chi_sq_87 8020 split .05 threshold

January 3, 2023

[ ]:
```
# Importing the packages
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxsize)
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import sklearn
import random
from sklearn.metrics import␣
↪confusion_matrix,accuracy_score,classification_report,RocCurveDisplay,ConfusionMatrixDispla
```

[ ]:
```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

[ ]:
```
# Importing the dataset
df = pd.read_csv('dataset_phishing.csv')
df.drop(['url'], axis=1, inplace=True)
#df.head(50)
```

[ ]:
```
# if your dataset contains missing value, check which column has missing values
#df.isnull().sum()
```

[ ]:
```
#df.dropna(inplace=True)
```

[ ]:
```
from sklearn import preprocessing


col = [df.columns[-1]]

lab_en= preprocessing.LabelEncoder()

for c in col:
    df[c]= lab_en.fit_transform(df[c])

#df.head(50)
```

[ ]:
```
a=len(df[df.status==0])
b=len(df[df.status==1])
```

[ ]:
```
print("Count of Legitimate Websites = ", a)
print("Count of Phishy Websites = ", b)
```

```
Count of Legitimate Websites =  5715
Count of Phishy Websites =  5715
```

[ ]:
```
X = df.drop(['status'], axis=1, inplace=False)
#X.head()
#same work
##inplace true modifies the og data & does not return anything
##inplace false does not modify og data but returns something whoch we store in␣
 ↪a var
# X= df.drop(columns='Result')
# X.head()
```

[ ]:
```
#df.head()
```

[ ]:
```
y = df['status']
y = pd.DataFrame(y)
y.head()
```

[ ]:

	status
0	0
1	1
2	1
3	0
4	0

[ ]:
```
# separate dataset into train and test
from cProfile import label
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=10)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[ ]:
```
((9144, 87), (2286, 87), (9144, 1), (2286, 1))
```

[ ]:
```
#X_test.head()
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler= MinMaxScaler()

col_X_train = [X_train.columns[:]]

for c in col_X_train:
    X_train[c]= scaler.fit_transform(X_train[c])

#X_train.head(5)
```

[ ]:
```
col_X_test = [X_test.columns[:]]

for c in col_X_test:
    X_test[c]= scaler.transform(X_test[c])

#X_test.head(5)
```

[ ]:
```
#perform chi square test
from sklearn.feature_selection import chi2
f_p_values = chi2(X_train,y_train)
```

[ ]:
```
f_p_values
```

[ ]:
```
(array([2.23370641e+01, 1.69566642e+01, 7.92634978e+02, 2.14222888e+01,
        8.35897474e+00, 5.30407662e+01, 2.53295162e+02, 6.10286729e+01,
                   nan, 9.21347603e+01, 2.48852581e+00, 5.32938701e+00,
        1.36127244e+00, 2.76337058e+01, 6.97249509e+00, 3.02335502e+01,
        3.37730377e-02, 2.83381683e+01, 3.15422397e+00, 1.94038283e-02,
        5.01524054e+02, 4.11373688e+01, 4.68200813e+01, 2.09854284e+01,
        4.59133775e+01, 2.65045377e+02, 1.94693671e+02, 2.98821218e+00,
        1.97645536e+00, 5.07490559e+01, 3.59432236e+02, 1.31736523e+02,
        1.89723928e+01, 3.31523214e+02, 1.71903269e+00, 8.25729068e+01,
        7.75004553e-06, 8.27679909e-01, 2.98821218e+01, 1.99134491e+01,
        9.70793366e-02, 9.44986388e-01, 4.73779494e+01, 4.67727391e+00,
        2.67052777e+01, 6.69644804e+00, 4.14067508e+01, 8.24823685e+00,
        1.81658385e+01, 1.75476572e+01, 2.20716199e+02, 7.15055802e+01,
        4.28310413e+01, 3.52188528e+01, 1.13779295e+02, 1.73543170e+02,
        7.52046451e+01, 1.25671887e+02, 2.92566949e+01,            nan,
        4.45199645e+00,            nan, 4.22631863e+01,            nan,
        3.50467989e+00, 2.62448221e+00, 1.10884273e+02, 1.01180269e+02,
                   nan, 1.71196163e+02, 1.05123590e+02,            nan,
        1.34913309e+00, 3.73506517e+01, 1.17153886e+02, 1.15083440e-01,
        7.30367272e-02, 3.23965734e+02, 2.40907086e+02, 1.55570635e+02,
        3.98630823e+01, 1.01301038e+01, 1.86177846e+02, 1.26203754e+01,
        1.30175372e+02, 2.24891968e+03, 4.76130808e+02]),
 array([2.28748708e-006, 3.82428387e-005, 2.15456012e-174, 3.68462862e-006,
        3.83787046e-003, 3.26697163e-013, 4.96687374e-057, 5.62495784e-015,
                    nan, 8.09705637e-022, 1.14679097e-001, 2.09687588e-002,
```

```
       2.43317079e-001, 1.46601796e-007, 8.27720043e-003, 3.83026070e-008,
       8.54190424e-001, 1.01867512e-007, 7.57306913e-002, 8.89214967e-001,
       4.42963423e-111, 1.41897299e-010, 7.78119832e-012, 4.62789878e-006,
       1.23599050e-011, 1.36390512e-059, 3.00516293e-044, 8.38727207e-002,
       1.59764326e-001, 1.04963046e-012, 3.74294379e-080, 1.70855173e-030,
       1.32623548e-005, 4.47745841e-074, 1.89817606e-001, 1.01843501e-019,
       9.97778780e-001, 3.62944266e-001, 4.59126916e-008, 8.10283030e-006,
       7.55363102e-001, 3.30998761e-001, 5.85374855e-012, 3.05642145e-002,
       2.36973375e-007, 9.66051752e-003, 1.23629710e-010, 4.07916034e-003,
       2.02478406e-005, 2.80195804e-005, 6.31194922e-050, 2.76475119e-017,
       5.96776479e-011, 2.94657202e-009, 1.45650442e-026, 1.24551325e-039,
       4.24367530e-018, 3.62767027e-029, 6.33968046e-008,             nan,
       3.48604695e-002,             nan, 7.97802714e-011,             nan,
       6.11956709e-002, 1.05226826e-001, 6.27276739e-026, 8.39817380e-024,
                   nan, 4.05431738e-039, 1.14750659e-024,             nan,
       2.45429728e-001, 9.86878980e-010, 2.65636383e-027, 7.34429313e-001,
       7.86965560e-001, 1.98196025e-072, 2.49414539e-054, 1.05073706e-035,
       2.72403538e-010, 1.45867197e-003, 2.17104281e-042, 3.81564706e-004,
       3.75126966e-030, 0.00000000e+000, 1.48483498e-105]))
```

[ ]:
```
#The less the p_values the more important that feature is
p_values = pd.Series(f_p_values[1])
p_values.index = X_train.columns
p_values
```

[ ]:
```
length_url                  2.287487e-06
length_hostname             3.824284e-05
ip                         2.154560e-174
nb_dots                     3.684629e-06
nb_hyphens                  3.837870e-03
nb_at                       3.266972e-13
nb_qm                       4.966874e-57
nb_and                      5.624958e-15
nb_or                                NaN
nb_eq                       8.097056e-22
nb_underscore               1.146791e-01
nb_tilde                    2.096876e-02
nb_percent                  2.433171e-01
nb_slash                    1.466018e-07
nb_star                     8.277200e-03
nb_colon                    3.830261e-08
nb_comma                    8.541904e-01
nb_semicolumn               1.018675e-07
nb_dollar                   7.573069e-02
nb_space                    8.892150e-01
nb_www                     4.429634e-111
nb_com                      1.418973e-10
```

nb_dslash	7.781198e-12
http_in_path	4.627899e-06
https_token	1.235990e-11
ratio_digits_url	1.363905e-59
ratio_digits_host	3.005163e-44
punycode	8.387272e-02
port	1.597643e-01
tld_in_path	1.049630e-12
tld_in_subdomain	3.742944e-80
abnormal_subdomain	1.708552e-30
nb_subdomains	1.326235e-05
prefix_suffix	4.477458e-74
random_domain	1.898176e-01
shortening_service	1.018435e-19
path_extension	9.977788e-01
nb_redirection	3.629443e-01
nb_external_redirection	4.591269e-08
length_words_raw	8.102830e-06
char_repeat	7.553631e-01
shortest_words_raw	3.309988e-01
shortest_word_host	5.853749e-12
shortest_word_path	3.056421e-02
longest_words_raw	2.369734e-07
longest_word_host	9.660518e-03
longest_word_path	1.236297e-10
avg_words_raw	4.079160e-03
avg_word_host	2.024784e-05
avg_word_path	2.801958e-05
phish_hints	6.311949e-50
domain_in_brand	2.764751e-17
brand_in_subdomain	5.967765e-11
brand_in_path	2.946572e-09
suspecious_tld	1.456504e-26
statistical_report	1.245513e-39
nb_hyperlinks	4.243675e-18
ratio_intHyperlinks	3.627670e-29
ratio_extHyperlinks	6.339680e-08
ratio_nullHyperlinks	NaN
nb_extCSS	3.486047e-02
ratio_intRedirection	NaN
ratio_extRedirection	7.978027e-11
ratio_intErrors	NaN
ratio_extErrors	6.119567e-02
login_form	1.052268e-01
external_favicon	6.272767e-26
links_in_tags	8.398174e-24
submit_email	NaN

```
ratio_intMedia                    4.054317e-39
ratio_extMedia                    1.147507e-24
sfh                                        NaN
iframe                            2.454297e-01
popup_window                      9.868790e-10
safe_anchor                       2.656364e-27
onmouseover                       7.344293e-01
right_clic                        7.869656e-01
empty_title                       1.981960e-72
domain_in_title                   2.494145e-54
domain_with_copyright             1.050737e-35
whois_registered_domain           2.724035e-10
domain_registration_length        1.458672e-03
domain_age                        2.171043e-42
web_traffic                       3.815647e-04
dns_record                        3.751270e-30
google_index                      0.000000e+00
page_rank                        1.484835e-105
dtype: float64
```

[ ]:
```
#sort p_values to check which feature has the lowest values
p_values = p_values.sort_values(ascending = False)
p_values
```

