Random Forest:

Its a part of Supervised Learning , where we know the Dependent and Independent variables.

Random Forest algorithm works on ensemble method which means grouping of all the results into one on below conditions:

All items in the group should be independent with each other.

All items should not be correlated with each other.

Example: If cricket captain is confused whether he select new player for the next match then he will be taking help from different people,like Batting Coach,Bowling Coach,Feilding Coach,Management and also some visitor's opinion then by taking average from all the 5 people Cricket captain will take/Predict the Decision for selecting a new player or not selecting,This is how Random Forest works.

In [ ]:

```

```

Bootstrapeed DataSet:

Subsets of Original Data which forms by Randomly sampling which involves Repetation,Duplication and Missing values.

After randomising if we form 3 Subsets of Data from original data then each subsets will split by chossing the Minimum and Best Variables and form the Root Node and calculates the Gini Index and then Decision Tree form.

Uses Random samples every time

Once all the Gini Index calculated for all the 3 subsets,then after calculating the Average,Model takes Decision or make Prediction.

Out of Bag points (OOB):

When Random Forest selects variables randomly then its for SURE that it will be missing or duplicating the Data in new form subsets,.

As it calculates that 37 % of Data are Missed/Duplicates from the Original Data and 63.2 % are the Unique Sample Data points.Every Bootstrapped Data will have the OOB points.So when we will try to evaluate the model accuracy we can compare it with OOB points by the ratio of out of bag samples correctly classified by the random forest model and proportion of OOB samples incorrectly classified-out of bag error.

Formula is (1-1/E).

SUMMARIZE: Bootstrapped > Each Tree spilts by giving class prediction > class with most votes/average becomes Model Prediction.

Usefull for both Regression and Calssification models.

Its not Explanatory as which Variable is most valuable can't be explanied like Decision Tree.

In [ ]:

```

```

In [ ]:

```

```

Now Hands On practise by using Random Forest Algorithm.

We have a Bank Data set with credit card details,by using this data set we will be building a model to check whether customers will respond to the Marketing Campaign or not.

In [1]:

```
# importing libraries.

import numpy as np
import pandas as pd

# Load and check the head of data set (to identify data is correctly loaded or not).

rf = pd.read_csv (r'E:\G\Banking Dataset.csv')
rf.head(10)
```

Out[1]:

	Cust_ID	Target	Age	Gender	Balance	Occupation	No_OF_CR_TXNS	AGE_BKT	SCR
0	C1	0	30	M	160378.60	SAL	2	26-30	826
1	C10	1	41	M	84370.59	SELF-EMP	14	41-45	843
2	C100	0	49	F	60849.26	PROF	49	46-50	328
3	C1000	0	49	M	10558.81	SAL	23	46-50	619
4	C10000	0	43	M	97100.48	SENP	3	41-45	397
5	C10001	0	30	M	160378.60	SAL	2	26-30	781
6	C10002	0	43	M	26275.55	PROF	23	41-45	354
7	C10003	0	53	M	33616.47	SAL	45	>50	239
8	C10004	0	45	M	1881.37	PROF	3	41-45	339
9	C10005	0	37	M	3274.37	PROF	33	36-40	535

In [3]:

```
rf.shape # There are 10 coloumns and 20,000 rows.
```

Out[3]:

```
(20000, 10)
```

In [3]:

```
# As all the variabls are self-explanotory.
# TARGET variable is giving information that "0" (Not responded to Marketing Campaign)
 and "1" (Responded to Marketing Campaign).
```

In [4]:

```
rf.info ()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 10 columns):
Cust_ID           20000 non-null object
Target            20000 non-null int64
Age               20000 non-null int64
Gender            20000 non-null object
Balance           20000 non-null float64
Occupation        20000 non-null object
No_OF_CR_TXNS     20000 non-null int64
AGE_BKT           20000 non-null object
SCR               20000 non-null int64
Holding_Period    20000 non-null int64
dtypes: float64(1), int64(5), object(4)
memory usage: 1.5+ MB
```

In [7]:

```
# For Random Forest we will need all the integers values for all the variables so all n
on int needs to convert into int.
# This can be done by converting object to the integers by label encoding.

for feature in rf.columns:
        if rf[feature].dtype == 'object':
            rf[feature] = pd.Categorical(rf[feature]).codes
```

In [8]:

```
rf.info() # All Variables are converted into Integers (its not matter wether its int8,i
nt16)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 10 columns):
Cust_ID           20000 non-null int16
Target            20000 non-null int64
Age               20000 non-null int64
Gender            20000 non-null int8
Balance           20000 non-null float64
Occupation        20000 non-null int8
No_OF_CR_TXNS     20000 non-null int64
AGE_BKT           20000 non-null int8
SCR               20000 non-null int64
Holding_Period    20000 non-null int64
dtypes: float64(1), int16(1), int64(5), int8(3)
memory usage: 1.0 MB
```

