correlation_target _label_87 8020 split .02 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)
```

[ ]:
```python
##print(df.corr()['Result'].sort_values())
## correlation values of features with target label
corr_col = abs(df.corr()['status']).sort_values(ascending=False)
corr_col = corr_col.rename_axis('Col').reset_index(name='Correlation')
corr_col
```

[ ]:

	Col	Correlation
0	status	1.000000e+00
1	google_index	7.311708e-01
2	page_rank	5.111371e-01
3	nb_www	4.434677e-01
4	ratio_digits_url	3.563946e-01
5	domain_in_title	3.428070e-01
6	nb_hyperlinks	3.426283e-01
7	phish_hints	3.353927e-01
8	domain_age	3.318891e-01
9	ip	3.216978e-01
10	nb_qm	2.943191e-01
11	length_url	2.485805e-01
12	ratio_intHyperlinks	2.439821e-01
13	nb_slash	2.422700e-01
14	length_hostname	2.383224e-01
15	nb_eq	2.333863e-01
16	ratio_digits_host	2.243349e-01
17	shortest_word_host	2.230840e-01
18	prefix_suffix	2.146807e-01
19	longest_word_path	2.127091e-01
20	tld_in_subdomain	2.088842e-01
21	empty_title	2.070428e-01
22	nb_dots	2.070288e-01
23	longest_words_raw	2.001466e-01
24	avg_word_path	1.972561e-01
25	avg_word_host	1.935017e-01
26	ratio_intMedia	1.933331e-01
27	length_words_raw	1.920105e-01
28	links_in_tags	1.844011e-01
29	safe_anchor	1.733973e-01
30	domain_with_copyright	1.730985e-01
31	nb_and	1.705464e-01
32	avg_words_raw	1.675637e-01
33	domain_registration_length	1.617188e-01
34	nb_com	1.562835e-01
35	ratio_extRedirection	1.508267e-01
36	external_favicon	1.465654e-01
37	statistical_report	1.439435e-01
38	nb_at	1.429146e-01
39	ratio_extMedia	1.404059e-01

40	abnormal_subdomain	1.281598e-01
41	longest_word_host	1.245156e-01
42	dns_record	1.221190e-01
43	https_token	1.146691e-01
44	nb_subdomains	1.128907e-01
45	suspecious_tld	1.100896e-01
46	shortening_service	1.061200e-01
47	nb_semicolumn	1.035541e-01
48	nb_hyphens	1.001075e-01
49	domain_in_brand	9.822216e-02
50	nb_colon	9.283531e-02
51	nb_extCSS	8.356663e-02
52	ratio_extHyperlinks	8.335725e-02
53	tld_in_path	7.914651e-02
54	shortest_word_path	7.436495e-02
55	nb_dslash	7.260234e-02
56	http_in_path	7.077624e-02
57	whois_registered_domain	6.697907e-02
58	brand_in_path	6.515575e-02
59	brand_in_subdomain	6.425702e-02
60	web_traffic	6.038772e-02
61	popup_window	5.760197e-02
62	nb_external_redirection	5.620994e-02
63	shortest_words_raw	3.936361e-02
64	nb_underscore	3.809134e-02
65	ratio_extErrors	3.470251e-02
66	nb_tilde	3.014233e-02
67	nb_percent	2.810129e-02
68	nb_star	2.646512e-02
69	nb_dollar	2.496206e-02
70	nb_redirection	2.440520e-02
71	random_domain	1.963062e-02
72	login_form	1.900010e-02
73	punycode	1.871039e-02
74	char_repeat	1.473217e-02
75	iframe	1.208332e-02
76	nb_comma	1.186465e-02
77	port	9.011116e-03
78	onmouseover	7.787061e-03
79	right_clic	4.680056e-03
80	nb_space	4.193222e-03
81	path_extension	5.592660e-17
82	nb_or	NaN
83	ratio_nullHyperlinks	NaN
84	ratio_intRedirection	NaN
85	ratio_intErrors	NaN
86	submit_email	NaN

```
87                          sfh          NaN
```

[ ]:
```
def correlation (corr_col, threshold):
        corr_feature = set()
        for index, row in corr_col.iterrows():
                if row['Correlation'] < threshold or np.
 ↪isnan(row['Correlation']):
                        corr_feature.add(row['Col'])
        return corr_feature
```

[ ]:
```
corr_feature = correlation(corr_col,.02)
len(set(corr_feature))
```

[ ]: 17

[ ]:
```
corr_feature
```

[ ]:
```
{'char_repeat',
 'iframe',
 'login_form',
 'nb_comma',
 'nb_or',
 'nb_space',
 'onmouseover',
 'path_extension',
 'port',
 'punycode',
 'random_domain',
 'ratio_intErrors',
 'ratio_intRedirection',
 'ratio_nullHyperlinks',
 'right_clic',
 'sfh',
 'submit_email'}
```

[ ]:
```
df.drop(corr_feature, axis=1, inplace=True)
```

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

[ ]: 71

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

[ ]:
```
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, 70), (2286, 70), (9144, 1), (2286, 1))
```

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

[ ]:
```
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.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)
```

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

# defining parameter range
param_grid = {'penalty' : ['l2'],
            'C' : [10, 20, 30], #0.1, 1, 10, 20,
            '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 15 candidates, totalling 150 fits
{'C': 30, 'max_iter': 2500, 'penalty': 'l2', 'solver': 'lbfgs'}
LogisticRegression(C=30, max_iter=2500)
0.9441158182970429
```

[ ]:
```
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)
```

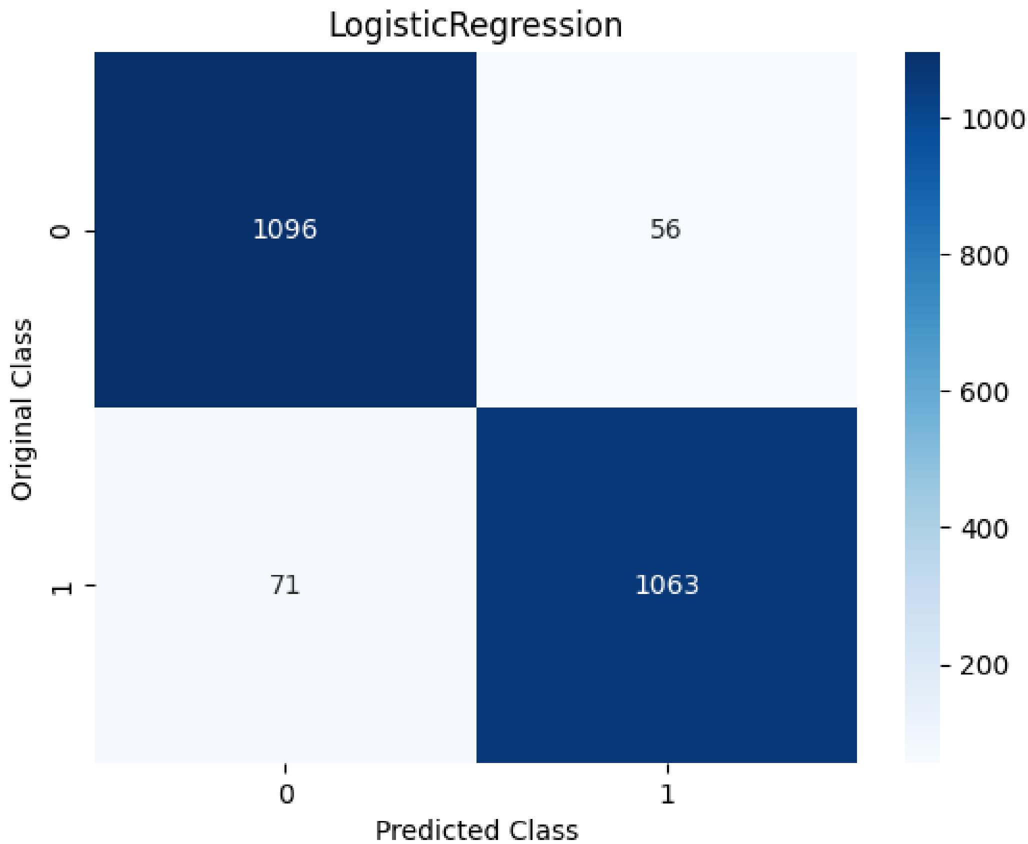
[ ]:
```
print ("Accuracy of logr classifier : ", accuracy_score(y_test,␣
↪logr_predict)*100)
```

```
Accuracy of logr classifier :  94.44444444444444
```