[ ]:
```
path_extension                    9.977788e-01
nb_space                          8.892150e-01
nb_comma                          8.541904e-01
right_clic                        7.869656e-01
char_repeat                       7.553631e-01
onmouseover                       7.344293e-01
nb_redirection                    3.629443e-01
shortest_words_raw                3.309988e-01
iframe                            2.454297e-01
nb_percent                        2.433171e-01
random_domain                     1.898176e-01
port                              1.597643e-01
nb_underscore                     1.146791e-01
login_form                        1.052268e-01
punycode                          8.387272e-02
nb_dollar                         7.573069e-02
ratio_extErrors                   6.119567e-02
nb_extCSS                         3.486047e-02
shortest_word_path                3.056421e-02
nb_tilde                          2.096876e-02
longest_word_host                 9.660518e-03
nb_star                           8.277200e-03
avg_words_raw                     4.079160e-03
```

nb_hyphens	3.837870e-03
domain_registration_length	1.458672e-03
web_traffic	3.815647e-04
length_hostname	3.824284e-05
avg_word_path	2.801958e-05
avg_word_host	2.024784e-05
nb_subdomains	1.326235e-05
length_words_raw	8.102830e-06
http_in_path	4.627899e-06
nb_dots	3.684629e-06
length_url	2.287487e-06
longest_words_raw	2.369734e-07
nb_slash	1.466018e-07
nb_semicolumn	1.018675e-07
ratio_extHyperlinks	6.339680e-08
nb_external_redirection	4.591269e-08
nb_colon	3.830261e-08
brand_in_path	2.946572e-09
popup_window	9.868790e-10
whois_registered_domain	2.724035e-10
nb_com	1.418973e-10
longest_word_path	1.236297e-10
ratio_extRedirection	7.978027e-11
brand_in_subdomain	5.967765e-11
https_token	1.235990e-11
nb_dslash	7.781198e-12
shortest_word_host	5.853749e-12
tld_in_path	1.049630e-12
nb_at	3.266972e-13
nb_and	5.624958e-15
domain_in_brand	2.764751e-17
nb_hyperlinks	4.243675e-18
shortening_service	1.018435e-19
nb_eq	8.097056e-22
links_in_tags	8.398174e-24
ratio_extMedia	1.147507e-24
external_favicon	6.272767e-26
suspecious_tld	1.456504e-26
safe_anchor	2.656364e-27
ratio_intHyperlinks	3.627670e-29
dns_record	3.751270e-30
abnormal_subdomain	1.708552e-30
domain_with_copyright	1.050737e-35
ratio_intMedia	4.054317e-39
statistical_report	1.245513e-39
domain_age	2.171043e-42
ratio_digits_host	3.005163e-44

```
phish_hints                  6.311949e-50
domain_in_title              2.494145e-54
nb_qm                        4.966874e-57
ratio_digits_url             1.363905e-59
empty_title                  1.981960e-72
prefix_suffix                4.477458e-74
tld_in_subdomain             3.742944e-80
page_rank                   1.484835e-105
nb_www                      4.429634e-111
ip                          2.154560e-174
google_index                 0.000000e+00
nb_or                                 NaN
ratio_nullHyperlinks                  NaN
ratio_intRedirection                  NaN
ratio_intErrors                       NaN
submit_email                          NaN
sfh                                   NaN
dtype: float64
```

[ ]:
```
def DropFeature (p_values, threshold):
        drop_feature = set()
        for index, values in p_values.items():
                if values > threshold or np.isnan(values):
                        drop_feature.add(index)
        return drop_feature
```

[ ]:
```
drop_feature = DropFeature(p_values,.05)
len(set(drop_feature))
```

[ ]:
```
23
```

[ ]:
```
drop_feature
```

[ ]:
```
{'char_repeat',
 'iframe',
 'login_form',
 'nb_comma',
 'nb_dollar',
 'nb_or',
 'nb_percent',
 'nb_redirection',
 'nb_space',
 'nb_underscore',
 'onmouseover',
 'path_extension',
 'port',
 'punycode',
```

```
 'random_domain',
 'ratio_extErrors',
 'ratio_intErrors',
 'ratio_intRedirection',
 'ratio_nullHyperlinks',
 'right_clic',
 'sfh',
 'shortest_words_raw',
 'submit_email'}
```

[ ]:
```
X_train.drop(drop_feature, axis=1, inplace=True)
X_test.drop(drop_feature, axis=1, inplace=True)
```

[ ]:
```
len(X_train.columns)
```

[ ]: 64

[ ]:
```
len(X_test.columns)
```

[ ]: 64

[ ]:
```
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 9144 samples.
Testing set has 2286 samples.
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# defining parameter range
param_grid = {'penalty' : ['l2'],
            'C' : [0.1, 1, 10, 20, 30],
            'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
            'max_iter' : [2500, 5000]}

grid_logr = GridSearchCV(LogisticRegression(), param_grid, refit = True, cv =
 ↪10, verbose = 3, n_jobs = -1)

# fitting the model for grid search
grid_logr.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_logr.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_logr.best_estimator_)
print(grid_logr.best_score_)
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
{'C': 30, 'max_iter': 2500, 'penalty': 'l2', 'solver': 'saga'}
LogisticRegression(C=30, max_iter=2500, solver='saga')
0.9425849266420346
```

[ ]:
```
logr_model = grid_logr.best_estimator_

# Performing training
#logr_model = logr.fit(X_train, y_train.values.ravel())
```

[ ]:
```
logr_predict  =  logr_model.predict(X_test)
```

[ ]:
```
# from sklearn.metrics import confusion_matrix,accuracy_score
# cm = confusion_matrix(y_test, dct_pred)
# ac = accuracy_score(y_test, dct_pred)
```

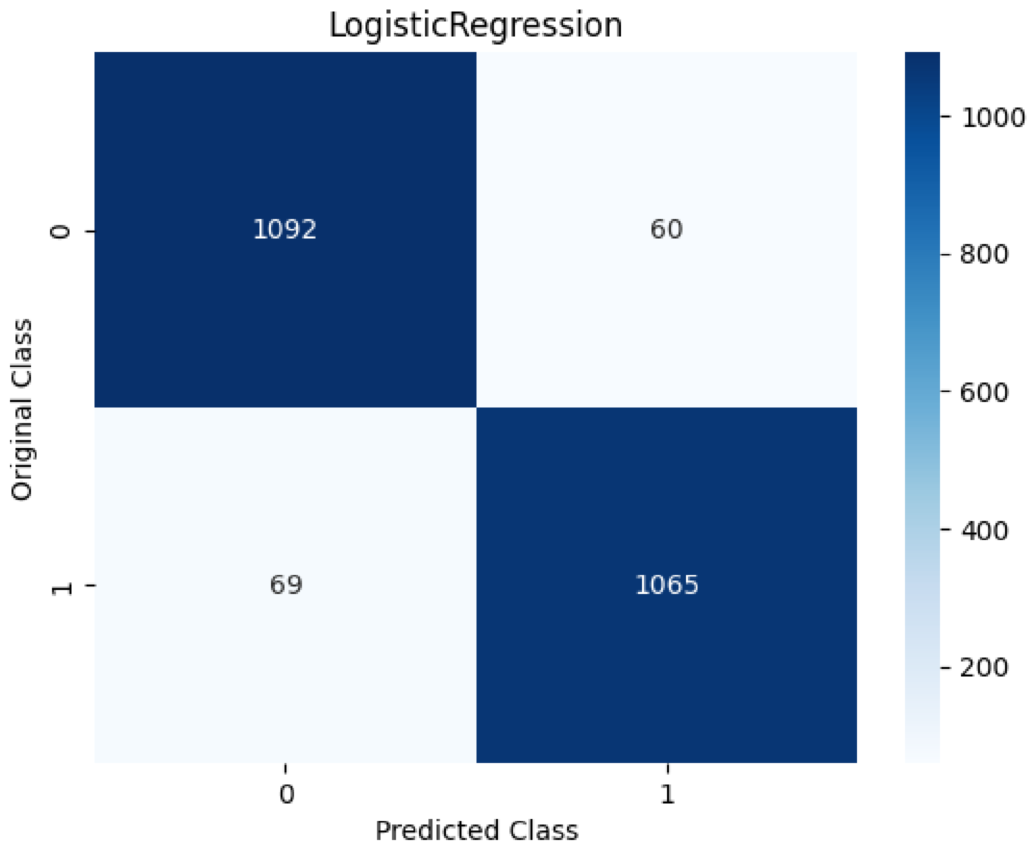
[ ]:
```
print ("Accuracy of logr classifier : ", accuracy_score(y_test,␣
 ↪logr_predict)*100)
```

```
Accuracy of logr classifier :  94.35695538057742
```

[ ]:
```
print(classification_report(y_test, logr_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      1152
           1       0.95      0.94      0.94      1134

    accuracy                           0.94      2286
   macro avg       0.94      0.94      0.94      2286
weighted avg       0.94      0.94      0.94      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, logr_predict), annot=True, fmt='g',␣
 ↪cmap='Blues')
plt.title("LogisticRegression")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
# from sklearn.neighbors import KNeighborsClassifier

# #training_accuracy=[]
# test_accuracy=[]

# neighbors=range(1,10)
# ##values.ravel() converts vector y to flattened array
# for i in neighbors:
#     knn=KNeighborsClassifier(n_neighbors=i)
#     knn_model = knn.fit(X_train,y_train.values.ravel())
#     #training_accuracy.append(knn.score(X_train,y_train.values.ravel()))
#     test_accuracy.append(knn_model.score(X_test,y_test.values.ravel()))
```

[ ]:
```
# plt.plot(neighbors,test_accuracy,label="test accuracy")
# plt.ylabel("Accuracy")
# plt.xlabel("number of neighbors")
# plt.legend()
# plt.show()
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# defining parameter range
param_grid = {'n_neighbors': [1,2,3,4,5,6,7,8,9,10]}

grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid, refit = True, cv =␣
↪10, verbose = 3, n_jobs = -1)

# fitting the model for grid search
grid_knn.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_knn.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_knn.best_estimator_)
print(grid_knn.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
{'n_neighbors': 3}
KNeighborsClassifier(n_neighbors=3)
0.9231213306070714
```

[ ]:
```
knn_model = grid_knn.best_estimator_
#knn_model = knn.fit(X_train,y_train.values.ravel())
```

[ ]:
```
#print ("Accuracy of knn classifier: ", max(test_accuracy)*100)
knn_predict = knn_model.predict(X_test)
```

[ ]:
```
print('The accuracy of knn Classifier is: ', 100.0 * accuracy_score(y_test,␣
↪knn_predict))
```

