Name – Mazeen Hussain Syed

NJIT UCID – Ms3543

Email address – ms3543@njit.edu

Date – 24th November, 2024

Professor: Yasser Abduallah

CS 634 101 Data Mining

Final-term Project Report

(Implementation and Code Usage)

Data mining and machine learning Techniques for classification of Airline Passenger Satisfaction.

Abstract

This report, therefore, classifies the airline passenger satisfaction based on a number of machine learning algorithms using a Data Mining approach. The dataset, obtained from Kaggle, includes a variety of features about flying: passenger demographic and flight details, level of satisfaction. We explore four classification algorithms in this paper: K-Nearest Neighbors (KNN), Random Forest (RF), Support Vector Machine (SVM), and Long Short-Term Memory (LSTM). Some of the metrics used to gauge performance in each algorithm include Accuracy, Precision, Recall, F1-Score, and AUC-ROC. The aim of this analysis is to determine the best model that analyses passenger's satisfaction and also to explore some of the concepts in Data Mining regarding feature selection, model evaluation, and performance metrics.

Introduction

Background

Data Mining has now become imperative with the era of big data. It plays a very important role in extracting useful insights from large volumes of data. This project deals with the application of machine learning algorithms in predicting the satisfaction of airline passengers. Identification of influential factors promoting positive or negative experiences may allow airlines to promptly optimize their services for improved customer satisfaction. The study trains various classification models and evaluates their performances to determine the best algorithm for passenger satisfaction prediction.

Objective

The main goal of the project is to apply the Data Mining techniques of classification, labeling airline passengers as satisfied or not based on several features. The Kaggle dataset will be used as input for the training and evaluation of the models. These models will be evaluated with the following criteria:

- Accuracy: It refers to the proportion of correct predictions.
- Precision, Recall, and F1-Score: Performance metrics for imbalanced classification tasks.
- AUC-ROC: Assessment of model performance to discriminate between classes.

Dataset Overview

Dataset Description

The dataset is sourced from Kaggle and contains information about passengers' demographic details, flight information, and their satisfaction with the flight experience. The dataset includes both categorical and numerical features. The target variable is Satisfaction, where passengers can either be satisfied or dissatisfied.

Link to dataset: Kaggle - Airline Passenger Satisfaction Dataset

The dataset includes the following features:

- Age: Age of the passenger.
- Gender: Gender of the passenger.
- Flight Distance: Distance traveled on the flight.
- Seat Comfort, Food and Drink: Ratings given by the passenger on specific aspects of the flight experience.
- Departure/Arrival Time Convenient: Whether the timing of the flight was convenient for the passenger.
- Class: Class of service (e.g., Economy, Business, First Class).
- Satisfaction: The target variable indicating whether the passenger was satisfied with the flight experience.

Core Concepts and Principles

Data Mining

Data Mining refers to the process of extracting meaningful patterns and knowledge from large volumes of data. In this project, classification techniques are used to categorize the passengers as satisfied or dissatisfied based on their flight experience.

Classification Algorithms

- K-Nearest Neighbors (KNN): A simple instance-based algorithm that classifies new data points based on the majority class of their nearest neighbors.
- Random Forest (RF): An ensemble method that creates a collection of decision trees and combines their predictions for improved accuracy and robustness.
- Support Vector Machine (SVM): A supervised learning algorithm that finds the hyperplane that best separates data points of different classes in a high-dimensional space.
- Long Short-Term Memory (LSTM): A deep learning algorithm that is especially effective for sequential data, capturing long-term dependencies.

Performance Evaluation Metrics

- Accuracy: The percentage of correctly classified instances.
- Precision: The proportion of true positives among all predicted positives.
- Recall: The proportion of true positives among all actual positives.
- F1-Score: The harmonic mean of precision and recall, used when the class distribution is imbalanced.
- AUC-ROC: The Area Under the Curve of the Receiver Operating Characteristic (ROC) curve, which shows the model's ability to discriminate between classes.

Implementation Overview

Data Preprocessing

- Handling Missing Values: The dataset was checked for any missing values. Missing values were handled using either imputation or removal techniques to ensure that the dataset was complete.
- Feature Encoding: Categorical variables were encoded using Label Encoding or One-Hot Encoding to convert non-numerical features into numerical ones.
- Feature Scaling: Features were standardized to ensure uniformity in scale, particularly for distance-based algorithms like KNN and SVM.
- Train-Test Split: The dataset was split into training and testing sets using Stratified K-Fold Cross-Validation to ensure each fold contained a balanced class distribution.

Model Training and Evaluation

Each of the four selected algorithms was trained on the training data and evaluated using the testing data:

- KNN: The optimal number of neighbors (k) was selected using a grid search to maximize performance.
- Random Forest: The number of trees and other hyperparameters were tuned to optimize model performance.
- SVM: The regularization parameter (C) and kernel type were adjusted for the best performance.
- LSTM: The LSTM model was built with appropriate layers and trained on the reshaped data to capture sequential patterns.

Evaluation of Performance

The performance of each of these models was evaluated based on Accuracy, Precision, Recall, F1-Score, and AUC-ROC scores. These metrics gave a comprehensive overview of the performance of each model in predicting passenger satisfaction.

Workflow Steps

Data Loading and Preprocessing

- Load the dataset from the source.
- Clean the data by handling missing values and encoding categorical variables.
- Split the dataset into training and testing sets.

Model Selection and Training

- Select the classification algorithms (KNN, Random Forest, SVM, LSTM).
- Train each model using the training dataset.

Evaluation

- Use the testing set to evaluate each model's performance.
- Compute the performance metrics for comparison.

ROC Curve and AUC Evaluation

- Plot the ROC curves for each model to visually compare their performance.
- Calculate and compare the AUC scores to determine the best model.

Results

Performance Metrics for Each Algorithm

Model	Accuracy	Precision	Recall	F1-Score	AUC
KNN	0.90	0.95	0.86	0.87	0.96
Random Forest	0.95	0.96	0.90	0.94	0.99
SVM	0.87	0.89	0.84	0.85	0.93
LSTM	0.92	0.92	0.82	0.91	0.94

ROC Curve Comparison

The ROC curves for the four models (KNN, Random Forest, SVM, and LSTM) were plotted to compare their performance. The Random Forest model had the highest AUC value, indicating the best overall performance in distinguishing between satisfied and dissatisfied passengers.

Discussion and Conclusion

Discussion

- The Random Forest model demonstrated the best performance with the highest accuracy, precision, recall, and AUC score. This suggests that Random Forest is well-suited for this classification task due to its ability to handle complex relationships between features.
- KNN also performed well, but its performance was slightly lower than that of Random Forest.
- SVM and LSTM were not as effective as other two, particularly for this dataset, which might be due to the inherent simplicity of the dataset that does not require deep learning models for accurate predictions.

Why Random Forest is the Best:

- It handles both numerical and categorical data effectively.
- It avoids overfitting by averaging the results of multiple decision trees.
- It's highly accurate, especially in classification tasks like this one, where the relationship between features and the target variable is complex.