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

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      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': 5}
KNeighborsClassifier()
0.9246507873874521
```

```
[ ]: 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))
```

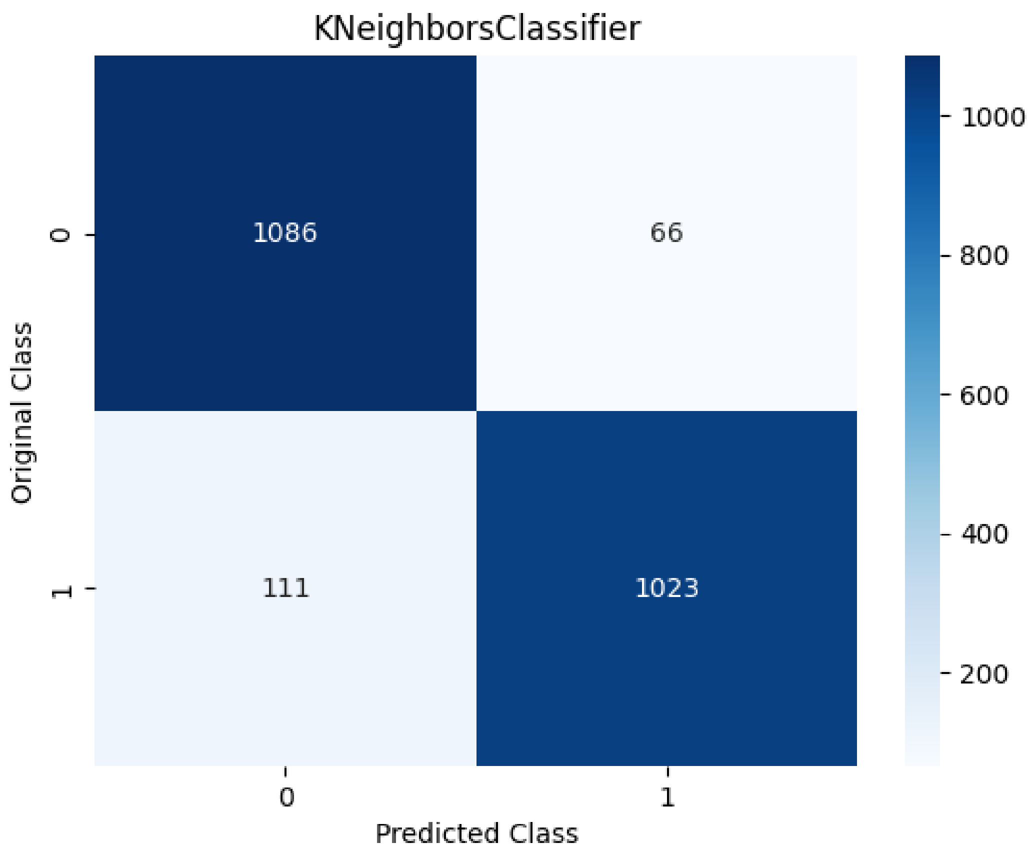
```
The accuracy of knn Classifier is:  92.25721784776903
```

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

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92      1152
           1       0.94      0.90      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': [1, 10],  #0.1, 1, 10
              'gamma': [1, 0.1], #
              'kernel': ['rbf']}   #'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 4 candidates, totalling 40 fits
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=10, gamma=0.1)
0.9569104757805121
```

[ ]:
```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)
```

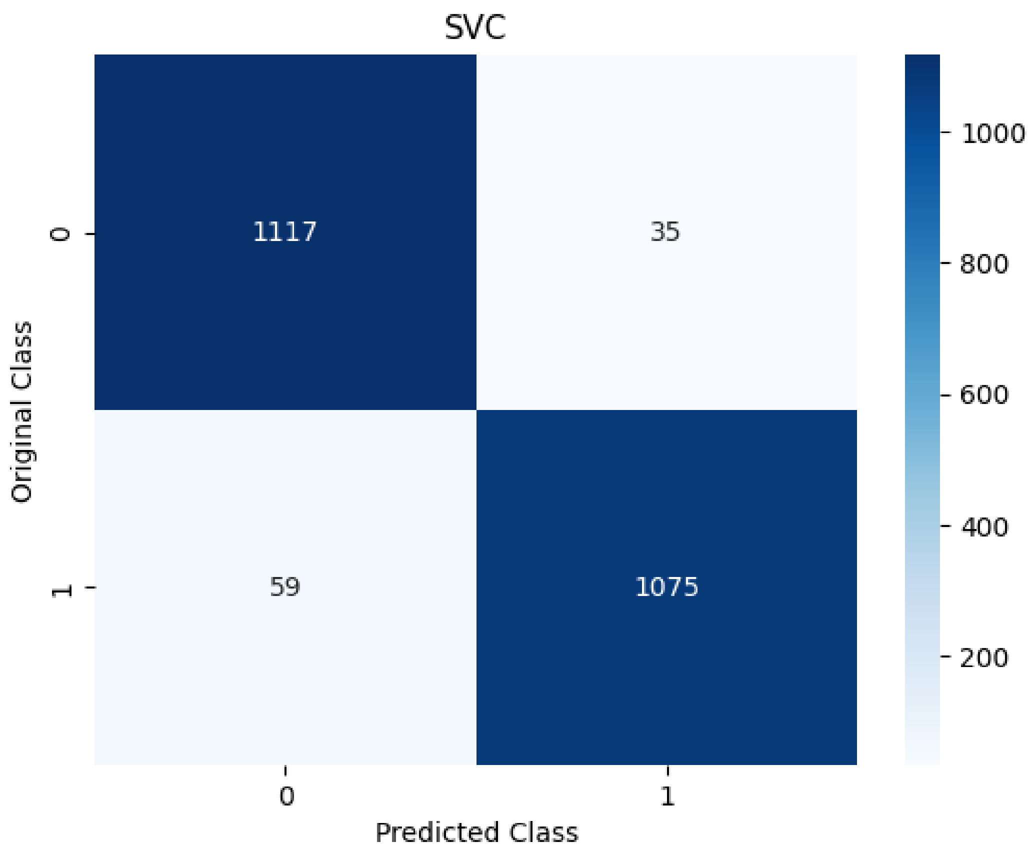
[ ]:
```python
print('The accuracy of svc Classifier is: ', 100.0 * accuracy_score(y_test,␣
↪svc_predict))
```

```
The accuracy of svc Classifier is:  95.88801399825022
```

[ ]:
```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], #.5
                         'gamma': [.1],  #1,.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 1 candidates, totalling 10 fits
{'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.1}
NuSVC(gamma=0.1, nu=0.1)
0.9587691167150936
```