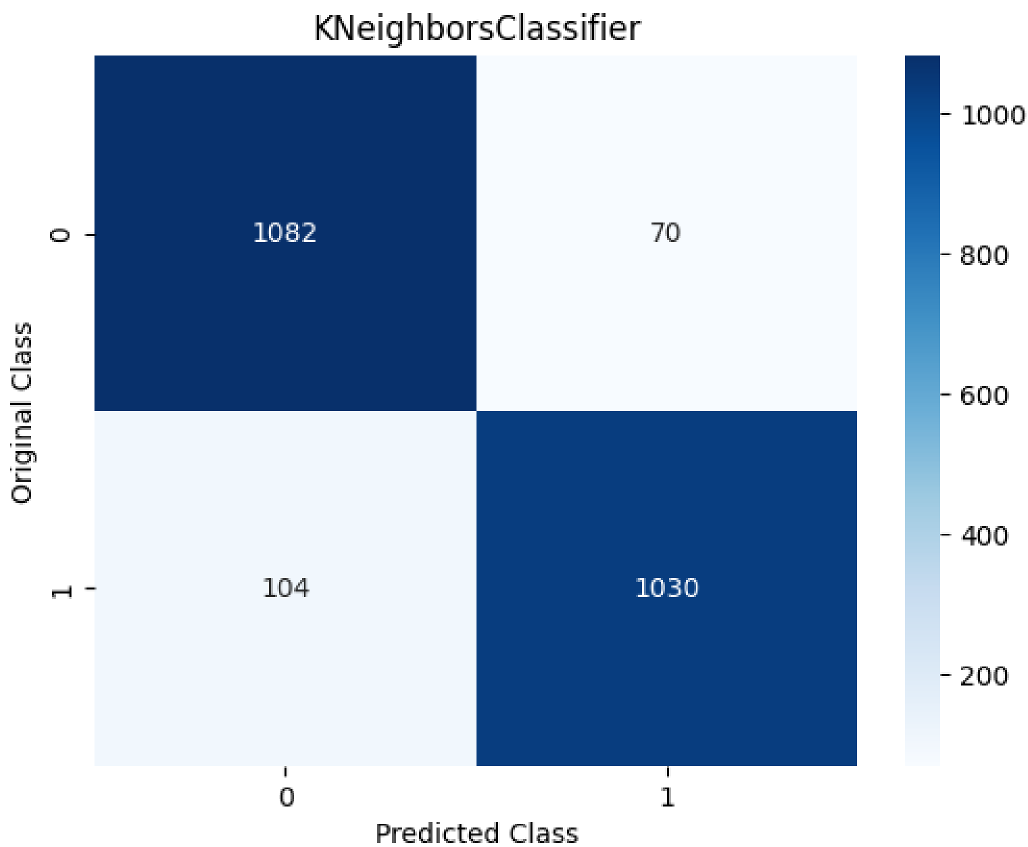
```
The accuracy of knn Classifier is:  92.38845144356955
```

[ ]:
```
print(classification_report(y_test, knn_predict))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.93      1152
           1       0.94      0.91      0.92      1134

    accuracy                           0.92      2286
   macro avg       0.92      0.92      0.92      2286
weighted avg       0.92      0.92      0.92      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, knn_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("KNeighborsClassifier")
```

```
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
# # here is the change
# knn_y_pred_proba = knn.predict_proba(X_test)
# knn_y_pred_proba_positive = knn_y_pred_proba[:, 1]

# RocCurveDisplay.from_predictions(y_test,knn_y_pred_proba_positive)

# fig, ax = plt.subplots()
# RocCurveDisplay.from_estimator(
#    logreg, X_test, y_test, ax = ax)

# logreg_y_decision = logreg.decision_function(X_test)
# metrics.RocCurveDisplay.
↪from_predictions(y_test,logreg_y_decision,ax=ax,name="logreg predictions")
```

[ ]:
```python
from sklearn.svm import SVC

# defining parameter range
param_grid = {'C': [0.1, 1, 10],
                        'gamma': [1, 0.1, 0.01],
                        'kernel': ['linear','poly', 'rbf', 'sigmoid']}

grid_svc = GridSearchCV(SVC(), param_grid, refit = True, cv = 10, verbose = 3,␣
 ↪n_jobs = -1)

# fitting the model for grid search
grid_svc.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_svc.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_svc.best_estimator_)
print(grid_svc.best_score_)
```

```
Fitting 10 folds for each of 36 candidates, totalling 360 fits
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=10, gamma=0.1)
0.9572385837787423
```

[ ]:
```python
svc_model = grid_svc.best_estimator_
#svc_model = svc.fit(X_train,y_train.values.ravel())
```

[ ]:
```python
svc_predict = svc_model.predict(X_test)
```

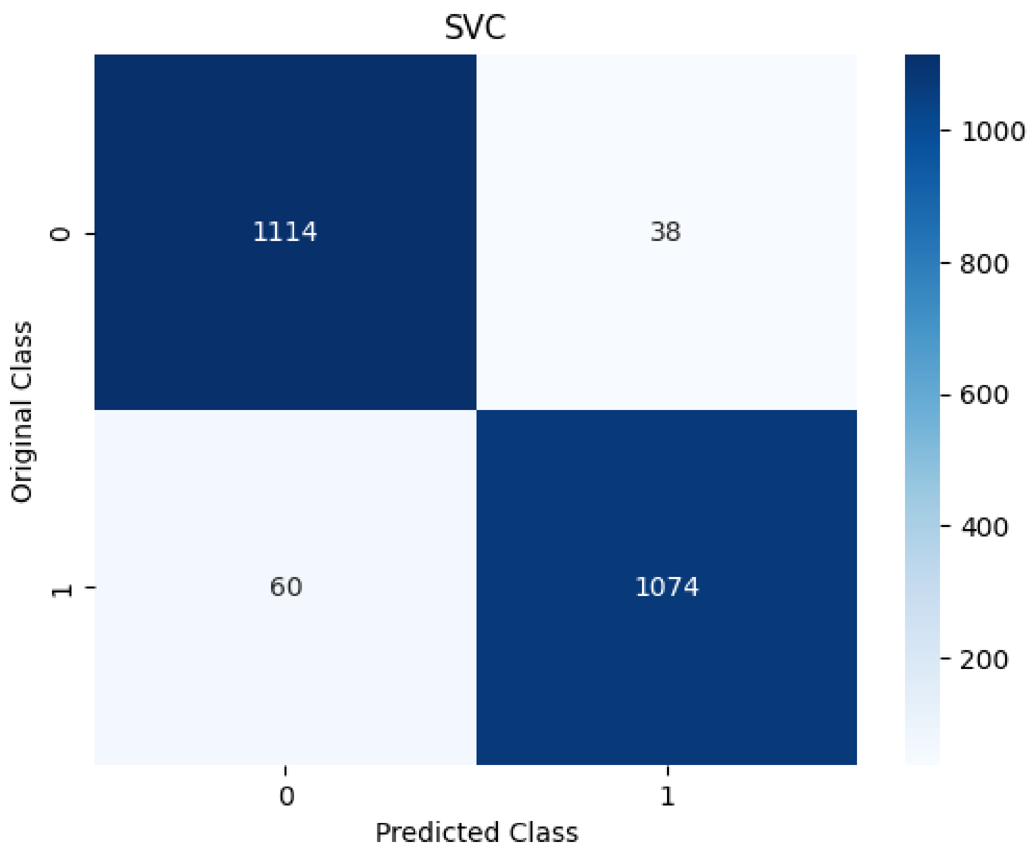
[ ]:
```python
print('The accuracy of svc Classifier is: ', 100.0 * accuracy_score(y_test,␣
 ↪svc_predict))
```

```
The accuracy of svc Classifier is:  95.71303587051618
```

[ ]:
```python
print(classification_report(y_test, svc_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1152
           1       0.97      0.95      0.96      1134

    accuracy                           0.96      2286
   macro avg       0.96      0.96      0.96      2286
weighted avg       0.96      0.96      0.96      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, svc_predict), annot=True, fmt='g',␣
    ↪cmap='Blues')
plt.title("SVC")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
from sklearn.svm import NuSVC

# defining parameter range
param_grid = {'nu': [0.1, 0.5],
              'gamma': [1, 0.1, 0.01],
              'kernel': ['rbf']} #'linear','poly', 'rbf', 'sigmoid'

grid_nusvc = GridSearchCV(NuSVC(), param_grid, refit = True, verbose = 3, cv =␣
    ↪10, n_jobs = -1)

# fitting the model for grid search
grid_nusvc.fit(X_train, y_train.values.ravel())
```

```
# print best parameter after tuning
print(grid_nusvc.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_nusvc.best_estimator_)
print(grid_nusvc.best_score_)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
{'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.1}
NuSVC(gamma=0.1, nu=0.1)
0.9580039698198037
```

[ ]:
```
nusvc_model = grid_nusvc.best_estimator_
#nusvc_model = nusvc.fit(X_train, y_train.values.ravel())
```

[ ]:
```
nusvc_predict = nusvc_model.predict(X_test)
```

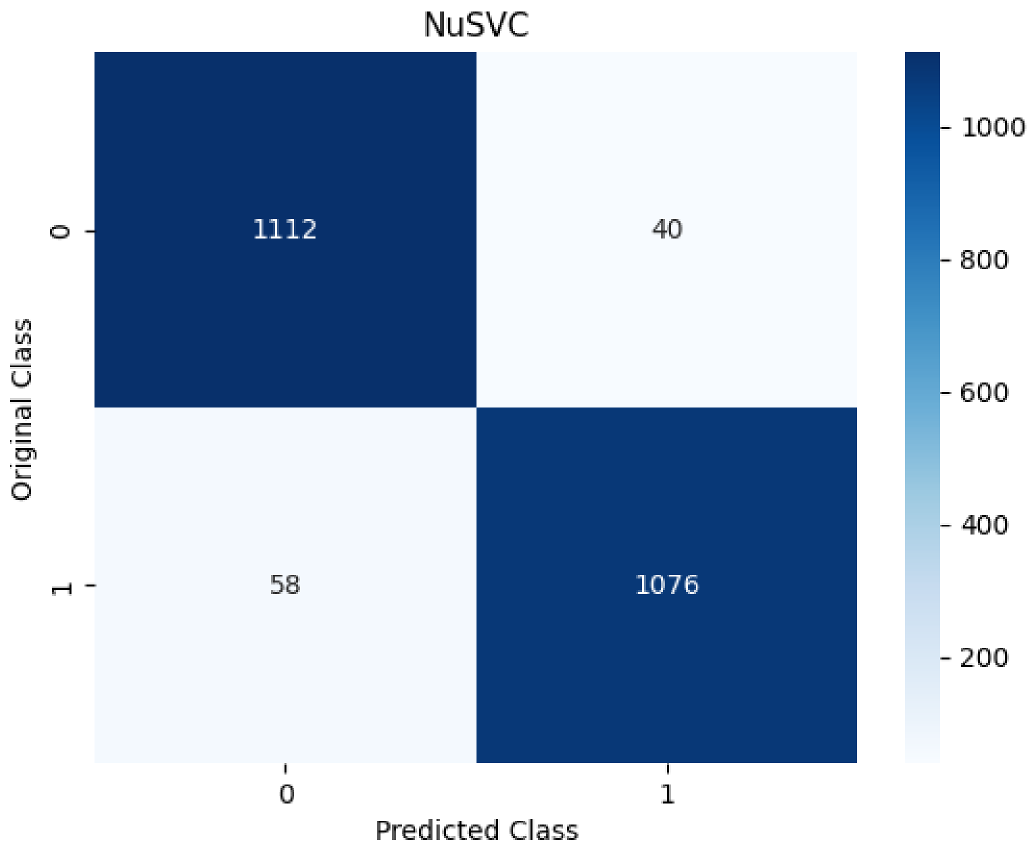
[ ]:
```
print('The accuracy of nusvc Classifier is: ', 100.0 * accuracy_score(y_test,␣
↪nusvc_predict))
```

```
The accuracy of nusvc Classifier is:  95.71303587051618
```

