Ensemble Technique

① What is Ensemble?

Combining Multiple Models => TRAIN => PREDICTION

Two types

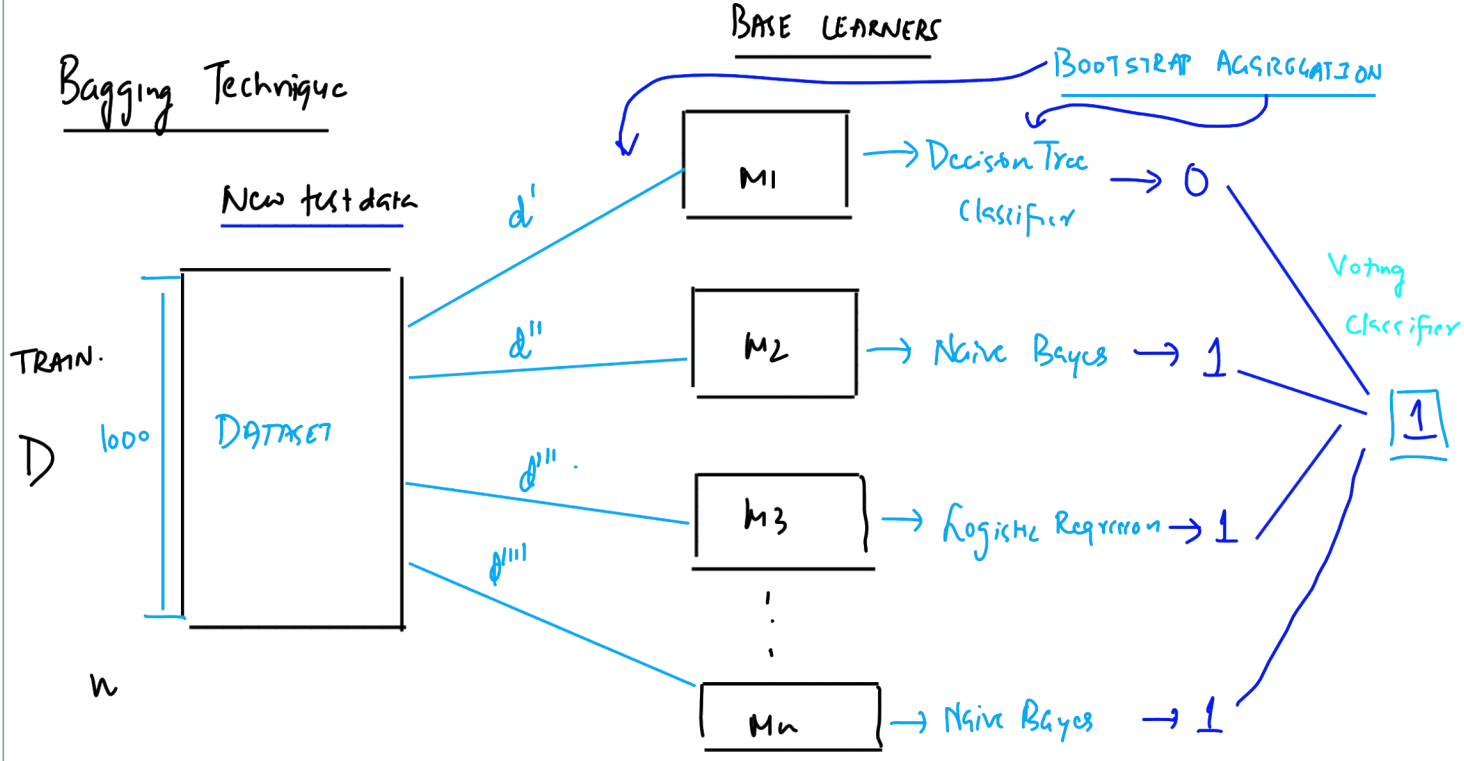
① Bagging.

