abid_mahum_finalproject

April 11, 2024

1 Final Project - Data Mining

2 Using KNN, SVM, RF and LSTM To Predict Diabetes

2.1 Goal

"My project aims to implement a variety of machine learning classification algorithms, along with a deep learning model, to predict the likelihood of a patient having diabetes. This prediction is based on specific diagnostic measurements provided in the dataset."

2.1.1 Importing the packages and libraries that are required for the project

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")


from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import brier_score_loss
from sklearn.metrics import auc

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

2.1.2 Loading Data And Preprocessing

[3]:
```
diab = pd.read_csv('diabetes.csv')
diab.describe()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	
std	3.369578	31.972618	19.355807	15.952218	115.244002	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

[4]:
```
diab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[5]:
```
def impute_missing_values(dataframe):
    columns_to_impute = ['Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin', 'BMI']
```

```
    for column in columns_to_impute:
        dataframe.loc[dataframe[column] == 0, column] = np.nan
        dataframe[column].fillna(dataframe[column].median(), inplace=True)

    return dataframe

# Assuming 'diab' is your DataFrame
diab = impute_missing_values(diab)
```

[6]:
```
diab.head()
```

[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

2.1.3 Separating The Dataset into Features and Output label

[7]:
```
# Feature and label separation
features = diab.iloc[:, :-1]
labels = diab.iloc[:, -1]
```

2.1.4 Data Visualization

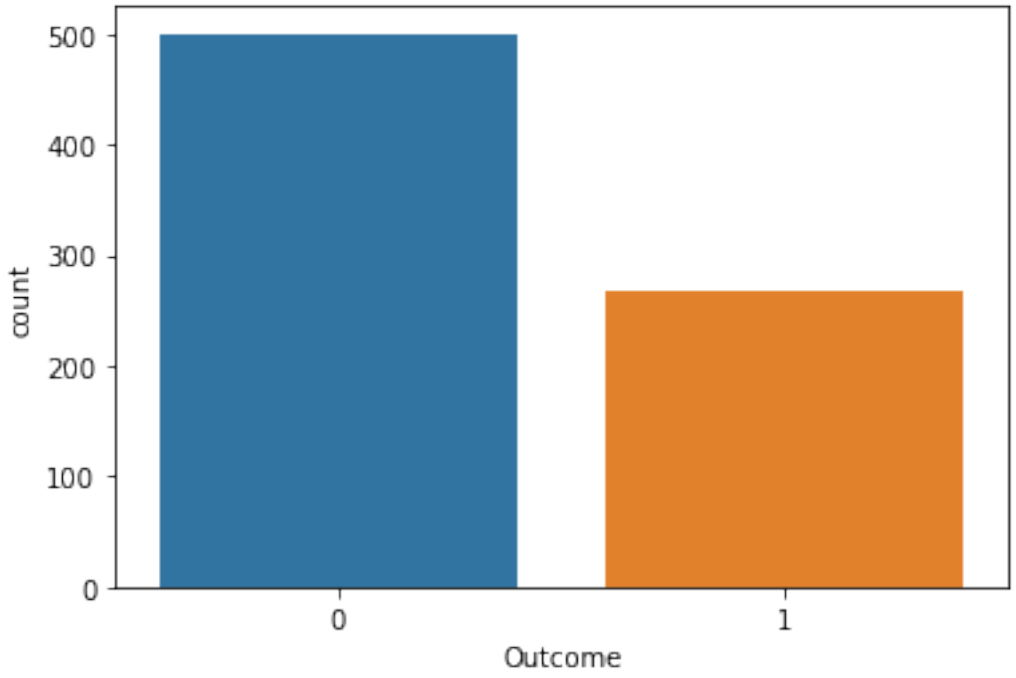
In the dataset, an observable data imbalance exists in the target label, where the number of patients without diabetes is twice the number of patients with diabetes.

Now, we have two potential approaches: one involves addressing the data imbalance in the train dataset that we will generate, while the other entails employing stratified sampling in the train-test split. Additionally, we will use stratified k-fold cross-validation to maintain a similar label ratio in both the training and testing sets.

For this project, we will adopt the strategy of stratified sampling and apply stratified k-fold cross-validation.

[8]:
```
# Visualizing the distribution of the target variable
sns.countplot(labels, label="Count")
plt.show()

# Checking for data imbalance
```

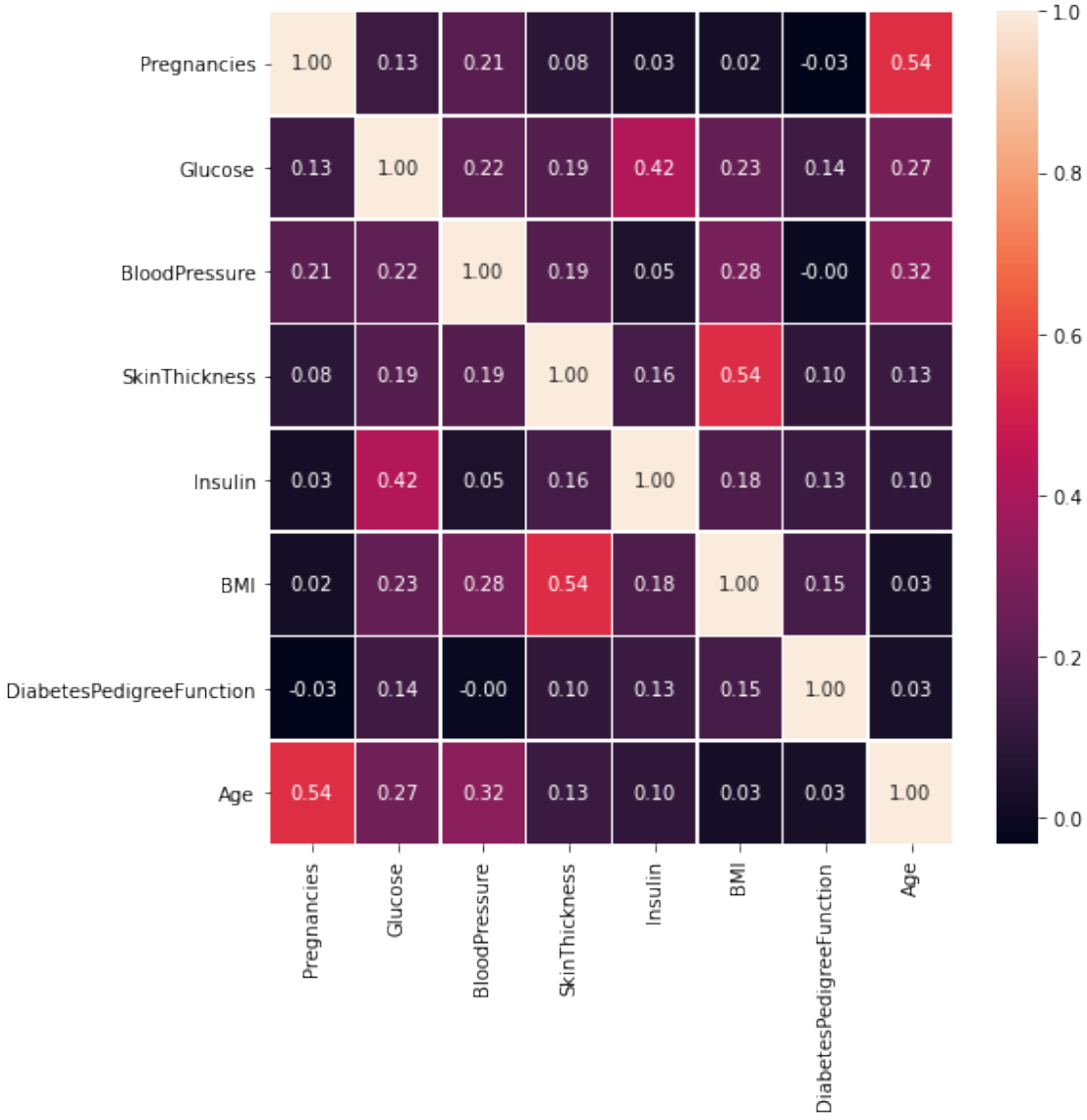
```
positive_outcomes, negative_outcomes = labels.value_counts()
total_samples = labels.count()

print('----------Checking for Data Imbalance------------')
print('Number of Positive Outcomes: ', positive_outcomes)
print('Percentage of Positive Outcomes: {}%'.format(round((positive_outcomes /␣
 ↪total_samples) * 100, 2)))
print('Number of Negative Outcomes : ', negative_outcomes)
print('Percentage of Negative Outcomes: {}%'.format(round((negative_outcomes /␣
 ↪total_samples) * 100, 2)))
print('\n')
```




```
----------Checking for Data Imbalance------------
Number of Positive Outcomes:  500
Percentage of Positive Outcomes: 65.1%
Number of Negative Outcomes :  268
Percentage of Negative Outcomes: 34.9%
```

