
## Machine Learning 🖵

---

## Natural Language Toolkit

# Text Analytics

𝓑𝔂 ⟹ 𝓟𝓡𝓘𝓝𝓒𝓔👑💖

https://github.com/pkvidyarthi/

---

## Porter Stemmer ⟶

- Stemming is a significant part of the pipelining procedure in Natural Language Processing.
- Stemming is the process of generating morphological modifications of a root/base word. Stemming programs are generally considered as stemming algorithms or stemmers.
- A stemming algorithm reduces the words like "retrieves", "retrieved", "retrieval" to the root word, "retrieve" and "Choco", "Chocolatey", "Chocolates" reduce to the stem "chocolate".

```
# Porter Stemming
words = ['program', 'programming', 'programmers', 'programmer', 'programs']
import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()

for x in words:
  print(x, " : ", ps.stem(x))
```

```
program  :  program
programming  :  program
programmers  :  programm
programmer  :  programm
programs  :  program
```

```
words2 = ["consult", "consultant", "consulting", "consultantative", "consultants", "consulting"]
import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

```
for x in words2:
  print(x, " : ", ps.stem(x))
```

```
consult  :  consult
consultant  :  consult
consulting  :  consult
consultantative  :  consult
consultants  :  consult
consulting  :  consult
```

## Stop Words ⟶

- Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.
- We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words.
- NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory.

## Text Analytics ⟶

```
import nltk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

### Mountong Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. <br /><br />The... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |

```
df.shape
```

```
(50000, 2)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

```
df['sentiment'].value_counts()
```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

### Preprocessing ⟶ LabelEncoder

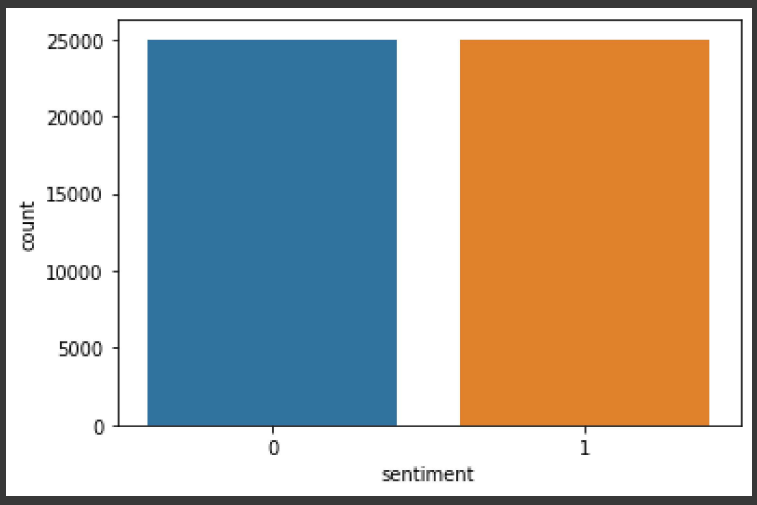
Preprocessing without using LabelEncoder, will use List Comperihension.

```
df['sentiment'] = [1 if sentiment == 'positive' else 0 for sentiment in df['sentiment']]
df['sentiment'].value_counts()
```

```
1    25000
0    25000
Name: sentiment, dtype: int64
```

## Plotting Countplot

```
sns.countplot(df['sentiment'])
plt.show()
```




```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

## What is stopwords ?

```
# Stopwords
stopwords.words('english')
```

```
      'same',
      'so',
      'than',
      'too',
      'very',
      's',
      't',
      'can',
      'will',
      'just',
      'don',
      "don't",
      'should',
      "should've",
      'now',
      'd',
      'll',
      'm',
      'o',
      're',
      've',
      'y',
      'ain',
      'aren',
      "aren't",
      'couldn',
      "couldn't",
      'didn',
      "didn't",
      'doesn',
      "doesn't",
      'hadn',
      "hadn't",
      'hasn',
      "hasn't",
      'haven',
      "haven't",
      'isn',
      "isn't",
      'ma',
      'mightn',
      "mightn't",
      'mustn',
      "mustn't",
      'needn',
      "needn't",
      'shan',
      "shan't",
      'shouldn',
      "shouldn't",
      'wasn',
      "wasn't",
      'weren',
      "weren't",
      'won',
      "won't",
      'wouldn',
      "wouldn't"]
```

## Regular Expression ⟶

- Regular expression is a set of characters, called as the pattern, which helps in finding substrings in a given string. The pattern is used to detect the substrings
- Example : suppose we have a dataset of customer reviews about a restaurant and we want to extract the emojis from the reviews because they are a good predictor os the sentiment of the review.
- Regular expressions are very powerful tool in text processing. It will help to clean and handle text in a much better way.

