataset, headers):
39     """
40     Add the headers to the dataset
41     :param dataset:
42     :param headers:
43     :return:
44     """
45     dataset.columns = headers
46     return dataset
47
48
49 def data_file_to_csv():
50     """
51
```

```
    :return:
    """

    # Headers
    headers = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",
               "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses",
               "CancerType"]
    # Load the dataset into Pandas data frame
    dataset = read_data(INPUT_PATH)
    # Add the headers to the loaded dataset
    dataset = add_headers(dataset, headers)
    # Save the loaded dataset into csv format
    dataset.to_csv(OUTPUT_PATH, index=False)
    print "File saved ...!"


def split_dataset(dataset, train_percentage, feature_headers, target_header):
    """
    Split the dataset with train_percentage
    :param dataset:
    :param train_percentage:
    :param feature_headers:
    :param target_header:
    :return: train_x, test_x, train_y, test_y
    """

    # Split dataset into train and test dataset
    train_x, test_x, train_y, test_y = train_test_split(dataset[feature_headers], dataset[target_header],
                                                        train_size=train_percentage)
    return train_x, test_x, train_y, test_y


def handel_missing_values(dataset, missing_values_header, missing_label):
    """
    Filter missing values from the dataset
    :param dataset:
    :param missing_values_header:
    :param missing_label:
    :return:
    """

    return dataset[dataset[missing_values_header] != missing_label]


def random_forest_classifier(features, target):
```

```
    """
    To train the random forest classifier with features and target data
    :param features:
    :param target:
    :return: trained random forest classifier
    """
    clf = RandomForestClassifier()
    clf.fit(features, target)
    return clf


def dataset_statistics(dataset):
    """
    Basic statistics of the dataset
    :param dataset: Pandas dataframe
    :return: None, print the basic statistics of the dataset
    """
    print dataset.describe()


def main():
    """
    Main function
    :return:
    """
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(OUTPUT_PATH)
    # Get basic statistics of the loaded dataset
    dataset_statistics(dataset)

    # Filter missing values
    dataset = handel_missing_values(dataset, HEADERS[6], '?')
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Train and Test dataset size details
    print "Train_x Shape :: ", train_x.shape
    print "Train_y Shape :: ", train_y.shape
    print "Test_x Shape :: ", test_x.shape
    print "Test_y Shape :: ", test_y.shape

    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)
    print "Trained model :: ", trained_model
    predictions = trained_model.predict(test_x)

```

```
    for i in xrange(0, 5):
        print "Actual outcome :: {} and Predicted outcome :: {}".format(list(test_y)[i], predictions[i])

    print "Train Accuracy :: ", accuracy_score(train_y, trained_model.predict(train_x))
    print "Test Accuracy  :: ", accuracy_score(test_y, predictions)
    print " Confusion matrix ", confusion_matrix(test_y, predictions)


