Machine Learning 🖥

Classification and Regression Tree (CART) Algorithm

Ensemble Technique/Algorithm

$\mathcal{By} \implies \mathcal{PRINCE}$ 👑❤️

https://github.com/pkvidyarthi/

Ensemble Technique $\longrightarrow$

- **Ensemble Algorithm is Classification And Regression Tree (CART) Based.**
- **Random Forest is also prone to overfitting.**
- **Random Forest and Decision Tree have same parameters :** ***criterion & max_depth.***
- **Random Forest $\longrightarrow$ Ensembles of Decision Trees.**

n_estimators :

- **A parameter of Random Forest**
- **n_estimators = 100,** means 100 trees are in the forest. 100 decision trees will be trained under random forest.

Random Forest is slow as compare to other Ensemble Algorithm.

Note :

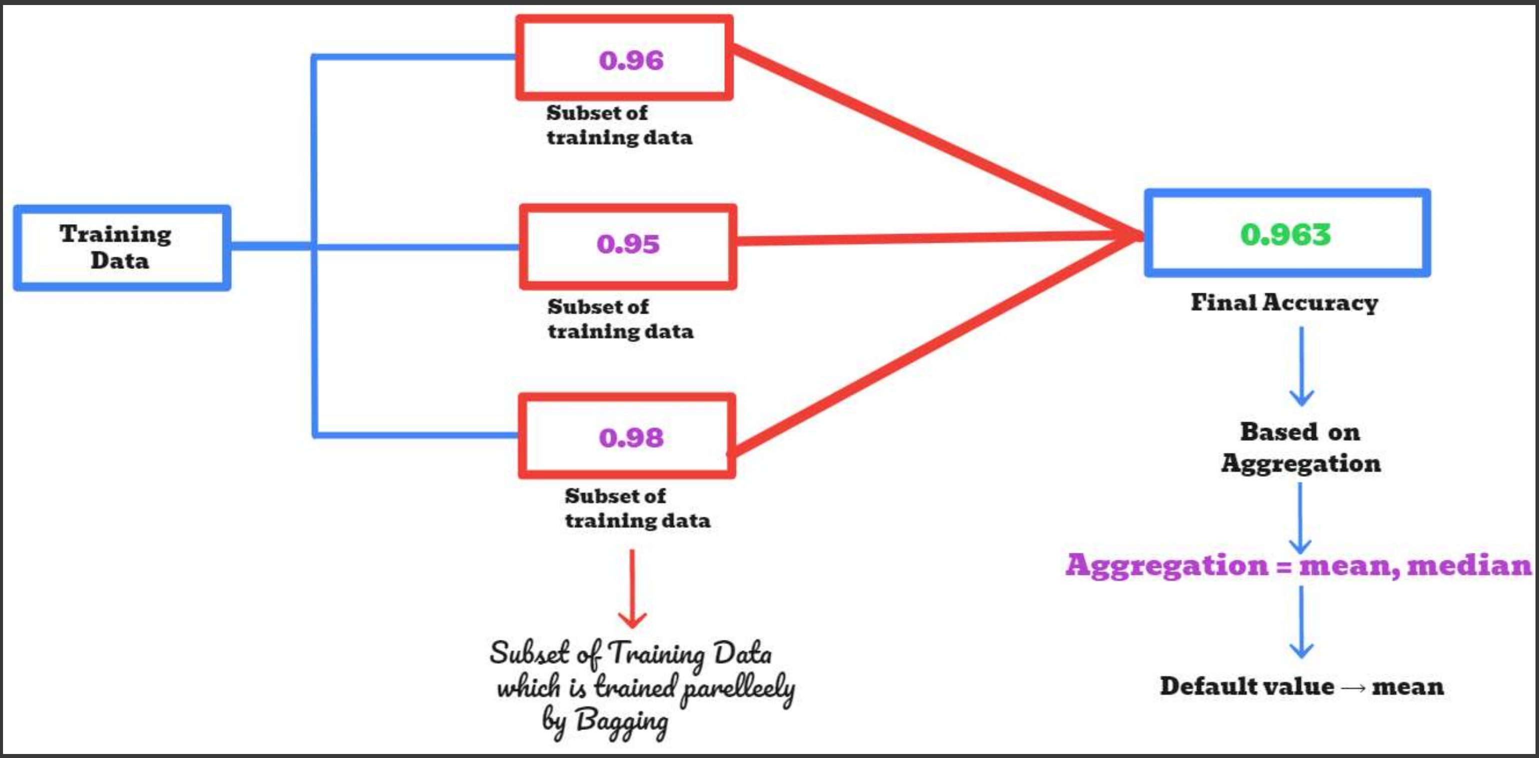
1. **Gini** calculation and **Entropy** calculation remain same for Random Forest as Decision Tree.**
2. All other concepts will be same as Decision Tree.
3. Random Forest and Logistic Regression are used most.

Types of Ensemble Techniques :

1. **Bagging**
2. **Boosting**

Bagging $\longrightarrow$

Bagging is used when our objective is to reduce the variable of a decision tree.