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

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

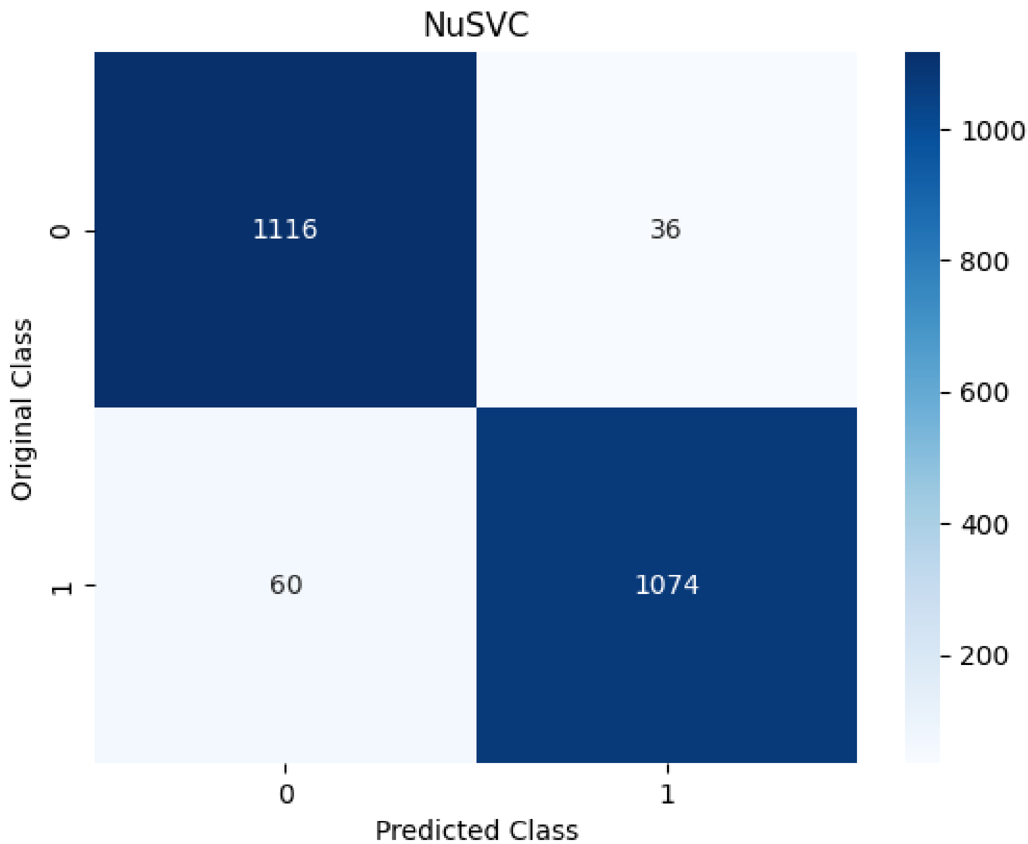
[ ]:
```
print('The accuracy of nusvc Classifier is: ', 100.0 * accuracy_score(y_test,␣
↪nusvc_predict))
```

```
The accuracy of nusvc Classifier is:  95.8005249343832
```

[ ]:
```
print(classification_report(y_test, nusvc_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, 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': 30, 'dual': False, 'loss': 'squared_hinge', 'penalty': 'l2', 'tol': 0.01}
LinearSVC(C=30, dual=False, tol=0.01)
0.9446634621133313
```

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

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

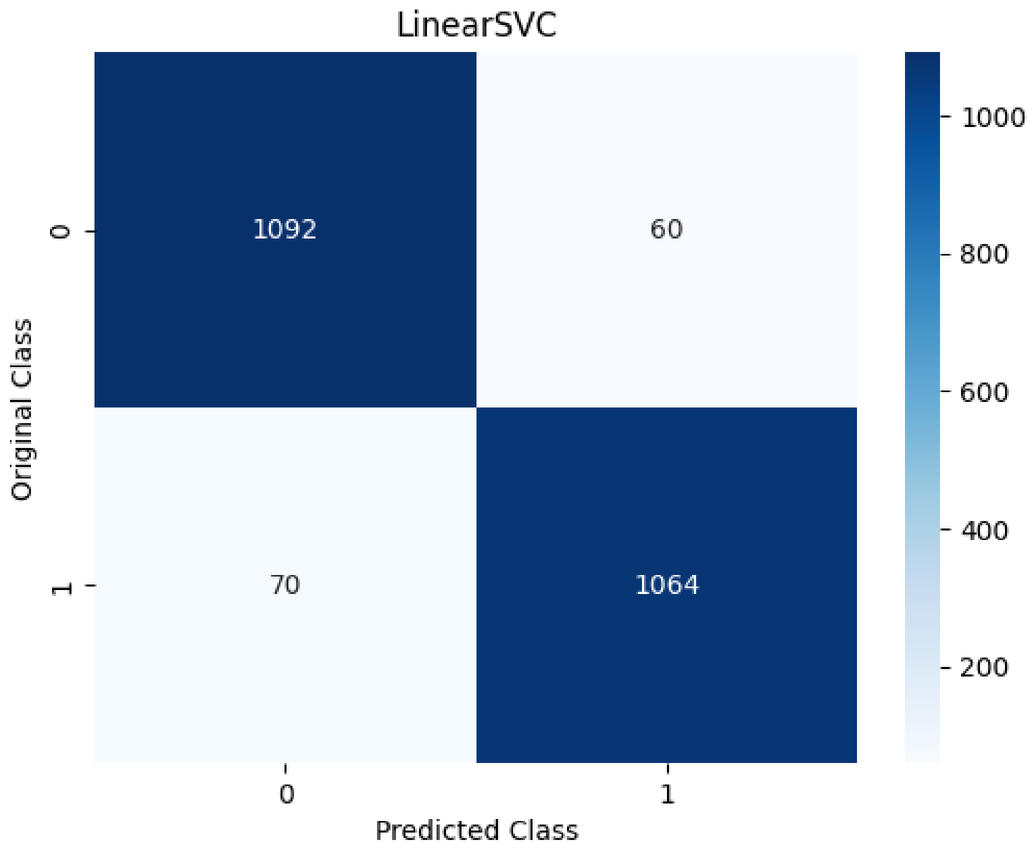
[ ]:
```
print('The accuracy of lsvc Classifier is: ', 100.0 * accuracy_score(y_test,␣
 ↪lsvc_predict))
```

```
The accuracy of lsvc Classifier is:  94.31321084864392
```

[ ]:
```
print(classification_report(y_test, lsvc_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, 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': [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 3 candidates, totalling 30 fits
{'n_estimators': 200}
```

```
AdaBoostClassifier(n_estimators=200)
0.9542872858150686
```