## tqdm ⟶

**Python external library tqdm :** to create simple and hassle-free progress bars which can add to code and make it look lively.

```
# ! pip install tdqm
```

```
# Regular Expression
import re
# tqdm
from tqdm import tqdm
```

```
corpus = []
for i in tqdm(range(0, len(df))):
  # to read every sentence in every column
  sentence = re.sub('[^a-zA-Z]', ' ', df['review'] [i])
  sentence = sentence.lower()
  sentence = sentence.split()
  sentence = [ps.stem(word) for word in sentence if not word in stopwords.words('english')]
  sentence = ' '.join(sentence)
  corpus.append(sentence)
```

```
100%|██████████| 50000/50000 [19:02<00:00, 43.75it/s]
```

### Bag of Words Model

```
# Bag of Words Model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 2500)
# Independent Variable
x1 = cv.fit_transform(corpus).toarray()
# Dependent Variable
y1 = pd.get_dummies(df['sentiment'])
y1 = y1.iloc[:,1].values
```

```
from sklearn.model_selection import train_test_split
xtrain1, xtest1, ytrain1, ytest1 = train_test_split(x1, y1, test_size = 0.2, random_state = 1)
```

```
xtrain1.shape, xtest1.shape
```

```
((40000, 2500), (10000, 2500))
```

```
ytrain1.shape, ytest1.shape
```

```
((40000,), (10000,))
```

## Naive Bayes Algorithm

```
# Naive Bayes
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(xtrain1, ytrain1)
```

```
MultinomialNB()
```

```
ypred1 = model.predict(xtest1)
ypred1
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=uint8)
```

```
# Importing accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
accuracy_score(ytest1, ypred1)
```

```
0.8442
```

```
model.score(xtrain1, ytrain1)
```

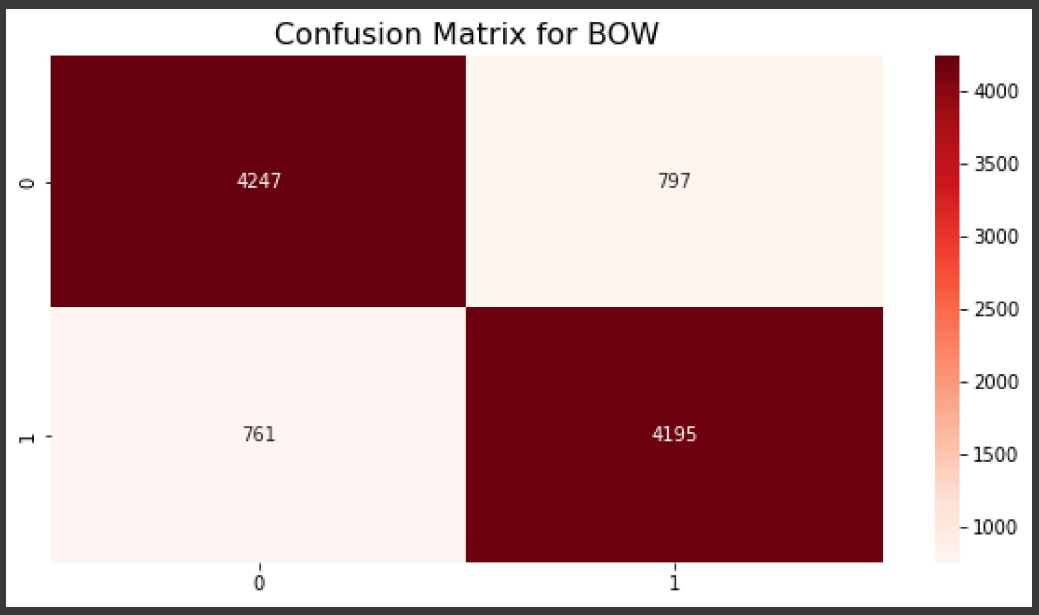
```
0.84725
```

## Confusion Matrix

```
# Confusion Matrix
cf1 = confusion_matrix(ytest1, ypred1)
cf1
```

```
array([[4247,  797],
       [ 761, 4195]])
```

```
plt.figure(figsize = (10,5))
plt.title("Confusion Matrix for BOW", fontsize = 16)
sns.heatmap(cf1, fmt = 'g', annot = True, cmap = 'Reds')
plt.show()
```