In [9]:

```
# Dropping the Independent Variable and storing in new Data frame x.
# Popping the Dependent Variable and storing in new Data frame y.

x = rf.drop (['Target','Cust_ID'],axis=1)
y = rf.pop ('Target')
```

In [12]:

```
x.head() # Independent Variable.
```

Out[12]:

	Age	Gender	Balance	Occupation	No_OF_CR_TXNS	AGE_BKT	SCR	Holding_Period
0	30	1	160378.60	1	2	0	826	9
1	41	1	84370.59	2	14	3	843	9
2	49	0	60849.26	0	49	4	328	26
3	49	1	10558.81	1	23	4	619	19
4	43	1	97100.48	3	3	3	397	8

In [121]:

```
x.shape
```

Out[121]:

```
(20000, 8)
```

In [13]:

```
y.head() # Dependent variable.
```

Out[13]:

```
0    0
1    1
2    0
3    0
4    0
Name: Target, dtype: int64
```

In [ ]:

```
```

Training and Test.

Training Data Set: We will be training the Model on training data set.

Once we train the model on training data set then we will be evaluating our model on Test data set by comparing the results on Training data set.

So Train our Model on Training Data and then give the Test Data to the Trained Model to check how it is performing by comparing Training and Test outputs on Models.

In [10]:

```
from sklearn.model_selection import train_test_split
```

In [11]:

```
X_train,X_test,train_labels,test_labels = train_test_split (x, y , test_size = .30 , ra
ndom_state=1)

#X_train - Training Data set
#X_test - Testing Data set on which we will evaluate how our Model performs.
#train_labels - dependent variables train.
#test_labels - dependent variables test.
#test_size - 30% as 70 % we will be using for the Training (which model will be train b
y looking this data)
#random_state=1 - everytime  model will select the data set it will be constant its not
matter whether we provie random_state = 1 or 100 or 1000 but whatever we will provide i
n the beginning
# it should be same for every time so that it will give same training and test samples.
```

In [12]:

```
# We will pull the RandomForest Classifier.

from sklearn.ensemble import RandomForestClassifier
```

In [17]:

```
# Now we will be calling Randon Forest Classifer.

rfcl = RandomForestClassifier(oob_score=True , n_estimators=501) # n_estimators = 501 ,
it means we want 501 Tress or subsets of orinigal data.
```

In [18]:

```
# Now we will be fitting the data and building the model.

rfcl = rfcl.fit(X_train,train_labels) # so we are using trianing data set (for depenede
nt and independent ) for building the model.
```

In [19]:

```
# Now we have build the model and now we can check the different parameters of the mode
l itslef.
# 1 : OBB score.

rfcl.oob_score
```

Out[19]:

```
True
```

In [20]:

```
rfcl.oob_score_
```

Out[20]:

```
0.9189285714285714
```

In [27]:

```
# So OOB score gives us the Accuracy of Model is 91.97 % (92%) so Error rate is 8 %.
# which means on our training data the accuracy is high and our model performs very goo
d which also means that by using the given data set we can use this model.
```

In [ ]:

```

```

Now we will check number of functions in RandomForestClassifier.

1: Max_depth = If defines the number of Levels within each Decision Tree, if we have max_depth = 8 then each decision tree will have Max 8 levels.

2: Max_features = It means Which feature will be the Best root node from each DT when split happens,suppose we have in this data set 8 independent variables and if we select max_feature = 5 then the best 5 variables from 8 will get selected and based on that (5 variables) best root node picked up and calculate the Gini index.

3: Min_Sample_leaf = Minimum leaf for every DT , benchmark is on all the training data i.e 70 % , out of which we select 1-3 % leaf value.

4: Min_Sample_split = Minimum splitting which goes 3 times the min_sample_leaf.

5: n_estimators = How many subsets will form from the main data (then all the DT will form from each subset,calculates Gini index and based on all DT gini index value we get final value).

In [62]:

```
rfcl = RandomForestClassifier (n_estimators=501 ,
                               max_features= 5 ,
                               max_depth= 10 ,
                               min_samples_leaf= 50 ,
                               min_samples_split= 110 ,
                               oob_score= True)
```

In [65]:

```
rfcl = rfcl.fit (X_train , train_labels) # We have fit in the model with new features.
```

In [67]:

```
rfcl.oob_score_
```

Out[67]:

```
0.9152857142857143
```

In [68]:

```
# As we can see that OOB score has given the accuracy of 0.9152 which is 0.40 less than
previous one which can consider as No change.
# Hence we can say that with this values of features Accuracy is not changing.
```

In [ ]:

```

```

So we have chosen the fetures in the model but whether all features with their values are correctly selected or not or which variable we need to choose with how many values, this can be evaluated by GRIDSEARCH CROSSVALIDATION MATRIX.

GRIDSEARCH - It means we can pass multiple values for each feature and it can also return or give us infomration how many values for which feature can give us the best results.

In [70]:

```
from sklearn.model_selection import GridSearchCV
```

FOR GRIDSEARCH :

1:First we need to create a Dictonary by specifying the values of each feature. 2:Each Feature name should be exactly the same example : {max_features : [8,16]} so this means max_features will try first 8 and then it will try for 16,we can give like {max_feature : [1,2,8,5,17,26] } but this will be huge for the gridsearch. 3: gridsearch (estimator = rfcl , param_grid = dictonary variable () , cv = 3)cross validation). so first RandomForest algorithm will be created 3 times using the specific Data and then it splits into 2 parts 70 % for Training and 1 part 30 % for Testing.

In [72]:

```
param_grid = {'max_depth' : [7,10],
              'max_features' : [4,6],
              'min_samples_leaf' : [50,100],
              'min_samples_split': [150,300],
              'n_estimators' : [301,501]
             }
```

In [73]:

```
# so above we have created one dictonary with all the values.
# So first RF model will perform on 7*4*50*150*301 and then 7*6*100*300*501 and so o
n...
# Total 32 RF model will be created and when we wo cv = 3 then it will again run and it
will run 32*3= 96 Times.
# And out of 96 times the BEST RF model we will get as an output.
```

In [74]:

```
rfcl = RandomForestClassifier()
```

In [75]:

```
grid_search = GridSearchCV (estimator=rfcl , param_grid=param_grid, cv=3) # so it run 3
2 times and calculate best RF out of 96.
```

In [76]:

```
# now we will use the model.

grid_search.fit (X_train , train_labels)
```

Out[76]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight
=None,
                                              criterion='gini', max_depth=
None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.
0,
                                              n_estimators='warn', n_jobs=
None,
                                              oob_score=False,
                                              random_state=None, verbose=
0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [7, 10], 'max_features': [4, 6],
                         'min_samples_leaf': [50, 100],
                         'min_samples_split': [150, 300],
                         'n_estimators': [301, 501]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

So now we can calculate the Best Parameters like values for each feautres. Also we can calculate the Best Estimators which means out of 32 models which model has used this best parameters,so we can use that model.

In [77]:

```
grid_search.best_params_ # This are the BEST Parameters out of 96 parameters.
```

Out[77]:

```
{'max_depth': 7,
 'max_features': 6,
 'min_samples_leaf': 50,
 'min_samples_split': 150,
 'n_estimators': 301}
```

In [79]:

```
grid_search.best_estimator_ # This is the BEST MODEL out of 32 which uses the above BES
T Parameters.
```

Out[79]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
                       max_depth=7, max_features=6, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=150,
                       min_weight_fraction_leaf=0.0, n_estimators=301,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [80]:

```
best_grid = grid_search.best_estimator_
```

In [93]:

```
# Now we will Predict the RF model on our Training and Testing Data.

rtrain_predict = best_grid.predict(X_train)
rtest_predict = best_grid.predict(X_test)

### we can use probability or prediction
### We have successfully build our Random Forest Model.
```

In [ ]:

```

```

Now we will Evaluate our Model , as how it is Performing by using Classification and Confusion Matrix.

In [94]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [95]:

```
confusion_matrix (train_labels , rtrain_predict)
```

Out[95]:

```
array([[12750,    32],
       [ 1145,    73]], dtype=int64)
```

In [96]:

```
confusion_matrix (test_labels , rtest_predict)
```

Out[96]:

```
array([[5473,   12],
       [ 487,   28]], dtype=int64)
```

In [97]:

```
print(classification_report(train_labels,rtrain_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     12782
           1       0.70      0.06      0.11      1218

    accuracy                           0.92     14000
   macro avg       0.81      0.53      0.53     14000
weighted avg       0.90      0.92      0.88     14000
```

In [98]:

```
print(classification_report(test_labels,rtest_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      5485
           1       0.70      0.05      0.10       515

    accuracy                           0.92      6000
   macro avg       0.81      0.53      0.53      6000
weighted avg       0.90      0.92      0.88      6000
```

In [99]:

```
# we can see that Precision is Same for Development (Training) and Testing data.
# if we see only value (1) then Recall is poor for both the Data and also F1 score is v
ery poor for both.
# we can increase the benchmark for probability which is 0.05.
```

Now we will be checking the ROC and AUC, which we will be able visualise the pattern of Develement and Testing Set.