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

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

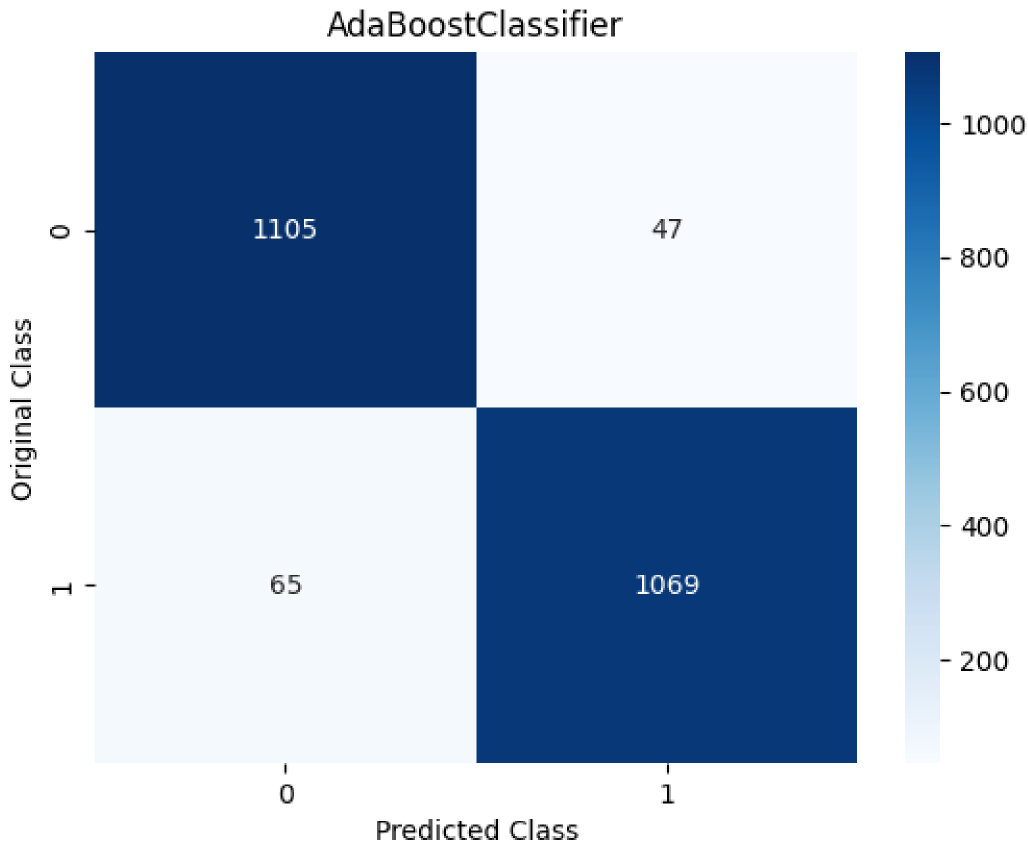
[ ]:
```python
print('The accuracy of Ada Boost Classifier is: ', 100.0 *
    ↪accuracy_score(ada_predict,y_test))
```

```
The accuracy of Ada Boost Classifier is:  95.10061242344707
```

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

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1152
           1       0.96      0.94      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
```

[ ]:
```python
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()
```



[ ]:
```
from xgboost import XGBClassifier


# defining parameter range
param_grid = {
    "gamma": [.01, .1, .5], #
    "n_estimators": [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 9 candidates, totalling 90 fits
{'gamma': 0.01, 'n_estimators': 200}
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=0.01, 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=200,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
0.9702528966531551
```

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

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

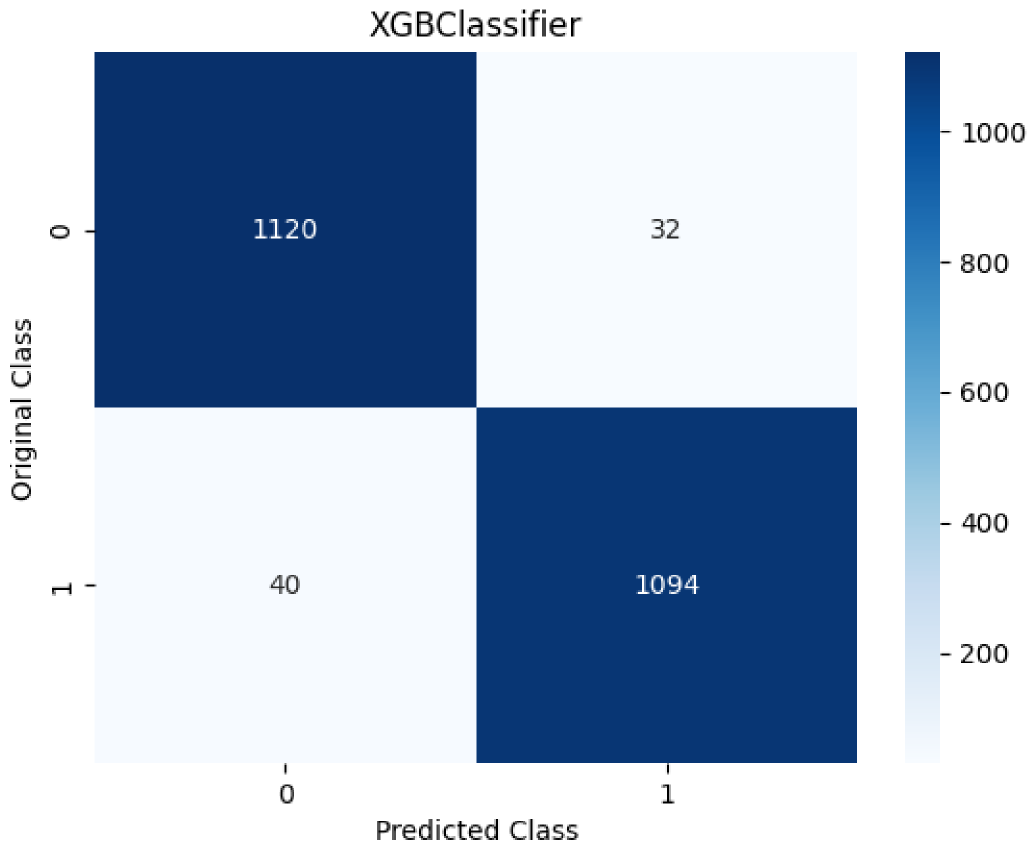
[ ]:
```
print('The accuracy of XGBoost Classifier is: ' , 100.0 *␣
 ↪accuracy_score(xgb_predict,y_test))
```

```
The accuracy of XGBoost Classifier is:  96.8503937007874
```

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

```
              precision    recall  f1-score   support

           0       0.97      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": [.5], #.1,.5,1
    "n_estimators": [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 3 candidates, totalling 30 fits
{'learning_rate': 0.5, 'n_estimators': 250}
GradientBoostingClassifier(learning_rate=0.5, n_estimators=250)
0.9652227044995279
```

[ ]:
```
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)
```

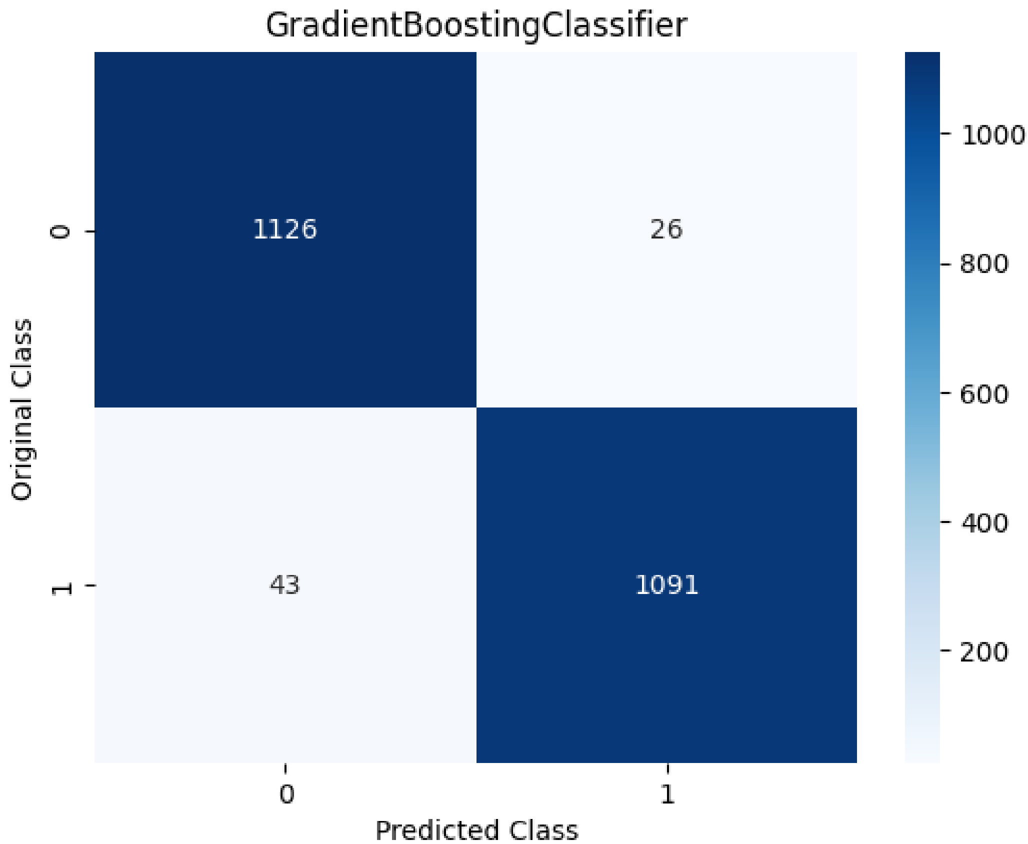
[ ]:
```
print('The accuracy of GradientBoost Classifier is: ' , 100.0 *␣
    ↪accuracy_score(gbc_predict,y_test))
```

