caRandom Forest Classifier

1. <https://www.datacamp.com/tutorial/random-forests-classifier-python>
2. Below Code for practice

**About dataset**

The dataset used in this is **‘titanic.csv’** which is available for free, which is available on Kaggle.com. This dataset includes the following features

**1. Importing Libraries and reading dataset**

**import** pandas **as** pd

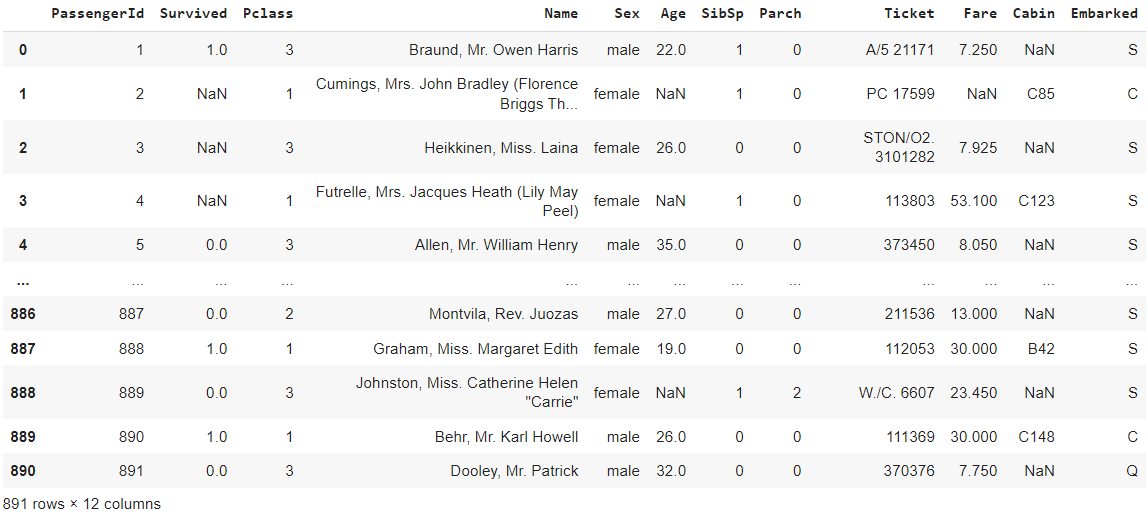
**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

df = pd.read\_csv("titanic.csv")

df



**2. Data preprocessing**

df.drop(['Cabin','PassengerId','Name','Ticket’],axis=1,inplace=True)

df = df.fillna(0)

#### 3. ****Handling categorical data****

**from** sklearn.preprocessing **import** LabelEncoder

le=LabelEncoder()

df['Sex']=le.fit\_transform(df['Sex'])

df['Embarked']=le.fit\_transform(df['Embarked'])

df

#### 4. ****Dependent and independent variables****

*# Putting feature variable to X*

X = df.drop('Survived',axis=1)

*# Putting response variable to y*

y = df['Survived']

#### ****5. Splitting dataset into Training and Testing Set****

*# Splitting the data into train and test*

**from** sklearn.model\_selection **import** train\_test\_split

*# Splitting the data into train and test*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, random\_state=42)

Next, split both x and y into training and testing sets with the help of the train\_test\_split() function. In this training data set is 0.8 which means 80%.

#### ****6. Implementing a Random forest classifier****

*#Import Random Forest Model*

**from** sklearn.ensemble **import** RandomForestClassifier

*#Create a Gaussian Classifier*

clf=RandomForestClassifier(n\_estimators=100)

*#Train the model using the training sets y\_pred=clf.predict(X\_test)*

clf.fit(X\_train,y\_train)

##### **Different parameters are used in the Random forest algorithm**

1. **N\_estimators-The number of decision trees in the forest.**

**Note:**The default value is **100**. You can increase the number of trees that can increase the accuracy but be careful that should not lead to overfitting

1. **criterion{“gini”, “entropy”}, default=”gini”**

This is to measure the quality of a split. These are the criterion by which the decision tree actually split the variables.

* **“gini”** for the Gini impurity
* **“entropy**” for the information gain

1. **Max\_depth int, default=None**

The maximum depth of the tree(root node to terminal node).

Note: If you are using a high value that means you are overcomplicating the things and that can lead to overfitting. So be careful while choosing the value.

1. **min\_samples\_split(int or float, default=2)**

The minimum number of samples actually required to split an internal node:

Remember the lower the value the higher the chance to fit errors but that doesn’t mean you choose a very high value because that will over generalize the model leading to overfitting. So choose value accordingly.

1. **min\_samples\_leaf(int or float, default=1)**

The minimum number of samples is required to be at a leaf node.

1. **Max\_features {“auto”, “sqrt”, “log2”}, int or float, default=”auto”**

a maximum number of features random forest considers when looking for the best split.

1. **n\_jobs(int, default=None)**

It is the number of jobs to run in parallel. This is used when you have the capability to do parallel processing**where n\_jobs=** -1 means using all processors and n\_jobs=1, it can use only one processor

1. **random\_state(int, RandomState instance or None, default=None)**

It Controls both the randomness of the samples used when building trees.

1. **verbose(int, default=0)**

Controls the verbosity when fitting and predicting. It gives you all the run-time information.

You can hyper-tune these by changing the values. You can read my blog on [**hyper-tuning**](https://www.shiksha.com/online-courses/articles/hyperparameter-tuning-beginners-tutorial/).

#### ****7. Predicting test cases using random forest****

*# Predicting the test set results*

Pred = classifier.predict(X\_test)

**print**(Pred)

**Output:**

[0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 0 1 0 1

 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 1 1 1 1 1

 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1]

#### 

#### ****8. Checking the accuracy score****

**from** sklearn.metrics **import** classification\_report

rand\_score=classifier.score(X\_test, y\_test)

'''rand\_score=classifier.accuracy\_score(y\_test,Pred)'''

classification\_report\_rf=classification\_report(y\_test,Pred)

**print**("Accuracy score:",rand\_score)

**Output:**

Accuracy score: 0.8268156424581006