## Classification Report

```
# Classification Report
print(classification_report(ytest1, ypred1))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.84      0.85      5044
               1       0.84      0.85      0.84      4956

        accuracy                           0.84     10000
       macro avg       0.84      0.84      0.84     10000
    weighted avg       0.84      0.84      0.84     10000
```

```
# Tfidf Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features = 2500)
# Independent Variable
x = tfidf.fit_transform(corpus).toarray()
# Dependent Variable
y = pd.get_dummies(df['sentiment'])
y = y.iloc[:,1].values
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, random_state = 1)
```

### Random Forest Algorithm

```
# Random Forest Algorithm
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 500, criterion = 'entropy', max_depth = 10)
rf.fit(xtrain, ytrain)
```

```
RandomForestClassifier(criterion='entropy', max_depth=10, n_estimators=500)
```

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

```
array([0, 0, 0, ..., 1, 1, 0], dtype=uint8)
```

```
# accuracy score
rf.score(xtest, ytest)
```

```
0.8238
```

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

```
0.85635
```

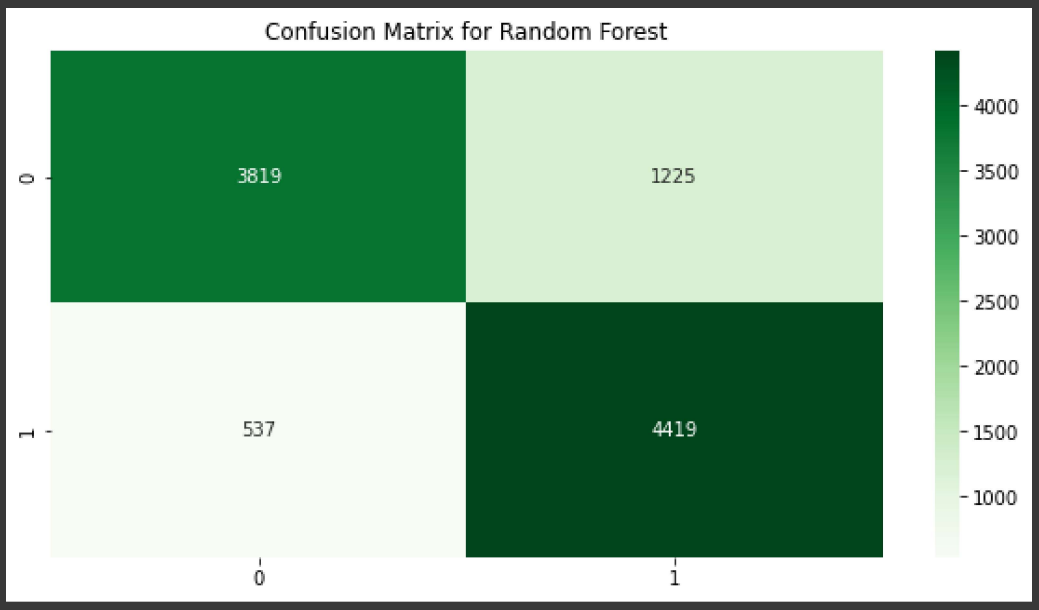
### Cnfusion Matrix for Random Forest

```
cf2 = confusion_matrix(ytest, rf_pred)
cf2
```

```
array([[3819, 1225],
       [ 537, 4419]])
```

```
plt.figure(figsize = (10, 5))
plt.title("Confusion Matrix for Random Forest")
```

```
sns.heatmap(cf2, fmt = 'g', annot = True, cmap = "Greens")
plt.show()
```




### Classification Report for Random Forest

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

```
                precision    recall  f1-score   support

             0       0.88      0.76      0.81      5044
             1       0.78      0.89      0.83      4956

      accuracy                           0.82     10000
     macro avg       0.83      0.82      0.82     10000
  weighted avg       0.83      0.82      0.82     10000
```

### Ada Boost Algorithm

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

```
ada_pred
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=uint8)
```

```
# Accuracy Score
ada.score(xtest, ytest)
```

```
0.8044
```

```
ada.score(xtrain, ytrain)
```