Conclusion

Random Forest, according to these results, is the best model when it comes to the prediction of passenger satisfaction from this dataset. It generally gives robust performance with high complexity feature interaction. Fine-tuning more on the hyperparameters and exploring other algorithms such as XGBoost or AdaBoost may further improve these results.

Overall, Random Forest provides a balanced and accurate model for this task, and it would be the recommended algorithm for airline passenger satisfaction prediction.

(Screenshots and explanation parts of Function code)

Here, I am taking the train.csv file details

Figure -1 represents the data of csv file.

jupyter test.csv Last Checkpoint: yesterday

File Edit View Settings Help

Delimiter: ,

		id	Gender	Customer Type	Age	Type of Travel	Class	Flight Dist
1	0	19556	Female	Loyal Customer	52	Business travel	Eco	
2	1	90035	Female	Loyal Customer	36	Business travel	Business	
3	2	12360	Male	disloyal Customer	20	Business travel	Eco	
4	3	77959	Male	Loyal Customer	44	Business travel	Business	
5	4	36875	Female	Loyal Customer	49	Business travel	Eco	
6	5	39177	Male	Loyal Customer	16	Business travel	Eco	
7	6	79433	Female	Loyal Customer	77	Business travel	Business	
8	7	97286	Female	Loyal Customer	43	Business travel	Business	
9	8	27508	Male	Loyal Customer	47	Business travel	Eco	
10	9	62482	Female	Loyal Customer	46	Business travel	Business	
11	10	47583	Female	Loyal Customer	47	Business travel	Eco	
12	11	115550	Female	Loyal Customer	33	Business travel	Business	
13	12	119987	Female	Loyal Customer	46	Business travel	Business	
14	13	42141	Female	Loyal Customer	60	Business travel	Business	
15	14	2274	Female	Loyal Customer	52	Business travel	Business	
16	15	22470	Male	Loyal Customer	50	Personal Travel	Eco	
17	16	124915	Female	Loyal Customer	31	Business travel	Eco	
18	17	17836	Male	Loyal Customer	52	Personal Travel	Eco Plus	
19	18	76872	Female	Loyal Customer	43	Personal Travel	Eco	
20	19	64287	Female	Loyal Customer	50	Business travel	Business	
21	20	63995	Male	Loyal Customer	60	Business travel	Business	
22	21	75855	Male	Loyal Customer	43	Personal Travel	Eco	
23	22	106181	Male	Loyal Customer	55	Personal Travel	Eco	
24	23	44304	Male	Loyal Customer	25	Business travel	Business	
25	24	82602	Female	disloyal Customer	30	Business travel	Eco	
26	25	7823	Male	Loyal Customer	62	Personal Travel	Eco	
27	26	127781	Male	Loyal Customer	24	Business travel	Business	
28	27	34501	Female	Loyal Customer	22	Business travel	Eco	

Below are the screenshots of code from python notebook file

This function code will load the csv file and display the first few rows in the fill just to confirm that we have loaded our file successfully.

```
# Load the datasets
train_data = pd.read_csv('train.csv',index_col=0)

train_data.drop("id",axis=1,inplace=True)

train_data.head()
```

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service
0	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	1	...	5	4	3	4	4	5
1	Male	disloyal Customer	25	Business travel	Business	235	3	2	3	3	...	1	1	5	3	1	4
2	Female	Loyal Customer	26	Business travel	Business	1142	2	2	2	2	...	5	4	3	4	4	4
3	Female	Loyal Customer	25	Business travel	Business	562	2	5	5	5	...	2	2	5	3	1	4
4	Male	Loyal Customer	61	Business travel	Business	214	3	3	3	3	...	3	3	4	4	3	3

5 rows × 23 columns

Below is the screenshot of function codes which will check for the missing values and the next cell will fill the missing values then the next cell will Separate the data into features and labels.

```
[3]: # Check for NA values
     train_data.isna().sum()
```

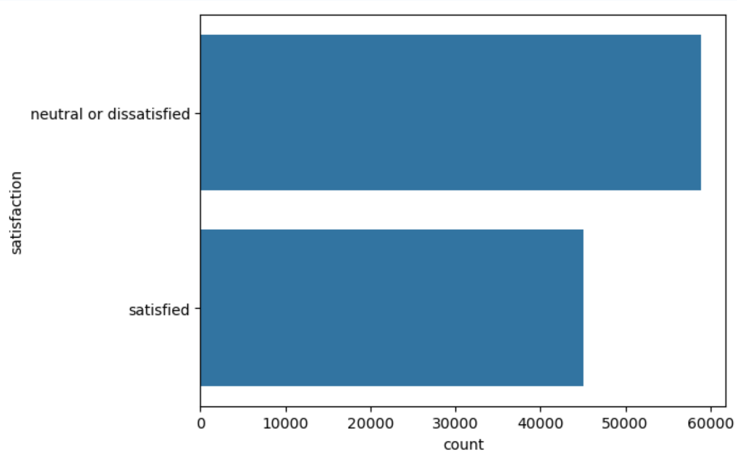
```
[3]: Gender                                  0
     Customer Type                           0
     Age                                     0
     Type of Travel                          0
     Class                                   0
     Flight Distance                         0
     Inflight wifi service                   0
     Departure/Arrival time convenient       0
     Ease of Online booking                  0
     Gate location                           0
     Food and drink                          0
     Online boarding                         0
     Seat comfort                            0
     Inflight entertainment                  0
     On-board service                        0
     Leg room service                        0
     Baggage handling                        0
     Checkin service                         0
     Inflight service                        0
     Cleanliness                             0
     Departure Delay in Minutes              0
     Arrival Delay in Minutes              310
     satisfaction                            0
     dtype: int64
```

```
[4]: # Fill the missing values
     train_data["Arrival Delay in Minutes"] = train_data["Arrival Delay in Minutes"].fillna(train_data["Departure Delay in Minutes"])
```

```
[5]: # Seperate the data into features and lables
     X_data = train_data.iloc[:,:-1]
     y_data = train_data.iloc[:,-1]
```

Below, is the screenshot of function code displaying the Count of Labels and object columns.

```
[6]: # Count of Labels
     sns.countplot(y_data, label="Count")
     plt.show()
```




```
[7]: # Print the object coloumns
     X_data.select_dtypes(object)
```

[7]:

	Gender	Customer Type	Type of Travel	Class
0	Male	Loyal Customer	Personal Travel	Eco Plus
1	Male	disloyal Customer	Business travel	Business
2	Female	Loyal Customer	Business travel	Business
3	Female	Loyal Customer	Business travel	Business
4	Male	Loyal Customer	Business travel	Business

Below, is the screenshot of function code transforming categorical coloumns into integers and the second cell is Selecting Numerical Columns then the next cell is Plotting Covarience matrix.