if __name__ == "__main__":
    main()
```

Summary

In this article, you learned how to implement the most popular classification algorithm random forest in python using python scikit learn package.

On process, you learned how to handle the missing values. Finally, you learned how to calculate the accuracy of any trained classifier using the scikit learn **accuray_score** method.

Follow us:

FACEBOOK| QUORA |TWITTER| GOOGLE+ | LINKEDIN| REDDIT | FLIPBOARD | MEDIUM | GITHUB

I hope you like this post. If you have any questions, then feel free to comment below. If you want me to write on one particular topic, then do tell it to me in the comments below.

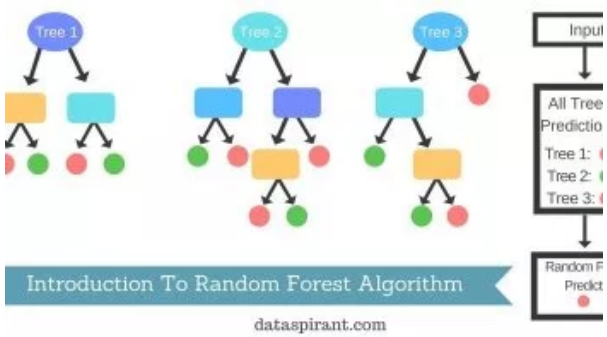
Recommended Data Science Courses

- Decision trees and random forest algorithm guide.
- Ensemble machine learning: Random forest and Adaboost.

- Machine learning classification concepts for beginners.
- Applying machine learning classification techniques case studies.

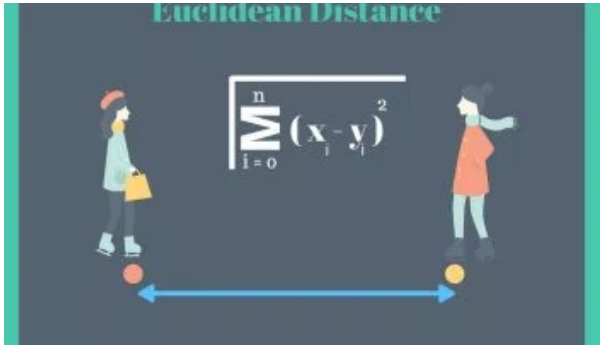
Share this:

Related



How the random forest algorithm works in machine learning
May 22, 2017
In "Machine Learning"

visualize decision tree in python with graphviz
April 21, 2017
In "Machine Learning"

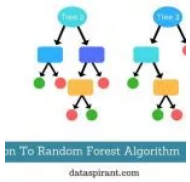


Knn sklearn, K-Nearest Neighbor implementation with scikit learn
December 30, 2016
In "Data Science"

classification algorithms | python | random forest

Previous Post
How the random forest algorithm works in machine learning

Next Post
How to perform the principal component analysis in R
Machine Learning



Machine Learning

Search ... Search

AWARDED TOP 75 DATA SCIENCE BLOG

FOLLOW AUTHOR

Dataaspirant awarded top 75 data science blog

BUILD YOUR CAREER IN AI WITH ANDREW NG DEEP LEARNING COURSES

Andrew ng Deep learning courses

MOST POPULAR POSTS

How Decision Tree Algorithm works

Building Decision Tree Algorithm in Python with scikit learn

Difference Between Softmax Function and Sigmoid Function

How the random forest algorithm works in machine learning

Five most popular similarity measures implementation in python

Support Vector Machine Classifier Implementation in R with caret package

Building Random Forest Classifier with Python Scikit learn

visualize decision tree in python with graphviz

How the Naive Bayes Classifier works in Machine Learning

Support vector machine (Svm classifier) implemenation in python with Scikit-learn

dataaspirant

Data Science Portal for Beginners

Follow

AEGIS GRAHAM BELL AWARDS 2018

RECENT POSTS

Twitter Sentiment analysis using R

How Q learning can be used in reinforcement learning

How to perform Reinforcement learning with R

Feature selection techniques with R

How to perform hierarchical clustering in R

10 Smart R programming tips to become better R programmer

How to create histograms in R

How to perform the principal component analysis in R

Building Random Forest Classifier with Python Scikit learn

How the random forest algorithm works in machine learning

FOLLOW ON TWITTER

CATEGORIES

- Amazon
- Big Data
- Courses
- Data Science
- Data Science Events
- DATAMINING
- Deep Learning

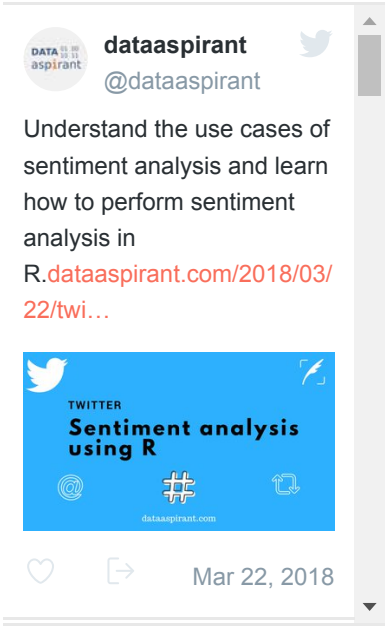
NEVER MISS A BIT

Hey Dude Subscribe to Dataaspirant. To get post updates in your inbox.

Email Address

Subscribe

Tweets by @dataaspirant



Embed View on Twitter

Interviews

Machine Learning

Newsletter

python

R

Recommendation Engine

QUICK LINKS

Home
About
Join us
Beginners Guide
Data scientists Interviews

HOURS & INFO

WhiteField, Bangalore, India
Opens: 9:00 AM
Closes: 11:00 PM

CATEGORIES

Amazon
Big Data
Courses
Data Science
Data Science Events

TAGS

Amazon go Big data Bigdata Classification classification algorithms clustering

partners

Data science courses

DATAMINING

Deep Learning

Interviews

Machine Learning

Newsletter

python

R

Recommendation Engine

algorithms datamining Data mining Datascience data science DataScienceCongress2017 Data science Courses Data Science Events data scientist Decision tree deep learning hierarchical clustering k-nearest neighbor kaggle Linear Regression logistic regression Machine learning monthly newsletter multinomial logistic regression Naive Bayes pca python python programming language random forest Recommendation_engine Recommendation_systems Regression regression coefficient Reinforcement Learning R Programming Scala scikit-learn SciPy Script output similarity_distance Supervised Learning Svm tensorflow Unsupervised Learning virtual_env