2.1.5 Checking for Correlation between attributes

[9]:
```
# Creating a correlation matrix and displaying it using a heatmap
fig, axis = plt.subplots(figsize=(8, 8))
correlation_matrix = features.corr()
sns.heatmap(correlation_matrix, annot=True, linewidths=.5, fmt='.2f', ax=axis)
plt.show()
```



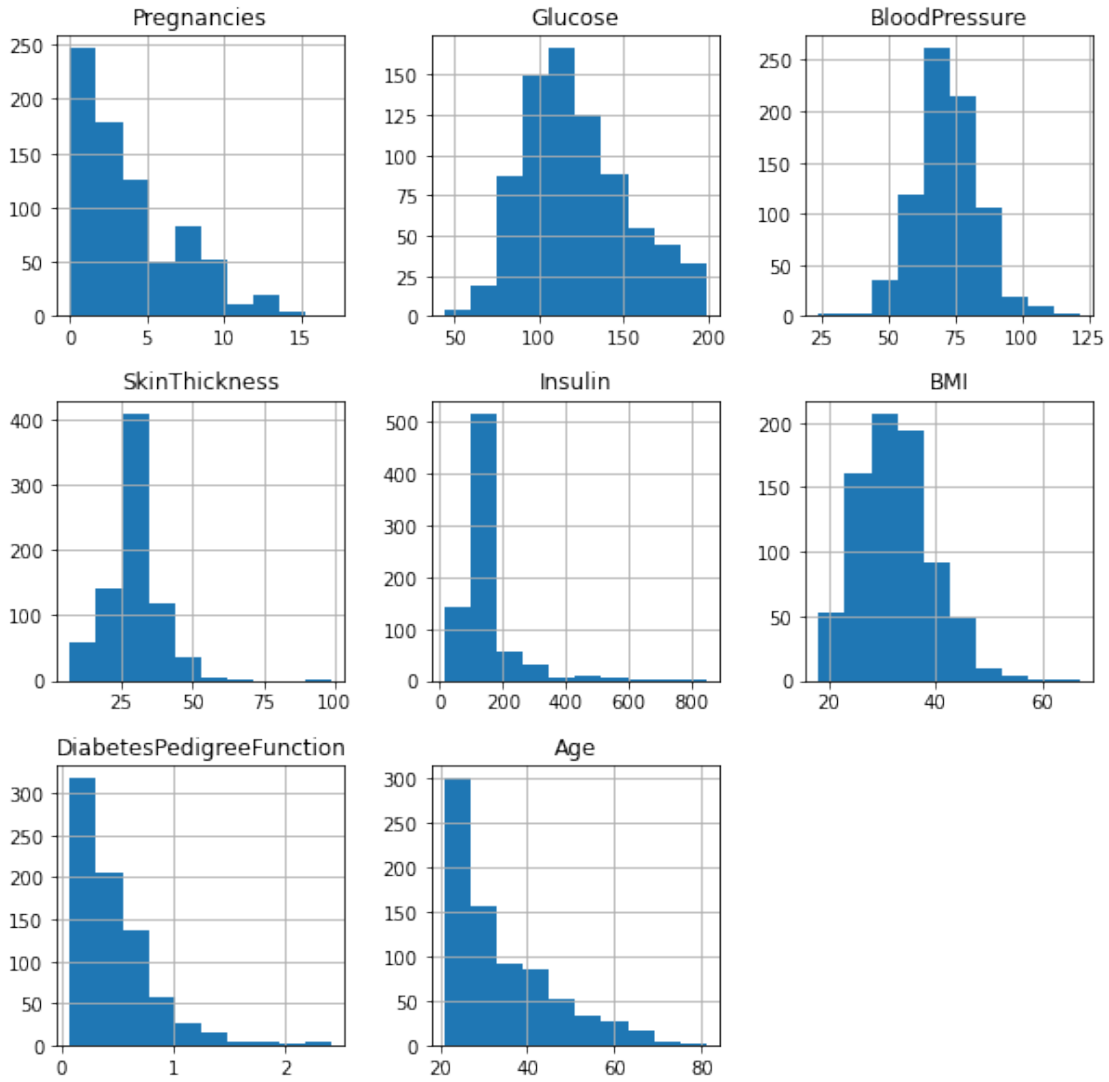
Upon examination, the most notable and heighest correlation is observed in two pairs:

```
The correlation between 'Age' and 'Pregnancies' is 0.54.

The correlation between 'BMI' and 'SkinThickness' is 0.54.
```

Notably, the attributes do not exhibit high correlation with each other. As a result, we can confidently utilize this set of attributes for our project.

2.1.6 Visualize the distribution of values for each attribute by plotting histograms.

[10]:
```
features.hist(figsize=(10, 10))
plt.show()
```



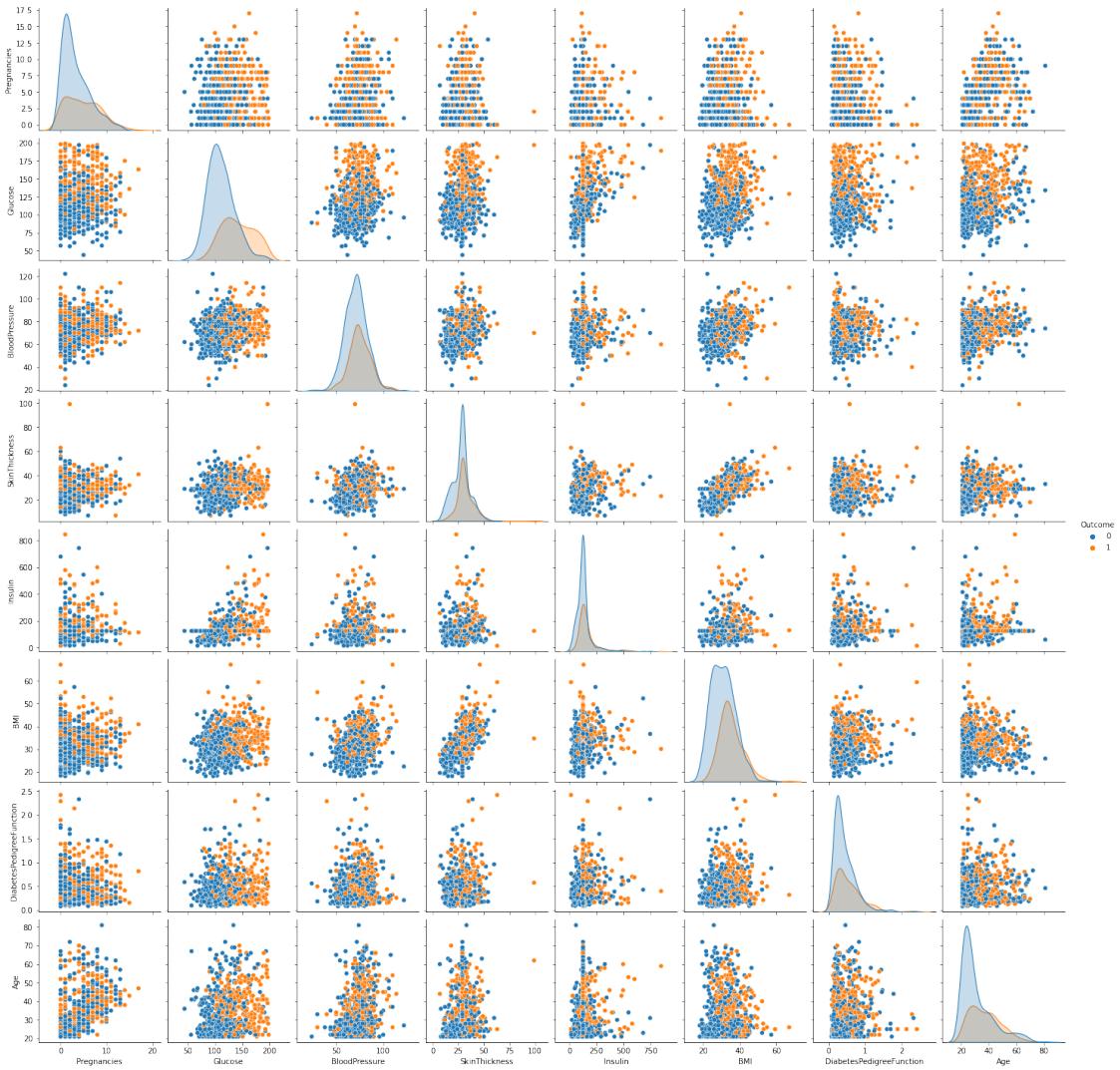
Following are the observations made from the above plot:

`The distributions of 'Glucose' and 'BloodPressure' exhibit a degree of symmetry.`

In contrast, the distributions of the remaining attributes are skewed towards one side.

2.1.7 Generate a pairplot to visualize multiple pairwise bivariate distributions within our dataset.

[11]:
```
# Creating a pair plot with a hue based on the 'Outcome' column
sns.pairplot(diab, hue='Outcome')
plt.show()
```



2.1.8 Train Test Data Split

[12]:
```
# Perform train-test split with a 10% test size and stratification
features_train_all, features_test_all, labels_train_all, labels_test_all =␣
↪train_test_split(features, labels, test_size=0.1, random_state=21,␣
↪stratify=labels)
```

```
# Reset indices for the training and testing sets
for dataset in [features_train_all, features_test_all, labels_train_all,␣
 ↪labels_test_all]:
    dataset.reset_index(drop=True, inplace=True)
```

2.1.9 Normalize the training dataset to enhance model performance.

Normalize the training dataset to enhance model performance.

The normalization process involves subtracting the mean from each value and then dividing by the standard deviation. This results in normalized attributes with a mean of 0 and a standard deviation of 1.

[13]:
```
# Standardize features for training set
features_train_all_std = (features_train_all - features_train_all.mean()) /␣
 ↪features_train_all.std()

# Standardize features for testing set
features_test_all_std = (features_test_all - features_test_all.mean()) /␣
 ↪features_test_all.std()
```

[14]:
```
features_train_all_std.describe()
```

[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
count	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02
mean	-5.141409e-18	9.897212e-17	-5.655550e-17	5.141409e-18	1.317486e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.171272e+00	-2.565904e+00	-3.968358e+00	-2.544585e+00	-1.444826e+00
25%	-8.708459e-01	-7.237508e-01	-6.337132e-01	-4.843358e-01	-2.026905e-01
50%	-2.699927e-01	-1.645256e-01	-6.438357e-02	-2.650264e-02	-2.026905e-01
75%	6.312872e-01	6.249688e-01	5.862789e-01	3.168722e-01	-1.467385e-01
max	3.034700e+00	2.467122e+00	4.002257e+00	7.985577e+00	7.865593e+00

	BMI	DiabetesPedigreeFunction	Age
count	6.910000e+02	6.910000e+02	6.910000e+02
mean	-6.169691e-17	-2.313634e-17	1.568130e-16
std	1.000000e+00	1.000000e+00	1.000000e+00
min	-2.063325e+00	-1.174590e+00	-1.058467e+00
25%	-7.104908e-01	-6.757402e-01	-8.065493e-01
50%	-3.407390e-02	-3.147111e-01	-3.866867e-01
75%	5.847756e-01	4.777553e-01	6.209834e-01
max	4.974289e+00	5.842258e+00	3.979884e+00

As observed in the table above, the mean of each attribute is approximately 0, and the standard deviation is 1.

2.1.10 Define the necessary function for model fitting and metric calculation.

[15]:
```
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
```

[16]:
```
def get_metrics(model, X_train, X_test, y_train, y_test, LSTM_flag):
    def calc_metrics(conf_matrix):
        TP, FN = conf_matrix[0][0], conf_matrix[0][1]
        FP, TN = conf_matrix[1][0], conf_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))

        return [TP, TN, FP, FN, TPR, TNR, FPR, FNR, Precision, F1_measure,␣
 ↪Accuracy, Error_rate, BACC, TSS, HSS]

    metrics = []

    if LSTM_flag == 1:
```

```
        # Convert data to numpy array
        Xtrain, Xtest, ytrain, ytest = map(np.array, [features_train,␣
↪features_test, labels_train, labels_test])

        # Reshape data
        shape = Xtrain.shape
        Xtrain_reshaped = Xtrain.reshape(len(Xtrain), shape[1], 1)
        Xtest_reshaped = Xtest.reshape(len(Xtest), shape[1], 1)

        model.fit(Xtrain_reshaped, ytrain, epochs=50,␣
↪validation_data=(Xtest_reshaped, ytest), verbose=0)
        lstm_scores = model.evaluate(Xtest_reshaped, ytest, verbose=0)
        predict_prob = model.predict(Xtest_reshaped)
        pred_labels = predict_prob > 0.5
        pred_labels_1 = pred_labels.astype(int)
        matrix = confusion_matrix(ytest, pred_labels_1, labels=[1, 0])
        lstm_brier_score = brier_score_loss(ytest, predict_prob)
        lstm_roc_auc = roc_auc_score(ytest, predict_prob)
        metrics.extend(calc_metrics(matrix))
        metrics.extend([lstm_brier_score, lstm_roc_auc, lstm_scores[1]])

    elif LSTM_flag == 0:
        model.fit(features_train, labels_train)
        predicted = model.predict(features_test)
        matrix = confusion_matrix(labels_test, predicted, labels=[1, 0])
        model_brier_score = brier_score_loss(labels_test, model.
↪predict_proba(features_test)[:, 1])
        model_roc_auc = roc_auc_score(y_test, model.
↪predict_proba(features_test)[:, 1])
        metrics.extend(calc_metrics(matrix))
        metrics.extend([model_brier_score, model_roc_auc, model.
↪score(features_test, labels_test)])

    return metrics
```

2.2 Selecting Classification Algorithms

2.2.1 I have decided to select following Classification algorithms

1.K-Nearest Neighbor

2.Random Forest

3.Support Vector Machine

2.2.2 For Deep learning algorithm, I have decided to use LSTM

Long Short-Term Memory

2.3 Parameter Tuning for KNN

[17]:
```
# Define KNN parameters for grid search
knn_parameters = {"n_neighbors": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,␣
 ↪14, 15]}

# 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(features_train_all_std, labels_train_all)

# Print the best parameters found by GridSearchCV
print("\nBest Parameters for KNN based on GridSearchCV: ", knn_cv.best_params_)
print('\n')
```

```
Best Parameters for KNN based on GridSearchCV:  {'n_neighbors': 13}
```

[18]:
```
# Extract the best value for 'n_neighbors' from the grid search results
best_n_neighbors = knn_cv.best_params_['n_neighbors']
```

2.3.1 RF Parameter Tuning

[19]:
```
# Define Random Forest parameters for grid search
param_grid_rf = {
    "n_estimators": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    "min_samples_split": [2, 4, 6, 8, 10]
}

# Create Random Forest model
rf_classifier = RandomForestClassifier()

# Perform grid search with cross-validation
grid_search_rf = GridSearchCV(estimator=rf_classifier,␣
 ↪param_grid=param_grid_rf, cv=10, n_jobs=-1)
grid_search_rf.fit(features_train_all_std, labels_train_all)

# Display the best parameters from the grid search
best_rf_params = grid_search_rf.best_params_
print("\nBest Parameters for Random Forest based on GridSearchCV: ",␣
 ↪best_rf_params)
print('\n')

# Extract the best values for 'min_samples_split' and 'n_estimators'
```

```
min_samples_split = best_rf_params['min_samples_split']
n_estimators = best_rf_params['n_estimators']
```

```
Best Parameters for Random Forest based on GridSearchCV:  {'min_samples_split':
10, 'n_estimators': 70}
```