[ ]:
```
print(classification_report(y_test, nusvc_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1152
           1       0.96      0.95      0.96      1134

    accuracy                           0.96      2286
   macro avg       0.96      0.96      0.96      2286
weighted avg       0.96      0.96      0.96      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, nusvc_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("NuSVC")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
from sklearn.svm import LinearSVC

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 20, 30],
                        'penalty': ['l1','l2'],
                        'loss': ['squared_hinge'],
                        'dual': [False],
                        'tol': [.1,.01,.001]}

grid_lsvc = GridSearchCV(LinearSVC(), param_grid, refit = True, verbose = 3, cv
 ↪= 10, n_jobs = -1)

# fitting the model for grid search
grid_lsvc.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_lsvc.best_params_)

# print how our model looks after hyper-parameter tuning
```

```
print(grid_lsvc.best_estimator_)
print(grid_lsvc.best_score_)
```

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits
{'C': 20, 'dual': False, 'loss': 'squared_hinge', 'penalty': 'l2', 'tol': 0.001}
LinearSVC(C=20, dual=False, tol=0.001)
0.9426942162595209
```

[ ]:
```
lsvc_model = grid_lsvc.best_estimator_
#lsvc_model = lsvc.fit(X_train, y_train.values.ravel())
```

[ ]:
```
lsvc_predict = lsvc_model.predict(X_test)
```

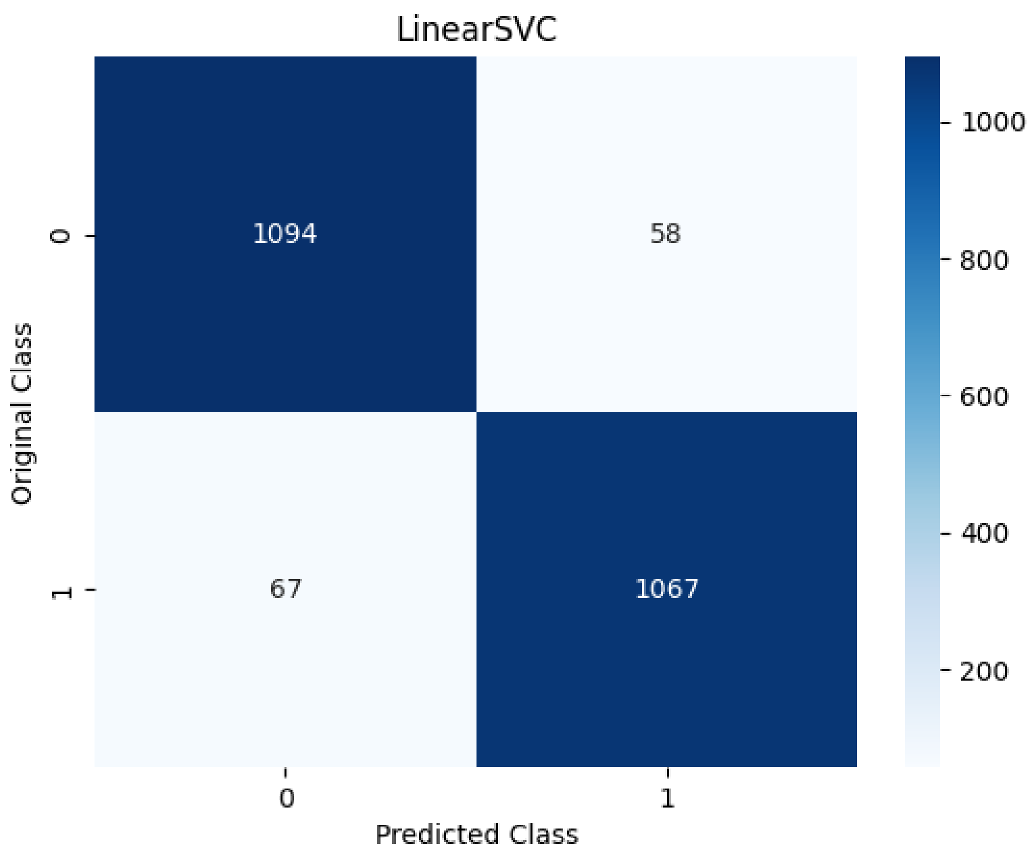
[ ]:
```
print('The accuracy of lsvc Classifier is: ', 100.0 * accuracy_score(y_test,␣
↪lsvc_predict))
```

```
The accuracy of lsvc Classifier is:  94.53193350831145
```

[ ]:
```
print(classification_report(y_test, lsvc_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      1152
           1       0.95      0.94      0.94      1134

    accuracy                           0.95      2286
   macro avg       0.95      0.95      0.95      2286
weighted avg       0.95      0.95      0.95      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, lsvc_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("LinearSVC")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
from sklearn.ensemble import AdaBoostClassifier

# defining parameter range
param_grid = {'n_estimators': [40,50,100,200,300]}

grid_ada = GridSearchCV(AdaBoostClassifier(), param_grid, refit = True, verbose␣
 ↪= 3, cv = 10, n_jobs = -1)

# fitting the model for grid search
grid_ada.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_ada.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_ada.best_estimator_)
print(grid_ada.best_score_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
{'n_estimators': 300}
```

```
AdaBoostClassifier(n_estimators=300)
0.9539591778168383
```

[ ]:
```
ada_model = grid_ada.best_estimator_
#ada_model = ada.fit(X_train,y_train.values.ravel())
```

[ ]:
```
ada_predict = ada_model.predict(X_test)
```

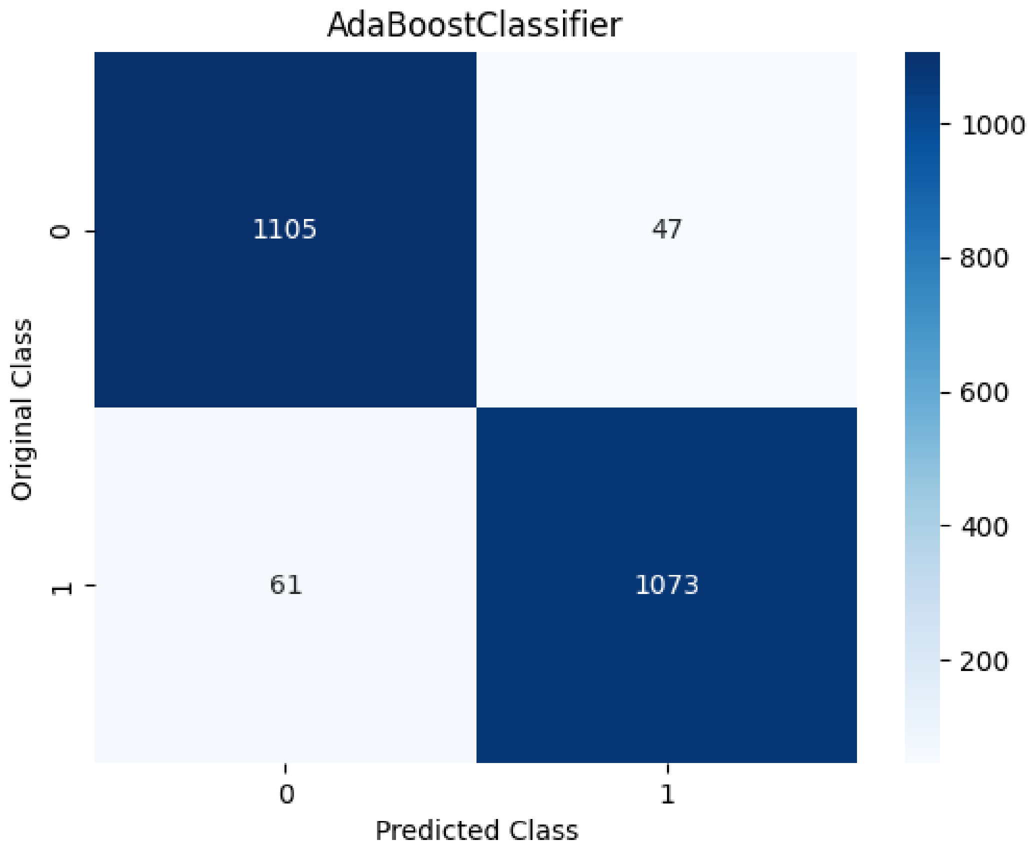
[ ]:
```
print('The accuracy of Ada Boost Classifier is: ', 100.0 *␣
↪accuracy_score(ada_predict,y_test))
```

```
The accuracy of Ada Boost Classifier is:  95.2755905511811
```

[ ]:
```
print(classification_report(y_test, ada_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95      1152
           1       0.96      0.95      0.95      1134

    accuracy                           0.95      2286
   macro avg       0.95      0.95      0.95      2286
weighted avg       0.95      0.95      0.95      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, ada_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("AdaBoostClassifier")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```python
from xgboost import XGBClassifier


# defining parameter range
param_grid = {
    "gamma": [.01, .1, .5],
    "n_estimators": [50,100,150,200,250]
}

grid_xgb = GridSearchCV(XGBClassifier(), param_grid, refit = True, verbose = 3,␣
 ↪cv = 10, n_jobs = -1)

# fitting the model for grid search
grid_xgb.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_xgb.best_params_)

# print how our model looks after hyper-parameter tuning
```

```
print(grid_xgb.best_estimator_)
print(grid_xgb.best_score_)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
{'gamma': 0.1, 'n_estimators': 150}
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.1, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=150,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
0.9704711171694707
```

[ ]:
```
xgb_model = grid_xgb.best_estimator_
#xgb_model = xgb.fit(X_train,y_train)
```

[ ]:
```
xgb_predict=xgb_model.predict(X_test)
```

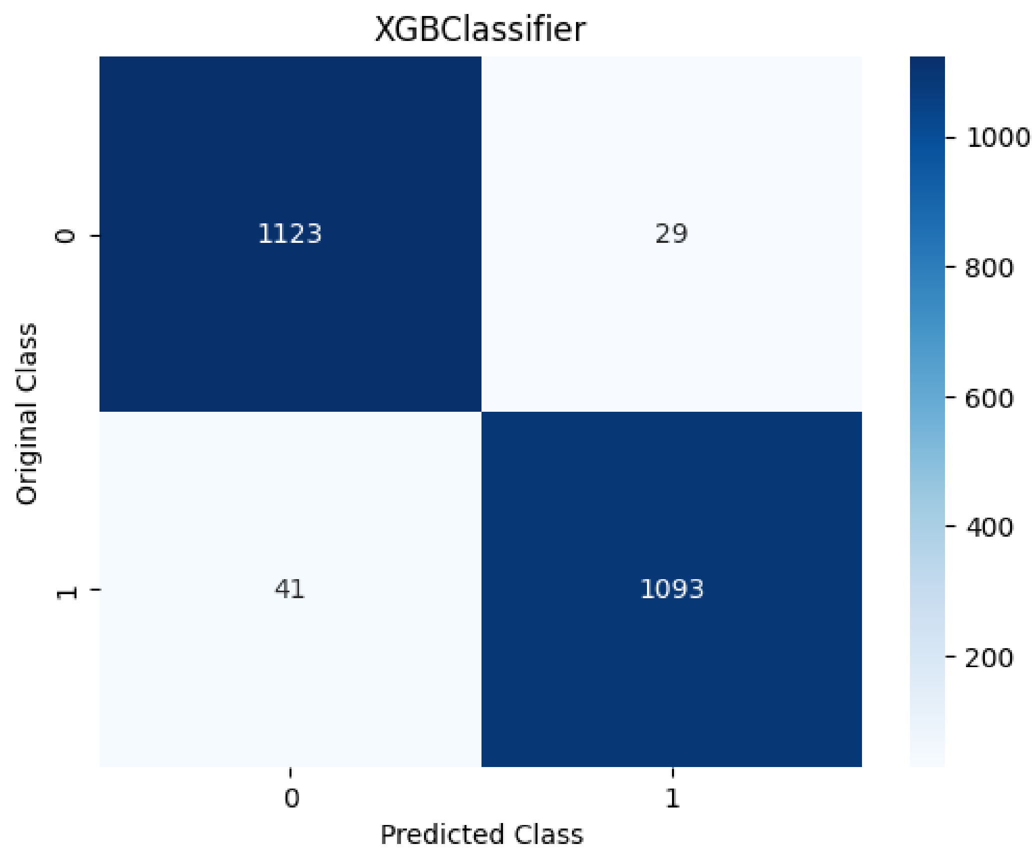
[ ]:
```
print('The accuracy of XGBoost Classifier is: ' , 100.0 *␣
↪accuracy_score(xgb_predict,y_test))
```

```
The accuracy of XGBoost Classifier is:  96.93788276465442
```

[ ]:
```
print(classification_report(y_test, xgb_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1152
           1       0.97      0.96      0.97      1134

    accuracy                           0.97      2286
   macro avg       0.97      0.97      0.97      2286
weighted avg       0.97      0.97      0.97      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, xgb_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("XGBClassifier")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
from sklearn.ensemble import GradientBoostingClassifier

# defining parameter range
param_grid = {
    "learning_rate": [.1,.5,1],
    "n_estimators": [50,100,150,200,250]
}

grid_gbc = GridSearchCV(GradientBoostingClassifier(), param_grid, refit = True,␣
 ↪verbose = 3, cv = 10, n_jobs = -1)

# fitting the model for grid search
grid_gbc.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_gbc.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_gbc.best_estimator_)
```

```
print(grid_gbc.best_score_)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
{'learning_rate': 0.5, 'n_estimators': 250}
GradientBoostingClassifier(learning_rate=0.5, n_estimators=250)
0.9658795183604166
```

[ ]:
```
gbc_model = grid_gbc.best_estimator_
#gbc_model = gbc.fit(X_train,y_train.values.ravel())

#clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
#    max_depth=1, random_state=0).fit(X_train, y_train)
#clf.score(X_test, y_test)
```

[ ]:
```
gbc_predict = gbc_model.predict(X_test)
```

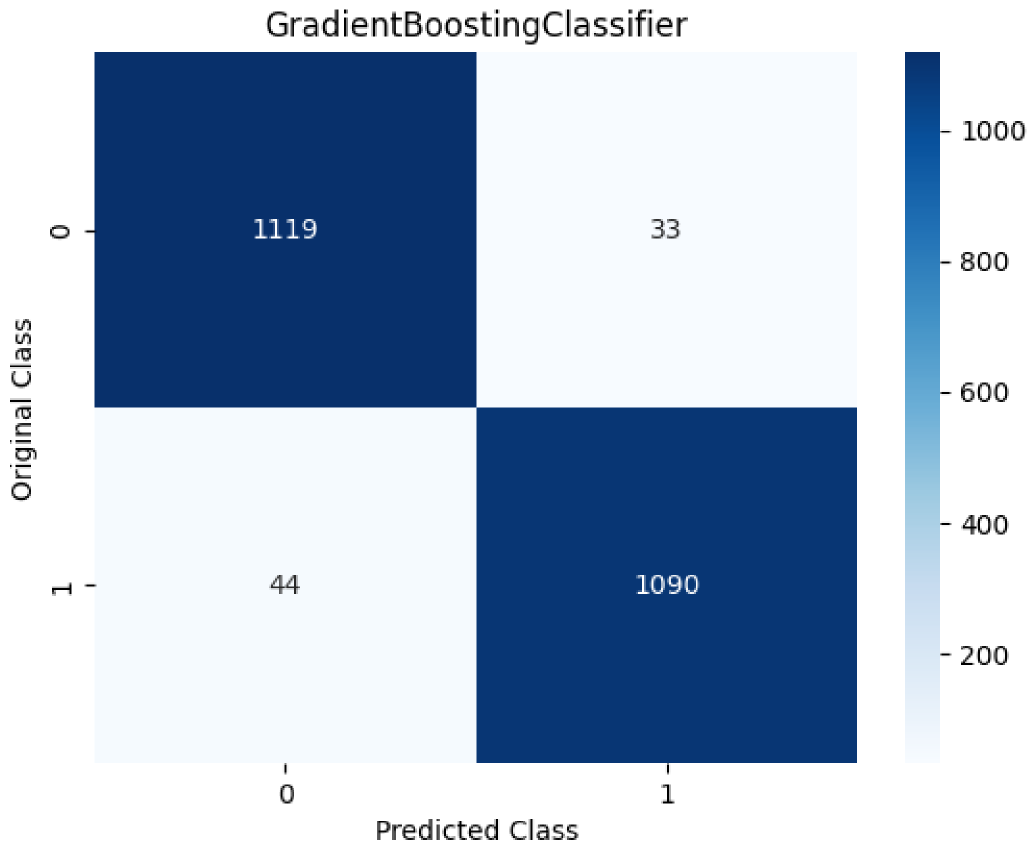
[ ]:
```
print('The accuracy of GradientBoost Classifier is: ' , 100.0 *␣
↪accuracy_score(gbc_predict,y_test))
```

```
The accuracy of GradientBoost Classifier is:  96.63167104111986
```

[ ]:
```
print(classification_report(y_test, gbc_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1152
           1       0.97      0.96      0.97      1134

    accuracy                           0.97      2286
   macro avg       0.97      0.97      0.97      2286
weighted avg       0.97      0.97      0.97      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, gbc_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("GradientBoostingClassifier")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
# gbc_model.get_params().keys()
```

[ ]:
```
# import inspect
# import sklearn
# import xgboost

# models = [xgboost.XGBClassifier]
# for m in models:
#     hyperparams = inspect.signature(m.__init__)
#     print(hyperparams)
# #or
# xgb_model.get_params().keys()
```

[ ]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# defining parameter range
param_grid = {
    "base_estimator": [DecisionTreeClassifier()],
    "n_estimators": [50,100,150,200,250]
```

```
}

grid_bag = GridSearchCV(BaggingClassifier(), param_grid, refit = True, verbose␣
↪= 3, cv = 10, n_jobs = -1)

# fitting the model for grid search
grid_bag.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_bag.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_bag.best_estimator_)
print(grid_bag.best_score_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
{'base_estimator': DecisionTreeClassifier(), 'n_estimators': 250}
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=250)
0.9575681266516005
```

[ ]:
```
bag_model = grid_bag.best_estimator_
#bag_model = bag.fit(X_train, y_train.values.ravel())
```

[ ]:
```
bag_predict = bag_model.predict(X_test)
```

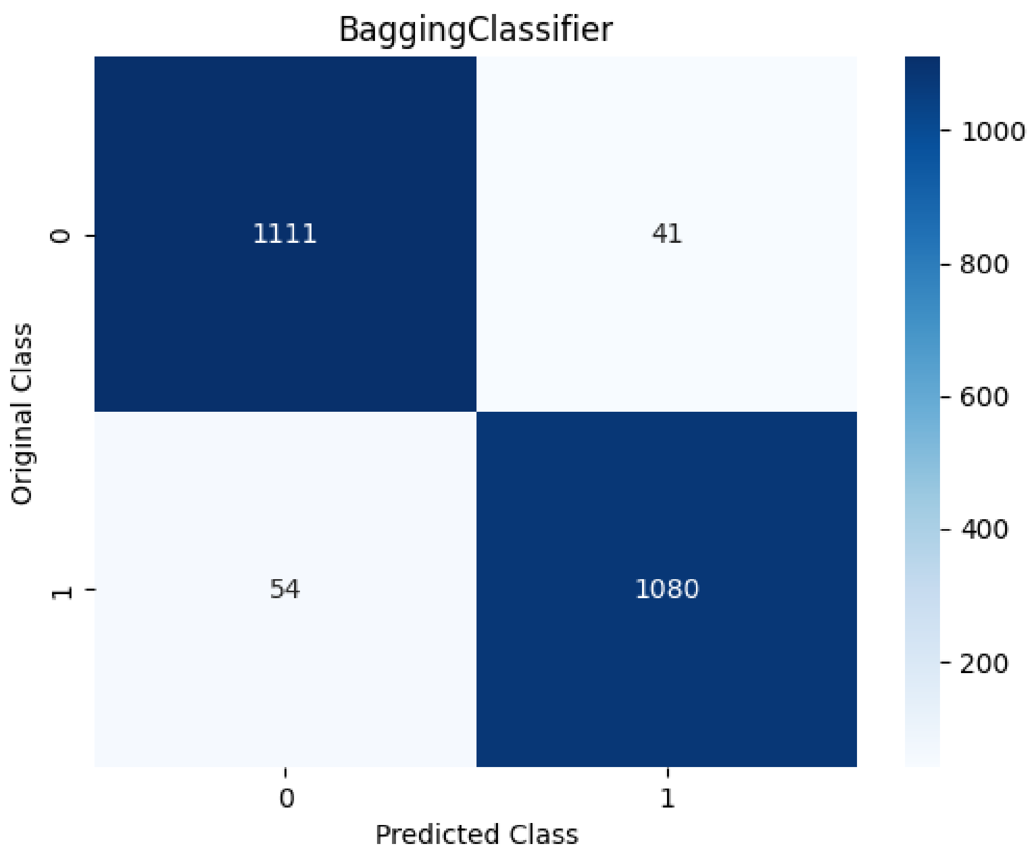
[ ]:
```
print('The accuracy of Bagging Classifier is: ' , 100.0 *␣
↪accuracy_score(y_test, bag_predict))
```

```
The accuracy of Bagging Classifier is:  95.84426946631672
```

[ ]:
```
print(classification_report(y_test, bag_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      1152
           1       0.96      0.95      0.96      1134

    accuracy                           0.96      2286
   macro avg       0.96      0.96      0.96      2286
weighted avg       0.96      0.96      0.96      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, bag_predict), annot=True, fmt='g',␣
↪cmap='Blues')
plt.title("BaggingClassifier")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



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

# defining parameter range
param_grid = {
    "n_estimators": [50,100,150,200,250]
}

grid_rfc = GridSearchCV(RandomForestClassifier(), param_grid, refit = True,␣
 ↪verbose = 3, cv = 10, n_jobs = -1)

# fitting the model for grid search
grid_rfc.fit(X_train, y_train.values.ravel())

# print best parameter after tuning
print(grid_rfc.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid_rfc.best_estimator_)
print(grid_rfc.best_score_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
{'n_estimators': 100}
RandomForestClassifier()
0.9666444261099352
```

[ ]:
```
rfc_model = grid_rfc.best_estimator_
#rfc_model = rfc.fit(X_train,y_train.values.ravel())
```

[ ]:
```
rfc_predict = rfc_model.predict(X_test)
```

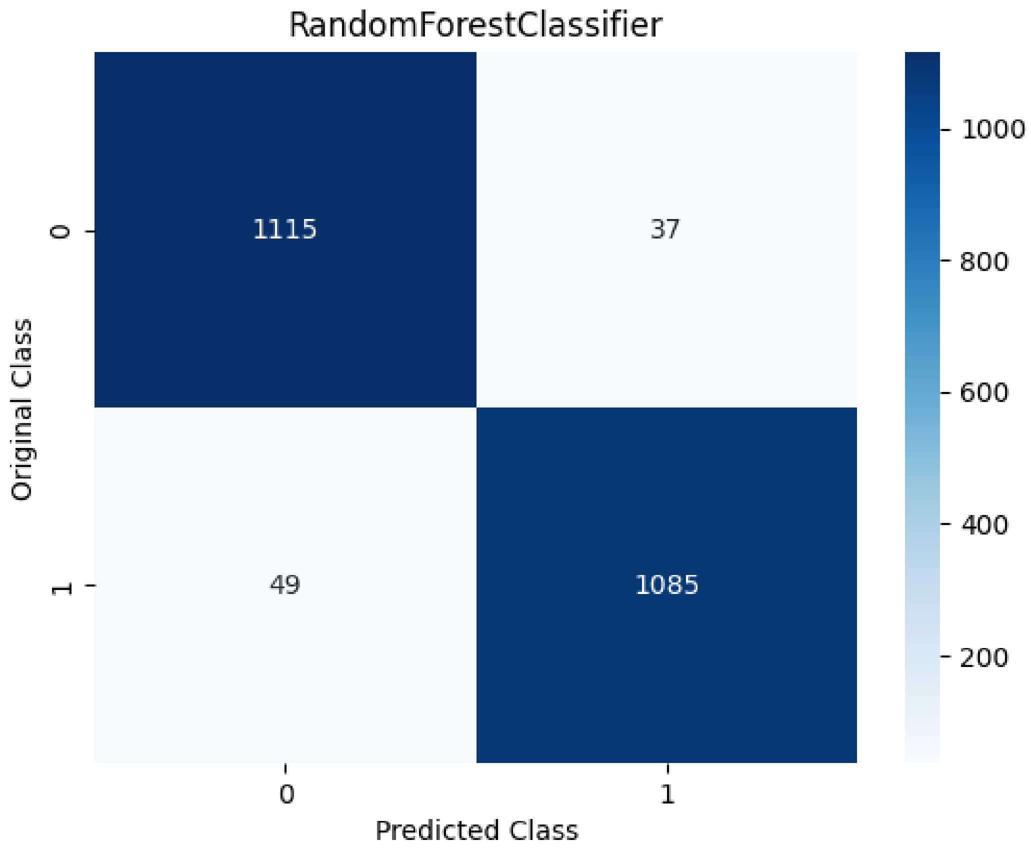
[ ]:
```
print('The accuracy of RandomForest Classifier is: ' , 100.0 *␣
    ↪accuracy_score(rfc_predict,y_test))
```

```
The accuracy of RandomForest Classifier is:  96.23797025371829
```

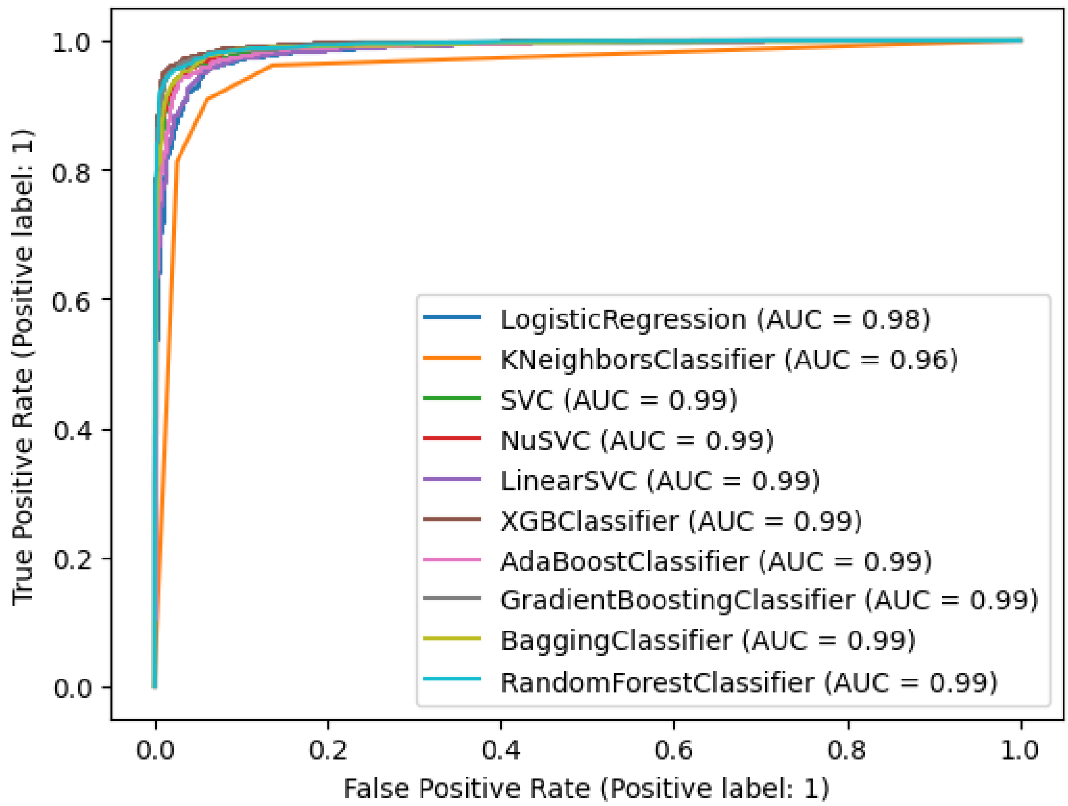
[ ]:
```
print(classification_report(y_test, rfc_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      1152
           1       0.97      0.96      0.96      1134

    accuracy                           0.96      2286
   macro avg       0.96      0.96      0.96      2286
weighted avg       0.96      0.96      0.96      2286
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, rfc_predict), annot=True, fmt='g',␣
    ↪cmap='Blues')
plt.title("RandomForestClassifier")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
estimators = ␣
↪[logr_model,knn_model,svc_model,nusvc_model,lsvc_model,xgb_model,ada_model,gbc_model,bag_mo

for estimator in estimators:
    RocCurveDisplay.from_estimator(estimator,X_test,y_test,ax=plt.gca())
```