```
The accuracy of GradientBoost Classifier is:  96.98162729658793
```

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

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1152
           1       0.98      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": [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 3 candidates, totalling 30 fits
{'base_estimator': DecisionTreeClassifier(), 'n_estimators': 150}
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=150)
0.9573495474166277
```

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

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

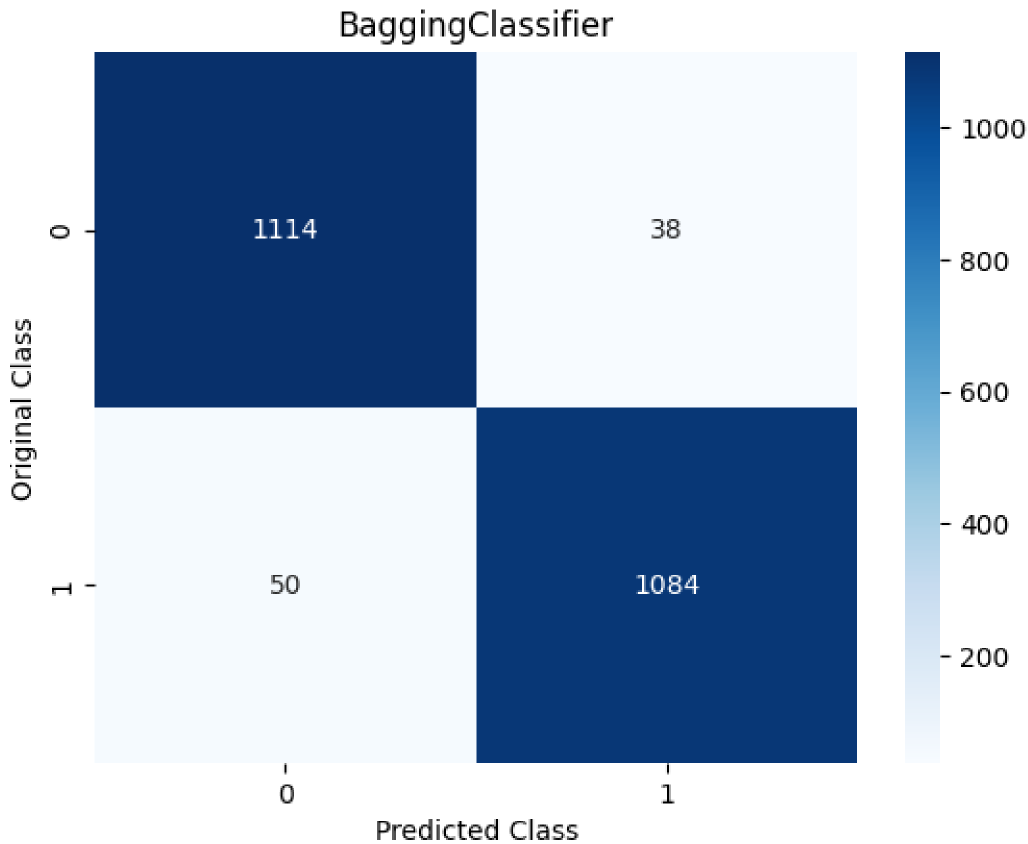
[ ]:
```
print('The accuracy of Bagging Classifier is: ' , 100.0 *
 ↪accuracy_score(y_test, bag_predict))
```

```
The accuracy of Bagging Classifier is:  96.15048118985126
```

[ ]:
```
print(classification_report(y_test, bag_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, 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]
}

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 4 candidates, totalling 40 fits
{'n_estimators': 150}
RandomForestClassifier(n_estimators=150)
0.9659880905405889
```

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

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

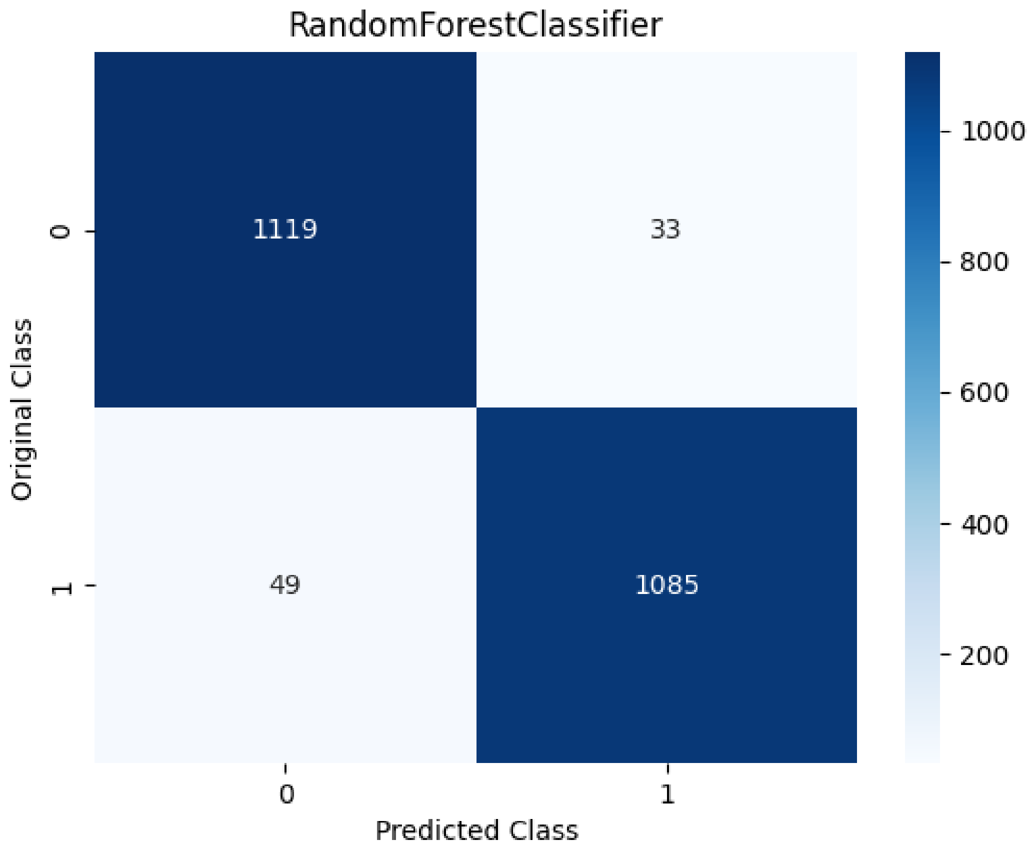
[ ]:
```
print('The accuracy of RandomForest Classifier is: ' , 100.0 *␣
↪accuracy_score(rfc_predict,y_test))
```

```
The accuracy of RandomForest Classifier is:  96.41294838145232
```