2.3.2 SVM Parameter Tuning

Here, I am using 'linear' kernel for SVC for this project

[20]:
```
# Define Support Vector Machine parameters for grid search
param_grid_svc = {"kernel": ["linear"], "C": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}

# Create Support Vector Machine model
svc_classifier = SVC()

# Perform grid search with cross-validation
grid_search_svc = GridSearchCV(estimator=svc_classifier,␣
 ↪param_grid=param_grid_svc, cv=10, n_jobs=-1)
grid_search_svc.fit(features_train_all_std, labels_train_all)

# Display the best parameters from the grid search
best_svc_params = grid_search_svc.best_params_
print("\nBest Parameters for Support Vector Machine based on GridSearchCV: ",␣
 ↪best_svc_params)
print('\n')

# Extract the best value for 'C'
C_value = best_svc_params['C']
```

```
Best Parameters for Support Vector Machine based on GridSearchCV:  {'C': 3,
'kernel': 'linear'}
```

2.3.3 Comparing the classifiers with selected parameters by using 10-Fold Stratified Cross-Validation to calculate all metrics

Implementing 10-Fold Stratified Cross-Validation In this project, I will be using the training data set for validation as well using Startefied 10-Fold Cross Validation

[21]:
```
# Define Stratified K-Fold cross-validator
cv_stratified = StratifiedKFold(n_splits=10, shuffle=True, random_state=21)
```

[22]:
```
# Initialize metric columns
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, svm_metrics_list, lstm_metrics_list = [],␣
 ↪[], [], []

C = 1.0

# 10 Iterations of 10-fold cross-validation
for iter_num, (train_index, test_index) in enumerate(cv_stratified.
 ↪split(features_train_all_std, labels_train_all), 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)
    # SVM Classifier Model
    svm_model = SVC(C=C, kernel='linear', probability=True)
    # LSTM model
    lstm_model = Sequential()
    lstm_model.add(LSTM(64, activation='relu', batch_input_shape=(None, 8, 1),␣
 ↪return_sequences=False))
    lstm_model.add(Dense(1, activation='sigmoid'))
    # Compile model
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    # Split data into training and testing sets
    features_train, features_test = features_train_all_std.iloc[train_index, :
 ↪], features_train_all_std.iloc[test_index, :]
    labels_train, labels_test = labels_train_all[train_index],␣
 ↪labels_train_all[test_index]

    # Get metrics for each algorithm
    knn_metrics = get_metrics(knn_model, features_train, features_test,␣
 ↪labels_train, labels_test, 0)
    rf_metrics = get_metrics(rf_model, features_train, features_test,␣
 ↪labels_train, labels_test, 0)
    svm_metrics = get_metrics(svm_model, features_train, features_test,␣
 ↪labels_train, labels_test, 0)
```

```
    lstm_metrics = get_metrics(lstm_model, features_train, features_test,␣
↪labels_train, labels_test, 1)

    # Append metrics to respective lists
    knn_metrics_list.append(knn_metrics)
    rf_metrics_list.append(rf_metrics)
    svm_metrics_list.append(svm_metrics)
    lstm_metrics_list.append(lstm_metrics)

    # Create a DataFrame for all metrics
    metrics_all_df = pd.DataFrame([knn_metrics, rf_metrics, svm_metrics,␣
↪lstm_metrics],
                                  columns=metric_columns, index=['KNN', 'RF',␣
↪'SVM', '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')
```

```
Iteration 1:


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

                     KNN     RF    SVM   LSTM
TP                 14.00  15.00  13.00  13.00
TN                 39.00  37.00  37.00  36.00
FP                  6.00   8.00   8.00   9.00
FN                 11.00  10.00  12.00  12.00
TPR                 0.56   0.60   0.52   0.52
TNR                 0.87   0.82   0.82   0.80
FPR                 0.13   0.18   0.18   0.20
FNR                 0.44   0.40   0.48   0.48
Precision           0.70   0.65   0.62   0.59
F1_measure          0.62   0.62   0.57   0.55
Accuracy            0.76   0.74   0.71   0.70
Error_rate          0.24   0.26   0.29   0.30
BACC                0.71   0.71   0.67   0.66
TSS                 0.43   0.42   0.34   0.32
HSS                 0.45   0.43   0.35   0.33
Brier_score         0.17   0.18   0.17   0.19
AUC                 0.80   0.78   0.79   0.78
Acc_by_package_fn   0.76   0.74   0.71   0.70
```

Iteration 2:

----- Metrics for all Algorithms in Iteration 2 -----

	KNN	RF	SVM	LSTM
TP	11.00	10.00	14.00	12.00
TN	37.00	41.00	42.00	37.00
FP	8.00	4.00	3.00	8.00
FN	13.00	14.00	10.00	12.00
TPR	0.46	0.42	0.58	0.50
TNR	0.82	0.91	0.93	0.82
FPR	0.18	0.09	0.07	0.18
FNR	0.54	0.58	0.42	0.50
Precision	0.58	0.71	0.82	0.60
F1_measure	0.51	0.53	0.68	0.55
Accuracy	0.70	0.74	0.81	0.71
Error_rate	0.30	0.26	0.19	0.29
BACC	0.64	0.66	0.76	0.66
TSS	0.28	0.33	0.52	0.32
HSS	0.29	0.36	0.55	0.34
Brier_score	0.17	0.17	0.14	0.19
AUC	0.79	0.80	0.86	0.75
Acc_by_package_fn	0.70	0.74	0.81	0.71

Iteration 3:

----- Metrics for all Algorithms in Iteration 3 -----

	KNN	RF	SVM	LSTM
TP	11.00	11.00	10.00	11.00
TN	34.00	35.00	34.00	36.00
FP	11.00	10.00	11.00	9.00
FN	13.00	13.00	14.00	13.00
TPR	0.46	0.46	0.42	0.46
TNR	0.76	0.78	0.76	0.80
FPR	0.24	0.22	0.24	0.20
FNR	0.54	0.54	0.58	0.54
Precision	0.50	0.52	0.48	0.55
F1_measure	0.48	0.49	0.44	0.50
Accuracy	0.65	0.67	0.64	0.68
Error_rate	0.35	0.33	0.36	0.32

BACC	0.61	0.62	0.59	0.63
TSS	0.21	0.24	0.17	0.26
HSS	0.22	0.24	0.18	0.27
Brier_score	0.21	0.23	0.24	0.22
AUC	0.72	0.68	0.68	0.73
Acc_by_package_fn	0.65	0.67	0.64	0.68

Iteration 4:

----- Metrics for all Algorithms in Iteration 4 -----

	KNN	RF	SVM	LSTM
TP	13.00	15.00	13.00	10.00
TN	37.00	42.00	41.00	42.00
FP	8.00	3.00	4.00	3.00
FN	11.00	9.00	11.00	14.00
TPR	0.54	0.62	0.54	0.42
TNR	0.82	0.93	0.91	0.93
FPR	0.18	0.07	0.09	0.07
FNR	0.46	0.38	0.46	0.58
Precision	0.62	0.83	0.76	0.77
F1_measure	0.58	0.71	0.63	0.54
Accuracy	0.72	0.83	0.78	0.75
Error_rate	0.28	0.17	0.22	0.25
BACC	0.68	0.78	0.73	0.68
TSS	0.36	0.56	0.45	0.35
HSS	0.37	0.59	0.49	0.39
Brier_score	0.16	0.15	0.14	0.19
AUC	0.81	0.84	0.87	0.79
Acc_by_package_fn	0.72	0.83	0.78	0.75

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000002B3194BF700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and https://www.tensorflow.org/api_docs/python/tf/function for more details.