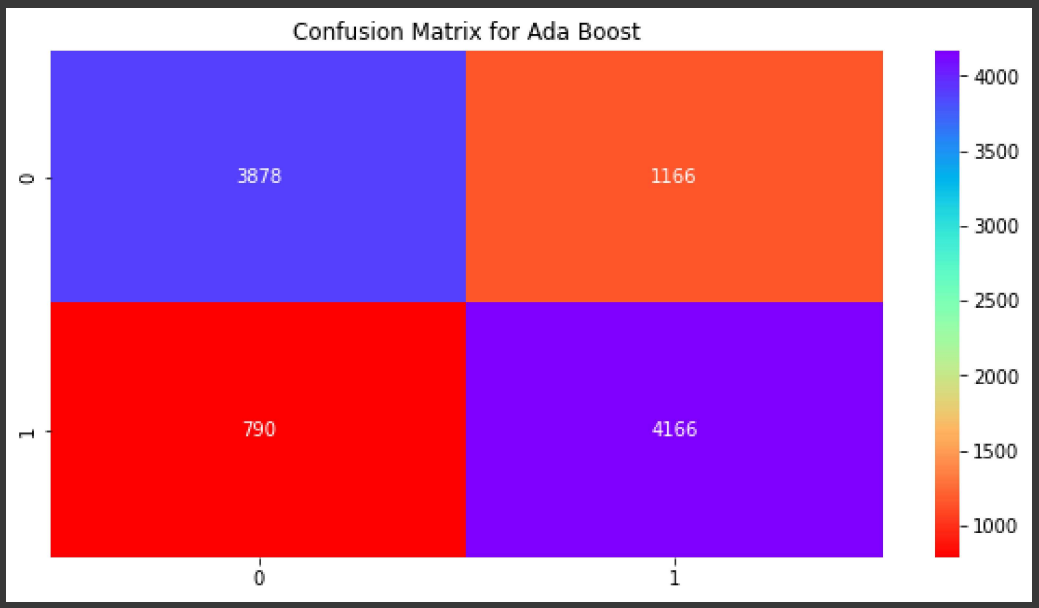
```
0.812
```

### Confusion Matrix for AdaBoost

```
cf3 =  confusion_matrix(ytest, ada_pred)
cf3
```

```
array([[3878, 1166],
       [ 790, 4166]])
```

```
plt.figure(figsize = (10, 5))
plt.title("Confusion Matrix for Ada Boost")
sns.heatmap(cf3, fmt = 'g', annot = True, cmap = 'rainbow_r')
plt.show()
```




### Classification Report for Ada Boost

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

```
                precision    recall  f1-score   support

           0       0.83      0.77      0.80      5044
           1       0.78      0.84      0.81      4956

    accuracy                           0.80     10000
   macro avg       0.81      0.80      0.80     10000
weighted avg       0.81      0.80      0.80     10000
```

### Gradient BoostingAlgorithm

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

```
array([0, 0, 0, ..., 1, 1, 0], dtype=uint8)
```

```
# Accuracy Score
gbc.score(xtest, ytest)
```

```
0.8128
```

```
gbc.score(xtrain, ytrain)
```