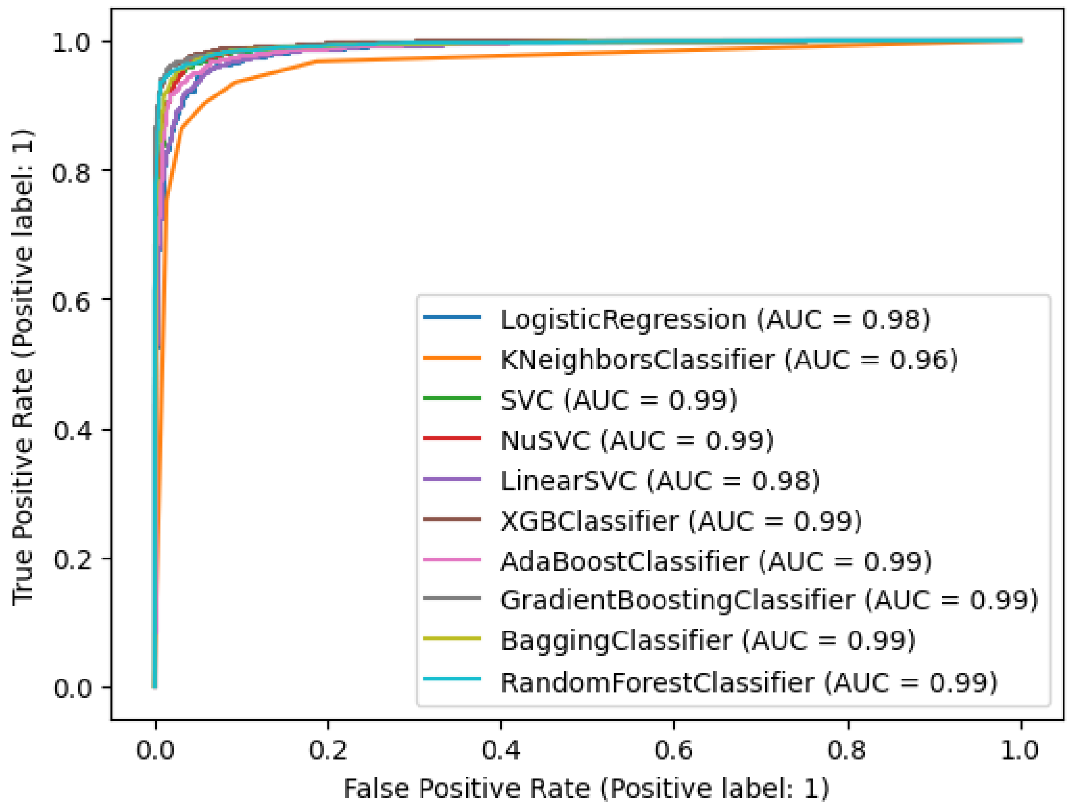
[ ]:
```
print(classification_report(y_test, rfc_predict))
```

```
              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
```

```python
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_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 100)               68400

 batch_normalization_6 (Batc  (None, 100)              400
 hNormalization)

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 100)               68400

 batch_normalization_6 (Batc  (None, 100)              400
 hNormalization)

 dense_24 (Dense)            (None, 50)                5050

 dense_25 (Dense)            (None, 25)                1275

 dense_26 (Dense)            (None, 10)                260

 dense_27 (Dense)            (None, 1)                 11

=================================================================
Total params: 75,396
Trainable params: 75,196
Non-trainable params: 200
_________________________________________________________________
Epoch 1/100
229/229 [==============================] - 4s 7ms/step - loss: 0.2298 -
accuracy: 0.9151 - val_loss: 0.3870 - val_accuracy: 0.9328
Epoch 2/100
229/229 [==============================] - 1s 5ms/step - loss: 0.1593 -
accuracy: 0.9386 - val_loss: 0.2143 - val_accuracy: 0.9399
Epoch 3/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.1353 -
accuracy: 0.9524 - val_loss: 0.1476 - val_accuracy: 0.9442
Epoch 4/100
229/229 [==============================] - 1s 5ms/step - loss: 0.1261 -
accuracy: 0.9526 - val_loss: 0.1498 - val_accuracy: 0.9508
Epoch 5/100
229/229 [==============================] - 1s 5ms/step - loss: 0.1230 -
accuracy: 0.9541 - val_loss: 0.1448 - val_accuracy: 0.9475
Epoch 6/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1100 -
accuracy: 0.9619 - val_loss: 0.1457 - val_accuracy: 0.9502
Epoch 7/100
229/229 [==============================] - 1s 4ms/step - loss: 0.1048 -
accuracy: 0.9632 - val_loss: 0.1559 - val_accuracy: 0.9470
Epoch 8/100
229/229 [==============================] - 1s 5ms/step - loss: 0.1007 -
accuracy: 0.9643 - val_loss: 0.1550 - val_accuracy: 0.9470
Epoch 9/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0868 -
accuracy: 0.9668 - val_loss: 0.1516 - val_accuracy: 0.9513
Epoch 10/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0907 -
accuracy: 0.9656 - val_loss: 0.1507 - val_accuracy: 0.9492
Epoch 11/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0791 -
accuracy: 0.9690 - val_loss: 0.1565 - val_accuracy: 0.9486
Epoch 12/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0770 -
accuracy: 0.9703 - val_loss: 0.1604 - val_accuracy: 0.9470
Epoch 13/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0708 -
accuracy: 0.9746 - val_loss: 0.1603 - val_accuracy: 0.9535
Epoch 14/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0739 -
accuracy: 0.9722 - val_loss: 0.1536 - val_accuracy: 0.9535
Epoch 15/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0685 -
accuracy: 0.9744 - val_loss: 0.1782 - val_accuracy: 0.9437
Epoch 16/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0621 -
accuracy: 0.9761 - val_loss: 0.1827 - val_accuracy: 0.9459
Epoch 17/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0618 -
accuracy: 0.9765 - val_loss: 0.1781 - val_accuracy: 0.9492
Epoch 18/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0548 -
accuracy: 0.9792 - val_loss: 0.1747 - val_accuracy: 0.9497
Epoch 19/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0503 -
accuracy: 0.9807 - val_loss: 0.1958 - val_accuracy: 0.9464
Epoch 20/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0515 -
accuracy: 0.9800 - val_loss: 0.2061 - val_accuracy: 0.9442
Epoch 21/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0493 -
accuracy: 0.9817 - val_loss: 0.2169 - val_accuracy: 0.9431
Epoch 22/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0459 -
accuracy: 0.9815 - val_loss: 0.2165 - val_accuracy: 0.9459
Epoch 23/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0430 -
accuracy: 0.9841 - val_loss: 0.2094 - val_accuracy: 0.9492
Epoch 24/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0434 -
accuracy: 0.9840 - val_loss: 0.1943 - val_accuracy: 0.9513
Epoch 25/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0388 -
accuracy: 0.9851 - val_loss: 0.2114 - val_accuracy: 0.9475
Epoch 26/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0384 -
accuracy: 0.9847 - val_loss: 0.2319 - val_accuracy: 0.9448
Epoch 27/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0387 -
accuracy: 0.9858 - val_loss: 0.2270 - val_accuracy: 0.9486
Epoch 28/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0364 -
accuracy: 0.9859 - val_loss: 0.2375 - val_accuracy: 0.9415
Epoch 29/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0266 -
accuracy: 0.9892 - val_loss: 0.2472 - val_accuracy: 0.9470
Epoch 30/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0307 -
accuracy: 0.9891 - val_loss: 0.2246 - val_accuracy: 0.9486
Epoch 31/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0269 -
accuracy: 0.9897 - val_loss: 0.2316 - val_accuracy: 0.9431
Epoch 32/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0270 -
accuracy: 0.9896 - val_loss: 0.2240 - val_accuracy: 0.9426
Epoch 33/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0269 -
accuracy: 0.9896 - val_loss: 0.2644 - val_accuracy: 0.9431
Epoch 34/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0258 -
accuracy: 0.9895 - val_loss: 0.2614 - val_accuracy: 0.9502
Epoch 35/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0252 -
accuracy: 0.9895 - val_loss: 0.2603 - val_accuracy: 0.9475
Epoch 36/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0313 -
accuracy: 0.9884 - val_loss: 0.2373 - val_accuracy: 0.9437
Epoch 37/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0236 -
accuracy: 0.9903 - val_loss: 0.2698 - val_accuracy: 0.9481
Epoch 38/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0188 -
accuracy: 0.9936 - val_loss: 0.2591 - val_accuracy: 0.9481
Epoch 39/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0218 -
accuracy: 0.9922 - val_loss: 0.2841 - val_accuracy: 0.9459
Epoch 40/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0158 -
accuracy: 0.9948 - val_loss: 0.2590 - val_accuracy: 0.9470
Epoch 41/100
229/229 [==============================] - 1s 4ms/step - loss: 0.0212 -
accuracy: 0.9928 - val_loss: 0.2868 - val_accuracy: 0.9508
Epoch 42/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0202 -
accuracy: 0.9914 - val_loss: 0.2959 - val_accuracy: 0.9453
Epoch 43/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0241 -
accuracy: 0.9904 - val_loss: 0.2805 - val_accuracy: 0.9492
Epoch 44/100
229/229 [==============================] - 2s 8ms/step - loss: 0.0198 -
accuracy: 0.9922 - val_loss: 0.3156 - val_accuracy: 0.9431
Epoch 45/100
229/229 [==============================] - 2s 9ms/step - loss: 0.0164 -
accuracy: 0.9947 - val_loss: 0.2915 - val_accuracy: 0.9492
Epoch 46/100
229/229 [==============================] - 2s 8ms/step - loss: 0.0162 -
accuracy: 0.9936 - val_loss: 0.2865 - val_accuracy: 0.9492
Epoch 47/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0188 -
accuracy: 0.9940 - val_loss: 0.3167 - val_accuracy: 0.9481
Epoch 48/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0153 -
accuracy: 0.9947 - val_loss: 0.2853 - val_accuracy: 0.9519
Epoch 49/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0161 -
accuracy: 0.9949 - val_loss: 0.3231 - val_accuracy: 0.9399
Epoch 50/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0148 -
accuracy: 0.9948 - val_loss: 0.3304 - val_accuracy: 0.9481
Epoch 51/100
```

```
229/229 [==============================] - 1s 6ms/step - loss: 0.0195 -
accuracy: 0.9934 - val_loss: 0.3343 - val_accuracy: 0.9497
Epoch 52/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0184 -
accuracy: 0.9930 - val_loss: 0.3207 - val_accuracy: 0.9459
Epoch 53/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0186 -
accuracy: 0.9925 - val_loss: 0.3145 - val_accuracy: 0.9508
Epoch 54/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0186 -
accuracy: 0.9926 - val_loss: 0.3025 - val_accuracy: 0.9502
Epoch 55/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0138 -
accuracy: 0.9956 - val_loss: 0.2915 - val_accuracy: 0.9497
Epoch 56/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0180 -
accuracy: 0.9930 - val_loss: 0.3257 - val_accuracy: 0.9470
Epoch 57/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0146 -
accuracy: 0.9944 - val_loss: 0.3130 - val_accuracy: 0.9486
Epoch 58/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0156 -
accuracy: 0.9940 - val_loss: 0.3551 - val_accuracy: 0.9442
Epoch 59/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0186 -
accuracy: 0.9941 - val_loss: 0.3086 - val_accuracy: 0.9464
Epoch 60/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0128 -
accuracy: 0.9963 - val_loss: 0.3086 - val_accuracy: 0.9502
Epoch 61/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0115 -
accuracy: 0.9959 - val_loss: 0.2958 - val_accuracy: 0.9524
Epoch 62/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0088 -
accuracy: 0.9966 - val_loss: 0.3253 - val_accuracy: 0.9486
Epoch 63/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0092 -
accuracy: 0.9973 - val_loss: 0.3320 - val_accuracy: 0.9448
Epoch 64/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0083 -
accuracy: 0.9967 - val_loss: 0.3162 - val_accuracy: 0.9481
Epoch 65/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0163 -
accuracy: 0.9937 - val_loss: 0.3340 - val_accuracy: 0.9382
Epoch 66/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0180 -
accuracy: 0.9940 - val_loss: 0.3368 - val_accuracy: 0.9513
Epoch 67/100
```

```
229/229 [==============================] - 1s 6ms/step - loss: 0.0139 -
accuracy: 0.9955 - val_loss: 0.3124 - val_accuracy: 0.9519
Epoch 68/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0118 -
accuracy: 0.9958 - val_loss: 0.3232 - val_accuracy: 0.9437
Epoch 69/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0162 -
accuracy: 0.9937 - val_loss: 0.3353 - val_accuracy: 0.9535
Epoch 70/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0147 -
accuracy: 0.9948 - val_loss: 0.3340 - val_accuracy: 0.9448
Epoch 71/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0115 -
accuracy: 0.9960 - val_loss: 0.3453 - val_accuracy: 0.9442
Epoch 72/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0123 -
accuracy: 0.9960 - val_loss: 0.3271 - val_accuracy: 0.9508
Epoch 73/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0088 -
accuracy: 0.9970 - val_loss: 0.3513 - val_accuracy: 0.9470
Epoch 74/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0095 -
accuracy: 0.9971 - val_loss: 0.3312 - val_accuracy: 0.9481
Epoch 75/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0104 -
accuracy: 0.9973 - val_loss: 0.3384 - val_accuracy: 0.9464
Epoch 76/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0128 -
accuracy: 0.9952 - val_loss: 0.3247 - val_accuracy: 0.9492
Epoch 77/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0064 -
accuracy: 0.9979 - val_loss: 0.3674 - val_accuracy: 0.9486
Epoch 78/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0083 -
accuracy: 0.9974 - val_loss: 0.3621 - val_accuracy: 0.9453
Epoch 79/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0079 -
accuracy: 0.9975 - val_loss: 0.3997 - val_accuracy: 0.9497
Epoch 80/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0182 -
accuracy: 0.9934 - val_loss: 0.4112 - val_accuracy: 0.9399
Epoch 81/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0102 -
accuracy: 0.9962 - val_loss: 0.3839 - val_accuracy: 0.9420
Epoch 82/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0094 -
accuracy: 0.9964 - val_loss: 0.3930 - val_accuracy: 0.9470
Epoch 83/100
```