Iteration 5:

----- Metrics for all Algorithms in Iteration 5 -----

	KNN	RF	SVM	LSTM
TP	10.00	8.00	6.00	13.00
TN	41.00	41.00	43.00	39.00
FP	4.00	4.00	2.00	6.00
FN	14.00	16.00	18.00	11.00
TPR	0.42	0.33	0.25	0.54
TNR	0.91	0.91	0.96	0.87
FPR	0.09	0.09	0.04	0.13
FNR	0.58	0.67	0.75	0.46
Precision	0.71	0.67	0.75	0.68
F1_measure	0.53	0.44	0.38	0.60
Accuracy	0.74	0.71	0.71	0.75
Error_rate	0.26	0.29	0.29	0.25
BACC	0.66	0.62	0.60	0.70
TSS	0.33	0.24	0.21	0.41
HSS	0.36	0.28	0.24	0.43
Brier_score	0.18	0.18	0.18	0.18
AUC	0.77	0.76	0.81	0.77
Acc_by_package_fn	0.74	0.71	0.71	0.75

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000002B318D1B8B0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and https://www.tensorflow.org/api_docs/python/tf/function for more details.

Iteration 6:

----- Metrics for all Algorithms in Iteration 6 -----

	KNN	RF	SVM	LSTM
TP	17.00	19.00	18.00	16.00
TN	41.00	40.00	40.00	39.00
FP	4.00	5.00	5.00	6.00
FN	7.00	5.00	6.00	8.00
TPR	0.71	0.79	0.75	0.67
TNR	0.91	0.89	0.89	0.87
FPR	0.09	0.11	0.11	0.13

FNR	0.29	0.21	0.25	0.33
Precision	0.81	0.79	0.78	0.73
F1_measure	0.76	0.79	0.77	0.70
Accuracy	0.84	0.86	0.84	0.80
Error_rate	0.16	0.14	0.16	0.20
BACC	0.81	0.84	0.82	0.77
TSS	0.62	0.68	0.64	0.53
HSS	0.64	0.68	0.65	0.54
Brier_score	0.13	0.12	0.12	0.15
AUC	0.91	0.91	0.94	0.86
Acc_by_package_fn	0.84	0.86	0.84	0.80

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000002B3165211F0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and https://www.tensorflow.org/api_docs/python/tf/function for more details.

Iteration 7:

----- Metrics for all Algorithms in Iteration 7 -----

	KNN	RF	SVM	LSTM
TP	16.00	16.00	13.00	15.00
TN	39.00	40.00	37.00	35.00
FP	6.00	5.00	8.00	10.00
FN	8.00	8.00	11.00	9.00
TPR	0.67	0.67	0.54	0.62
TNR	0.87	0.89	0.82	0.78
FPR	0.13	0.11	0.18	0.22
FNR	0.33	0.33	0.46	0.38
Precision	0.73	0.76	0.62	0.60
F1_measure	0.70	0.71	0.58	0.61
Accuracy	0.80	0.81	0.72	0.72
Error_rate	0.20	0.19	0.28	0.28
BACC	0.77	0.78	0.68	0.70
TSS	0.53	0.56	0.36	0.40
HSS	0.54	0.57	0.37	0.40
Brier_score	0.15	0.15	0.15	0.17
AUC	0.86	0.85	0.86	0.81
Acc_by_package_fn	0.80	0.81	0.72	0.72

```
WARNING:tensorflow:5 out of the last 13 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000002B315F7F550>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Iteration 8:


----- Metrics for all Algorithms in Iteration 8 -----

                     KNN     RF    SVM   LSTM
TP                 13.00  14.00  10.00  12.00
TN                 41.00  40.00  42.00  38.00
FP                  4.00   5.00   3.00   7.00
FN                 11.00  10.00  14.00  12.00
TPR                 0.54   0.58   0.42   0.50
TNR                 0.91   0.89   0.93   0.84
FPR                 0.09   0.11   0.07   0.16
FNR                 0.46   0.42   0.58   0.50
Precision           0.76   0.74   0.77   0.63
F1_measure          0.63   0.65   0.54   0.56
Accuracy            0.78   0.78   0.75   0.72
Error_rate          0.22   0.22   0.25   0.28
BACC                0.73   0.74   0.68   0.67
TSS                 0.45   0.47   0.35   0.34
HSS                 0.49   0.50   0.39   0.36
Brier_score         0.16   0.17   0.17   0.17
AUC                 0.83   0.81   0.82   0.79
Acc_by_package_fn   0.78   0.78   0.75   0.72


WARNING:tensorflow:5 out of the last 13 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000002B31DAB9310>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
```

https://www.tensorflow.org/api_docs/python/tf/function for more details.

Iteration 9:

----- Metrics for all Algorithms in Iteration 9 -----

	KNN	RF	SVM	LSTM
TP	12.00	17.00	16.00	14.00
TN	38.00	36.00	38.00	39.00
FP	7.00	9.00	7.00	6.00
FN	12.00	7.00	8.00	10.00
TPR	0.50	0.71	0.67	0.58
TNR	0.84	0.80	0.84	0.87
FPR	0.16	0.20	0.16	0.13
FNR	0.50	0.29	0.33	0.42
Precision	0.63	0.65	0.70	0.70
F1_measure	0.56	0.68	0.68	0.64
Accuracy	0.72	0.77	0.78	0.77
Error_rate	0.28	0.23	0.22	0.23
BACC	0.67	0.75	0.76	0.73
TSS	0.34	0.51	0.51	0.45
HSS	0.36	0.50	0.52	0.47
Brier_score	0.17	0.16	0.15	0.16
AUC	0.80	0.82	0.85	0.84
Acc_by_package_fn	0.72	0.77	0.78	0.77

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000002B31B3B2AF0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and https://www.tensorflow.org/api_docs/python/tf/function for more details.

Iteration 10:

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

	KNN	RF	SVM	LSTM
TP	16.00	17.00	17.00	17.00
TN	38.00	39.00	39.00	38.00
FP	7.00	6.00	6.00	7.00

```
FN                  8.00   7.00   7.00   7.00
TPR                 0.67   0.71   0.71   0.71
TNR                 0.84   0.87   0.87   0.84
FPR                 0.16   0.13   0.13   0.16
FNR                 0.33   0.29   0.29   0.29
Precision           0.70   0.74   0.74   0.71
F1_measure          0.68   0.72   0.72   0.71
Accuracy            0.78   0.81   0.81   0.80
Error_rate          0.22   0.19   0.19   0.20
BACC                0.76   0.79   0.79   0.78
TSS                 0.51   0.58   0.58   0.55
HSS                 0.52   0.58   0.58   0.55
Brier_score         0.14   0.11   0.13   0.12
AUC                 0.87   0.93   0.90   0.92
Acc_by_package_fn   0.78   0.81   0.81   0.80
```

[23]:
```
# 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)
svm_metrics_df = pd.DataFrame(svm_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
for i, metrics_df in enumerate([knn_metrics_df, rf_metrics_df, svm_metrics_df,␣
 ↪lstm_metrics_df], start=1):
    print('\nMetrics for Algorithm {}:\n'.format(i))
    print(metrics_df.round(decimals=2).T)
    print('\n')
```

```
Metrics for Algorithm 1:

      iter1  iter2  iter3  iter4  iter5  iter6  iter7  iter8  \
TP    14.00  11.00  11.00  13.00  10.00  17.00  16.00  13.00
TN    39.00  37.00  34.00  37.00  41.00  41.00  39.00  41.00
FP     6.00   8.00  11.00   8.00   4.00   4.00   6.00   4.00
FN    11.00  13.00  13.00  11.00  14.00   7.00   8.00  11.00
TPR    0.56   0.46   0.46   0.54   0.42   0.71   0.67   0.54
```

TNR	0.87	0.82	0.76	0.82	0.91	0.91	0.87	0.91
FPR	0.13	0.18	0.24	0.18	0.09	0.09	0.13	0.09
FNR	0.44	0.54	0.54	0.46	0.58	0.29	0.33	0.46
Precision	0.70	0.58	0.50	0.62	0.71	0.81	0.73	0.76
F1_measure	0.62	0.51	0.48	0.58	0.53	0.76	0.70	0.63
Accuracy	0.76	0.70	0.65	0.72	0.74	0.84	0.80	0.78
Error_rate	0.24	0.30	0.35	0.28	0.26	0.16	0.20	0.22
BACC	0.71	0.64	0.61	0.68	0.66	0.81	0.77	0.73
TSS	0.43	0.28	0.21	0.36	0.33	0.62	0.53	0.45
HSS	0.45	0.29	0.22	0.37	0.36	0.64	0.54	0.49
Brier_score	0.17	0.17	0.21	0.16	0.18	0.13	0.15	0.16
AUC	0.80	0.79	0.72	0.81	0.77	0.91	0.86	0.83
Acc_by_package_fn	0.76	0.70	0.65	0.72	0.74	0.84	0.80	0.78

	iter9	iter10
TP	12.00	16.00
TN	38.00	38.00
FP	7.00	7.00
FN	12.00	8.00
TPR	0.50	0.67
TNR	0.84	0.84
FPR	0.16	0.16
FNR	0.50	0.33
Precision	0.63	0.70
F1_measure	0.56	0.68
Accuracy	0.72	0.78
Error_rate	0.28	0.22
BACC	0.67	0.76
TSS	0.34	0.51
HSS	0.36	0.52
Brier_score	0.17	0.14
AUC	0.80	0.87
Acc_by_package_fn	0.72	0.78