① Random Forest Classifier And Regressor

② Boosting

① AdaBoost
② Gradient Boost
③ Xgboost

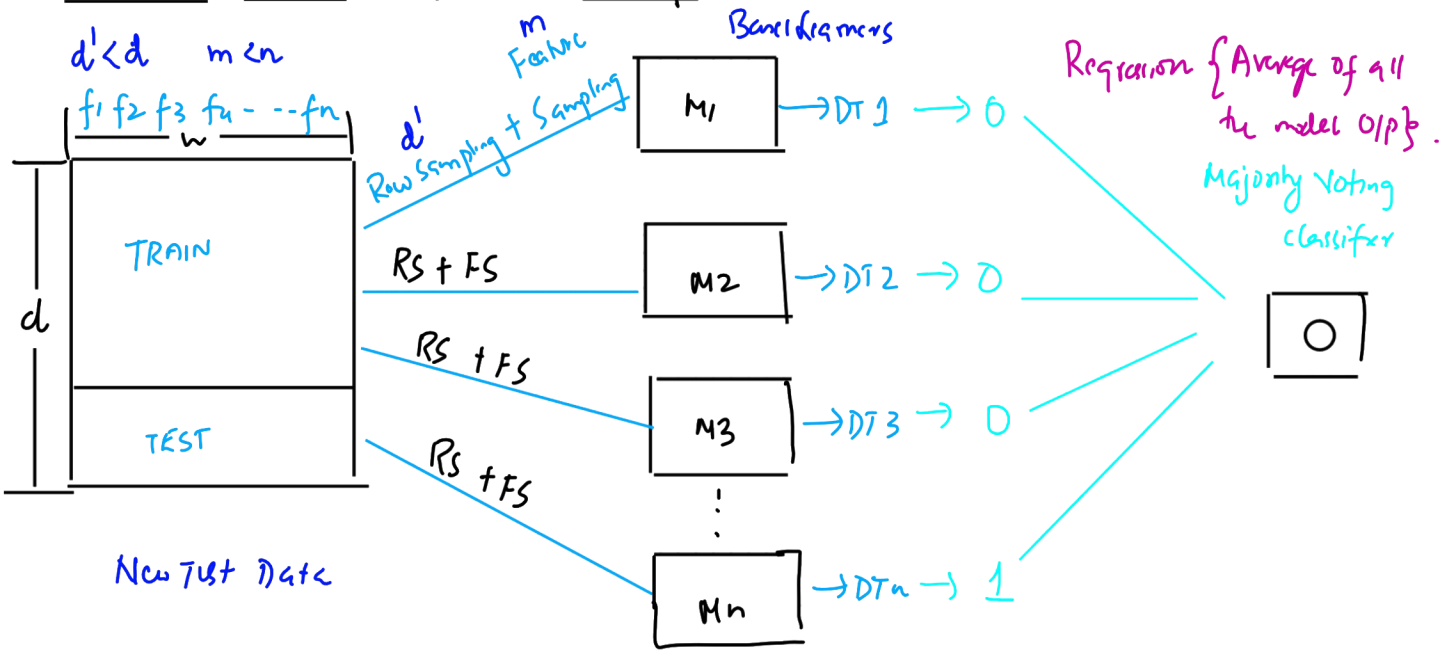
Bagging Technique



Custom Bagging Technique

Regression { Average of all o/p → Prediction }

Random Forest Classifier And Regressor



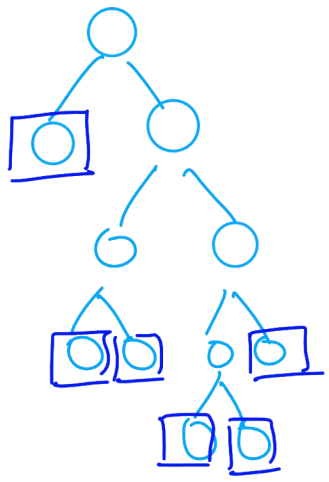
Note:

Classification ⟶ Majority Voting Classifier

Regression ⟶ Average O/p of all the models.