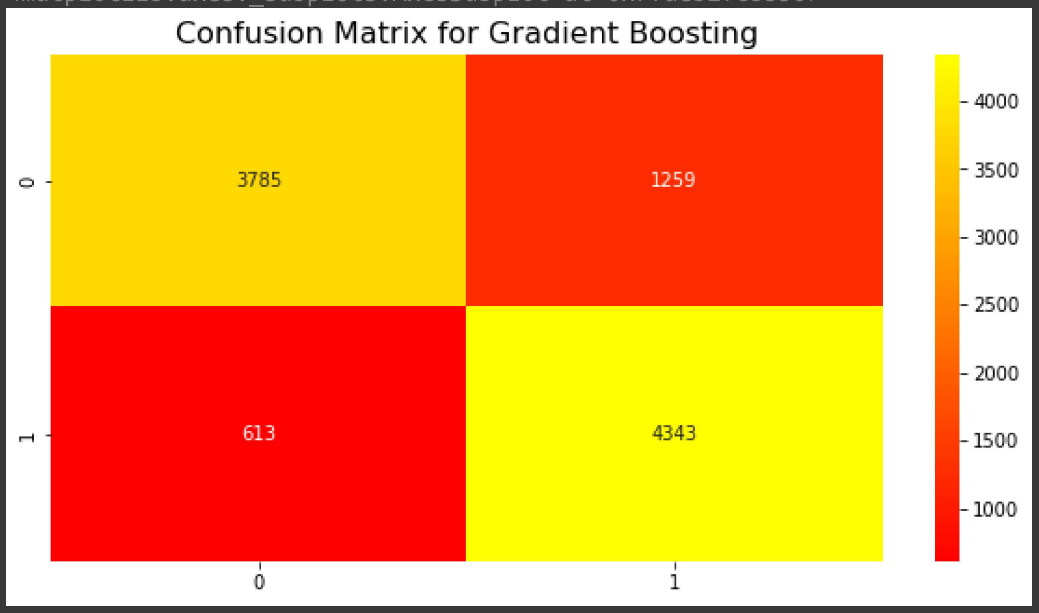
```
0.8273
```

```
# Confusion Matrix
cf4 = confusion_matrix(ytest, gbc_pred)
cf4
```

```
array([[3785, 1259],
       [ 613, 4343]])
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix for Gradient Boosting', fontsize = 16)
sns.heatmap(cf4, fmt = 'g', annot = True, cmap = 'autumn')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faeb2fcbb50>
```




```
# Classification Report
print(classification_report(ytest, gbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.75      0.80      5044
           1       0.78      0.88      0.82      4956

    accuracy                           0.81     10000
   macro avg       0.82      0.81      0.81     10000
weighted avg       0.82      0.81      0.81     10000
```

### XGBoost Algorithm

```
# XGBoost
from xgboost import XGBClassifier
xgbc = XGBClassifier()
xgbc.fit(xtrain, ytrain)
xgbc_pred = xgbc.predict(xtest)
```

```
xgbc_pred
```

```
array([0, 0, 1, ..., 1, 1, 0], dtype=uint8)
```

```
# Accuracy Score
xgbc.score(xtest, ytest)
```

```
0.8084
```

```
accuracy_score(ytest, xgbc_pred)
```

```
0.8084
```

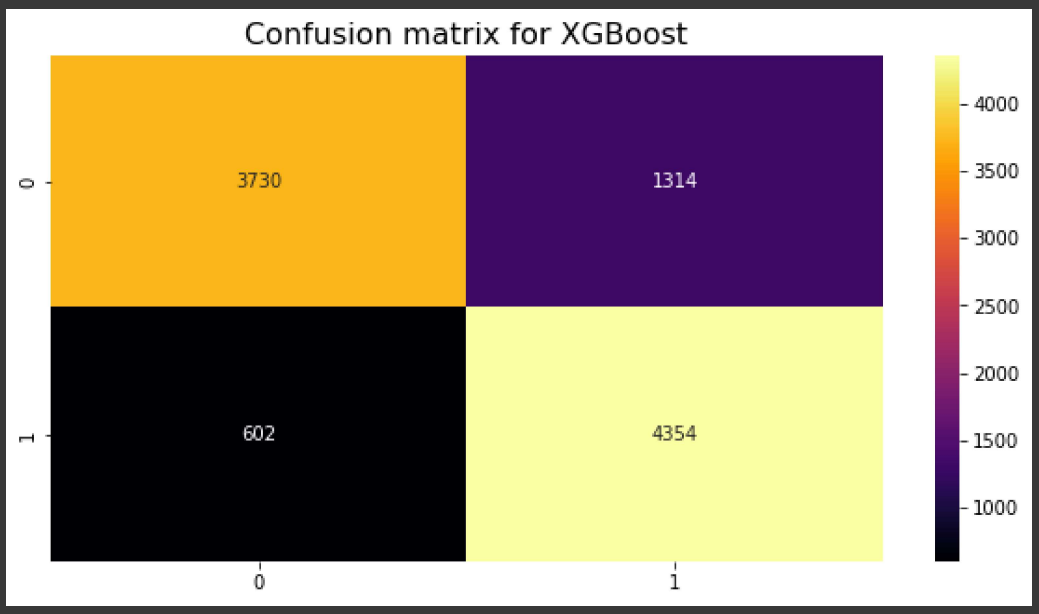
```
xgbc.score(xtrain, ytrain)
```

```
0.823525
```

```
# Confusion Matrix
xgbc_cf = confusion_matrix(ytest, xgbc_pred)
xgbc_cf
```

```
array([[3730, 1314],
       [ 602, 4354]])
```

```
plt.figure(figsize = (10,5))
plt.title("Confusion matrix for XGBoost", fontsize = 16)
sns.heatmap(xgbc_cf, fmt = 'g', annot =True, cmap = 'inferno')
plt.show()
```




```
# Classification Report
print(classification_report(ytest, xgbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.74      0.80      5044
           1       0.77      0.88      0.82      4956

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.82      0.81      0.81     10000
```