Metrics for Algorithm 2:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	\
TP	15.00	10.00	11.00	15.00	8.00	19.00	16.00	14.00	
TN	37.00	41.00	35.00	42.00	41.00	40.00	40.00	40.00	
FP	8.00	4.00	10.00	3.00	4.00	5.00	5.00	5.00	
FN	10.00	14.00	13.00	9.00	16.00	5.00	8.00	10.00	
TPR	0.60	0.42	0.46	0.62	0.33	0.79	0.67	0.58	
TNR	0.82	0.91	0.78	0.93	0.91	0.89	0.89	0.89	
FPR	0.18	0.09	0.22	0.07	0.09	0.11	0.11	0.11	
FNR	0.40	0.58	0.54	0.38	0.67	0.21	0.33	0.42	
Precision	0.65	0.71	0.52	0.83	0.67	0.79	0.76	0.74	

F1_measure	0.62	0.53	0.49	0.71	0.44	0.79	0.71	0.65
Accuracy	0.74	0.74	0.67	0.83	0.71	0.86	0.81	0.78
Error_rate	0.26	0.26	0.33	0.17	0.29	0.14	0.19	0.22
BACC	0.71	0.66	0.62	0.78	0.62	0.84	0.78	0.74
TSS	0.42	0.33	0.24	0.56	0.24	0.68	0.56	0.47
HSS	0.43	0.36	0.24	0.59	0.28	0.68	0.57	0.50
Brier_score	0.18	0.17	0.23	0.15	0.18	0.12	0.15	0.17
AUC	0.78	0.80	0.68	0.84	0.76	0.91	0.85	0.81
Acc_by_package_fn	0.74	0.74	0.67	0.83	0.71	0.86	0.81	0.78

	iter9	iter10
TP	17.00	17.00
TN	36.00	39.00
FP	9.00	6.00
FN	7.00	7.00
TPR	0.71	0.71
TNR	0.80	0.87
FPR	0.20	0.13
FNR	0.29	0.29
Precision	0.65	0.74
F1_measure	0.68	0.72
Accuracy	0.77	0.81
Error_rate	0.23	0.19
BACC	0.75	0.79
TSS	0.51	0.58
HSS	0.50	0.58
Brier_score	0.16	0.11
AUC	0.82	0.93
Acc_by_package_fn	0.77	0.81

Metrics for Algorithm 3:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	\
TP	13.00	14.00	10.00	13.00	6.00	18.00	13.00	10.00	
TN	37.00	42.00	34.00	41.00	43.00	40.00	37.00	42.00	
FP	8.00	3.00	11.00	4.00	2.00	5.00	8.00	3.00	
FN	12.00	10.00	14.00	11.00	18.00	6.00	11.00	14.00	
TPR	0.52	0.58	0.42	0.54	0.25	0.75	0.54	0.42	
TNR	0.82	0.93	0.76	0.91	0.96	0.89	0.82	0.93	
FPR	0.18	0.07	0.24	0.09	0.04	0.11	0.18	0.07	
FNR	0.48	0.42	0.58	0.46	0.75	0.25	0.46	0.58	
Precision	0.62	0.82	0.48	0.76	0.75	0.78	0.62	0.77	
F1_measure	0.57	0.68	0.44	0.63	0.38	0.77	0.58	0.54	
Accuracy	0.71	0.81	0.64	0.78	0.71	0.84	0.72	0.75	
Error_rate	0.29	0.19	0.36	0.22	0.29	0.16	0.28	0.25	
BACC	0.67	0.76	0.59	0.73	0.60	0.82	0.68	0.68	

TSS	0.34	0.52	0.17	0.45	0.21	0.64	0.36	0.35
HSS	0.35	0.55	0.18	0.49	0.24	0.65	0.37	0.39
Brier_score	0.17	0.14	0.24	0.14	0.18	0.12	0.15	0.17
AUC	0.79	0.86	0.68	0.87	0.81	0.94	0.86	0.82
Acc_by_package_fn	0.71	0.81	0.64	0.78	0.71	0.84	0.72	0.75

	iter9	iter10
TP	16.00	17.00
TN	38.00	39.00
FP	7.00	6.00
FN	8.00	7.00
TPR	0.67	0.71
TNR	0.84	0.87
FPR	0.16	0.13
FNR	0.33	0.29
Precision	0.70	0.74
F1_measure	0.68	0.72
Accuracy	0.78	0.81
Error_rate	0.22	0.19
BACC	0.76	0.79
TSS	0.51	0.58
HSS	0.52	0.58
Brier_score	0.15	0.13
AUC	0.85	0.90
Acc_by_package_fn	0.78	0.81

Metrics for Algorithm 4:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	\
TP	13.00	12.00	11.00	10.00	13.00	16.00	15.00	12.00	
TN	36.00	37.00	36.00	42.00	39.00	39.00	35.00	38.00	
FP	9.00	8.00	9.00	3.00	6.00	6.00	10.00	7.00	
FN	12.00	12.00	13.00	14.00	11.00	8.00	9.00	12.00	
TPR	0.52	0.50	0.46	0.42	0.54	0.67	0.62	0.50	
TNR	0.80	0.82	0.80	0.93	0.87	0.87	0.78	0.84	
FPR	0.20	0.18	0.20	0.07	0.13	0.13	0.22	0.16	
FNR	0.48	0.50	0.54	0.58	0.46	0.33	0.38	0.50	
Precision	0.59	0.60	0.55	0.77	0.68	0.73	0.60	0.63	
F1_measure	0.55	0.55	0.50	0.54	0.60	0.70	0.61	0.56	
Accuracy	0.70	0.71	0.68	0.75	0.75	0.80	0.72	0.72	
Error_rate	0.30	0.29	0.32	0.25	0.25	0.20	0.28	0.28	
BACC	0.66	0.66	0.63	0.68	0.70	0.77	0.70	0.67	
TSS	0.32	0.32	0.26	0.35	0.41	0.53	0.40	0.34	
HSS	0.33	0.34	0.27	0.39	0.43	0.54	0.40	0.36	
Brier_score	0.19	0.19	0.22	0.19	0.18	0.15	0.17	0.17	
AUC	0.78	0.75	0.73	0.79	0.77	0.86	0.81	0.79	

```
Acc_by_package_fn   0.70   0.71   0.68   0.75   0.75   0.80   0.72   0.72
```

```
                   iter9  iter10
TP                 14.00   17.00
TN                 39.00   38.00
FP                  6.00    7.00
FN                 10.00    7.00
TPR                 0.58    0.71
TNR                 0.87    0.84
FPR                 0.13    0.16
FNR                 0.42    0.29
Precision           0.70    0.71
F1_measure          0.64    0.71
Accuracy            0.77    0.80
Error_rate          0.23    0.20
BACC                0.73    0.78
TSS                 0.45    0.55
HSS                 0.47    0.55
Brier_score         0.16    0.12
AUC                 0.84    0.92
Acc_by_package_fn   0.77    0.80
```