(Q) Why should we use Random Forest instead of DT?

Decision Tree



Overfitting

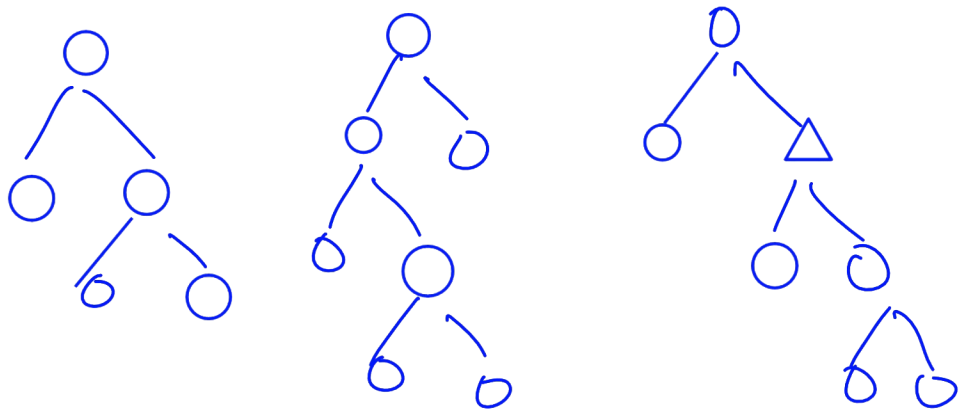
Generalized model

Low Bias, Low Variance

Random Forest

TRAINING ACC ↑↑ → Low Bias ⟶ Low Bias

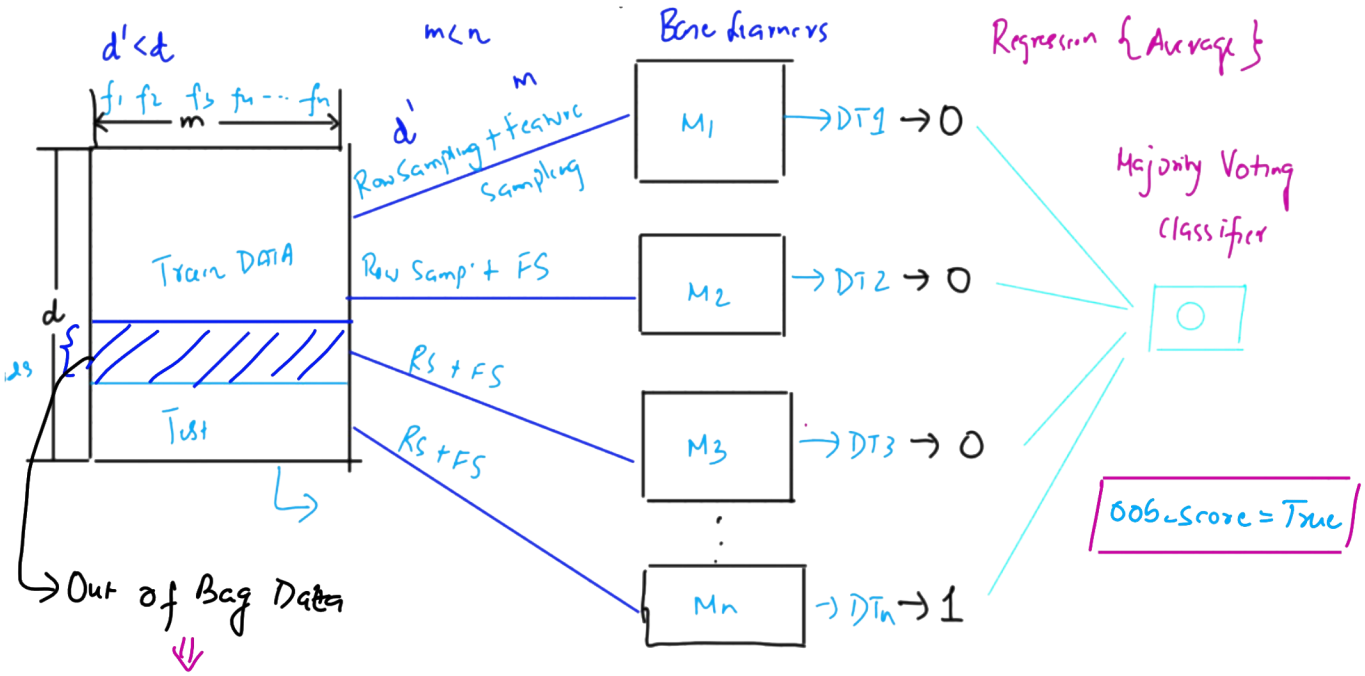
TEST DATA ACC ↓↓ → High Variance ⟶ Low Variance