[ ]:
```
import tensorflow as tf
#from tensorflow.keras.datasets import imdb
from keras.layers import Embedding, Dense, LSTM, BatchNormalization
from keras.losses import BinaryCrossentropy
from keras.models import Sequential
from keras.optimizers import Adam
#from tensorflow.keras.preprocessing.sequence import pad_sequences

# Model configuration
additional_metrics = ['accuracy']
batch_size = 32
#embedding_output_dims = (X_train.shape[1])
loss_function = BinaryCrossentropy()
#max_sequence_length = (X_train.shape[1])
#num_distinct_words = (X_train.shape[1])
number_of_epochs = 100
optimizer = Adam()
validation_split = 0.20
verbosity_mode = 1

# reshape from [samples, features] into [samples, timesteps, features]
```

```
timesteps = 1
X_train_reshape = X_train.values.ravel().reshape(X_train.shape[0],timesteps,␣
↪X_train.shape[1])
X_test_reshape = X_test.values.ravel().reshape(X_test.shape[0],timesteps,␣
↪X_test.shape[1])

# Disable eager execution
#tf.compat.v1.disable_eager_execution()

# Load dataset
# (x_train, y_train), (x_test, y_test) = imdb.
↪load_data(num_words=num_distinct_words)
# print(x_train.shape)
# print(x_test.shape)

# Pad all sequences
# padded_inputs = pad_sequences(X_train, maxlen=max_sequence_length, value = 0.
↪0) # 0.0 because it corresponds with <PAD>
# padded_inputs_test = pad_sequences(X_test, maxlen=max_sequence_length, value␣
↪= 0.0) # 0.0 because it corresponds with <PAD>

# Define the Keras model
def build_model_lstm():
    model = Sequential()
    #model.add(Embedding(num_distinct_words, embedding_output_dims,␣
↪input_length=max_sequence_length))
    model.add(LSTM(100, input_shape = (timesteps,X_train_reshape.shape[2])))
    model.add(BatchNormalization())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(25, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    model.compile(optimizer=optimizer, loss=loss_function,␣
↪metrics=additional_metrics)
    return model

#from keras.wrappers.scikit_learn import KerasClassifier
lstm_model = build_model_lstm()
# Give a summary
lstm_model.summary()

# Train the model
```

```
history = lstm_model.fit(X_train_reshape, y_train.values.ravel(),
    batch_size=batch_size, epochs=number_of_epochs, verbose=verbosity_mode,
    validation_split=validation_split)

# Test the model after training
#lstm_predict = lstm_model.predict(X_test_reshape)
test_results = lstm_model.evaluate(X_test_reshape, y_test.values.ravel(),
    verbose=False)
print(f'Test results - Loss: {test_results[0]} - Accuracy:
    {100*test_results[1]}%')
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               66000

 batch_normalization (BatchN  (None, 100)              400
 ormalization)

 dense (Dense)               (None, 50)                5050


_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               66000

 batch_normalization (BatchN  (None, 100)              400
 ormalization)

 dense (Dense)               (None, 50)                5050

 dense_1 (Dense)             (None, 25)                1275

 dense_2 (Dense)             (None, 10)                260

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 72,996
Trainable params: 72,796
Non-trainable params: 200
_________________________________________________________________
Epoch 1/100
229/229 [==============================] - 4s 7ms/step - loss: 0.2223 -
accuracy: 0.9136 - val_loss: 0.4193 - val_accuracy: 0.9295
Epoch 2/100
```

```
229/229 [==============================] - 1s 4ms/step - loss: 0.1568 -
accuracy: 0.9423 - val_loss: 0.2086 - val_accuracy: 0.9399
Epoch 3/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1376 -
accuracy: 0.9489 - val_loss: 0.1596 - val_accuracy: 0.9404
Epoch 4/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1270 -
accuracy: 0.9535 - val_loss: 0.1526 - val_accuracy: 0.9442
Epoch 5/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1111 -
accuracy: 0.9571 - val_loss: 0.1533 - val_accuracy: 0.9459
Epoch 6/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1045 -
accuracy: 0.9610 - val_loss: 0.1502 - val_accuracy: 0.9497
Epoch 7/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1006 -
accuracy: 0.9628 - val_loss: 0.1514 - val_accuracy: 0.9492
Epoch 8/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0983 -
accuracy: 0.9613 - val_loss: 0.1578 - val_accuracy: 0.9399
Epoch 9/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0953 -
accuracy: 0.9632 - val_loss: 0.1507 - val_accuracy: 0.9486
Epoch 10/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0871 -
accuracy: 0.9668 - val_loss: 0.1687 - val_accuracy: 0.9442
Epoch 11/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0802 -
accuracy: 0.9714 - val_loss: 0.1812 - val_accuracy: 0.9426
Epoch 12/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0722 -
accuracy: 0.9729 - val_loss: 0.1630 - val_accuracy: 0.9541
Epoch 13/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0682 -
accuracy: 0.9733 - val_loss: 0.1557 - val_accuracy: 0.9535
Epoch 14/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0654 -
accuracy: 0.9754 - val_loss: 0.1801 - val_accuracy: 0.9486
Epoch 15/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0701 -
accuracy: 0.9721 - val_loss: 0.1736 - val_accuracy: 0.9502
Epoch 16/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0623 -
accuracy: 0.9765 - val_loss: 0.1818 - val_accuracy: 0.9546
Epoch 17/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0597 -
accuracy: 0.9761 - val_loss: 0.1913 - val_accuracy: 0.9453
Epoch 18/100
```

```
229/229 [==============================] - 1s 4ms/step - loss: 0.0554 -
accuracy: 0.9813 - val_loss: 0.1735 - val_accuracy: 0.9502
Epoch 19/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0521 -
accuracy: 0.9821 - val_loss: 0.1722 - val_accuracy: 0.9524
Epoch 20/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0537 -
accuracy: 0.9796 - val_loss: 0.1708 - val_accuracy: 0.9459
Epoch 21/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0496 -
accuracy: 0.9824 - val_loss: 0.1913 - val_accuracy: 0.9431
Epoch 22/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0508 -
accuracy: 0.9795 - val_loss: 0.1987 - val_accuracy: 0.9431
Epoch 23/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0451 -
accuracy: 0.9828 - val_loss: 0.1873 - val_accuracy: 0.9475
Epoch 24/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0455 -
accuracy: 0.9829 - val_loss: 0.1895 - val_accuracy: 0.9519
Epoch 25/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0409 -
accuracy: 0.9846 - val_loss: 0.1957 - val_accuracy: 0.9519
Epoch 26/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0411 -
accuracy: 0.9862 - val_loss: 0.2121 - val_accuracy: 0.9453
Epoch 27/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0376 -
accuracy: 0.9870 - val_loss: 0.2127 - val_accuracy: 0.9502
Epoch 28/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0349 -
accuracy: 0.9882 - val_loss: 0.2015 - val_accuracy: 0.9453
Epoch 29/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0425 -
accuracy: 0.9848 - val_loss: 0.1991 - val_accuracy: 0.9481
Epoch 30/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0362 -
accuracy: 0.9877 - val_loss: 0.2377 - val_accuracy: 0.9470
Epoch 31/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0364 -
accuracy: 0.9867 - val_loss: 0.2108 - val_accuracy: 0.9486
Epoch 32/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0343 -
accuracy: 0.9866 - val_loss: 0.2275 - val_accuracy: 0.9442
Epoch 33/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0292 -
accuracy: 0.9892 - val_loss: 0.2329 - val_accuracy: 0.9502
Epoch 34/100
```

```
229/229 [==============================] - 1s 4ms/step - loss: 0.0330 -
accuracy: 0.9888 - val_loss: 0.2292 - val_accuracy: 0.9453
Epoch 35/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0289 -
accuracy: 0.9897 - val_loss: 0.2268 - val_accuracy: 0.9530
Epoch 36/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0338 -
accuracy: 0.9869 - val_loss: 0.2525 - val_accuracy: 0.9371
Epoch 37/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0275 -
accuracy: 0.9906 - val_loss: 0.2191 - val_accuracy: 0.9519
Epoch 38/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0295 -
accuracy: 0.9889 - val_loss: 0.2277 - val_accuracy: 0.9442
Epoch 39/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0217 -
accuracy: 0.9928 - val_loss: 0.2287 - val_accuracy: 0.9535
Epoch 40/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0210 -
accuracy: 0.9933 - val_loss: 0.2635 - val_accuracy: 0.9431
Epoch 41/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0240 -
accuracy: 0.9910 - val_loss: 0.2428 - val_accuracy: 0.9481
Epoch 42/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0271 -
accuracy: 0.9907 - val_loss: 0.2515 - val_accuracy: 0.9568
Epoch 43/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0245 -
accuracy: 0.9921 - val_loss: 0.2799 - val_accuracy: 0.9366
Epoch 44/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0201 -
accuracy: 0.9926 - val_loss: 0.2768 - val_accuracy: 0.9475
Epoch 45/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0235 -
accuracy: 0.9915 - val_loss: 0.2470 - val_accuracy: 0.9502
Epoch 46/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0233 -
accuracy: 0.9906 - val_loss: 0.2689 - val_accuracy: 0.9475
Epoch 47/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0214 -
accuracy: 0.9919 - val_loss: 0.2691 - val_accuracy: 0.9481
Epoch 48/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0208 -
accuracy: 0.9922 - val_loss: 0.2809 - val_accuracy: 0.9541
Epoch 49/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0198 -
accuracy: 0.9930 - val_loss: 0.2851 - val_accuracy: 0.9486
Epoch 50/100
```

```
229/229 [==============================] - 1s 4ms/step - loss: 0.0208 -
accuracy: 0.9930 - val_loss: 0.2764 - val_accuracy: 0.9442
Epoch 51/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0206 -
accuracy: 0.9921 - val_loss: 0.2710 - val_accuracy: 0.9453
Epoch 52/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0237 -
accuracy: 0.9922 - val_loss: 0.2615 - val_accuracy: 0.9519
Epoch 53/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0189 -
accuracy: 0.9937 - val_loss: 0.2734 - val_accuracy: 0.9513
Epoch 54/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0131 -
accuracy: 0.9960 - val_loss: 0.3044 - val_accuracy: 0.9426
Epoch 55/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0160 -
accuracy: 0.9940 - val_loss: 0.2800 - val_accuracy: 0.9519
Epoch 56/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0178 -
accuracy: 0.9929 - val_loss: 0.3062 - val_accuracy: 0.9393
Epoch 57/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0238 -
accuracy: 0.9914 - val_loss: 0.2737 - val_accuracy: 0.9508
Epoch 58/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0164 -
accuracy: 0.9940 - val_loss: 0.3027 - val_accuracy: 0.9513
Epoch 59/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0239 -
accuracy: 0.9922 - val_loss: 0.2741 - val_accuracy: 0.9524
Epoch 60/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0175 -
accuracy: 0.9941 - val_loss: 0.2563 - val_accuracy: 0.9524
Epoch 61/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0139 -
accuracy: 0.9944 - val_loss: 0.2518 - val_accuracy: 0.9513
Epoch 62/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0139 -
accuracy: 0.9955 - val_loss: 0.2698 - val_accuracy: 0.9492
Epoch 63/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0193 -
accuracy: 0.9937 - val_loss: 0.2888 - val_accuracy: 0.9420
Epoch 64/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0196 -
accuracy: 0.9930 - val_loss: 0.3050 - val_accuracy: 0.9475
Epoch 65/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0129 -
accuracy: 0.9952 - val_loss: 0.2995 - val_accuracy: 0.9470
Epoch 66/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0115 -
accuracy: 0.9962 - val_loss: 0.2883 - val_accuracy: 0.9470
Epoch 67/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0139 -
accuracy: 0.9945 - val_loss: 0.3136 - val_accuracy: 0.9497
Epoch 68/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0122 -
accuracy: 0.9955 - val_loss: 0.3187 - val_accuracy: 0.9486
Epoch 69/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0128 -
accuracy: 0.9944 - val_loss: 0.3468 - val_accuracy: 0.9481
Epoch 70/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0191 -
accuracy: 0.9928 - val_loss: 0.3448 - val_accuracy: 0.9431
Epoch 71/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0141 -
accuracy: 0.9952 - val_loss: 0.3339 - val_accuracy: 0.9420
Epoch 72/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0077 -
accuracy: 0.9973 - val_loss: 0.3257 - val_accuracy: 0.9502
Epoch 73/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0181 -
accuracy: 0.9937 - val_loss: 0.3440 - val_accuracy: 0.9431
Epoch 74/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0147 -
accuracy: 0.9945 - val_loss: 0.3110 - val_accuracy: 0.9459
Epoch 75/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0140 -
accuracy: 0.9954 - val_loss: 0.3554 - val_accuracy: 0.9437
Epoch 76/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0123 -
accuracy: 0.9959 - val_loss: 0.3196 - val_accuracy: 0.9497
Epoch 77/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0108 -
accuracy: 0.9962 - val_loss: 0.3644 - val_accuracy: 0.9464
Epoch 78/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0180 -
accuracy: 0.9941 - val_loss: 0.3778 - val_accuracy: 0.9420
Epoch 79/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0127 -
accuracy: 0.9958 - val_loss: 0.3327 - val_accuracy: 0.9420
Epoch 80/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0094 -
accuracy: 0.9967 - val_loss: 0.3699 - val_accuracy: 0.9475
Epoch 81/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0147 -
accuracy: 0.9943 - val_loss: 0.3265 - val_accuracy: 0.9486
Epoch 82/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0106 -
accuracy: 0.9963 - val_loss: 0.3280 - val_accuracy: 0.9481
Epoch 83/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0101 -
accuracy: 0.9967 - val_loss: 0.3406 - val_accuracy: 0.9486
Epoch 84/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0102 -
accuracy: 0.9967 - val_loss: 0.3587 - val_accuracy: 0.9464
Epoch 85/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0114 -
accuracy: 0.9958 - val_loss: 0.3676 - val_accuracy: 0.9442
Epoch 86/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0119 -
accuracy: 0.9952 - val_loss: 0.3554 - val_accuracy: 0.9410
Epoch 87/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0089 -
accuracy: 0.9974 - val_loss: 0.3610 - val_accuracy: 0.9492
Epoch 88/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0091 -
accuracy: 0.9970 - val_loss: 0.4277 - val_accuracy: 0.9404
Epoch 89/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0178 -
accuracy: 0.9937 - val_loss: 0.3436 - val_accuracy: 0.9508
Epoch 90/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0098 -
accuracy: 0.9967 - val_loss: 0.3558 - val_accuracy: 0.9513
Epoch 91/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0109 -
accuracy: 0.9962 - val_loss: 0.4206 - val_accuracy: 0.9415
Epoch 92/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0192 -
accuracy: 0.9941 - val_loss: 0.3428 - val_accuracy: 0.9453
Epoch 93/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0116 -
accuracy: 0.9962 - val_loss: 0.3141 - val_accuracy: 0.9492
Epoch 94/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0078 -
accuracy: 0.9970 - val_loss: 0.3599 - val_accuracy: 0.9459
Epoch 95/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0058 -
accuracy: 0.9978 - val_loss: 0.3747 - val_accuracy: 0.9475
Epoch 96/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0063 -
accuracy: 0.9971 - val_loss: 0.3997 - val_accuracy: 0.9459
Epoch 97/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0159 -
accuracy: 0.9958 - val_loss: 0.3611 - val_accuracy: 0.9420
Epoch 98/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0114 -
accuracy: 0.9958 - val_loss: 0.3565 - val_accuracy: 0.9464
Epoch 99/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0104 -
accuracy: 0.9959 - val_loss: 0.3956 - val_accuracy: 0.9464
Epoch 100/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0109 -
accuracy: 0.9966 - val_loss: 0.3779 - val_accuracy: 0.9502
Test results - Loss: 0.32264989614486694 - Accuracy: 95.31933665275574%
```

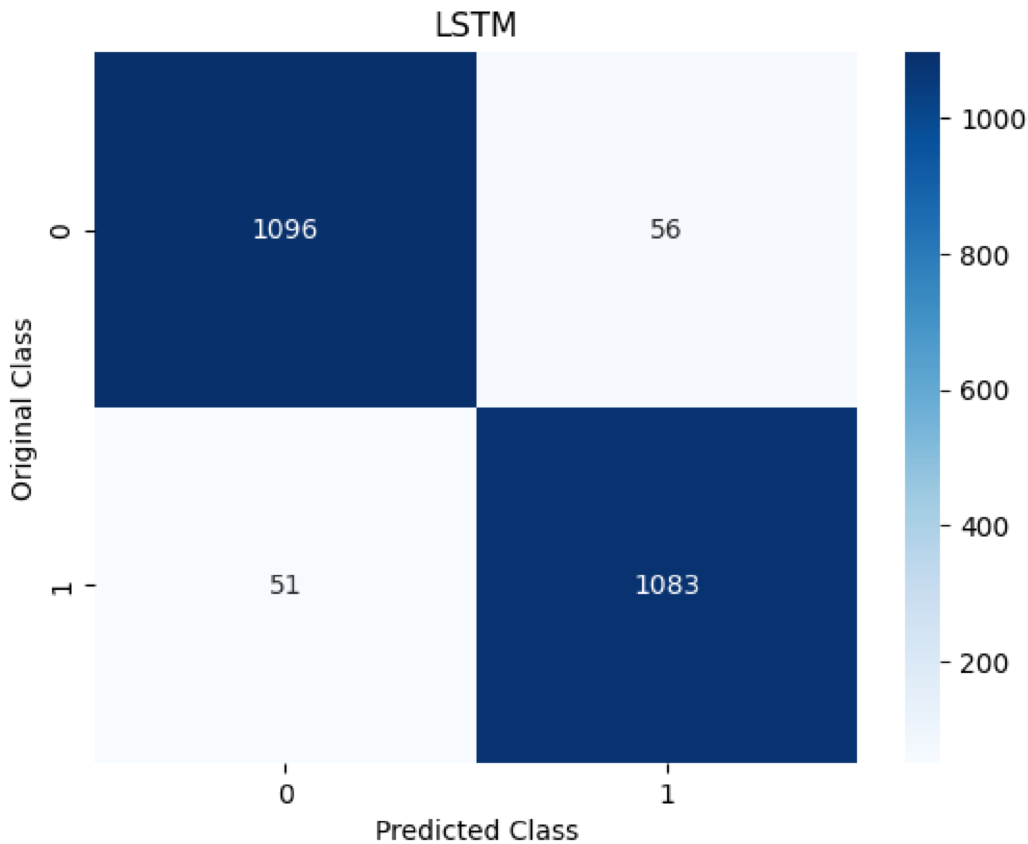
[ ]:

```
lstm_predict_proba = lstm_model.predict(X_test_reshape, batch_size=32)
lstm_predict_class = (lstm_predict_proba > 0.5).astype("int32")
print(classification_report(y_test, lstm_predict_class))
```

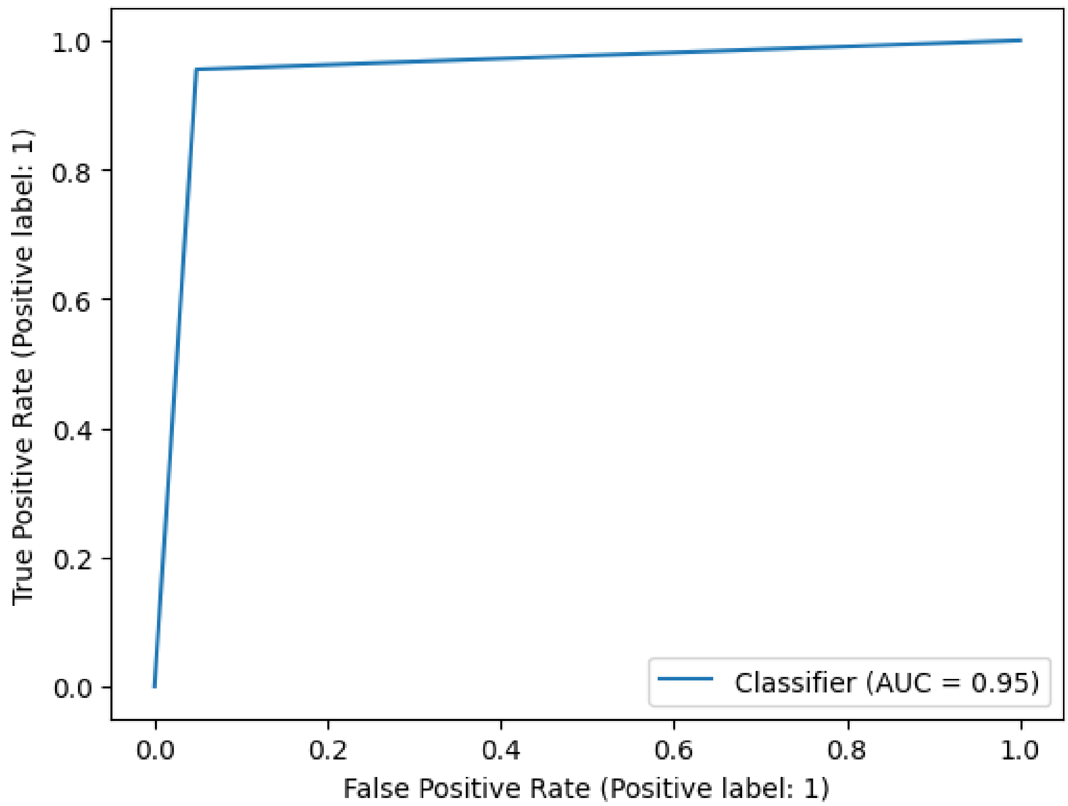
```
72/72 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.96      0.95      0.95      1152
           1       0.95      0.96      0.95      1134

    accuracy                           0.95      2286
   macro avg       0.95      0.95      0.95      2286
weighted avg       0.95      0.95      0.95      2286
```

[ ]:

```
sns.heatmap(confusion_matrix(y_test, lstm_predict_class), annot=True, fmt='g',␣
 ↪cmap='Blues')
plt.title("LSTM")
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



[ ]:
```
RocCurveDisplay.from_predictions(y_test,lstm_predict_class)
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
Classifier (AUC = 0.95)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)