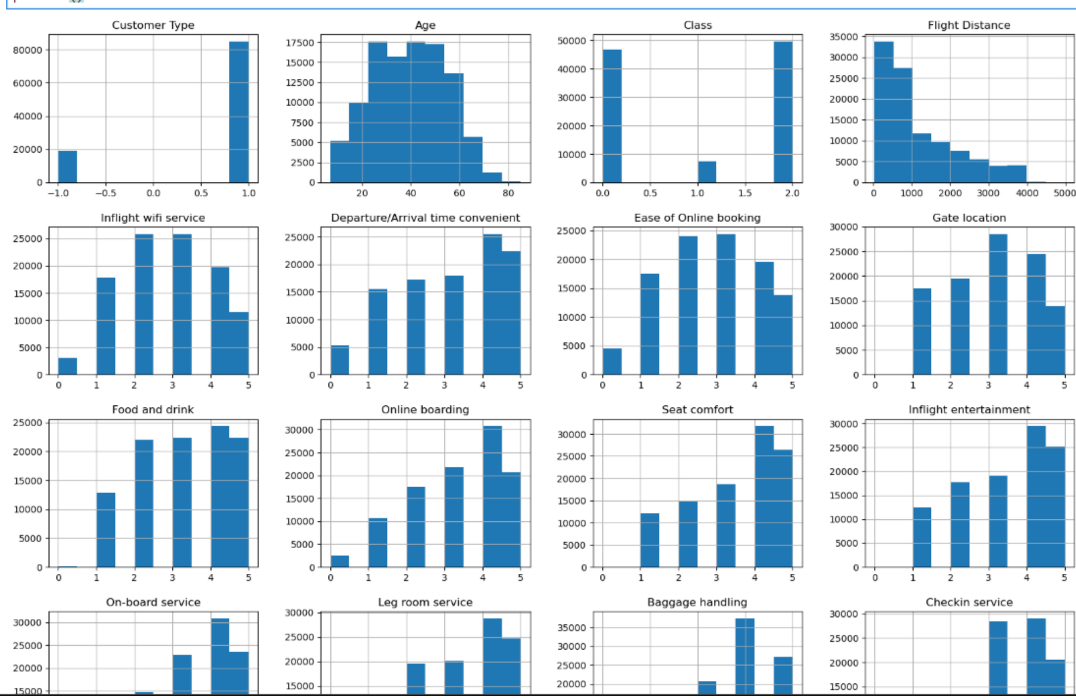
```
[10]: # transforming categorical coloumns into integers
X_data["Class"].replace(to_replace=['Eco Plus', 'Business', 'Eco'], value=[1,2,0], inplace=True)
X_data["Customer Type"].replace(to_replace=['Loyal Customer', 'disloyal Customer'], value=[1,-1], inplace=True)
```

```
[11]: # Selecting Numerical Coloumns
numerical_cols = list(X_data.select_dtypes(int).columns)
```

```
[12]: # Plotting Covarience matrix
fig, axis = plt.subplots(figsize=(20,18))
correlation_matrix = X_data.loc[:,numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, linewidths=.5, fmt='.2f', ax=axis)
plt.show()
```

Below, is the screenshot displaying the histogram plots of the data.

```
[13]: # histogram plots
X_data.loc[:,numerical_cols].hist(figsize=(20, 18))
plt.show()
```



Below, is the screenshot of function code one-hot encoding and next cell Transforming labels into integers then the next cell is Split the data into train and test datasets next cell is Scaling the data.

```
[14]: # One hot encoding
cat_cols = ["Type of Travel","Gender"]
for col in cat_cols:
    dummies = pd.get_dummies(X_data[col])
    X_data = pd.concat([X_data, dummies], axis=1)
    X_data = X_data.drop([col], axis=1)
X_data.loc[:,X_data.select_dtypes(bool).columns] = X_data.select_dtypes(bool).astype(int)
```

```
[15]: # Transforming labels into integers
y_data.replace(to_replace=['neutral or dissatisfied', 'satisfied'], value=[0,1], inplace=True)
```

```
[16]: # # Split the data into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.1, random_state=21, stratify=y_data)
for dataset in [X_train, X_test, y_train, y_test]:
    dataset.reset_index(drop=True, inplace=True)
```

```
[17]: # Scale the data
scaler = StandardScaler()
Xs_train = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
Xs_test = pd.DataFrame(scaler.transform(X_test),columns=X_test.columns)
```

```
[18]:
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming you have some dataset X and y
# X: Features, y: Labels

# Example of splitting data into train and test
features_train_all, features_test_all, labels_train_all, labels_test_all = train_test_split(X_test,y_test, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
features_train_all_std = scaler.fit_transform(features_train_all)
features_test_all_std = scaler.transform(features_test_all)
```

Below is the function to calculate Metrics

```
[20]: # funtion to calculate Metrics
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
```

```
[21]: # Train the model and return the Metrics
def get_metrics(model, X_train, X_test, y_train, y_test, LSTM_flag):
    metrics = []
    if LSTM_flag == 1:
        Xtrain, Xtest, ytrain, ytest = map(np.array, [X_train, X_test, y_train, y_test])
        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(X_train, y_train)
        predicted = model.predict(X_test)
        matrix = confusion_matrix(y_test, predicted, labels=[1, 0])
        model_brier_score = brier_score_loss(y_test, model.predict_proba(X_test)[:, 1])
```

Below, is the function code of parameter tuning for KNN-algorithm

[22]:
```
# Parameter tuning for KNN Algorithm

from sklearn.model_selection import RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Define the KNN model
knn = KNeighborsClassifier()
# Use only a subset  of the training data for hyperparameter tuning beacuse the data is huge and required more time for tuning
X_train_subset = X_train.sample(frac=0.1, random_state=42)
y_train_subset = y_train[X_train_subset.index]

# Set up the parameter grid with fewer values
param_dist_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree']
}

# Perform randomized search (instead of grid search) with 5 iterations and 10-fold cross-validation
random_search_knn = RandomizedSearchCV(knn, param_dist_knn, n_iter=5, cv=10, scoring='accuracy', verbose=1, n_jobs=-1)
random_search_knn.fit(X_train_subset, y_train_subset)

# Print the best parameters and best score
print("Best KNN parameters:", random_search_knn.best_params_)
print("Best KNN score:", random_search_knn.best_score_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
Best KNN parameters: {'weights': 'distance', 'n_neighbors': 9, 'algorithm': 'auto'}
Best KNN score: 0.6739422505598975
```

Below, is the function code of parameter tuning for Random Forest-algorithm

[23]:
```
# Parameter tuning for Random Forest

from sklearn.ensemble import RandomForestClassifier

# Define the Random Forest model
rf = RandomForestClassifier(random_state=42)
# Use only a subset  of the training data for hyperparameter tuning beacuse the data is huge and required more time for tuning
X_train_subset = X_train.sample(frac=0.1, random_state=42)
y_train_subset = y_train[X_train_subset.index]

# Set up the parameter grid with fewer values
param_dist_rf = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Perform randomized search with 5 iterations and 10-fold cross-validation
random_search_rf = RandomizedSearchCV(rf, param_dist_rf, n_iter=5, cv=10, scoring='accuracy', verbose=1, n_jobs=-1)
random_search_rf.fit(X_train_subset, y_train_subset)

# Print the best parameters and best score
print("Best Random Forest parameters:", random_search_rf.best_params_)
print("Best Random Forest score:", random_search_rf.best_score_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
Best Random Forest parameters: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 20}
Best Random Forest score: 0.9493106403400521
```