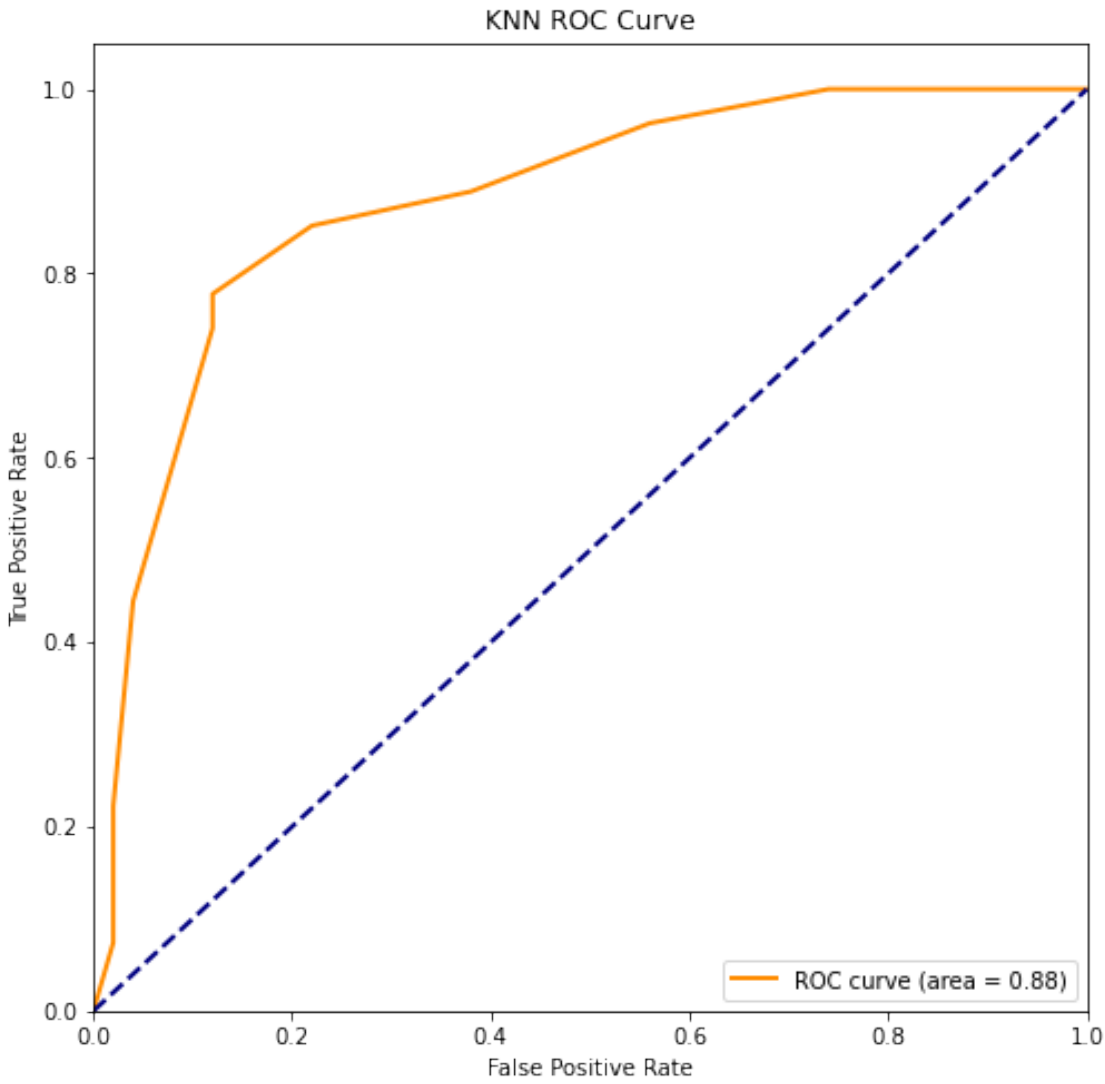
[24]:
```
# Calculate the average metrics for each algorithm
knn_avg_df = knn_metrics_df.mean()
rf_avg_df = rf_metrics_df.mean()
svm_avg_df = svm_metrics_df.mean()
lstm_avg_df = lstm_metrics_df.mean()

# Create a DataFrame with the average performance for each algorithm
avg_performance_df = pd.DataFrame({'KNN': knn_avg_df, 'RF': rf_avg_df, 'SVM':␣
 ↪svm_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    SVM   LSTM
TP                13.30  14.20  13.00  13.30
TN                38.50  39.10  39.30  37.90
FP                 6.50   5.90   5.70   7.10
FN                10.80   9.90  11.10  10.80
TPR                0.55   0.59   0.54   0.55
TNR                0.86   0.87   0.87   0.84
FPR                0.14   0.13   0.13   0.16
FNR                0.45   0.41   0.46   0.45
Precision          0.67   0.71   0.70   0.66
```

```
F1_measure          0.60   0.64   0.60   0.60
Accuracy            0.75   0.77   0.76   0.74
Error_rate          0.25   0.23   0.24   0.26
BACC                0.70   0.73   0.71   0.70
TSS                 0.41   0.46   0.41   0.39
HSS                 0.42   0.47   0.43   0.41
Brier_score         0.16   0.16   0.16   0.17
AUC                 0.82   0.82   0.84   0.80
Acc_by_package_fn   0.75   0.77   0.76   0.74
```

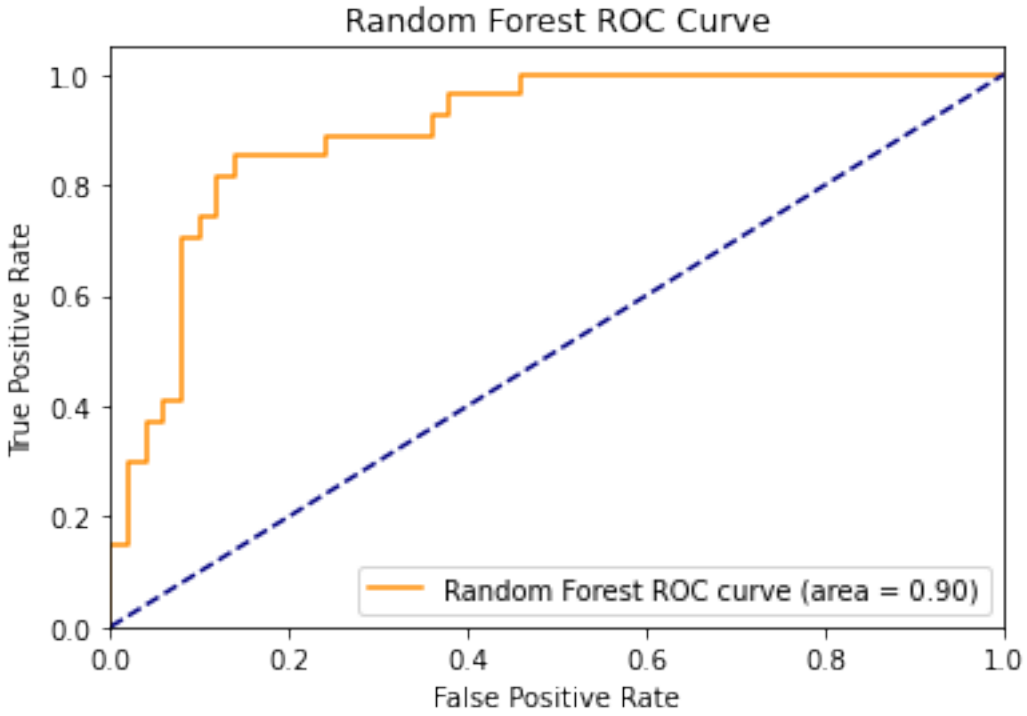
2.3.4 Evaluating the performance of various algorithms by comparing their ROC curves and AUC scores on the test dataset.