Uses mutiple Decision Trees in samples of data

and give Low Bias and Low Variance
which is our Generalized model

Out of Bag Score



Validation data → Performance and Accuracy of Random Forest → OOb score = 85%, 90%, 75%

Machine_Learning Ensemble_technique Bagging - decision_tree and pipeline with multiple models

1 Random Forest Classifier Implementation with Pipelines and Hyperparameter Tuning

- pipelines : model training is not one time activity, whenever we get more new data we need to again train it...
- so again we don't keep on writing code for feature engineering, scaling etc, that's-why we write generic code which can automate the entire process of training(feature engineering, model training etc)
- example each month new data is coming so have to automate repeating process on new data

-we don't automate EDA because it's all about analysis

[6]:
```
import seaborn as sns
df = sns.load_dataset("tips")
## this data can be come from anywhere like mongoDB, SQL etc
df.head(2)
```

[6]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

[7]:
```
## this dataset is about customer coming to restuarant
### their total-bill, tip, time etc all information

## for this bagging technique-Random Forest
### we are going to predict 'time'-dependent feature, rest all features are␣
 ↪independent
```

[8]:
```
df['time'].unique()
## only 2 unique categories, means binary classification problem
```

[8]:
```
['Dinner', 'Lunch']
Categories (2, object): ['Lunch', 'Dinner']
```

[5]:
```
## Handling Missing Values
## handling Categorical features
```

```
## handling outliers
## feature scaling
## Automating the entire process
```

[ ]:

2 1. EDA

- we cannot automate EDA process, because it's all about analysis

[12]:
```
df.head(2)
## there are 3 categorical features(independent):sex,smoker,day
## we can use .info() to get all information about dataset
```

[12]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

[16]:
```
## our dependent variable is in category so we are converting it into 0 and␣
 ↪1(for binary classification)
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit_transform(df['time'])## encoding time feature
```

[16]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

[17]:
```
df['time'] = encoder.fit_transform(df['time'])
```

[19]:
```
df.head(2)
## now time(dependent feature) variable is converted to 0 and 1
```

[19]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	0	2
1	10.34	1.66	Male	No	Sun	0	3

[21]:
```
df.time.value_counts()
## checking total no. of 0 and 1
```

[21]:
```
0    176
1     68
Name: time, dtype: int64
```

[24]:
```
## independent and dependent features
X = df.drop(labels=['time'],axis=1) ##droping time from independent features
y = df['time'] ## dependent feature is only, because we are predicting time
```

[25]:
```
X.head(2)
```

[25]:

	total_bill	tip	sex	smoker	day	size
0	16.99	1.01	Female	No	Sun	2
1	10.34	1.66	Male	No	Sun	3

[39]:
```
## train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20,
                                                    random_state=42)
```

[40]:
```
X_train.head(2)
```

[40]:

	total_bill	tip	sex	smoker	day	size
228	13.28	2.72	Male	No	Sat	2
208	24.27	2.03	Male	Yes	Sat	2