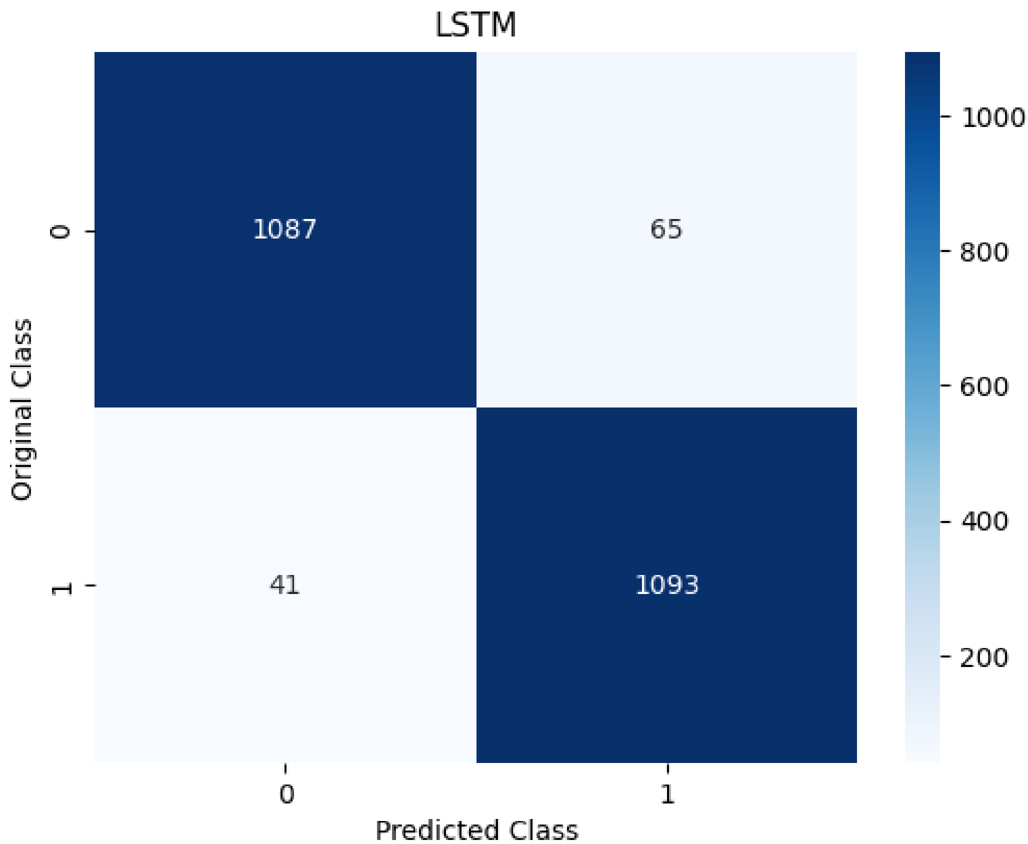
```
229/229 [==============================] - 1s 5ms/step - loss: 0.0099 -
accuracy: 0.9964 - val_loss: 0.3831 - val_accuracy: 0.9486
Epoch 84/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0053 -
accuracy: 0.9984 - val_loss: 0.4060 - val_accuracy: 0.9470
Epoch 85/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0069 -
accuracy: 0.9977 - val_loss: 0.3951 - val_accuracy: 0.9513
Epoch 86/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0082 -
accuracy: 0.9978 - val_loss: 0.3972 - val_accuracy: 0.9453
Epoch 87/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0164 -
accuracy: 0.9952 - val_loss: 0.3722 - val_accuracy: 0.9453
Epoch 88/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0146 -
accuracy: 0.9947 - val_loss: 0.3521 - val_accuracy: 0.9470
Epoch 89/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0084 -
accuracy: 0.9967 - val_loss: 0.3890 - val_accuracy: 0.9486
Epoch 90/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0084 -
accuracy: 0.9969 - val_loss: 0.3869 - val_accuracy: 0.9470
Epoch 91/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0052 -
accuracy: 0.9981 - val_loss: 0.3904 - val_accuracy: 0.9492
Epoch 92/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0166 -
accuracy: 0.9952 - val_loss: 0.3850 - val_accuracy: 0.9464
Epoch 93/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0148 -
accuracy: 0.9951 - val_loss: 0.3615 - val_accuracy: 0.9453
Epoch 94/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0086 -
accuracy: 0.9977 - val_loss: 0.3637 - val_accuracy: 0.9470
Epoch 95/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0067 -
accuracy: 0.9981 - val_loss: 0.4498 - val_accuracy: 0.9410
Epoch 96/100
229/229 [==============================] - 1s 6ms/step - loss: 0.0081 -
accuracy: 0.9982 - val_loss: 0.3746 - val_accuracy: 0.9497
Epoch 97/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0037 -
accuracy: 0.9990 - val_loss: 0.3796 - val_accuracy: 0.9459
Epoch 98/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0037 -
accuracy: 0.9990 - val_loss: 0.3903 - val_accuracy: 0.9415
Epoch 99/100
```

```
229/229 [==============================] - 1s 5ms/step - loss: 0.0057 -
accuracy: 0.9979 - val_loss: 0.4402 - val_accuracy: 0.9437
Epoch 100/100
229/229 [==============================] - 1s 5ms/step - loss: 0.0122 -
accuracy: 0.9956 - val_loss: 0.4576 - val_accuracy: 0.9437
Test results - Loss: 0.30803728103637695 - Accuracy: 95.3630805015564%
```