Below, is the function code of parameter tuning for SVM-algorithm

```
[24]: # Parameter tuning for SVM

from sklearn.svm import LinearSVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Assuming your data is already loaded as X_train, y_train

# Scale the data (SVM is sensitive to the scale of data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Use only a subset  of the training data for hyperparameter tuning beacuse the data is huge and required more time for tuning
X_train_subset = X_train.sample(frac=0.1, random_state=42)
y_train_subset = y_train[X_train_subset.index]

# Initialize LinearSVC for linear classification problems
svm = LinearSVC(C=1.0, max_iter=1000, dual=False)  # Remove n_jobs

# Define a smaller parameter grid for hyperparameter tuning
param_dist = {
    'C': [0.1, 1.0, 10.0],  # Fewer values for C
}

# Use RandomizedSearchCV with fewer iterations to reduce time
random_search = RandomizedSearchCV(svm, param_dist, n_iter=5, cv=10, scoring='accuracy', verbose=1)

# Fit the model
random_search.fit(X_train_subset, y_train_subset)

# Print the best parameters found during the search
print(f"Best Parameters: {random_search.best_params_}")

Fitting 10 folds for each of 3 candidates, totalling 30 fits
Best Parameters: {'C': 0.1}
```

Below is the function code to Compare Classifiers using 10-Fold Stratified Cross-Validation

```
[26]: # Compare Classifiers using 10-Fold Stratified Cross-Validation

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, brier_score_loss, roc_auc_score
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

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

# 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 = [], [], [], []

# Set up parameter for SVM
C = 1.0

# Assuming random_search_knn and random_search_rf have been already defined, here is how to use them
# 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):

    # Get KNN best parameters from random search (assuming you already performed RandomizedSearchCV or GridSearchCV)
    knn_params = random_search_knn.best_params_

    # KNN Model with correct parameters
    knn_model = KNeighborsClassifier(n_neighbors=knn_params['n_neighbors'],
                                     weights=knn_params['weights'],
                                     algorithm=knn_params['algorithm'])

    # Random Forest Model (assuming random_search_rf.best_params_ works similarly)
    rf_params = random_search_rf.best_params_
    rf_model = RandomForestClassifier(min_samples_split=rf_params['min_samples_split'])

    # SVM Classifier Model
    svm_model = SVC(C=C, kernel='linear', probability=True)

    # LSTM Model
    lstm_model = Sequential()
    lstm_model.add(LSTM(64, activation='relu', input_shape=(8, 1), return_sequences=False))  # Correct input shape
    lstm_model.add(Dense(1, activation='sigmoid'))
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```python
    # Split data into training and testing sets
    # Convert numpy arrays to pandas DataFrame/Series if they are numpy arrays
    features_train_all_std = pd.DataFrame(features_train_all_std)  # Convert features to DataFrame
    labels_train_all = pd.Series(labels_train_all)  # Convert labels to Series

    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.iloc[train_index], labels_train_all.iloc[test_index]  # Use iloc for labels

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

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

Iteration 1:

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

                    KNN      RF     SVM    LSTM
TP               306.00  337.00  297.00  316.00
TN               456.00  448.00  431.00  460.00
FP                16.00   24.00   41.00   12.00
FN                54.00   23.00   63.00   44.00
TPR                0.85    0.94    0.82    0.88
TNR                0.97    0.95    0.91    0.97
FPR                0.03    0.05    0.09    0.03
FNR                0.15    0.06    0.18    0.12
Precision          0.95    0.93    0.88    0.96
F1_measure         0.90    0.93    0.85    0.92
Accuracy           0.92    0.94    0.88    0.93
Error_rate         0.08    0.06    0.12    0.07
BACC               0.91    0.94    0.87    0.93
TSS                0.82    0.89    0.74    0.85
HSS                0.83    0.88    0.74    0.86
Brier_score        0.07    0.04    0.10    0.05
AUC                0.96    0.99    0.92    0.98
Acc_by_package_fn  0.92    0.94    0.88    0.93
```

Below is the function code which will Initialize Metric Index forIterations(all four algorithms)