[42]:
```
from sklearn.impute import SimpleImputer ## Handling Missing Values
from sklearn.preprocessing import OneHotEncoder## handling Categorical features,
```

```
#all are nominal-categorical variable not-ordinal value, cannot assign ranks so␣
 ↪we'll use OneHotEncoding

## handling outliers(not handling for this dataset we'll see it in end to end␣
 ↪project

from sklearn.preprocessing import StandardScaler## feature scaling, using␣
 ↪standard scaler technique
#for random-forest(decision tree) we don't have to compulsary scale data,␣
 ↪because it splits all
#for Linear/logistic-regression etc we have to preform scaling


from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer## Automating the entire process

## we have 3 library and we create pipeline for this to automate process
# because whenever we get new data we've to perform this steps regularly(handle␣
 ↪missing-value, feature scaling etc)
```

[43]:
```
## seperating categorical and numerical variables/features/column
categorical_cols = ['sex', 'smoker','day']
numerical_cols = ['total_bill', 'tip', 'size']
```

[44]:
```
## Feature Engineering Automation

##creating numerical-pipeline, it'll be responsible for any feature-engineering␣
 ↪activity on these columns/variable
### like missing-value, scaling etc..
num_pipeline = Pipeline(     ## creating pipeline
    steps =[
    ##define steps in steps-parameter in 'tuples' to perform by pipeline
        ('imputer', SimpleImputer(strategy='median')),# missing values
        ##object-name , object-initializing
        ('scaler',StandardScaler())# feature scaling
    ]
)

## Categorical Pipeline
cat_pipeline = Pipeline(
        steps=[
            ('imputer',SimpleImputer(strategy='most_frequent')), ## handling␣
 ↪missing values
            ## replace NaN with most_frequent value like 'mode'
            ('onehotencoder',OneHotEncoder())## Categorical features to␣
 ↪numerical
```

```
        ]
)

## as soon as we get new data we have to perform both pipeline, so we can␣
 ↪create a Wrapper
## to create WRAPPER on top of both pipeline we use 'ColumnTransformer()'
```

[45]:
```
## adding WRAPPER on top of both pipelines
preprocessor = ColumnTransformer([
    ('num_pipeline', num_pipeline, numerical_cols),
    ('cat_pipeline',cat_pipeline, categorical_cols)

])
#basically combining both pipeline to perform together
```

[46]:
```
##now we don't have to do imputing, scaling etc.. seperately,
### we'll just use aboove preprocessor on training data to do automatically

preprocessor.fit_transform(X_train)
```

[46]:
```
array([[-0.79306155, -0.2580329 , -0.61214068, …,  1.        ,
         0.        ,  0.        ],
       [ 0.46322744, -0.74211442, -0.61214068, …,  1.        ,
         0.        ,  0.        ],
       [ 0.80730659,  0.6399734 , -0.61214068, …,  0.        ,
         0.        ,  0.        ],
       …,
       [-1.65383098, -1.46472887, -0.61214068, …,  0.        ,
         0.        ,  0.        ],
       [ 1.64749986,  0.32426806, -0.61214068, …,  0.        ,
         1.        ,  0.        ],
       [ 2.75289699, -0.41237773,  0.45363997, …,  1.        ,
         0.        ,  0.        ]])
```

[47]:
```
X_train=preprocessor.fit_transform(X_train)##saving it in a variable
X_test = preprocessor.transform(X_test)

## feature Engineering completed
```

[48]:
```
from sklearn.ensemble import RandomForestClassifier
```

[49]:
```
## Automate Model Training Process
### because we train multiple model and select which one is giving better␣
 ↪accuracy

### to automate a model we first create a dictionary
models={
```

```
    'Random Forest': RandomForestClassifier()
}

## which-ever model/algorithm we want to train with, have to put
### in this dictionary
```

[50]:
```
from sklearn.metrics import accuracy_score
```

[53]:
```
## creating a function
def evaluate_model(X_train,y_train, X_test, y_test, models):

    report = {}
    for i in range(len(models)):
        model = list(models.values())[i]

        #Train model
        model.fit(X_train,y_train)


        # Predict Testing data
        y_pred = model.predict(X_test)

        ## Get accuracy for test data prediction
        test_model_score = accuracy_score(y_test,y_pred)

        report[list(models.keys())[i]] = test_model_score


    return report
```