[25]:
```
# KNN Model
knn_model = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn_model.fit(features_train_all_std, labels_train_all)

# Obtain predicted probabilities
y_score = knn_model.predict_proba(features_test_all_std)[:, 1]

# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(labels_test_all, y_score)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.
  ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('KNN ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



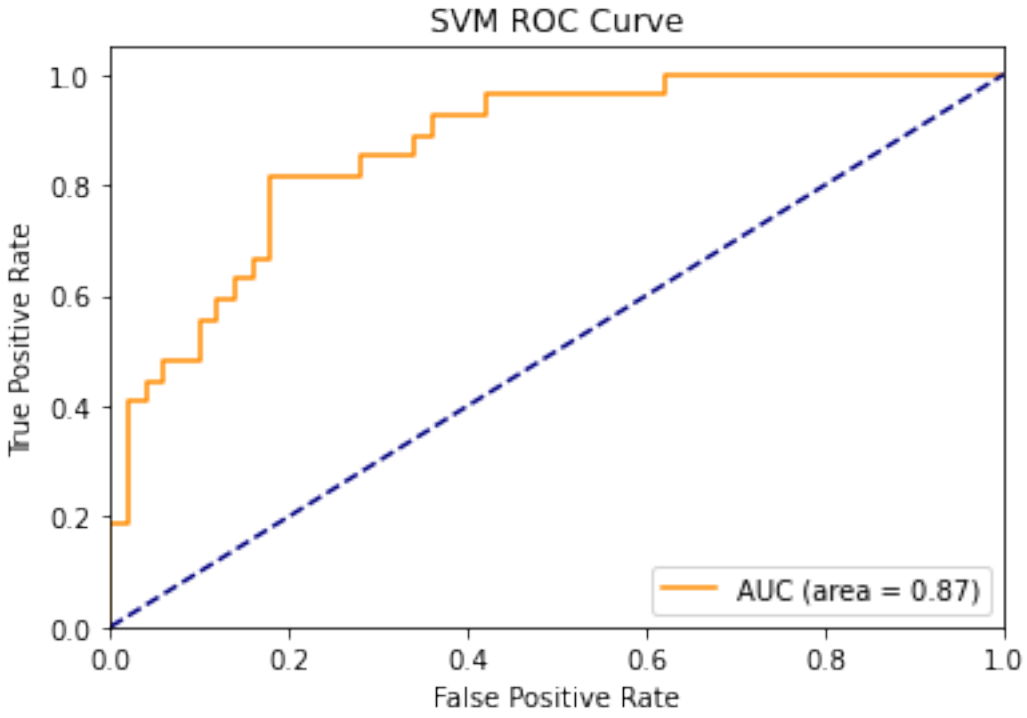
[26]:
```
# Random Forest Model
rf_model = RandomForestClassifier(min_samples_split=min_samples_split,␣
 ↪n_estimators=n_estimators)
rf_model.fit(features_train_all_std, labels_train_all)

# Obtain predicted probabilities
y_score_rf = rf_model.predict_proba(features_test_all_std)[:, 1]

# Compute ROC curve and ROC area
fpr_rf, tpr_rf, _ = roc_curve(labels_test_all, y_score_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot Random Forest ROC curve
```

```
plt.figure()
plt.plot(fpr_rf, tpr_rf, color="darkorange", label="Random Forest ROC curve␣
 ↪(area = {:.2f})".format(roc_auc_rf))
plt.plot([0, 1], [0, 1], color="navy", 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("Random Forest ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



[27]:
```
# SVM Classifier Model
svm_model = SVC(C=C, kernel='linear', probability=True)
svm_model.fit(features_train_all_std, labels_train_all)

# Obtain predicted probabilities
y_score_svm = svm_model.predict_proba(features_test_all_std)[:, 1]

# Compute ROC curve and ROC area
fpr_svm, tpr_svm, _ = roc_curve(labels_test_all, y_score_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)
```

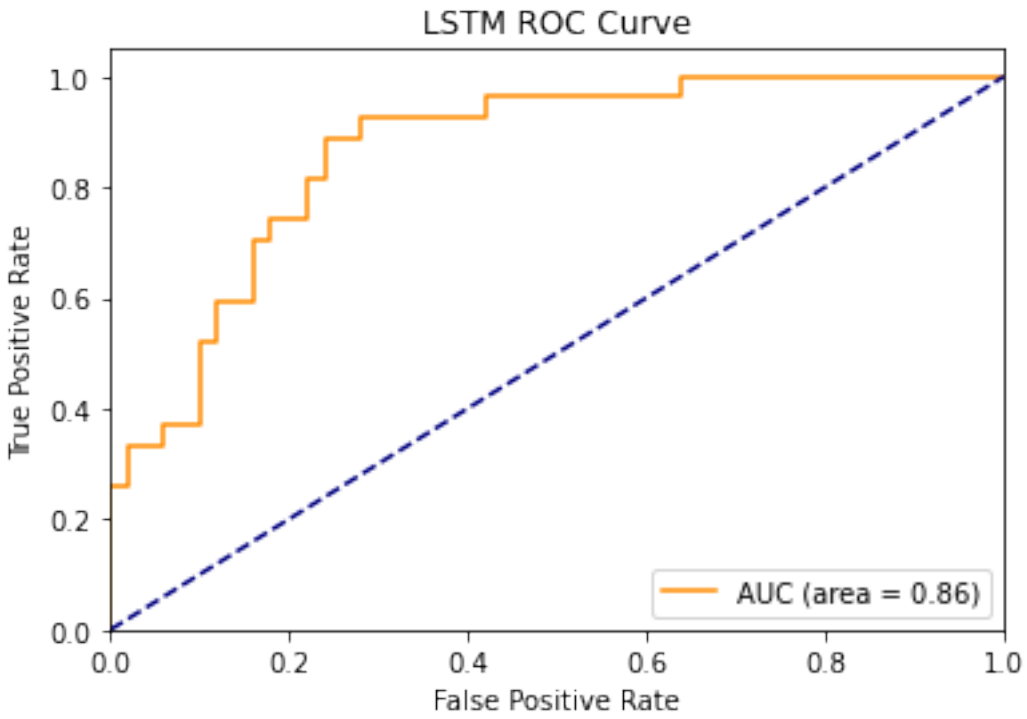
```
# Plot SVM ROC curve
plt.figure()
plt.plot(fpr_svm, tpr_svm, color="darkorange", label="AUC (area = {:.2f})".
  ↪format(roc_auc_svm))
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("SVM ROC Curve")
plt.legend(loc="lower right")
plt.show()
```



[28]:
```
# LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(64, activation='relu', batch_input_shape=(None, 8, 1),␣
  ↪return_sequences=False))
lstm_model.add(Dense(1, activation='sigmoid'))
# Compile model
lstm_model.compile(loss='binary_crossentropy', optimizer='adam',␣
  ↪metrics=['accuracy'])

# Convert data to numpy array
```

```
X_train_array = features_train_all_std.to_numpy()
X_test_array = features_test_all_std.to_numpy()
y_train_array = labels_train_all.to_numpy()
y_test_array = labels_test_all.to_numpy()

# Reshape data
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
lstm_model.fit(input_train, output_train, epochs=50,␣
 ↪validation_data=(input_test, output_test), verbose=0)

# Predict probabilities for the test set
predict_lstm = lstm_model.predict(input_test)

# Compute ROC curve and ROC area
fpr_lstm, tpr_lstm, _ = roc_curve(labels_test_all, predict_lstm)
roc_auc_lstm = auc(fpr_lstm, tpr_lstm)

# Plot LSTM ROC curve
plt.figure()
plt.plot(fpr_lstm, tpr_lstm, color="darkorange", label="AUC (area = {:.2f})".
 ↪format(roc_auc_lstm))
plt.plot([0, 1], [0, 1], color="navy", 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 Curve")
plt.legend(loc="lower right")
plt.show()
```

```
WARNING:tensorflow:5 out of the last 13 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000002B3195023A0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



2.3.5 Comparing All Models

[29]:
```
print(avg_performance_df.round(decimals=2))
print('\n')
```

```
                    KNN     RF    SVM   LSTM
TP                13.30  14.20  13.00  13.30
TN                38.50  39.10  39.30  37.90
FP                 6.50   5.90   5.70   7.10
FN                10.80   9.90  11.10  10.80
TPR                0.55   0.59   0.54   0.55
TNR                0.86   0.87   0.87   0.84
FPR                0.14   0.13   0.13   0.16
FNR                0.45   0.41   0.46   0.45
Precision          0.67   0.71   0.70   0.66
F1_measure         0.60   0.64   0.60   0.60
Accuracy           0.75   0.77   0.76   0.74
Error_rate         0.25   0.23   0.24   0.26
BACC               0.70   0.73   0.71   0.70
TSS                0.41   0.46   0.41   0.39
HSS                0.42   0.47   0.43   0.41
Brier_score        0.16   0.16   0.16   0.17
AUC                0.82   0.82   0.84   0.80
Acc_by_package_fn  0.75   0.77   0.76   0.74
```