Bagging

- Create a few subsets of data from the training sample.
- Now each collection of subsets data is used to prepare their decision trees.
- The aggregation (mean or median) of all assumption is taken as final accuracy.

Aggregation :

Calulate mean or median of all subsets of Training Data.

⟶ Bagging is also known as **Bootstrap Aggregation**.

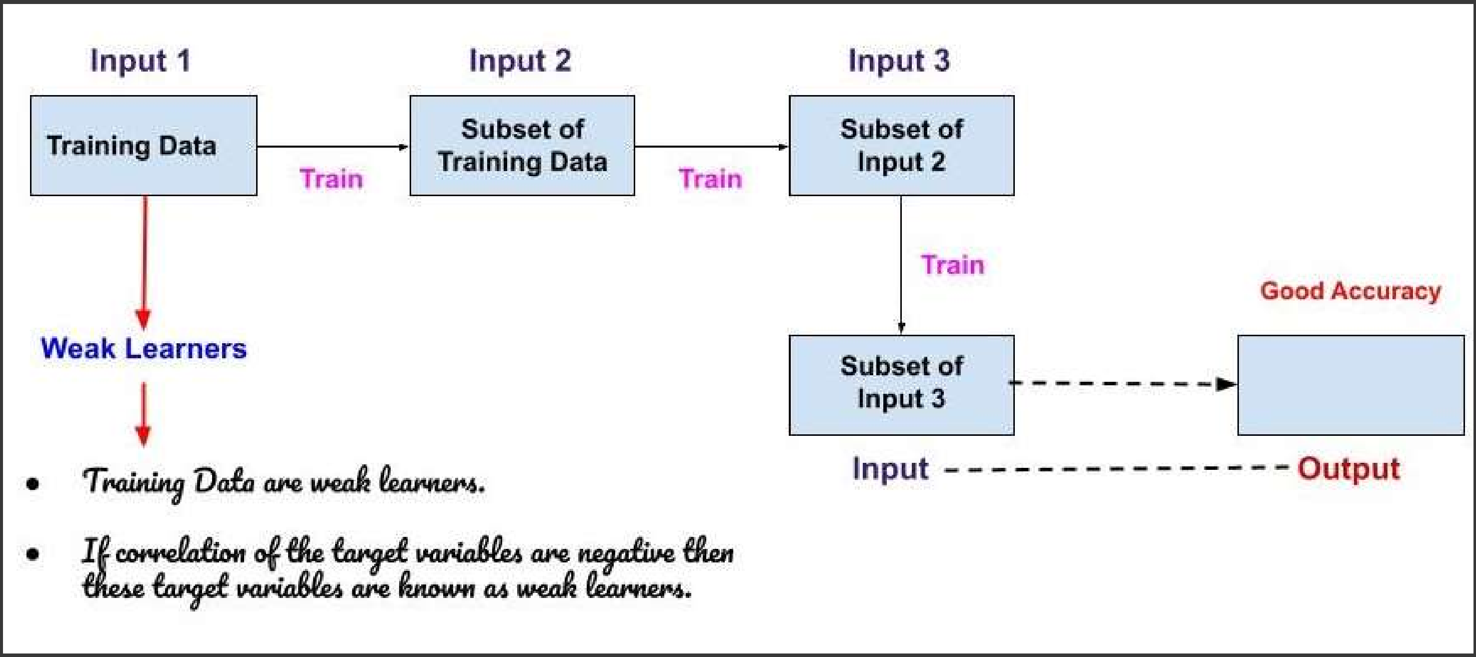
⟶ Bagging or Bootstrap Aggregation is designed to improve accuracy and reducing impurity in the algorithm.

Boosting ⟶

- **Boosting is another ansemble procedure to make a collection of predictors. In other words, we fit cosecutive trees usually random samples, and at each step, the objective is to solve net error from the prior trees.**
- **If a given input is misclassified by theory, then its weight is increased so that the upcoming hypothesis is more likely to classify it correctly by combining the entire set at least converts weak learners into better performing models.**

Types of Boosting :

- **Adaptive Boosting (Ada Boost)**
- **Gradient Boost**
- **eXtreame Gradient Boost (XG Boost)**
- **Light Gradient Boosting Machine (LGBM)**
- **Categorical Boosting (Cat Boost)**
- **Ada Net ⟶ Debveloped by Google :** Basically used in Deep Learning



Boosting

- Boosting trains data sequentially. It trains the trained data till it gets good accuracy.
- Very less chance of overfitting as compare to Random Forest.

Weak Learners :

- Training data are weak learners.
- If correlation of the target variables are negative then these target variables are known as weak learners.

---

eXtreme Gradient Boosting [XGBoost] ⟶

- In this XGBoosting algorithm, decision trees are created in sequential form.
- Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.
- The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ansemble to give a strong and more precious model.
- It can work on regression, classification, ranking, and user-defined prediction problems.
- XGBoost is designed to be both *fast to excute and highly effective*, perheps more effective than other open-source implementations.
- XGBoost takes weak learners under consideration first and sequentially trains them into strong learners.
- Relatively low speed and high accuracy in some cases as compared to Random Forest and Bagging.