[54]:
```
evaluate_model(X_train,y_train,X_test,y_test,models)
```

[54]:
```
{'Random Forest': 0.9591836734693877}
```

[56]:
```
## lets add decision tree to our model
from sklearn.tree import DecisionTreeClassifier

models={
    'Random Forest': RandomForestClassifier(),
    'Decision Tree' : DecisionTreeClassifier()
}
```

[57]:
```
evaluate_model(X_train,y_train,X_test,y_test,models)
```

[57]:
```
{'Random Forest': 0.9591836734693877, 'Decision Tree': 0.9387755102040817}
```

[58]:
```
## lets add Support Vector Machine to our model
## SVC-classifier because classification problem
from sklearn.svm import SVC

models={
    'Random Forest': RandomForestClassifier(),
    'Decision Tree' : DecisionTreeClassifier(),
    'SVC' : SVC()
}
```

[59]:
```
evaluate_model(X_train,y_train,X_test,y_test,models)
```

[59]:
```
{'Random Forest': 0.9591836734693877,
 'Decision Tree': 0.9387755102040817,
 'SVC': 0.9591836734693877}
```

[60]:
```
## after this step which ever model is giving better accuracy
### we can perform Hyperparameter tuning
#### for now 'Random Forest' is giving better accuracy
#because it uses multiple Decision Trees so it gives godd accuracy
## even in hackaton we mostly use 'Random Forest of ensemble technique

classifier = RandomForestClassifier()
```

[62]:
```
## Hyperpatameter Tuning with RandomizedSearchCV
params={'max_depth':[3,5,10,None],
        'n_estimators':[100,200,300],
        'criterion':['gini', 'entropy']
}
```

[63]:
```
from sklearn.model_selection import RandomizedSearchCV
```

[65]:
```
cv =RandomizedSearchCV(classifier, param_distributions=params, cv=3,␣
 ↪scoring='accuracy',verbose=3)
```

[66]:
```
cv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV 1/3] END criterion=entropy, max_depth=None, n_estimators=200;, score=0.954
total time=   0.4s
[CV 2/3] END criterion=entropy, max_depth=None, n_estimators=200;, score=0.969
total time=   0.4s
[CV 3/3] END criterion=entropy, max_depth=None, n_estimators=200;, score=0.954
total time=   0.4s
[CV 1/3] END criterion=entropy, max_depth=None, n_estimators=100;, score=0.969
total time=   0.2s
[CV 2/3] END criterion=entropy, max_depth=None, n_estimators=100;, score=0.985
```

```
total time=   0.2s
[CV 3/3] END criterion=entropy, max_depth=None, n_estimators=100;, score=0.954
total time=   0.2s
[CV 1/3] END criterion=gini, max_depth=10, n_estimators=300;, score=0.954 total
time=   0.5s
[CV 2/3] END criterion=gini, max_depth=10, n_estimators=300;, score=0.985 total
time=   0.6s
[CV 3/3] END criterion=gini, max_depth=10, n_estimators=300;, score=0.954 total
time=   0.5s
[CV 1/3] END criterion=entropy, max_depth=5, n_estimators=200;, score=0.969
total time=   0.4s
[CV 2/3] END criterion=entropy, max_depth=5, n_estimators=200;, score=0.969
total time=   0.4s
[CV 3/3] END criterion=entropy, max_depth=5, n_estimators=200;, score=0.969
total time=   0.4s
[CV 1/3] END criterion=entropy, max_depth=10, n_estimators=200;, score=0.954
total time=   0.4s
[CV 2/3] END criterion=entropy, max_depth=10, n_estimators=200;, score=0.969
total time=   0.4s
[CV 3/3] END criterion=entropy, max_depth=10, n_estimators=200;, score=0.954
total time=   0.4s
[CV 1/3] END criterion=entropy, max_depth=5, n_estimators=300;, score=0.969
total time=   0.6s
[CV 2/3] END criterion=entropy, max_depth=5, n_estimators=300;, score=0.985
total time=   0.6s
[CV 3/3] END criterion=entropy, max_depth=5, n_estimators=300;, score=0.969
total time=   0.6s
[CV 1/3] END criterion=entropy, max_depth=3, n_estimators=100;, score=0.969
total time=   0.2s
[CV 2/3] END criterion=entropy, max_depth=3, n_estimators=100;, score=0.954
total time=   0.2s
[CV 3/3] END criterion=entropy, max_depth=3, n_estimators=100;, score=0.954
total time=   0.2s
[CV 1/3] END criterion=gini, max_depth=3, n_estimators=100;, score=0.969 total
time=   0.2s
[CV 2/3] END criterion=gini, max_depth=3, n_estimators=100;, score=0.954 total
time=   0.2s
[CV 3/3] END criterion=gini, max_depth=3, n_estimators=100;, score=0.938 total
time=   0.2s
[CV 1/3] END criterion=entropy, max_depth=10, n_estimators=100;, score=0.954
total time=   0.2s
[CV 2/3] END criterion=entropy, max_depth=10, n_estimators=100;, score=0.969
total time=   0.2s
[CV 3/3] END criterion=entropy, max_depth=10, n_estimators=100;, score=0.954
total time=   0.2s
[CV 1/3] END criterion=gini, max_depth=3, n_estimators=300;, score=0.969 total
time=   0.5s
[CV 2/3] END criterion=gini, max_depth=3, n_estimators=300;, score=0.954 total
```

```
time=   0.6s
[CV 3/3] END criterion=gini, max_depth=3, n_estimators=300;, score=0.938 total
time=   0.6s
```