In [101]:

```
# we will only looking for probabilities and also for only (1's).

prob = best_grid.predict_proba(X_train)
prob = prob [:,1]
```

In [102]:

```
import matplotlib.pyplot as plt
```

In [105]:

```
from sklearn.metrics import roc_auc_score
```

In [106]:

```
auc = roc_auc_score (train_labels , prob)
auc
```

Out[106]:

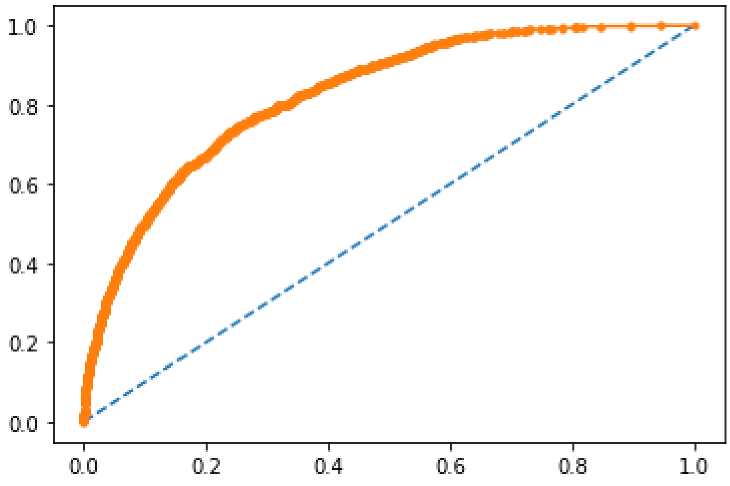
```
0.8267420330673344
```

In [107]:

```
# we will building the ROC curve for Training Data.

from sklearn.metrics import roc_curve

fpr , tpr, thresholds = roc_curve (train_labels , prob)
plt.plot ([0,1] , [0,1] , linestyle = '--')
plt.plot  (fpr, tpr, marker = '.') # plot the ROC curve
plt.show()
```



In [108]:

```
# AUC is 82 % as per the value.
```

In [112]:

```
# We will be building the ROC curve for Testing Data.

prob = best_grid.predict_proba (X_test)
prob = prob [:,1]
```

In [115]:

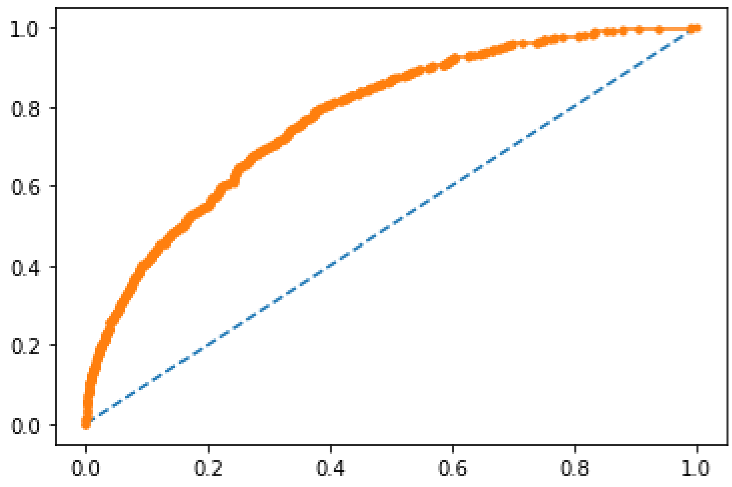
```
auc = roc_auc_score (test_labels,prob)
auc
```

Out[115]:

```
0.776378295616465
```

In [117]:

```
fpr , tpr ,thresholds = roc_curve (test_labels,prob)
plt.plot ([0,1],[0,1], linestyle ='--')
plt.plot (fpr, tpr, marker='.')
plt.show ()
```



In [118]:

```
# AUC is 77% as per the value
```

In [ ]:

```
```

Conclusion:Training Data (Development) (82 %) is performing good as compared to the Testing Data (77 %).

So we know that 1s are for Responding to Marketing Email Campaign and 0s are Not responding.

We can say that Result are more towards responding as the difference is only 15 % in Training and Testing Result.

If we add more data then also it will give us the 1s only which means it is having an Overfitting issue and it is Biased.

We need to have look for other different models.