### LightGBM Algorithm

```
# LightBGM
import lightgbm as lgb
```

```
lgbc = lgb.LGBMClassifier()
lgbc.fit(xtrain, ytrain)
lgbc_pred = lgbc.predict(xtest)
```

```
lgbc_pred
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=uint8)
```

```
# Accuracy Score
lgbc.score(xtest, ytest)
```

```
0.8606
```

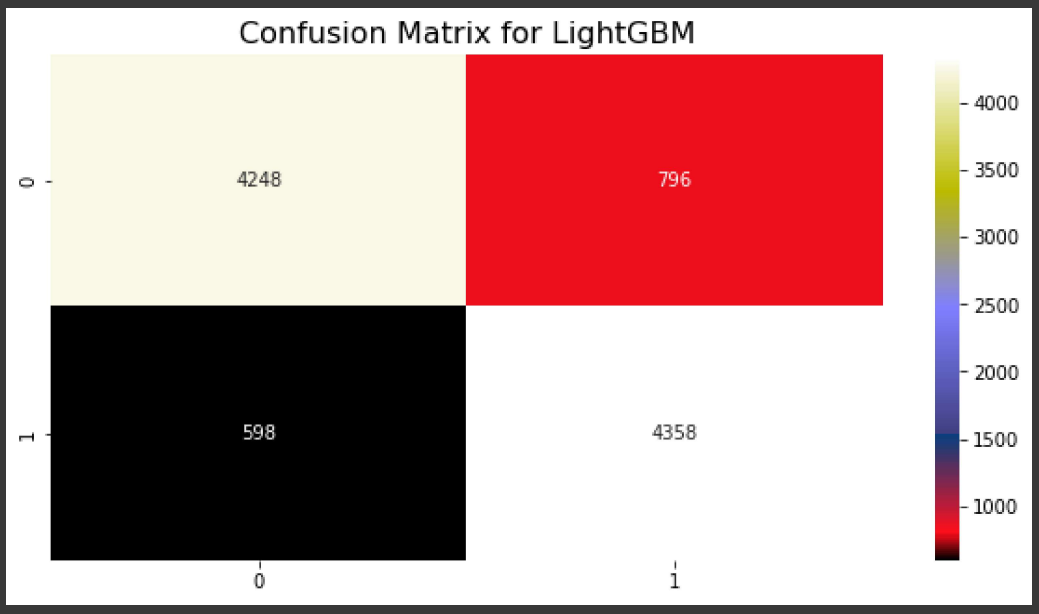
```
lgbc.score(xtrain, ytrain)
```

```
0.896125
```

```
# Confusion Matrix
lgbc_cf = confusion_matrix(ytest, lgbc_pred)
lgbc_cf
```

```
array([[4248,  796],
       [ 598, 4358]])
```

```
plt.figure(figsize = (10,5))
plt.title('Confusion Matrix for LightGBM', fontsize = 16)
sns.heatmap(lgbc_cf, fmt = 'g', annot = True, cmap = 'gist_stern')
plt.show()
```




```
# Classification Report
print(classification_report(ytest, lgbc_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      5044
           1       0.85      0.88      0.86      4956

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

## CatBoost Algorithm

```
! pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp38-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 12.6 MB/s eta 0:00:00
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.8/dist-packages (from catboost)
Requirement already satisfied: plotly in /usr/local/lib/python3.8/dist-packages (from catboost) (5.5.0)
Requirement already satisfied: graphviz in /usr/local/lib/python3.8/dist-packages (from catboost) (0.10
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from catboost) (3
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from catboost) (1.15.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.8/dist-packages (from catboost)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from catboost) (1.7.3)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from p
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=0.2
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplo
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib-
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.8/dist-packages (from plotly->
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
from catboost import CatBoostClassifier
ctb = CatBoostClassifier()
ctb.fit(xtrain, ytrain)
ctb_pred = ctb.predict(xtest)
ctb_pred
```