[66]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [3, 5, 10, None],
                                        'n_estimators': [100, 200, 300]},
                   scoring='accuracy', verbose=3)
```

[67]:
```
cv.best_params_
```

[67]:
```
{'n_estimators': 300, 'max_depth': 5, 'criterion': 'entropy'}
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

2.0.1 lets do for Regression problem with same Dataset of 'total_bill' as output feature

- repeating all above steps but using Regressor model

[1]:
```
## dataset
import seaborn as sns
df = sns.load_dataset("tips")
## this data can be come from anywhere like mongoDB, SQL etc
df.head(2)
```

[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

[31]:
```
## Independent and Dependent Variable
X = df.drop(labels=['total_bill'],axis=1)
y=df['total_bill']
```

[32]:
```
X.head(2)
```

[32]:
```
     tip     sex smoker  day    time  size
0   1.01  Female     No  Sun  Dinner     2
1   1.66    Male     No  Sun  Dinner     3
```

[33]:
```
y.head(2)
```

[33]:
```
0    16.99
1    10.34
Name: total_bill, dtype: float64
```

[34]:
```
## train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20,
                                                    random_state=42)
```

[35]:
```
X_train.head(2)
```

[35]:
```
      tip   sex smoker  day    time  size
228  2.72  Male     No  Sat  Dinner     2
208  2.03  Male    Yes  Sat  Dinner     2
```

[36]:
```
## to create Pipeline (automate Feature Engineering)

from sklearn.impute import SimpleImputer ## Handling Missing Values
from sklearn.preprocessing import OneHotEncoder## handling Categorical features,
#all are nominal-categorical variable not-ordinal value, cannot assign ranks so␣
 ↪we'll use OneHotEncoding

## handling outliers(not handling for this dataset we'll see it in end to end␣
 ↪project

from sklearn.preprocessing import StandardScaler## feature scaling, using␣
 ↪standard scaler technique
#for random-forest(decision tree) we don't have to compulsary scale data,␣
 ↪because it splits all
#for Linear/logistic-regression etc we have to preform scaling


from sklearn.pipeline import Pipeline## Automating the entire process
from sklearn.compose import ColumnTransformer ## wrap both numerical and␣
 ↪categorical pipeline

## we have 3 library and we create pipeline for this to automate process
# because whenever we get new data we've to perform this steps regularly(handle␣
 ↪missing-value, feature scaling etc)
```