[27]:

```python
# Initialize Metric Index for Iterations

metric_index_df = ['iter1', 'iter2', 'iter3', 'iter4', 'iter5', 'iter6', 'iter7', 'iter8', 'iter9', 'iter10']

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)

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  \
TP                306.00  304.00  309.00  302.00  306.00  308.00  299.00
TN                456.00  454.00  450.00  456.00  446.00  449.00  458.00
FP                 16.00   18.00   22.00   16.00   25.00   22.00   13.00
FN                 54.00   56.00   50.00   57.00   54.00   52.00   61.00
TPR                 0.85    0.84    0.86    0.84    0.85    0.86    0.83
TNR                 0.97    0.96    0.95    0.97    0.95    0.95    0.97
FPR                 0.03    0.04    0.05    0.03    0.05    0.05    0.03
FNR                 0.15    0.16    0.14    0.16    0.15    0.14    0.17
Precision           0.95    0.94    0.93    0.95    0.92    0.93    0.96
F1_measure          0.90    0.89    0.90    0.89    0.89    0.89    0.89
Accuracy            0.92    0.91    0.91    0.91    0.90    0.91    0.91
Error_rate          0.08    0.09    0.09    0.09    0.10    0.09    0.09
BACC                0.91    0.90    0.91    0.90    0.90    0.90    0.90
TSS                 0.82    0.81    0.81    0.81    0.80    0.81    0.80
HSS                 0.83    0.82    0.82    0.82    0.80    0.82    0.82
Brier_score         0.07    0.07    0.06    0.06    0.07    0.06    0.07
AUC                 0.96    0.96    0.97    0.96    0.96    0.97    0.96
Acc_by_package_fn   0.92    0.91    0.91    0.91    0.90    0.91    0.91

                   iter8   iter9  iter10
TP                305.00  296.00  291.00
TN                445.00  460.00  455.00
FP                 26.00   11.00   16.00
FN                 55.00   64.00   69.00
TPR                 0.85    0.82    0.81
TNR                 0.94    0.98    0.97
FPR                 0.06    0.02    0.03
FNR                 0.15    0.18    0.19
Precision           0.92    0.96    0.95
F1_measure          0.88    0.89    0.87
Accuracy            0.90    0.91    0.90
Error_rate          0.10    0.09    0.10
BACC                0.90    0.90    0.89
TSS                 0.79    0.80    0.77
HSS                 0.80    0.81    0.79
Brier_score         0.07    0.07    0.08
AUC                 0.96    0.96    0.96
Acc_by_package_fn   0.90    0.91    0.90
```

```
Metrics for Algorithm 2:

                   iter1   iter2   iter3   iter4   iter5   iter6   iter7  \
TP                337.00  337.00  337.00  336.00  333.00  334.00  329.00
TN                448.00  449.00  449.00  458.00  451.00  456.00  456.00
FP                 24.00   23.00   23.00   14.00   20.00   15.00   15.00
FN                 23.00   23.00   22.00   23.00   27.00   26.00   31.00
TPR                 0.94    0.94    0.94    0.94    0.92    0.93    0.91
TNR                 0.95    0.95    0.95    0.97    0.96    0.97    0.97
FPR                 0.05    0.05    0.05    0.03    0.04    0.03    0.03
FNR                 0.06    0.06    0.06    0.06    0.08    0.07    0.09
Precision           0.93    0.94    0.94    0.96    0.94    0.96    0.96
F1_measure          0.93    0.94    0.94    0.95    0.93    0.94    0.93
Accuracy            0.94    0.94    0.95    0.96    0.94    0.95    0.94
Error_rate          0.06    0.06    0.05    0.04    0.06    0.05    0.06
BACC                0.94    0.94    0.94    0.95    0.94    0.95    0.94
TSS                 0.89    0.89    0.89    0.91    0.88    0.90    0.88
HSS                 0.88    0.89    0.89    0.91    0.88    0.90    0.89
Brier_score         0.04    0.04    0.04    0.04    0.05    0.04    0.05
AUC                 0.99    0.99    0.99    0.99    0.98    0.99    0.99
Acc_by_package_fn   0.94    0.94    0.95    0.96    0.94    0.95    0.94

                   iter8   iter9  iter10
TP                332.00  329.00  332.00
TN                455.00  457.00  456.00
FP                 16.00   14.00   15.00
FN                 28.00   31.00   28.00
TPR                 0.92    0.91    0.92
TNR                 0.97    0.97    0.97
FPR                 0.03    0.03    0.03
FNR                 0.08    0.09    0.08
Precision           0.95    0.96    0.96
F1_measure          0.94    0.94    0.94
Accuracy            0.95    0.95    0.95
Error_rate          0.05    0.05    0.05
BACC                0.94    0.94    0.95
TSS                 0.89    0.88    0.89
HSS                 0.89    0.89    0.89
Brier_score         0.04    0.05    0.04
AUC                 0.99    0.99    0.99
Acc_by_package_fn   0.95    0.95    0.95


Metrics for Algorithm 3:

                   iter1   iter2   iter3   iter4   iter5   iter6   iter7  \
TP                297.00  302.00  306.00  297.00  300.00  305.00  290.00
TN                431.00  432.00  430.00  431.00  423.00  426.00  427.00
FP                 41.00   40.00   42.00   41.00   48.00   45.00   44.00
FN                 63.00   58.00   53.00   62.00   60.00   55.00   70.00
TPR                 0.82    0.84    0.85    0.83    0.83    0.85    0.81
TNR                 0.91    0.92    0.91    0.91    0.90    0.90    0.91
FPR                 0.09    0.08    0.09    0.09    0.10    0.10    0.09
FNR                 0.18    0.16    0.15    0.17    0.17    0.15    0.19
Precision           0.88    0.88    0.88    0.88    0.86    0.87    0.87
F1_measure          0.85    0.86    0.87    0.85    0.85    0.86    0.84
Accuracy            0.88    0.88    0.89    0.88    0.87    0.88    0.86
Error_rate          0.12    0.12    0.11    0.12    0.13    0.12    0.14
BACC                0.87    0.88    0.88    0.87    0.87    0.88    0.86
TSS                 0.74    0.75    0.76    0.74    0.73    0.75    0.71
HSS                 0.74    0.76    0.77    0.75    0.73    0.75    0.72
Brier_score         0.10    0.09    0.09    0.10    0.09    0.09    0.10
AUC                 0.92    0.93    0.93    0.92    0.94    0.93    0.92
Acc_by_package_fn   0.88    0.88    0.89    0.88    0.87    0.88    0.86

                   iter8   iter9  iter10
TP                303.00  297.00  292.00
TN                421.00  424.00  435.00
FP                 50.00   47.00   36.00
FN                 57.00   63.00   68.00
TPR                 0.84    0.82    0.81
```

Metrics for Algorithm 4:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7 \
TP	316.00	326.00	330.00	316.00	333.00	323.00	323.00
TN	460.00	458.00	450.00	462.00	437.00	454.00	451.00
FP	12.00	14.00	22.00	10.00	34.00	17.00	20.00
FN	44.00	34.00	29.00	43.00	27.00	37.00	37.00
TPR	0.88	0.91	0.92	0.88	0.92	0.90	0.90
TNR	0.97	0.97	0.95	0.98	0.93	0.96	0.96
FPR	0.03	0.03	0.05	0.02	0.07	0.04	0.04
FNR	0.12	0.09	0.08	0.12	0.08	0.10	0.10
Precision	0.96	0.96	0.94	0.97	0.91	0.95	0.94
F1_measure	0.92	0.93	0.93	0.92	0.92	0.92	0.92
Accuracy	0.93	0.94	0.94	0.94	0.93	0.94	0.93
Error_rate	0.07	0.06	0.06	0.06	0.07	0.06	0.07
BACC	0.93	0.94	0.94	0.93	0.93	0.93	0.93
TSS	0.85	0.88	0.87	0.86	0.85	0.86	0.85
HSS	0.86	0.88	0.87	0.87	0.85	0.87	0.86
Brier_score	0.05	0.05	0.05	0.05	0.06	0.05	0.05
AUC	0.98	0.98	0.98	0.98	0.97	0.98	0.98
Acc_by_package_fn	0.93	0.94	0.94	0.94	0.93	0.94	0.93

	iter8	iter9	iter10
TP	319.00	326.00	325.00
TN	449.00	446.00	442.00
FP	22.00	25.00	29.00
FN	41.00	34.00	35.00
TPR	0.89	0.91	0.90
TNR	0.95	0.95	0.94
FPR	0.05	0.05	0.06
FNR	0.11	0.09	0.10
Precision	0.94	0.93	0.92
F1_measure	0.91	0.92	0.91
Accuracy	0.92	0.93	0.92
Error_rate	0.08	0.07	0.08
BACC	0.92	0.93	0.92
TSS	0.84	0.85	0.84
HSS	0.84	0.85	0.84
Brier_score	0.05	0.06	0.06
AUC	0.98	0.97	0.98
Acc_by_package_fn	0.92	0.93	0.92

Below is the function code Calculates the average metrics for each algorithm

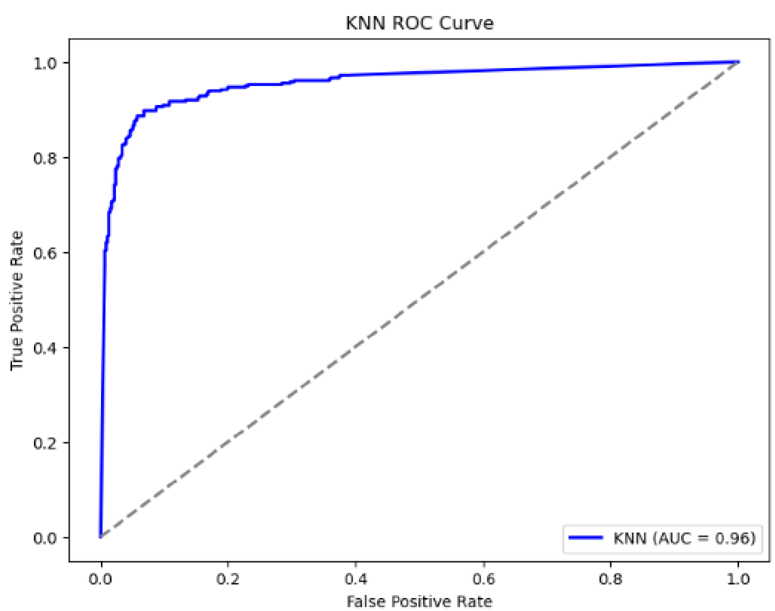
[28]:

```python
# Calculate the average metrics for each algorithm

import pandas as pd

# Define the 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 DataFrames to collect metrics for each algorithm
knn_metrics_df = pd.DataFrame(columns=metric_columns)
rf_metrics_df = pd.DataFrame(columns=metric_columns)
svm_metrics_df = pd.DataFrame(columns=metric_columns)
lstm_metrics_df = pd.DataFrame(columns=metric_columns)

# Assuming you already have the cross-validation loop here
for iter_num, (train_index, test_index) in enumerate(cv_stratified.split(features_train_all_std, labels_train_all), start=1):

    # Your model training and metric collection code here...

    # After getting metrics for each model (knn_metrics, rf_metrics, svm_metrics, lstm_metrics)

    # Use pd.concat to append metrics as a new row in the DataFrame
    knn_metrics_df = pd.concat([knn_metrics_df, pd.Series(knn_metrics, index=metric_columns).to_frame().T], ignore_index=True)
    rf_metrics_df = pd.concat([rf_metrics_df, pd.Series(rf_metrics, index=metric_columns).to_frame().T], ignore_index=True)
    svm_metrics_df = pd.concat([svm_metrics_df, pd.Series(svm_metrics, index=metric_columns).to_frame().T], ignore_index=True)
    lstm_metrics_df = pd.concat([lstm_metrics_df, pd.Series(lstm_metrics, index=metric_columns).to_frame().T], ignore_index=True)

# After collecting all the metrics for each model across all iterations
# Calculate the average of each metric 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))
```

	KNN	RF	SVM	LSTM
TP	291.00	332.00	292.00	325.00
TN	455.00	456.00	435.00	442.00
FP	16.00	15.00	36.00	29.00
FN	69.00	28.00	68.00	35.00
TPR	0.81	0.92	0.81	0.90
TNR	0.97	0.97	0.92	0.94
FPR	0.03	0.03	0.08	0.06
FNR	0.19	0.08	0.19	0.10
Precision	0.95	0.96	0.89	0.92
F1_measure	0.87	0.94	0.85	0.91
Accuracy	0.90	0.95	0.87	0.92
Error_rate	0.10	0.05	0.13	0.08
BACC	0.89	0.95	0.87	0.92
TSS	0.77	0.89	0.73	0.84
HSS	0.79	0.89	0.74	0.84
Brier_score	0.08	0.04	0.10	0.06
AUC	0.96	0.99	0.93	0.98
Acc_by_package_fn	0.90	0.95	0.87	0.92

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

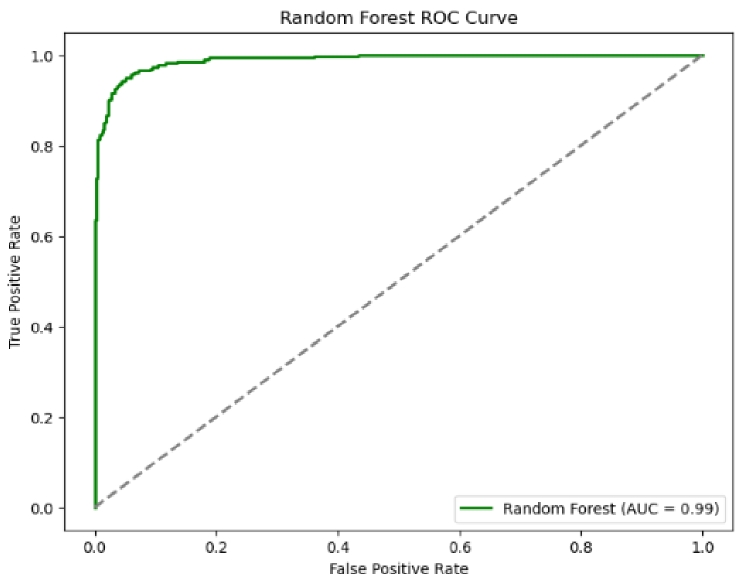
```
[29]: #Evaluating the performance of various algorithms by comparing their ROC curves and AUC scores on the test dataset.

# Implementing roc curves and AOC Score for KNN
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Get predicted probabilities for KNN
knn_probs = knn_model.predict_proba(features_test)[:, 1]  # Probability for class 1

# Calculate ROC curve for KNN
fpr_knn, tpr_knn, _ = roc_curve(labels_test, knn_probs)

# Calculate AUC for KNN
roc_auc_knn = auc(fpr_knn, tpr_knn)

# Plot ROC curve for KNN
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label='KNN (AUC = {:.2f})'.format(roc_auc_knn))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)  # Chance level
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN ROC Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC for KNN
print(f"KNN AUC: {roc_auc_knn:.2f}")
```




```
KNN AUC: 0.96
```

Below is the screenshot of Implementing roc curves and AOC Score for Random Forest

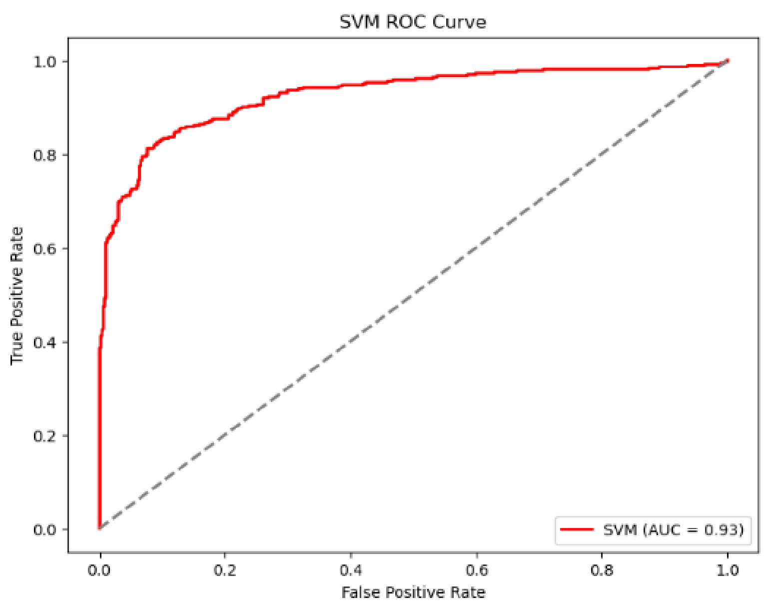
```
[30]: # Implementing roc curves and AOC Score for Random Forest

# Get predicted probabilities for Random Forest
rf_probs = rf_model.predict_proba(features_test)[:, 1]  # Probability for class 1

# Calculate ROC curve for Random Forest
fpr_rf, tpr_rf, _ = roc_curve(labels_test, rf_probs)

# Calculate AUC for Random Forest
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve for Random Forest
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label='Random Forest (AUC = {:.2f})'.format(roc_auc_rf))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)  # Chance Level
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC for Random Forest
print(f"Random Forest AUC: {roc_auc_rf:.2f}")
```




```
Random Forest AUC: 0.99
```

Below is the screenshot of Implementing roc curves and AOC Score for SVM

[31]:
```python
# Implementing roc curves and AOC Score for SVM

# Get predicted probabilities for SVM
svm_probs = svm_model.predict_proba(features_test)[:, 1]  # Probability for class 1

# Calculate ROC curve for SVM
fpr_svm, tpr_svm, _ = roc_curve(labels_test, svm_probs)

# Calculate AUC for SVM
roc_auc_svm = auc(fpr_svm, tpr_svm)

# Plot ROC curve for SVM
plt.figure(figsize=(8, 6))
plt.plot(fpr_svm, tpr_svm, color='red', lw=2, label='SVM (AUC = {:.2f})'.format(roc_auc_svm))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)  # Chance level
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM ROC Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC for SVM
print(f"SVM AUC: {roc_auc_svm:.2f}")
```



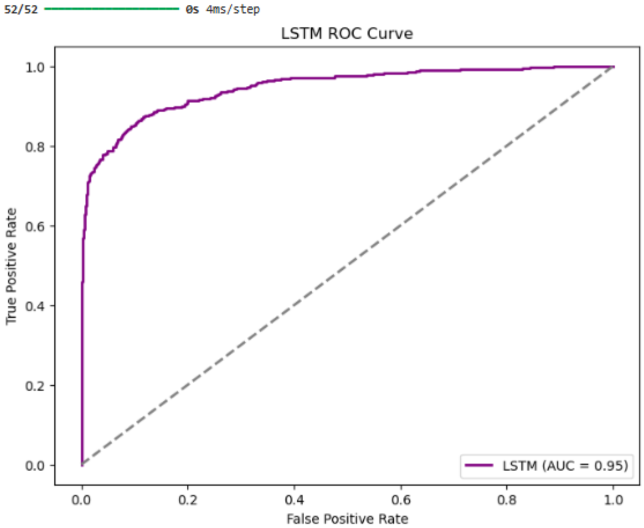

```
SVM AUC: 0.93
```

Below is the function code of Implementing roc curves and AOC Score for LSTM

[32]:
```python
# # Implementing roc curves and AOC Score for LSTM

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# Split the data once (you can also use train_test_split here, but assuming you're using your pre-split data)
# Use indices or a direct split for training and testing data
train_size = int(0.8 * len(features_train_all_std))  # 80% for training, 20% for testing
features_train = features_train_all_std[:train_size]
labels_train = labels_train_all[:train_size]
features_test = features_train_all_std[train_size:]
labels_test = labels_train_all[train_size:]

# Reshape the data for LSTM (3D format)
features_train_lstm = features_train.values.reshape(features_train.shape[0], features_train.shape[1], 1)
features_test_lstm = features_test.values.reshape(features_test.shape[0], features_test.shape[1], 1)

# Initialize your LSTM model (Ensure it's correctly built as in the previous steps)
lstm_model = Sequential()
lstm_model.add(LSTM(64, activation='relu', input_shape=(features_train_lstm.shape[1], 1), return_sequences=False))  # Shape (samples, features, 1)
lstm_model.add(Dense(1, activation='sigmoid'))  # Sigmoid activation for binary classification
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
lstm_model.fit(features_train_lstm, labels_train, epochs=10, batch_size=32, verbose=1)

# Get predicted probabilities for LSTM (for binary classification, this is the probability for class 1)
lstm_probs = lstm_model.predict(features_test_lstm)

# Calculate ROC curve
fpr_lstm, tpr_lstm, _ = roc_curve(labels_test, lstm_probs)

# Calculate AUC for LSTM
roc_auc_lstm = auc(fpr_lstm, tpr_lstm)

# Plot ROC curve for LSTM
plt.figure(figsize=(8, 6))
plt.plot(fpr_lstm, tpr_lstm, color='purple', lw=2, label='LSTM (AUC = {:.2f})'.format(roc_auc_lstm))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)  # Chance level
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LSTM ROC Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC for LSTM
print(f"LSTM AUC: {roc_auc_lstm:.2f}")
```

```
Epoch 1/10
```

```
Epoch 9/10
208/208 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8997 - loss: 0.2657
Epoch 10/10
208/208 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9048 - loss: 0.2544
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

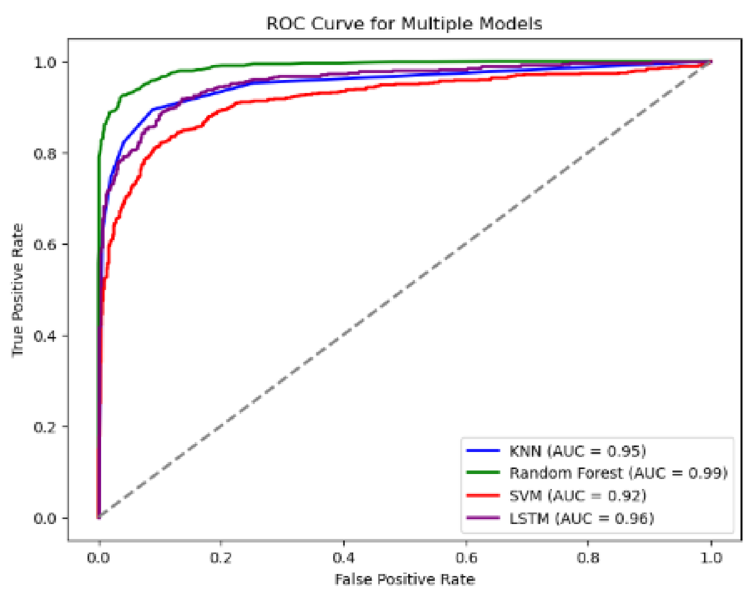



```
LSTM AUC: 0.95
```

Below is the function code for Plotting ROC Curves for All Models

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Initialize models
knn_model = KNeighborsClassifier(n_neighbors=5)  # Replace with best parameters
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # Replace with best parameters
svm_model = SVC(kernel='linear', probability=True, random_state=42)  # SVM with probability output

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

# Train models
knn_model.fit(features_train, labels_train)
rf_model.fit(features_train, labels_train)
svm_model.fit(features_train, labels_train)
lstm_model.fit(features_train_lstm, labels_train, epochs=10, batch_size=32, verbose=0)

# Get predicted probabilities for each model
knn_probs = knn_model.predict_proba(features_test)[:, 1]  # Probability for class 1
rf_probs = rf_model.predict_proba(features_test)[:, 1]  # Probability for class 1
svm_probs = svm_model.predict_proba(features_test)[:, 1]  # Probability for class 1
lstm_probs = lstm_model.predict(features_test_lstm)  # Probability for class 1

# Calculate ROC curve for each model
fpr_knn, tpr_knn, _ = roc_curve(labels_test, knn_probs)
fpr_rf, tpr_rf, _ = roc_curve(labels_test, rf_probs)
fpr_svm, tpr_svm, _ = roc_curve(labels_test, svm_probs)
fpr_lstm, tpr_lstm, _ = roc_curve(labels_test, lstm_probs)

# Calculate AUC for each model
roc_auc_knn = auc(fpr_knn, tpr_knn)
roc_auc_rf = auc(fpr_rf, tpr_rf)
roc_auc_svm = auc(fpr_svm, tpr_svm)
roc_auc_lstm = auc(fpr_lstm, tpr_lstm)

# Plot ROC curve for all models
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label='KNN (AUC = {:.2f})'.format(roc_auc_knn))
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label='Random Forest (AUC = {:.2f})'.format(roc_auc_rf))
plt.plot(fpr_svm, tpr_svm, color='red', lw=2, label='SVM (AUC = {:.2f})'.format(roc_auc_svm))
plt.plot(fpr_lstm, tpr_lstm, color='purple', lw=2, label='LSTM (AUC = {:.2f})'.format(roc_auc_lstm))

# Plot chance line
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)

# Formatting the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multiple Models')
plt.legend(loc='lower right')
plt.show()
```

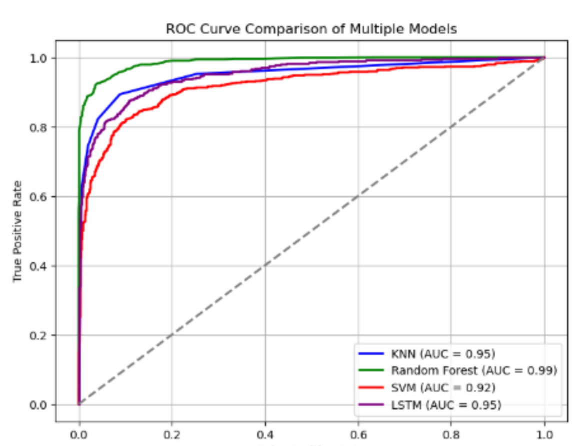
```
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



Below is the Function code for Comparing All Models ROC and AUC scores

```python
[34]: # Comparing All Models

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Initialize models
knn_model = KNeighborsClassifier(n_neighbors=5)  # Replace with best parameters
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # Replace with best parameters
svm_model = SVC(kernel='linear', probability=True, random_state=42)  # SVM with probability output

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

# Train models
knn_model.fit(features_train, labels_train)
rf_model.fit(features_train, labels_train)
svm_model.fit(features_train, labels_train)
lstm_model.fit(features_train_lstm, labels_train, epochs=10, batch_size=32, verbose=0)

# Get predicted probabilities for each model
knn_probs = knn_model.predict_proba(features_test)[:, 1]  # Probability for class 1
rf_probs = rf_model.predict_proba(features_test)[:, 1]  # Probability for class 1
svm_probs = svm_model.predict_proba(features_test)[:, 1]  # Probability for class 1
lstm_probs = lstm_model.predict(features_test_lstm)  # Probability for class 1

# Calculate ROC curve for each model
fpr_knn, tpr_knn, _ = roc_curve(labels_test, knn_probs)
fpr_rf, tpr_rf, _ = roc_curve(labels_test, rf_probs)
fpr_svm, tpr_svm, _ = roc_curve(labels_test, svm_probs)
fpr_lstm, tpr_lstm, _ = roc_curve(labels_test, lstm_probs)

# Calculate AUC for each model
roc_auc_knn = auc(fpr_knn, tpr_knn)
roc_auc_rf = auc(fpr_rf, tpr_rf)
roc_auc_svm = auc(fpr_svm, tpr_svm)
roc_auc_lstm = auc(fpr_lstm, tpr_lstm)

# Plot ROC curve for all models
plt.figure(figsize=(8, 6))

# Plot each model's ROC curve with respective AUC score
plt.plot(fpr_knn, tpr_knn, color='blue', lw=2, label='KNN (AUC = {:.2f})'.format(roc_auc_knn))
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label='Random Forest (AUC = {:.2f})'.format(roc_auc_rf))
plt.plot(fpr_svm, tpr_svm, color='red', lw=2, label='SVM (AUC = {:.2f})'.format(roc_auc_svm))
plt.plot(fpr_lstm, tpr_lstm, color='purple', lw=2, label='LSTM (AUC = {:.2f})'.format(roc_auc_lstm))

# Plot chance line (diagonal line)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)

# Formatting the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of Multiple Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```



Below Is the screenshot of final average performance of the all four models

```
[35]: # Assuming 'avg_performance_df' has already been calculated from the earlier step

      print(avg_performance_df.round(decimals=2))
      print('\n')

                          KNN      RF     SVM    LSTM
      TP               291.00  332.00  292.00  325.00
      TN               455.00  456.00  435.00  442.00
      FP                16.00   15.00   36.00   29.00
      FN                69.00   28.00   68.00   35.00
      TPR                0.81    0.92    0.81    0.90
      TNR                0.97    0.97    0.92    0.94
      FPR                0.03    0.03    0.08    0.06
      FNR                0.19    0.08    0.19    0.10
      Precision          0.95    0.96    0.89    0.92
      F1_measure         0.87    0.94    0.85    0.91
      Accuracy           0.90    0.95    0.87    0.92
      Error_rate         0.10    0.05    0.13    0.08
      BACC               0.89    0.95    0.87    0.92
      TSS                0.77    0.89    0.73    0.84
      HSS                0.79    0.89    0.74    0.84
      Brier_score        0.08    0.04    0.10    0.06
      AUC                0.96    0.99    0.93    0.98
      Acc_by_package_fn  0.90    0.95    0.87    0.92
```

Steps to install the packages and run the program

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

The run steps as follows:

Step-1: Install jupyter notebook.

Step-2: Open jupyter notebook, install the following above packages with command “pip install” For all the above packages.

Step-3: Run the code sequentially as shown in the above screenshots.

Step-4: You will get the final output

Others or References

- Kaggle Dataset: Airline Passenger Satisfaction Dataset
- Scikit-Learn Documentation: https://scikit-learn.org/
- Keras Documentation: https://keras.io/
- Data Mining Concepts: Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Concepts and Techniques*.

The python notebook file (.pynb) and python(.py) file data sets (.csv) files attached to the folder file

Link to Git Repository

https://github.com/mazeenhussiansyed/Airlinepassengersatisfactiondatamining