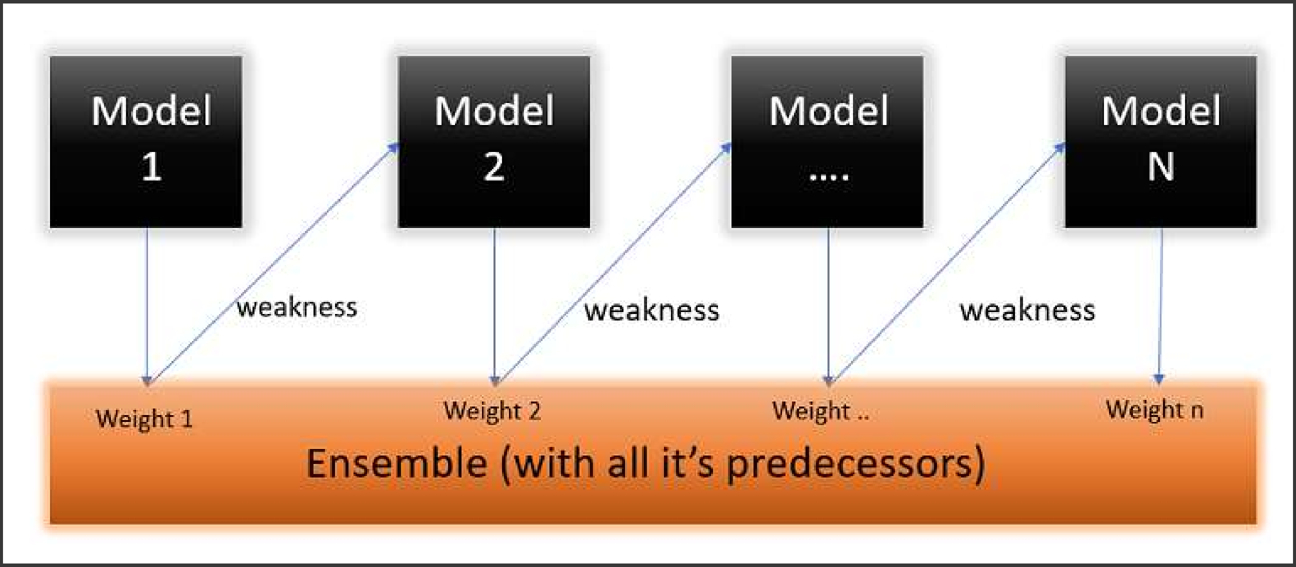
XGBoost Optimizations :

XGBoost features various optimizaations built to make the training faster when working with large datasets, in addition to its unique method of **generating** and **pruning** trees,

---

Adaptive Boosting [AdaBoost] Algorithm ⟶

- AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called Decision Stumps.
- This algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lowe error is received.



Light Gradient Boosting Machine [LightGBM] ⟶

- **LightGBM is a fast, distributed, high performance gradient boosting framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.**
- **LightGBM can handle the large size of data and takes lower memory to run.**
- **It is designed to be distributed and efficient with the following advantages:**
  1. **Faster training speed and higher efficiency.**
  2. **Lower memory usage.**
  3. **Better accuracy.**
  4. **Support of parallel and GPU learning.**
  5. **Capable of handling large-scale data.**
- **Light GBM is so popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.**
- **It is not advisable to use LGBM on small datasets. Light GBM is sensitive to overfitting and can easily overfit small data.**

LightGBM Parameters :

Control Parameters :

- max_depth, min_data_in_leaf, feature_fraction, bagging_fraction, lambda, min_gain_to_split, max_cat_group.

Core Parameters :

- Task, Boosting, learning_rate, num_leaves (default : 31), device (default : CPU)

Metric Parameters :

- mae (mean absolute error), mse (mean square error), binary_logloss, multi_logloss

---

Categorical Boosting [CatBoost] ⟶

- **CatBoost or Categorical Boosting is an open-source boosting library developed by Yandex. In addition to regression and classification, CatBoost can be used in ranking, recommendation systems, forecasting and even personal assistants.**
- **It has effective usage with default parameters thereby reducing the time needed for parameter tuning.**

AdaNet ⟶

Developed by Google, will be discussed in **Deep Learning**

- AdaNet is a lightweight TensorFlow-based framework that can automatically learn high-quality models with minimal expert intervention.
- AdaNet provides a general framework for not only learning a neural network architecture, but also for learning to ensemble to obtain even better models.
- AdaNet is also capable of automatically tuning the number of parameters in an ensemble to optimize performance.
- AdaNet is easy to use, and creates high-quality models, saving the time normally spent selecting optimal neural network architectures.

Python Implementation For Ensemble Learning Algorithm :

Mounting Google Drive

```
# Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_
```

Importing Libraries

```
# Imporitng Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('/content/drive/MyDrive/Dataset/Flight_Satisfaction.csv')
```

```
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214

```
# To show all columns
pd.set_option('display.max_column', None)
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214

```
df.shape
```

```
(103904, 25)
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	103904.0	51951.500000	29994.645522	0.0	25975.75	51951.5	77927.25	103903.0
id	103904.0	64924.210502	37463.812252	1.0	32533.75	64856.5	97368.25	129880.0
Age	103904.0	39.379706	15.114964	7.0	27.00	40.0	51.00	85.0
Flight Distance	103904.0	1189.448375	997.147281	31.0	414.00	843.0	1743.00	4983.0
Inflight wifi service	103904.0	2.729683	1.327829	0.0	2.00	3.0	4.00	5.0
Departure/Arrival time convenient	103904.0	3.060296	1.525075	0.0	2.00	3.0	4.00	5.0
Ease of Online booking	103904.0	2.756901	1.398929	0.0	2.00	3.0	4.00	5.0
Gate location	103904.0	2.976883	1.277621	0.0	2.00	3.0	4.00	5.0
Food and drink	103904.0	3.202129	1.329533	0.0	2.00	3.0	4.00	5.0
Online boarding	103904.0	3.250375	1.349509	0.0	2.00	3.0	4.00	5.0
Seat comfort	103904.0	3.439396	1.319088	0.0	2.00	4.0	5.00	5.0
Inflight entertainment	103904.0	3.358158	1.332991	0.0	2.00	4.0	4.00	5.0
On-board service	103904.0	3.382363	1.288354	0.0	2.00	4.0	4.00	5.0
Leg room service	103904.0	3.351055	1.315605	0.0	2.00	4.0	4.00	5.0
Baggage handling	103904.0	3.631833	1.180903	1.0	3.00	4.0	5.00	5.0
Checkin service	103904.0	3.304290	1.265396	0.0	3.00	3.0	4.00	5.0
Inflight service	103904.0	3.640428	1.175663	0.0	3.00	4.0	5.00	5.0
Cleanliness	103904.0	3.286351	1.312273	0.0	2.00	3.0	4.00	5.0
Departure Delay in Minutes	103904.0	14.815618	38.230901	0.0	0.00	0.0	12.00	1592.0
Arrival Delay in Minutes	103594.0	15.178678	38.698682	0.0	0.00	0.0	13.00	1584.0

Checking For Null Values

```
df.isna().apply(pd.value_counts).T
```

	False	True
Unnamed: 0	103904.0	NaN
id	103904.0	NaN
Gender	103904.0	NaN
Customer Type	103904.0	NaN
Age	103904.0	NaN
Type of Travel	103904.0	NaN
Class	103904.0	NaN
Flight Distance	103904.0	NaN
Inflight wifi service	103904.0	NaN
Departure/Arrival time convenient	103904.0	NaN
Ease of Online booking	103904.0	NaN
Gate location	103904.0	NaN
Food and drink	103904.0	NaN
Online boarding	103904.0	NaN
Seat comfort	103904.0	NaN
Inflight entertainment	103904.0	NaN
On-board service	103904.0	NaN
Leg room service	103904.0	NaN
Baggage handling	103904.0	NaN
Checkin service	103904.0	NaN
Inflight service	103904.0	NaN
Cleanliness	103904.0	NaN
Departure Delay in Minutes	103904.0	NaN
Arrival Delay in Minutes	103594.0	310.0
satisfaction	103904.0	NaN

Filling Null Values

```
df['Arrival Delay in Minutes'].fillna(df['Arrival Delay in Minutes'].median(), inplace = True)
```

```
df.isna().sum()
```

```
Unnamed: 0                           0
id                                   0
Gender                               0
Customer Type                        0
Age                                  0
Type of Travel                       0
Class                                0
Flight Distance                      0
Inflight wifi service                0
Departure/Arrival time convenient    0
Ease of Online booking               0
Gate location                        0
Food and drink                       0
Online boarding                      0
Seat comfort                         0
Inflight entertainment               0
On-board service                     0
```

```
Leg room service                     0
Baggage handling                     0
Checkin service                      0
Inflight service                     0
Cleanliness                          0
Departure Delay in Minutes           0
Arrival Delay in Minutes             0
satisfaction                         0
dtype: int64
```

Data Preprocessing

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
df['Gender'] = le.fit_transform(df['Gender'])
df['Customer Type'] = le.fit_transform(df['Customer Type'])
df['Type of Travel'] = le.fit_transform(df['Type of Travel'])
df['Class'] = le.fit_transform(df['Class'])
df['satisfaction'] = le.fit_transform(df['satisfaction'])
```

Train Test Split

```
# Train Test Split

x = df.drop(['Unnamed: 0','id','satisfaction'], axis = 1)
y = df[['satisfaction']]

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size = 0.3, random_state= 1)
```

```
xtrain.shape, xtrain.shape
```

```
((72732, 22), (72732, 22))
```

```
ytrain.shape, ytest.shape
```

```
((72732, 1), (31172, 1))
```

Decision Tree Algorithm

```
# Decision Tree Algorithm
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = 11)
dt.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=11)
```

```
dt_pred = dt.predict(xtest)
dt_pred
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

```
# Test Score
dt.score(xtest, ytest)
```

```
0.9515270114205056
```

```
# Training Score
dt.score(xtrain, ytrain)
```

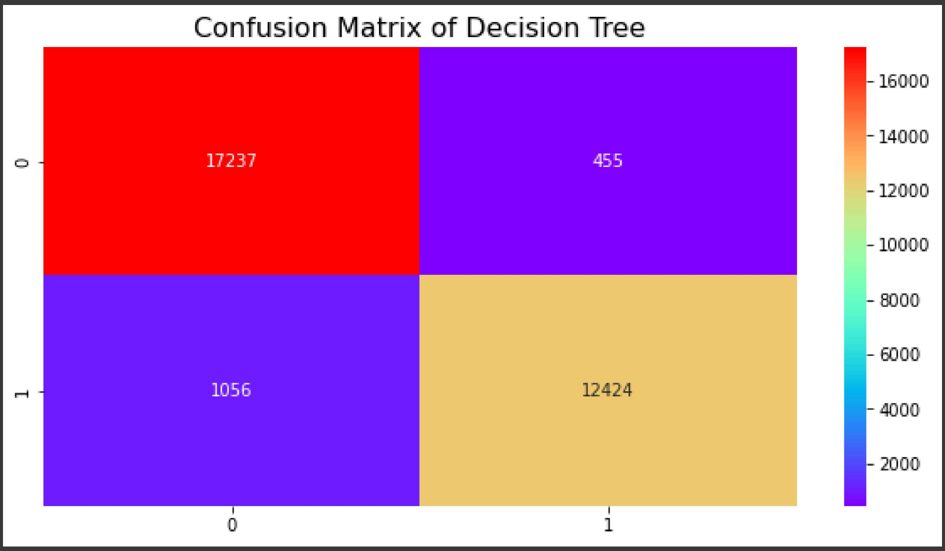
```
0.9587114337568058
```

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

```
cf = confusion_matrix(ytest, dt_pred)
print(cf)
```

```
[[17237   455]
 [ 1056 12424]]
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Decision Tree', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, dt_pred), fmt = 'g', annot = True, cmap = 'rainbow')
plt.show()
```



```
print(classification_report(ytest, dt_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96     17692
           1       0.96      0.92      0.94     13480

    accuracy                           0.95     31172
   macro avg       0.95      0.95      0.95     31172
weighted avg       0.95      0.95      0.95     31172
```

Seventh Machine Algorithm

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
```

```
rf.fit(xtrain, ytrain)
```

```
RandomForestClassifier()
```

```
rf_pred = rf.predict(xtest)
rf_pred
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

```
# Random Forest Test Score
rf.score(xtest, ytest)
```

```
0.9616322340562042
```

```
# Random Forest Training Score
rf.score(xtrain, ytrain)
```

```
0.9999862508936919
```

```
# Overfitting Occurs
rf = RandomForestClassifier(n_estimators = 500, criterion = 'entropy', max_depth = 11)
rf.fit(xtrain, ytrain)
rf_pred = rf.predict(xtest)
```

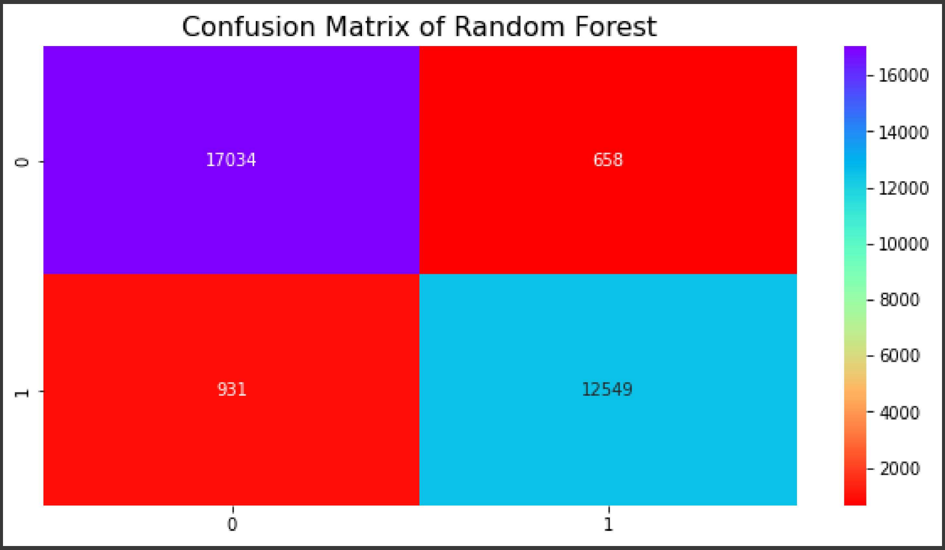
```
# Test Score after applyingparameters
rf.score(xtest, ytest)
```

```
0.9490247658154755
```

```
# Train Score after applyingparameters
rf.score(xtrain, ytrain)
```

```
0.9550266732662377
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Random Forest', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, rf_pred), fmt = 'g', annot = True, cmap = 'rainbow_r')
plt.show()
```



```
print(classification_report(ytest, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96     17692
           1       0.95      0.93      0.94     13480

    accuracy                           0.95     31172
   macro avg       0.95      0.95      0.95     31172
weighted avg       0.95      0.95      0.95     31172
```

Bagging