[37]:
```
## seperating categorical and numerical variables/features/column
categorical_cols = ['sex', 'smoker','day','time']
numerical_cols = ['tip', 'size']

## we can also use loop to select but for now we are directly writing all␣
 ↪columns name
```

[38]:
```
## Feature Engineering Automation

## Numerical Pipeline
num_pipeline = Pipeline(     ## creating pipeline
    steps =[
    ##define steps in steps-parameter in 'tuples' to perform by pipeline
        ('imputer', SimpleImputer(strategy='median')),# missing values
        ('scaler',StandardScaler())# feature scaling
    ]
)

## Categorical Pipeline
cat_pipeline = Pipeline(
        steps=[
            ('imputer',SimpleImputer(strategy='most_frequent')), ## handling␣
 ↪missing values
            ## replace NaN with most_frequent value like 'mode'
            ('onehotencoder',OneHotEncoder())## Categorical features to␣
 ↪numerical
        ]
)

## as soon as we get new data we have to perform both pipeline, so we can␣
 ↪create a Wrapper
## to create WRAPPER on top of both pipeline we use 'ColumnTransformer()'
```

[39]:
```
## adding WRAPPER on top of both pipelines
preprocessor = ColumnTransformer([
    ('num_pipeline', num_pipeline, numerical_cols),
    ('cat_pipeline',cat_pipeline, categorical_cols)

])
#basically combining both pipeline to perform together
```

[40]:
```
##now we don't have to do imputing, scaling etc.. seperately,
### we'll just use aboove preprocessor on training data to do automatically

X_train=preprocessor.fit_transform(X_train)##saving it in a variable
X_test = preprocessor.transform(X_test)
```

```
## feature Engineering completed

## we can now save preprocessor() to pickle for use it in any application for scaling and preprocessing
```

[41]:
```
X_train[2]
```

[41]:
```
array([ 0.6399734 , -0.61214068,  0.        ,  1.        ,  0.        ,
        1.        ,  1.        ,  0.        ,  0.        ,  0.        ,
        1.        ,  0.        ])
```

[42]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
```

[43]:
```
## Automate Model Training Process
### because we train multiple model and select which one is giving better accuracy

### to automate a model we first create a dictionary
models={
    'Random Forest': RandomForestRegressor(),
    'Linear Regression': LinearRegression(),
    'Support Vector Regressor': SVR(),
    'Decision Tree Regressor': DecisionTreeRegressor()
}

## which-ever model/algorithm we want to train with, have to put
### in this dictionary
```

[44]:
```
from sklearn.metrics import r2_score
```

[49]:
```
## creating a function
def evaluate_model(X_train,y_train, X_test, y_test, models):

    report = {}
    for i in range(len(models)):
        model = list(models.values())[i]

        #Train model
        model.fit(X_train,y_train)


        # Predict Testing data
        y_pred = model.predict(X_test)
```

```
        ## Get accuracy for test data prediction
        test_model_score = r2_score(y_test,y_pred)

        report[list(models.keys())[i]] = test_model_score


    return report
```

[52]:
```
evaluate_model(X_train,y_train,X_test,y_test,models)
```

[52]:
```
{'Random Forest': 0.5060168947967473,
 'Linear Regression': 0.6240808714290969,
 'Support Vector Regressor': 0.4509552360836214,
 'Decision Tree Regressor': 0.48681182575793536}
```

[ ]:

[ ]:

[47]:
```
### checking random forest model accuracy seperately
randomforR= RandomForestRegressor()
randomforR.fit(X_train,y_train)
y_pred = randomforR.predict(X_test)
randomforR.score(X_test,y_test)
```

[47]:
```
0.5051708667827985
```

[48]:
```
r2_score(y_test,y_pred)
```

[48]:
```
0.5051708667827985
```

[ ]:

[ ]: