

Name :- Vansh Vig
UCID :- vv455
Email :- vv455@njit.edu
Date :- 11/16/2024
Professor : - Yasser Abdulllah
CS634 :- Data Mining

FinalProject-Report

November 16, 2024

1. Project Overview :

This project focuses on evaluating the performance of different machine learning algorithms, namely K-Nearest Neighbors (KNN), Random Forest (RF), and Long Short-Term Memory (LSTM) models, for a classification task. The dataset used contains features and labels, where the goal is to predict the class labels based on the features.

2. How to Run the Program :

1. Clone the Repository :

```
git clone https://github.com/vanshvigggg/DATA_MINING_FINAL_TERM_PROJECT
```

2. Navigate to the Project Directory :

```
cd DATA_MINING_FINAL_TERM_PROJECT
```

3. Install Required Packages :

```
pip install -r requirements.txt
```

4. Run the Script / Notebook :

```
python ds634_finalproject_cardiovascular_disease_prediction.py
```

3. Required Packages :

- pandas (v1.3.3)
- numpy (v1.21.2)
- matplotlib (v3.4.3)
- seaborn (v0.11.2)
- scikit-learn (v0.24.2)
- keras (v2.6.0)

4. Dataset :

The dataset "health_data.csv"

contains features and labels necessary for training and evaluating the models. Ensure the dataset is in the CSV format and has appropriate columns.

5. Model Evaluation :

The program evaluates three types of models:

- K-Nearest Neighbors (KNN)
- Random Forest (RF)
- Long Short-Term Memory (LSTM)

For each model, the script performs 10 iterations of 10-fold cross-validation and calculates various performance metrics including confusion matrix metrics, Brier score, Area Under Curve (AUC), etc.

6. Output :

The program generates the following outputs:

- Evaluation metrics for each algorithm in each iteration.
- Average performance metrics for each algorithm across all iterations.
- ROC curves for KNN and Random Forest models.

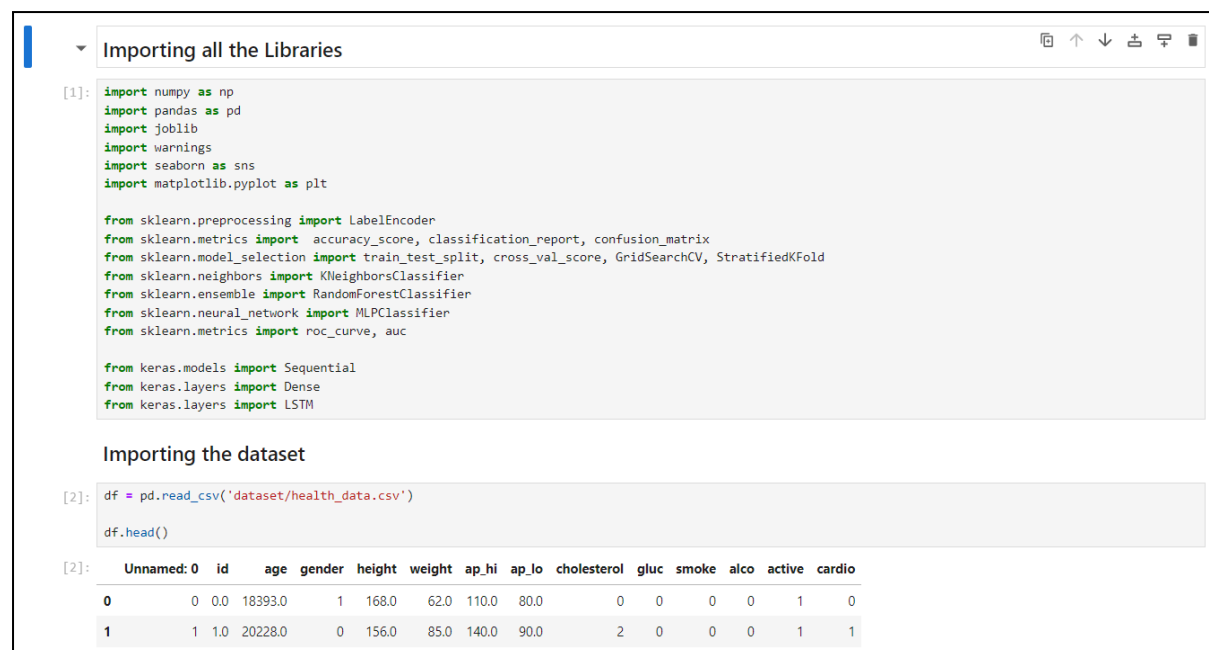
7. Conclusion :

The project demonstrates how to evaluate and compare the performance of different machine learning algorithms for a classification task. Users can modify the script, dataset, or parameters to suit their specific requirements.

8. Github Repository Link :

https://github.com/vanshvigggg/DATA_MINING_FINAL_TERM_PROJECT

9. Result and Screenshots :



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, titled 'Importing all the Libraries', contains a series of import statements for various Python libraries including numpy, pandas, joblib, warnings, seaborn, matplotlib, sklearn (preprocessing, metrics, model_selection, neighbors, ensemble, neural_network, metrics), and keras (models, layers). The second cell, titled 'Importing the dataset', shows the loading of a CSV file named 'health_data.csv' and the first few rows of the resulting DataFrame.

```
[1]: import numpy as np
import pandas as pd
import joblib
import warnings
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

Importing the dataset

```
[2]: df = pd.read_csv('dataset/health_data.csv')
df.head()
```

Unnamed: 0	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	
0	0	0.0	18393.0	1	168.0	62.0	110.0	80.0	0	0	0	0	1	0
1	1	1.0	20228.0	0	156.0	85.0	140.0	90.0	2	0	0	0	1	1

Checking missing values

```
[4]: print(df.isnull().sum())

Unnamed: 0    0
id            0
age           0
gender        0
height        0
weight        0
ap_hi         0
ap_lo         0
cholesterol   0
gluc          0
smoke         0
alco          0
active        0
cardio        0
dtype: int64

[5]: null_rows = df[df.isnull().any(axis=1)]
print("Rows with null values:")
print(null_rows)

Rows with null values:
Empty DataFrame
Columns: [Unnamed: 0, id, age, gender, height, weight, ap_hi, ap_lo, cholesterol, gluc, smoke, alco, active, cardio]
Index: []

[6]: #drop id
df = df.drop(['id', 'Unnamed: 0'], axis=1)
```

Removing Outliers:

It is important to remove outliers to improve the performance of our prediction models. We have removed outliers that fall outside the range of 2.5% to 97.5% in all instances of ap_hi, ap_lo, weight, and height features. This process has decreased the entries in the data set from 70,000 to 60,142 records.

```
[7]: df.drop(df[(df['height'] > df['height'].quantile(0.975)) | (df['height'] < df['height'].quantile(0.025))].index,inplace=True)
df.drop(df[(df['weight'] > df['weight'].quantile(0.975)) | (df['weight'] < df['weight'].quantile(0.025))].index,inplace=True)
df.drop(df[(df['ap_hi'] > df['ap_hi'].quantile(0.975)) | (df['ap_hi'] < df['ap_hi'].quantile(0.025))].index,inplace=True)
df.drop(df[(df['ap_lo'] > df['ap_lo'].quantile(0.975)) | (df['ap_lo'] < df['ap_lo'].quantile(0.025))].index,inplace=True)
len(df)

[7]: 60142

[8]: #Cases where diastolic pressure is higher than systolic
df[df['ap_lo'] > df['ap_hi']].shape[0]

[8]: 0

[9]: df.describe()

[9]:
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
count	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000
mean	19468.719979	0.347311	164.554854	73.426805	125.770526	81.046307	0.350953	0.220229	0.085631	0.051877	0.803648
std	2460.510296	0.476120	6.830174	11.614806	13.761847	8.239157	0.670076	0.567607	0.279820	0.221781	0.397241
min	10798.000000	0.000000	150.000000	52.000000	100.000000	60.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	17677.250000	0.000000	160.000000	65.000000	120.000000	80.000000	0.000000	0.000000	0.000000	0.000000	1.000000
50%	19705.000000	0.000000	165.000000	72.000000	120.000000	80.000000	0.000000	0.000000	0.000000	0.000000	1.000000
75%	21321.000000	1.000000	169.000000	80.000000	135.000000	90.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	23713.000000	1.000000	180.000000	106.000000	163.000000	100.000000	2.000000	2.000000	1.000000	1.000000	1.000000

Data Processing and cleaning

```
[10]: df['age'] = (df['age'] / 365).round().astype('int') #Converting age from days to years

print(df.head())

   age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  \
0   50      1   168.0   62.0  110.0   80.0           0      0      0      0
1   55      0   156.0   85.0  140.0   90.0           2      0      0      0
2   52      0   165.0   64.0  130.0   70.0           2      0      0      0
3   48      1   169.0   82.0  150.0  100.0           0      0      0      0
4   48      0   156.0   56.0  100.0   60.0           0      0      0      0

   active  cardio
0         1         0
1         1         1
2         0         1
3         1         1
4         0         0

[11]: ## Define the bin edges and labels
age_edges = [30, 35, 40, 45, 50, 55, 60, 65]
age_labels = [0, 1, 2, 3, 4, 5, 6]

# bin in 5 years span
df['age_group'] = pd.cut(df['age'], bins=7, labels=range(7), include_lowest=True, right=True)
df.head()

[11]:
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	age_group
0	50	1	168.0	62.0	110.0	80.0	0	0	0	0	1	0	3
1	55	0	156.0	85.0	140.0	90.0	2	0	0	0	1	1	4
2	52	0	165.0	64.0	130.0	70.0	2	0	0	0	0	1	4
3	48	1	169.0	82.0	150.0	100.0	0	0	0	0	1	1	3

```
[14]: df_og=df

df=df.drop(['height','weight','ap_hi','ap_lo','age'],axis=1)

df.head()
```

```
[14]:
```

	gender	cholesterol	gluc	smoke	alco	active	cardio	age_group	bmi	map
0	1	0	0	0	0	1	0	3	1	2
1	0	2	0	0	0	1	1	4	3	4
2	0	2	0	0	0	0	1	4	1	2
3	1	0	0	0	0	1	1	3	2	5
4	0	0	0	0	0	0	0	3	1	0

```
[15]: le = LabelEncoder()
df = df.apply(le.fit_transform)
df.describe()
```

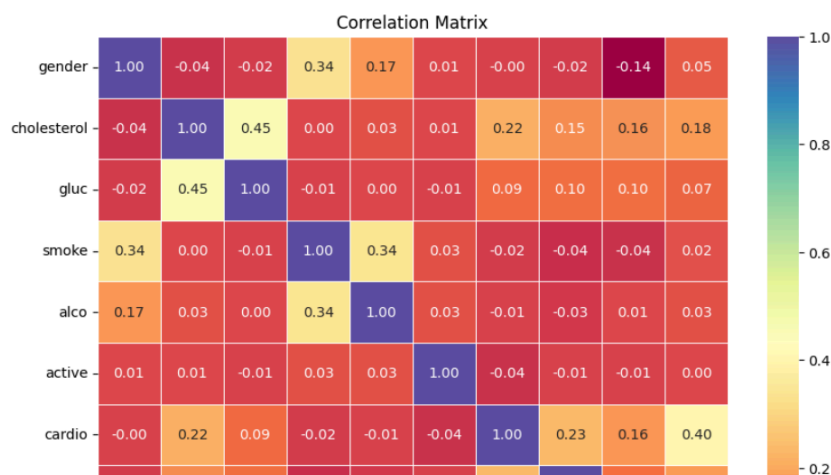
```
[15]:
```

	gender	cholesterol	gluc	smoke	alco	active	cardio	age_group	bmi	map
count	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000	60142.000000
mean	0.347311	0.350953	0.220229	0.085631	0.051877	0.803648	0.488228	4.042233	1.673440	2.359449
std	0.476120	0.670076	0.567607	0.279820	0.221781	0.397241	0.499866	1.377070	0.898707	1.186906
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	3.000000	1.000000	2.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	4.000000	2.000000	2.000000
75%	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	5.000000	2.000000	3.000000

```
[16]: # Set up figure
plt.figure(figsize=(10, 8))

# Draw correlation matrix
sns.heatmap(df.corr(), annot=True, cmap='Spectral', fmt=".2f", linewidths=.5)

# Show the figure
plt.title('Correlation Matrix')
plt.show()
```



```
[17]: cardio_0 = df[df['cardio'] == 0].sample(n=5000, random_state=42)
cardio_1 = df[df['cardio'] == 1].sample(n=5000, random_state=42)

# Concatenate the sliced dataframes
data = pd.concat([cardio_0, cardio_1])

# Shuffle the data to mix 0s and 1s
data = data.sample(frac=1, random_state=42).reset_index(drop=True)
data.shape
```

```
[17]: (10000, 10)
```

```
[18]: x = data.drop(['cardio','gender','alco'], axis=1)
y = data['cardio']

x.head()
```

```
[18]:
```

	cholesterol	gluc	smoke	active	age_group	bmi	map
0	0	0	0	1	2	2	4
1	0	0	0	1	3	1	2
2	1	1	1	1	4	2	3
3	1	0	1	1	3	2	3
4	0	0	0	0	6	1	2

Splitting the dataset into train and test

```
[19]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1)
```

```
[20]: x_train.info()
```

Function to calculate performance metrics

```
[21]: def calc_metrics(confusion_matrix):
    TP, FN = confusion_matrix[0][0], confusion_matrix[0][1]
    FP, TN = confusion_matrix[1][0], confusion_matrix[1][1]
    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
    FPR = FP / (TN + FP)
    FNR = FN / (TP + FN)
    Precision = TP / (TP + FP)
    F1_measure = 2 * TP / (2 * TP + FP + FN)
    Accuracy = (TP + TN) / (TP + FP + FN + TN)
    Error_rate = (FP + FN) / (TP + FP + FN + TN)
    BACC = (TPR + TNR) / 2
    TSS = TPR - FPR
    HSS = 2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))
    metrics = [TP, TN, FP, FN, TPR, TNR, FPR, FNR, Precision, F1_measure, Accuracy, Error_rate, BACC, TSS, HSS]
    return metrics

[22]: import numpy as np
from sklearn.metrics import confusion_matrix, brier_score_loss, roc_auc_score

def get_metrics(model, X_train, X_test, y_train, y_test, LSTM_flag):
    metrics = []

    if LSTM_flag == 1:
        # Convert data to numpy array
        Xtrain, Xtest, ytrain, ytest = map(np.array, [X_train, X_test, y_train, y_test])
        # Reshape data
        shape = Xtrain.shape
        Xtrain_resaped = Xtrain.reshape(len(Xtrain), shape[1], 1)
        Xtest_resaped = Xtest.reshape(len(Xtest), shape[1], 1)
        model.fit(Xtrain_resaped, ytrain, epochs=50, validation_data=(Xtest_resaped, ytest), verbose=0)
        lstm_scores = model.evaluate(Xtest_resaped, ytest, verbose=0)
        predict_prob = model.predict(Xtest_resaped)
```

Finding best parameters for Random Forest model

```
[23]: # build the model
rfModel = RandomForestClassifier(random_state=42)

# Fit the model
rfModel.fit(x_train, y_train)

# Make predictions
rf_pred = rfModel.predict(x_test)

# accuracy
rf_accuracy = accuracy_score(y_test, rf_pred)*100
print(f"Accuracy without CV: {rf_accuracy:.2f}")

Accuracy without CV: 71.45

[24]: param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10, 20],
    'max_features': ['sqrt', 'log2', None],
}

# Create grid search
rf_gridsearch = GridSearchCV(estimator=rfModel, param_grid=param_grid, cv=10, scoring='accuracy', n_jobs=-1)

# Fit grid search
rf_gridsearch.fit(x_train, y_train)

[24]: GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=42),
                 n_jobs=-1,
                 param_grid={'max_depth': [None, 10, 20, 30],
                             'max_features': ['sqrt', 'log2', None],
                             'min_samples_split': [2, 5, 10, 20],
                             'n_estimators': [100, 200, 300, 500]}
```

```
[25]: rf_best_params = rf_gridsearch.best_params_
best_estimator = rf_gridsearch.best_estimator_

print(f"Best Parameters : {rf_best_params}")
print(f"Best Estimator : {best_estimator}")

Best Parameters : {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_split': 20, 'n_estimators': 100}
Best Estimator : RandomForestClassifier(max_depth=10, min_samples_split=20, random_state=42)

[26]: max_depth = rf_gridsearch.best_params_['max_depth']
min_samples_split = rf_gridsearch.best_params_['min_samples_split']
n_estimators = rf_gridsearch.best_params_['n_estimators']

[27]: rf_pred_CV = best_estimator.predict(x_test)

[28]: rf_accuracy_cv = accuracy_score(y_test, rf_pred_CV)*100
print(f"Best Accuracy: {rf_accuracy_cv:.2f}")

Best Accuracy: 73.05

[29]: print(f"Random Forest accuracy without CV : {rf_accuracy:.2f}")
print(f"Random Forest accuracy with CV : {rf_accuracy_cv:.2f}")

Random Forest accuracy without CV : 71.45
Random Forest accuracy with CV : 73.05
```

▾ Finding best parameters for the KNN model ¶

```
[30]: knn_parameters = {"n_neighbors": [ 3, 4, 5, 6, 8, 10]}
# Create KNN model
knn_model = KNeighborsClassifier()
# Perform grid search with cross-validation
knn_cv = GridSearchCV(knn_model, knn_parameters, cv=10, n_jobs=-1)
knn_cv.fit(x_train, y_train)
# Print the best parameters found by GridSearchCV
print("\nBest Parameters for KNN based on GridSearchCV: ", knn_cv.best_params_)
```

Best Parameters for KNN based on GridSearchCV: {'n_neighbors': 8}

```
[31]: best_n_neighbors = knn_cv.best_params_['n_neighbors']

[32]: metric_columns = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR', 'Precision',
                      'F1_measure', 'Accuracy', 'Error_rate', 'BACC', 'TSS', 'HSS', 'Brier_score',
                      'AUC', 'Acc_by_package_fn']

# Initialize metrics lists for each algorithm
knn_metrics_list, rf_metrics_list, lstm_metrics_list = [], [], []

# 10 Iterations of 10-fold cross-validation
cv_stratified = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

for iter_num, (train_index, test_index) in enumerate(cv_stratified.split(x, y), start=1):
    # KNN Model
    knn_model = KNeighborsClassifier(n_neighbors=best_n_neighbors)

    # Random Forest Model
    rf_model = RandomForestClassifier(min_samples_split=min_samples_split, n_estimators=n_estimators, max_depth=max_depth)

    # LSTM model
    lstm_model = Sequential()
    lstm_model.add(LSTM(64, activation='relu', return_sequences=False))
    lstm_model.add(Dense(1, activation='sigmoid'))

    # Compile model
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Get metrics for each algorithm
    knn_metrics = get_metrics(knn_model, x_train, x_test, y_train, y_test, 0)
    rf_metrics = get_metrics(rf_model, x_train, x_test, y_train, y_test, 0)
    lstm_metrics = get_metrics(lstm_model, x_train, x_test, y_train, y_test, 1)
```

```
metrics_all_df = pd.DataFrame([knn_metrics, rf_metrics, lstm_metrics],
                              columns=metric_columns, index=['KNN', 'RF', 'LSTM'])
```

```
# Display metrics for all algorithms in each iteration
print('\nIteration {}: \n'.format(iter_num))
print('\n----- Metrics for all Algorithms in Iteration {} ----- \n'.format(iter_num))
print(metrics_all_df.round(decimals=2).T)
print('\n')
```

63/63  1s 9ms/step

Iteration 1:

----- Metrics for all Algorithms in Iteration 1 -----

	KNN	RF	LSTM
TP	616.00	677.00	692.00
TN	777.00	778.00	758.00
FP	219.00	218.00	238.00
FN	388.00	327.00	312.00
TPR	0.61	0.67	0.69
TNR	0.78	0.78	0.76
FPR	0.22	0.22	0.24
FNR	0.39	0.33	0.31
Precision	0.74	0.76	0.74
F1_measure	0.67	0.71	0.72
Accuracy	0.70	0.73	0.72
Error_rate	0.30	0.27	0.28
BACC	0.70	0.73	0.73
TSS	0.39	0.46	0.45
HSS	0.39	0.46	0.45
Brier_score	0.21	0.19	0.19
AUC	0.74	0.79	0.79
Acc_by_package_fn	0.70	0.73	0.73

63/63  1s 8ms/step

Precision	0.74	0.76	0.74
F1_measure	0.67	0.72	0.72
Accuracy	0.70	0.73	0.73
Error_rate	0.30	0.27	0.27
BACC	0.70	0.73	0.73
TSS	0.39	0.46	0.46
HSS	0.39	0.46	0.46
Brier_score	0.21	0.19	0.19
AUC	0.74	0.78	0.79
Acc_by_package_fn	0.70	0.73	0.73

63/63  1s 6ms/step

Iteration 10:

----- Metrics for all Algorithms in Iteration 10 -----

	KNN	RF	LSTM
TP	616.00	675.00	704.00
TN	777.00	780.00	744.00
FP	219.00	216.00	252.00
FN	388.00	329.00	300.00
TPR	0.61	0.67	0.70
TNR	0.78	0.78	0.75
FPR	0.22	0.22	0.25
FNR	0.39	0.33	0.30
Precision	0.74	0.76	0.74
F1_measure	0.67	0.71	0.72
Accuracy	0.70	0.73	0.72
Error_rate	0.30	0.27	0.28
BACC	0.70	0.73	0.72
TSS	0.39	0.46	0.45
HSS	0.39	0.46	0.45
Brier_score	0.21	0.19	0.19
AUC	0.74	0.78	0.79
Acc_by_package_fn	0.70	0.73	0.72

```
[33]: # Initialize metric index for each iteration
metric_index_df = ['iter1', 'iter2', 'iter3', 'iter4', 'iter5', 'iter6', 'iter7', 'iter8', 'iter9', 'iter10']

# Create DataFrames for each algorithm's metrics
knn_metrics_df = pd.DataFrame(knn_metrics_list, columns=metric_columns, index=metric_index_df)
rf_metrics_df = pd.DataFrame(rf_metrics_list, columns=metric_columns, index=metric_index_df)
lstm_metrics_df = pd.DataFrame(lstm_metrics_list, columns=metric_columns, index=metric_index_df)

# Display metrics for each algorithm in each iteration
algorithm_names = ['KNN', 'RF', 'LSTM']
for i, metrics_df in enumerate([knn_metrics_df, rf_metrics_df, lstm_metrics_df], start=1):
    print('\nMetrics for Algorithm {}: \n'.format(algorithm_names[i-1]))
    print(metrics_df.round(decimals=2).T)
    print('\n')
```

Metrics for Algorithm KNN:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7 \
TP	616.00	616.00	616.00	616.00	616.00	616.00	616.00
TN	777.00	777.00	777.00	777.00	777.00	777.00	777.00
FP	219.00	219.00	219.00	219.00	219.00	219.00	219.00
FN	388.00	388.00	388.00	388.00	388.00	388.00	388.00
TPR	0.61	0.61	0.61	0.61	0.61	0.61	0.61
TNR	0.78	0.78	0.78	0.78	0.78	0.78	0.78
FPR	0.22	0.22	0.22	0.22	0.22	0.22	0.22
FNR	0.39	0.39	0.39	0.39	0.39	0.39	0.39
Precision	0.74	0.74	0.74	0.74	0.74	0.74	0.74
F1_measure	0.67	0.67	0.67	0.67	0.67	0.67	0.67
Accuracy	0.70	0.70	0.70	0.70	0.70	0.70	0.70
Error_rate	0.30	0.30	0.30	0.30	0.30	0.30	0.30
BACC	0.70	0.70	0.70	0.70	0.70	0.70	0.70
TSS	0.39	0.39	0.39	0.39	0.39	0.39	0.39
HSS	0.39	0.39	0.39	0.39	0.39	0.39	0.39
Brier_score	0.21	0.21	0.21	0.21	0.21	0.21	0.21
AUC	0.74	0.74	0.74	0.74	0.74	0.74	0.74
Acc_by_package_fn	0.70	0.70	0.70	0.70	0.70	0.70	0.70

Metrics for Algorithm RF:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7 \
TP	677.00	680.00	678.00	680.00	685.00	678.00	676.00
TN	778.00	783.00	781.00	783.00	775.00	779.00	782.00
FP	218.00	213.00	215.00	213.00	221.00	217.00	214.00
FN	327.00	324.00	326.00	324.00	319.00	326.00	328.00
TPR	0.67	0.68	0.68	0.68	0.68	0.68	0.67
TNR	0.78	0.79	0.78	0.79	0.78	0.78	0.79
FPR	0.22	0.21	0.22	0.21	0.22	0.22	0.21
FNR	0.33	0.32	0.32	0.32	0.32	0.32	0.33
Precision	0.76	0.76	0.76	0.76	0.76	0.76	0.76
F1_measure	0.71	0.72	0.71	0.72	0.72	0.71	0.71
Accuracy	0.73	0.73	0.73	0.73	0.73	0.73	0.73
Error_rate	0.27	0.27	0.27	0.27	0.27	0.27	0.27
BACC	0.73	0.73	0.73	0.73	0.73	0.73	0.73
TSS	0.46	0.46	0.46	0.46	0.46	0.46	0.46
HSS	0.46	0.46	0.46	0.46	0.46	0.46	0.46
Brier_score	0.19	0.19	0.19	0.19	0.19	0.19	0.19
AUC	0.79	0.78	0.78	0.78	0.78	0.78	0.78
Acc_by_package_fn	0.73	0.73	0.73	0.73	0.73	0.73	0.73

	iter8	iter9	iter10
TP	661.00	679.00	675.00
TN	792.00	782.00	780.00
FP	204.00	214.00	216.00
FN	343.00	325.00	329.00
TPR	0.66	0.68	0.67
TNR	0.80	0.79	0.78
FPR	0.20	0.21	0.22
FNR	0.34	0.32	0.33
Precision	0.76	0.76	0.76
F1_measure	0.71	0.72	0.71
Accuracy	0.73	0.73	0.73
Error_rate	0.27	0.27	0.27
BACC	0.73	0.73	0.73
TSS	0.45	0.46	0.46
HSS	0.45	0.46	0.46

Metrics for Algorithm LSTM:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7 \
TP	692.00	684.00	619.00	650.00	659.00	664.00	711.00
TN	758.00	770.00	828.00	800.00	791.00	789.00	744.00
FP	238.00	226.00	168.00	196.00	205.00	207.00	252.00
FN	312.00	320.00	385.00	354.00	345.00	340.00	293.00
TPR	0.69	0.68	0.62	0.65	0.66	0.66	0.71
TNR	0.76	0.77	0.83	0.80	0.79	0.79	0.75
FPR	0.24	0.23	0.17	0.20	0.21	0.21	0.25
FNR	0.31	0.32	0.38	0.35	0.34	0.34	0.29
Precision	0.74	0.75	0.79	0.77	0.76	0.76	0.74
F1_measure	0.72	0.71	0.69	0.70	0.71	0.71	0.72
Accuracy	0.72	0.73	0.72	0.72	0.72	0.73	0.73
Error_rate	0.28	0.27	0.28	0.28	0.28	0.27	0.27
BACC	0.73	0.73	0.72	0.73	0.73	0.73	0.73
TSS	0.45	0.45	0.45	0.45	0.45	0.45	0.46
HSS	0.45	0.45	0.45	0.45	0.45	0.45	0.46
Brier_score	0.19	0.19	0.19	0.19	0.19	0.19	0.19
AUC	0.79	0.79	0.79	0.79	0.79	0.79	0.79
Acc_by_package_fn	0.73	0.73	0.72	0.73	0.73	0.73	0.73

	iter8	iter9	iter10
TP	663.00	690.00	704.00
TN	793.00	765.00	744.00
FP	203.00	231.00	252.00
FN	341.00	314.00	300.00
TPR	0.66	0.69	0.70
TNR	0.80	0.77	0.75
FPR	0.20	0.23	0.25
FNR	0.34	0.31	0.30
Precision	0.77	0.75	0.74
F1_measure	0.71	0.72	0.72
Accuracy	0.73	0.73	0.72
Error_rate	0.27	0.27	0.28
BACC	0.73	0.73	0.72
TSS	0.46	0.46	0.45
HSS	0.46	0.46	0.45

Average Result for each model

```
[34]: # Calculate the average metrics for each algorithm
knn_avg_df = knn_metrics_df.mean()
rf_avg_df = rf_metrics_df.mean()
lstm_avg_df = lstm_metrics_df.mean()
# Create a DataFrame with the average performance for each algorithm
avg_performance_df = pd.DataFrame({'KNN': knn_avg_df, 'RF': rf_avg_df, 'LSTM': lstm_avg_df}, index=metric_columns)
# Display the average performance for each algorithm
print(avg_performance_df.round(decimals=2))
print('\n')
```

	KNN	RF	LSTM
TP	616.00	676.90	673.60
TN	777.00	781.50	778.20
FP	219.00	214.50	217.80
FN	388.00	327.10	330.40
TPR	0.61	0.67	0.67
TNR	0.78	0.78	0.78
FPR	0.22	0.22	0.22
FNR	0.39	0.33	0.33
Precision	0.74	0.76	0.76
F1_measure	0.67	0.71	0.71
Accuracy	0.70	0.73	0.73
Error_rate	0.30	0.27	0.27
BACC	0.70	0.73	0.73
TSS	0.39	0.46	0.45
HSS	0.39	0.46	0.45
Brier_score	0.21	0.19	0.19
AUC	0.74	0.78	0.79
Acc_by_package_fn	0.70	0.73	0.73

Plotting ROC-SUC Curve for all the models

```
[35]: # Train models with best parameters
best_knn_model = KNeighborsClassifier(n_neighbors= best_n_neighbors)
best_rf_model = RandomForestClassifier(n_estimators=rf_best_params['n_estimators'], min_samples_split=rf_best_params['min_samples_split'])

# Fit models
best_knn_model.fit(x_train, y_train)
best_rf_model.fit(x_train, y_train)

#Lstm model
lstm_model = Sequential()
lstm_model.add(LSTM(64, activation='relu', return_sequences=False))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Convert data to numpy array
X_train_array = x_train.to_numpy()
X_test_array = x_test.to_numpy()
y_train_array = y_train.to_numpy()
y_test_array = y_test.to_numpy()

# Reshape data for LSTM model compatibility
input_shape = X_train_array.shape
input_train = X_train_array.reshape(len(X_train_array), input_shape[1], 1)
input_test = X_test_array.reshape(len(X_test_array), input_shape[1], 1)
output_train = y_train_array
output_test = y_test_array

# Train the LSTM model
history = lstm_model.fit(input_train, output_train, epochs=50, validation_data=(input_test, output_test), verbose=0)

# Predict probabilities for test set
knn_probs = best_knn_model.predict_proba(x_test)[: , 1]
rf_probs = best_rf_model.predict_proba(x_test)[: , 1]
lstm_probs = lstm_model.predict(x_test).ravel() # Assuming lstm model is already trained
```

```
# LSTM ROC AUC curve
plt.subplot(1, 3, 3)
plt.plot(lstm_fpr, lstm_tpr, color='red', lw=2, label='LSTM ROC curve (AUC = %0.2f)' % lstm_roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LSTM ROC AUC Curve')
plt.legend(loc="lower right")

plt.tight_layout()
plt.show()
```

63/63 — 1s 6ms/step