```
# Bagging
from sklearn.ensemble import BaggingClassifier
bgc = BaggingClassifier()
```

```
bgc.fit(xtrain, ytrain)
bgc.predict(xtest)
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

```
# Bagging Test Score
bgc.score(xtest, ytest)
```

```
0.9585525471577057
```

```
# Bagging Train Score
bgc.score(xtrain, ytrain)
```

```
0.9965902216355936
```

```
# Overfitting Occurs
```

```
from sklearn.ensemble import BaggingClassifier
bgc = BaggingClassifier(base_estimator=dt, n_estimators=100, random_state=1)
bgc.fit(xtrain, ytrain)
bgc_pred = bgc.predict(xtest)
```

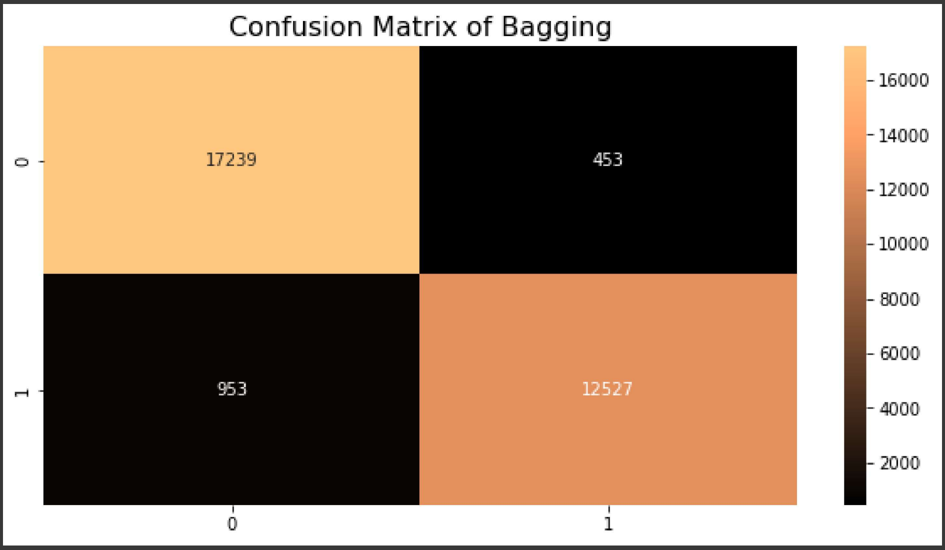
```
# Test Score after applying Bagging parameters
bgc.score(xtest, ytest)
```

```
0.9548954189657385
```

```
# Train Score after applying Bagging parameters
bgc.score(xtrain, ytrain)
```

```
0.9643210691305065
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Bagging', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, bgc_pred), fmt = 'g', annot = True, cmap = 'copper')
plt.show()
```



```
print(classification_report(ytest, bgc_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     17692
           1       0.97      0.93      0.95     13480

    accuracy                           0.95     31172
   macro avg       0.96      0.95      0.95     31172
weighted avg       0.96      0.95      0.95     31172
```

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(xtrain, ytrain)
ada_pred = ada.predict(xtest)
```

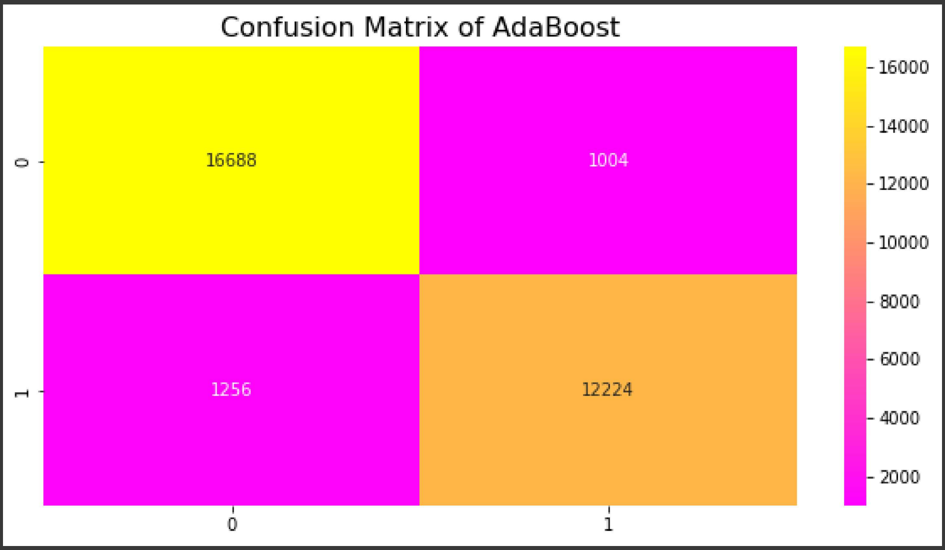
```
# AdaBoost Test Score
ada.score(xtest, ytest)
```

```
0.9274990375978442
```

```
# AdaBoost Training Score
ada.score(xtrain, ytrain)
```

```
0.9285596436231645
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of AdaBoost', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, ada_pred), fmt = 'g', annot = True, cmap = 'spring')
plt.show()
```



```
print(classification_report(ytest, ada_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     17692
           1       0.92      0.91      0.92     13480

    accuracy                           0.93     31172
   macro avg       0.93      0.93      0.93     31172
weighted avg       0.93      0.93      0.93     31172
```

Gradient Boosting Classifier

```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(xtrain, ytrain)
gbc_pred = gbc.predict(xtest)
```

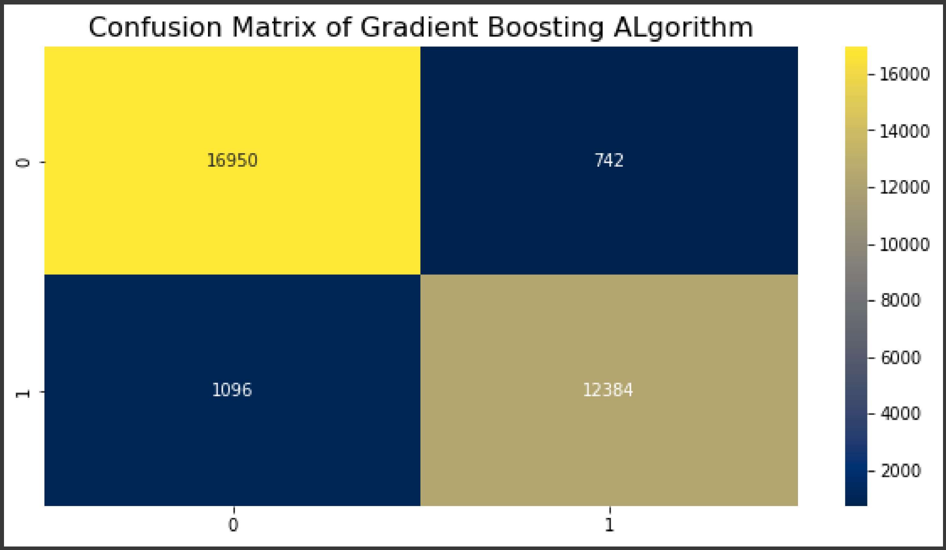
```
# Gradient Boost Test Score
gbc.score(xtest,ytest)
```

```
0.9410368279224945
```

```
# Gradient Boost Training Score
gbc.score(xtrain, ytrain)
```

```
0.9422537535060221
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of Gradient Boosting ALgorithm', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, gbc_pred), fmt = 'g', annot = True, cmap = 'cividis')
plt.show()
```



```
print(classification_report(ytest, gbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     17692
           1       0.94      0.92      0.93     13480

    accuracy                           0.94     31172
   macro avg       0.94      0.94      0.94     31172
weighted avg       0.94      0.94      0.94     31172
```

XGBoost

```
# XGBoost

# !pip install xgboost
```

```
import xgboost as xgb
xgbc = xgb.XGBClassifier()
xgbc.fit(xtrain, ytrain)
```

```
XGBClassifier()
```

```
xgbc_pred = xgbc.predict(xtest)
xgbc_pred
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

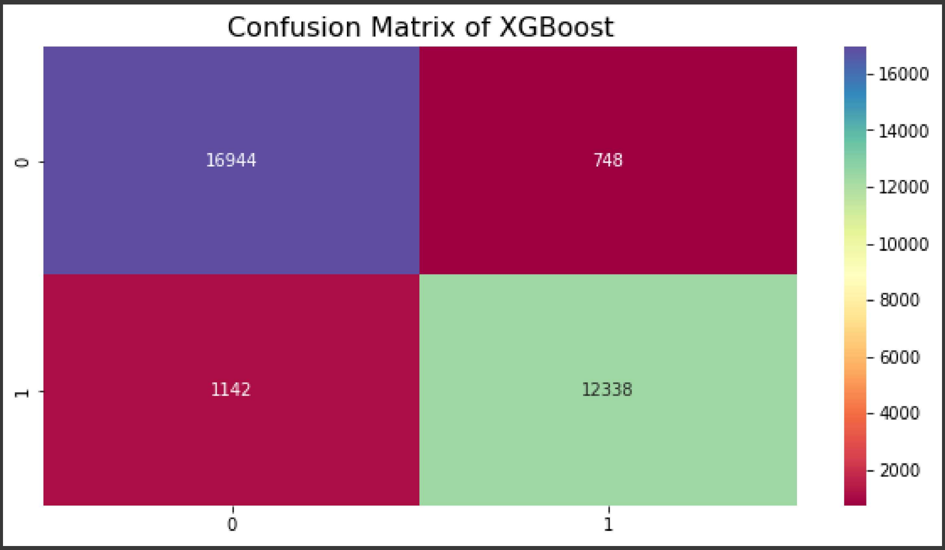
```
# XGBoost Test Score
xgbc.score(xtest, ytest)
```

```
0.9393686641858078
```

```
# XGBoost Training Score
xgbc.score(xtrain, ytrain)
```

```
0.9402738821976572
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of XGBoost', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, xgbc_pred), fmt = 'g', annot = True, cmap = 'Spectral')
plt.show()
```



```
print(classification_report(ytest, xgbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     17692
           1       0.94      0.92      0.93     13480

    accuracy                           0.94     31172
   macro avg       0.94      0.94      0.94     31172
weighted avg       0.94      0.94      0.94     31172
```

LightBGM

```
# LightGBM
import lightgbm as lg
lgbc = lg.LGBMClassifier()
lgbc.fit(xtrain, ytrain)
```

```
LGBMClassifier()
```

```
lgbc_pred = lgbc.predict(xtest)
lgbc_pred
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

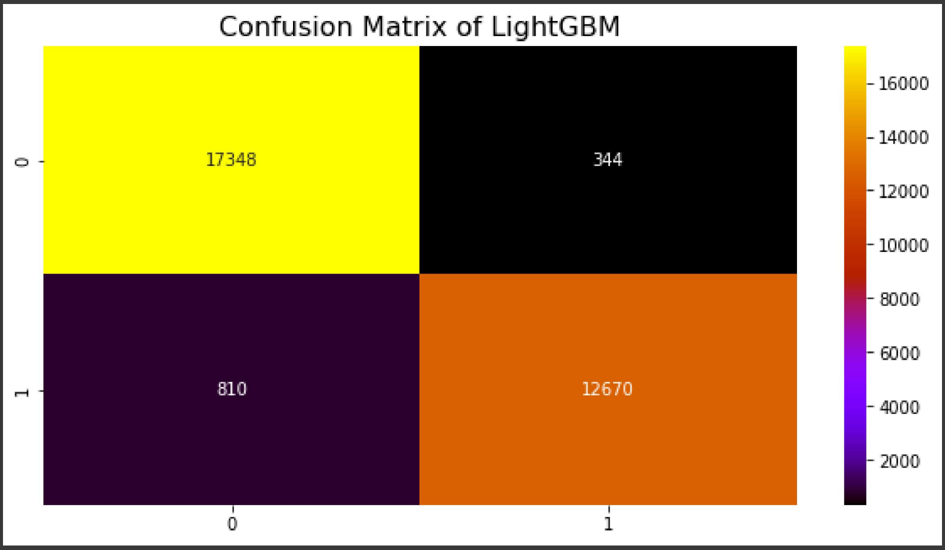
```
# LightGBM Test Score
lgbc.score(xtest, ytest)
```

```
0.9629795970742975
```

```
# LightGBM Training Score
lgbc.score(xtrain, ytrain)
```

```
0.9671121377110488
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of LightGBM', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, lgbc_pred), fmt = 'g', annot = True, cmap = 'gnuplot')
plt.show()
```



```
print(classification_report(ytest, lgbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     17692
           1       0.97      0.94      0.96     13480

    accuracy                           0.96     31172
   macro avg       0.96      0.96      0.96     31172
weighted avg       0.96      0.96      0.96     31172
```

CatBoost Algorithm

```
# CatBoost
# !pip install catboost
```

```
import catboost as ctb
ctb = ctb.CatBoostClassifier()
ctb.fit(xtrain, ytrain)
```

```
ctb_pred = ctb.predict(xtest)
```

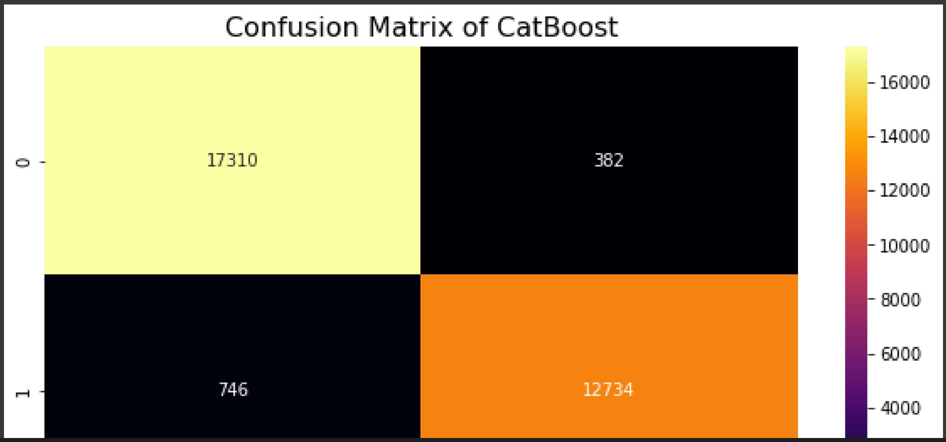
```
# CatBoost Test Score
ctb.score(xtest, ytest)
```

```
0.9638136789426408
```

```
# CatBoost Training Score
ctb.score(xtrain, ytrain)
```

```
0.9766952648077875
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix of CatBoost', fontsize = 16)
sns.heatmap(confusion_matrix(ytest, ctb_pred), fmt = 'g', annot = True, cmap = 'inferno')
plt.show()
```



```
print(classification_report(ytest, ctb_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     17692
           1       0.97      0.94      0.96     13480

    accuracy                           0.96     31172
   macro avg       0.96      0.96      0.96     31172
weighted avg       0.96      0.96      0.96     31172
```

✓ 0s completed at 9:06 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.