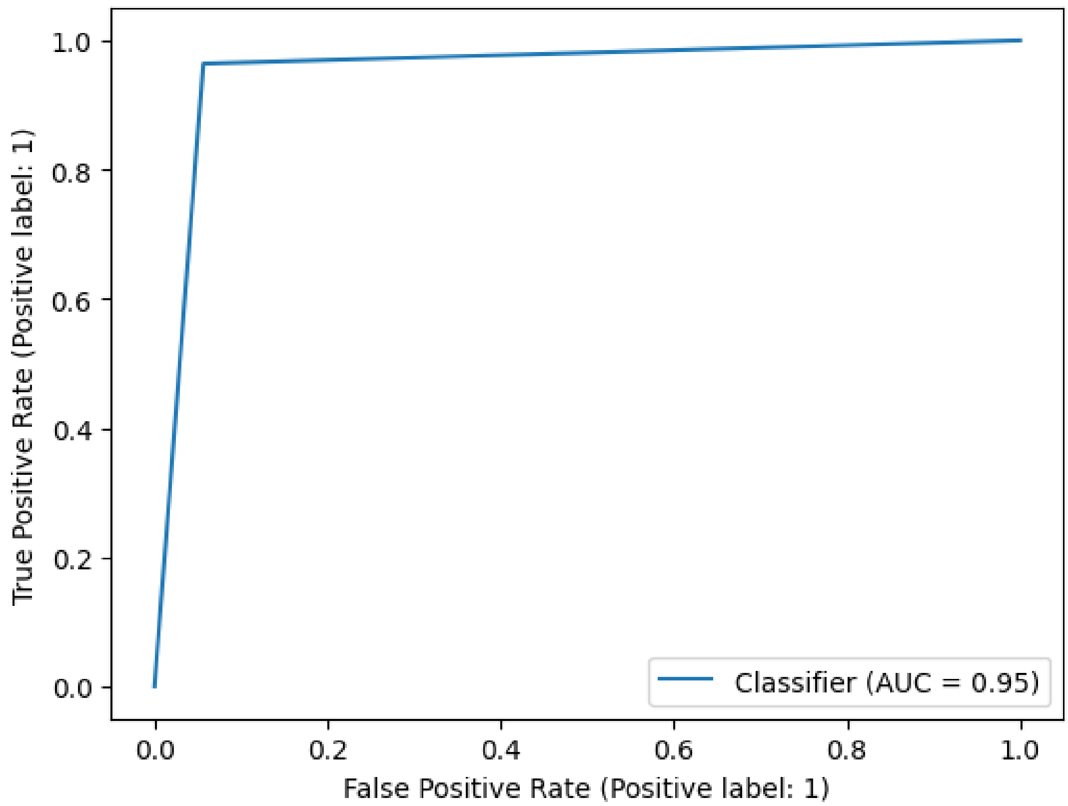
[ ]:
```
lstm_predict_proba = lstm_model.predict(X_test_reshape, batch_size=32)
lstm_predict_class = (lstm_predict_proba > 0.5).astype("int32")
print(classification_report(y_test, lstm_predict_class))
```

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

           0       0.96      0.94      0.95      1152
           1       0.94      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)