```
978:    learn: 0.2440632        total: 9m 49s   remaining: 12.6s
979:    learn: 0.2439588        total: 9m 50s   remaining: 12s
980:    learn: 0.2438583        total: 9m 50s   remaining: 11.4s
981:    learn: 0.2437352        total: 9m 51s   remaining: 10.8s
982:    learn: 0.2436390        total: 9m 52s   remaining: 10.2s
983:    learn: 0.2435368        total: 9m 52s   remaining: 9.64s
984:    learn: 0.2434421        total: 9m 53s   remaining: 9.03s
985:    learn: 0.2433998        total: 9m 53s   remaining: 8.43s
986:    learn: 0.2432960        total: 9m 54s   remaining: 7.83s
987:    learn: 0.2432047        total: 9m 54s   remaining: 7.23s
988:    learn: 0.2431233        total: 9m 55s   remaining: 6.62s
989:    learn: 0.2430025        total: 9m 56s   remaining: 6.02s
990:    learn: 0.2429124        total: 9m 56s   remaining: 5.42s
991:    learn: 0.2428235        total: 9m 57s   remaining: 4.82s
992:    learn: 0.2427199        total: 9m 57s   remaining: 4.21s
993:    learn: 0.2426092        total: 9m 58s   remaining: 3.61s
994:    learn: 0.2424928        total: 9m 59s   remaining: 3.01s
995:    learn: 0.2423925        total: 9m 59s   remaining: 2.41s
996:    learn: 0.2423251        total: 10m      remaining: 1.81s
997:    learn: 0.2422326        total: 10m      remaining: 1.2s
998:    learn: 0.2421328        total: 10m 1s   remaining: 602ms
999:    learn: 0.2420169        total: 10m 2s   remaining: 0us
array([0, 0, 0, ..., 0, 1, 0])
```

```
# Accuracy Score
ctb.score(xtest, ytest)
```

```
0.8692
```

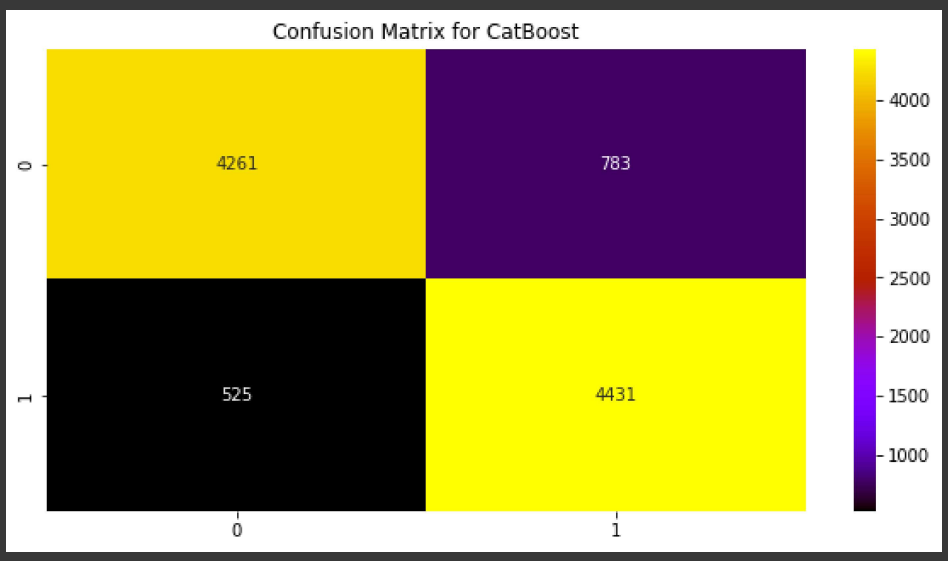
```
ctb.score(xtrain, ytrain)
```

```
0.91975
```

```
# Confusion Matrix
ctb_cf = confusion_matrix(ytest, ctb_pred)
ctb_cf
```

```
array([[4261,  783],
       [ 525, 4431]])
```

```
plt.figure(figsize =(10,5))
plt.title('Confusion Matrix for CatBoost')
sns.heatmap(ctb_cf, fmt = 'g', annot = True, cmap = 'gnuplot')
plt.show()
```




```
# Classification Report
```

```
# classification Report
print(classification_report(ytest, ctb_pred))
```

```
                precision    recall  f1-score   support

             0       0.89      0.84      0.87      5044
             1       0.85      0.89      0.87      4956

      accuracy                           0.87     10000
     macro avg       0.87      0.87      0.87     10000
  weighted avg       0.87      0.87      0.87     10000
```