NAME: Kartikey Patel
NJIT UCID: kp888
Email Address: kp888@njit.edu
23/11/2024
Professor: Yasser Abduallah
CS 634 101 Data Mining

Final Project Report

Implementation and Code Usage

Abstract:

Exploring the realm of binary classification opens up avenues for discerning patterns and characteristics in complex data sets, particularly when determining the potability of water. This project will delve into the comparative analysis of several machine learning approaches, including the K-Nearest Neighbour's classifier, , Random Forest, and LSTM. Let's dive into the core concepts and methodologies behind this endeavour.

Primary Objective:

The primary objective is to classify raisin samples into two categories: cammeo and non-cammeo. This classification helps ensure the assessment of raisin safety and quality, which is essential for public health and environmental management.

Machine learning models

1. **K-Nearest Neighbours Classifier:**

- It uses the k-nearest neighbors algorithm to perform classification based on similarity measure.
- Effective for small to medium-sized datasets and simple decision boundaries.
- The k parameter controls the number of neighbors considered during classification.

2. **Random forest:**

- It uses a set of decision trees to perform the classification.
- Known for its robustness and efficiency in handling large datasets with high dimensionality.
- Ability to capture complex relationships between input functions and target variable.

3. **LSTM:**

- Long Short-Term Memory (LSTM) neural network architecture.
- Designed to efficiently model sequential data using both past and future information.
- Particularly adept at capturing temporal dependencies and patterns in time series data.

Project workflow:

1. Data preprocessing:

- Load and pre-process raisin varieties data, including features such as Area, Perimeter and Eccentricity.
- Treat missing values, outliers, and data normalization to ensure model compatibility.

Classes 0 and 1 are almost balanced. Thus, no upsampling or downsampling.
Null values present.
Therefore, all null values are replaced with Median Values.

2. Model training:
• Implement each model using popular machine learning frameworks such as scikit-learn and TensorFlow/Keras.
• Split the data set into training and test sets for model evaluation.
• Train each model with the training data and evaluate its performance on the test set.

3. Performance Rating:
Assess model performance using metrics such as True Positive Rate (TPR), Specificity (SPC), Precision (PPV) and others specified.

1. **True Positive Rate (TPR):**
   - Also known as sensitivity or recall, TPR measures the proportion of actual positive cases that were correctly identified by the model.
     TPR = TP / (TP + FN)

2. **Specificity (SPC):**
   - Specificity measures the proportion of actual negative cases that were correctly identified by the model.
     TN / (TN + FP)

3. **Precision (PPV):**
   - Precision measures the proportion of true positive predictions among all positive predictions made by the model.
     TP / (TP + FP)

4. **Negative Predictive Value (NPV):**
   - NPV measures the proportion of true negative predictions among all negative predictions made by the model.
     NPV = TN / (TN + FN)

5. **False Positive Rate (FPR):**
   - FPR measures the proportion of actual negative cases that were incorrectly classified as positive by the model.
     FPR = FP / (FP + TN)

6. **False Discovery Rate (FDR):**
   - FDR measures the proportion of false positive predictions among all positive predictions made by the model.
     FDR = FP / (FP + TP)

7. **False Negative Rate (FNR):**
   - FNR measures the proportion of actual positive cases that were incorrectly classified as negative by the model.
     FNR = FN / (FN + TP)

8. **Accuracy (ACC):**

- Accuracy measures the overall correctness of the model's predictions, calculated as the ratio of correctly classified cases to the total number of cases.
  ACC = (TP + TN) / (TP + TN + FP + FN)

9. **F1 Score:**

- The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is particularly useful when the classes are imbalanced.
  F1 = 2 * TP / (2*TP + FP + FN)

10. **Balanced Accuracy (BACC):**

- BACC is the arithmetic mean of sensitivity (TPR) and specificity (SPC), providing a balanced evaluation of the model's performance across both classes.
  BACC = (TPR + SPC) / 2

11. **True Skill Statistics (TSS):**

- TSS measures the discriminatory power of the model by subtracting the false positive rate from the true positive rate.
  TSS = TPR - FPR

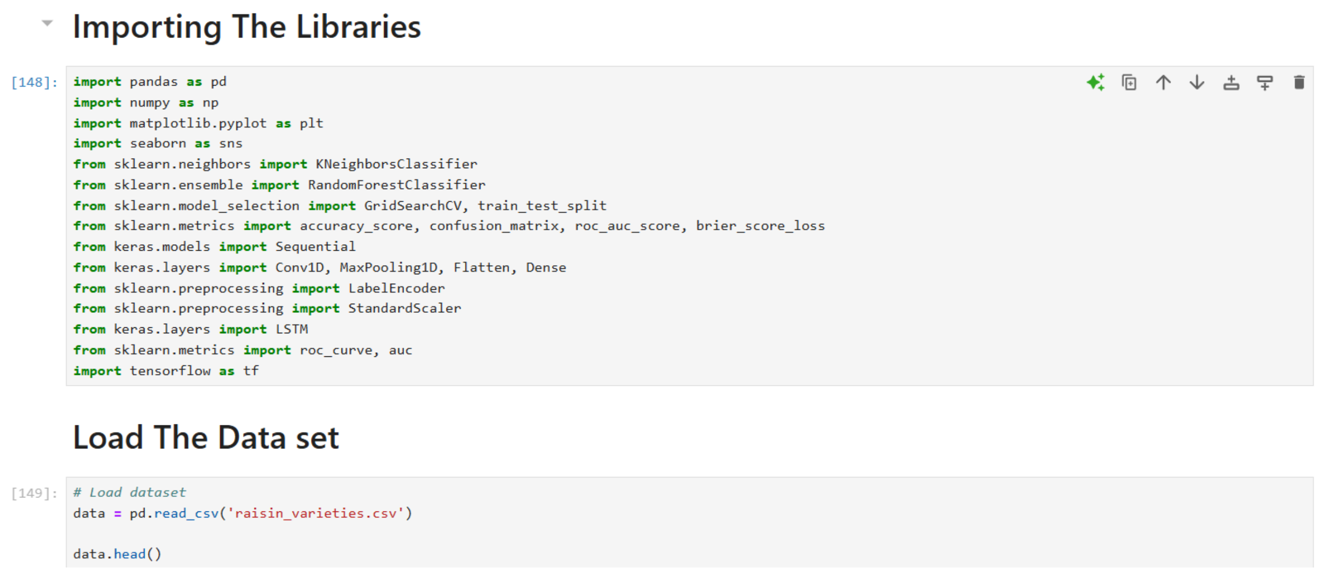
12. **Heidke Skill Score (HSS):**

- HSS measures the skill of the model compared to random chance, taking into account correct and incorrect classifications.
  HSS = (2 * (TP * TN - FP * FN)) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))

Conclusion:

In conclusion, this project showcases the effectiveness of various machine learning models in the binary classification of raisin variety. Each model, including K-Nearest Neighbours Classifier, Random Forest, and LSTM, offers unique advantages and performance characteristics. The comparative analysis provides valuable insights into their suitability for raisin quality assessment and management.

Screenshots

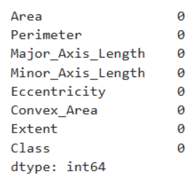


Importing The Libraries

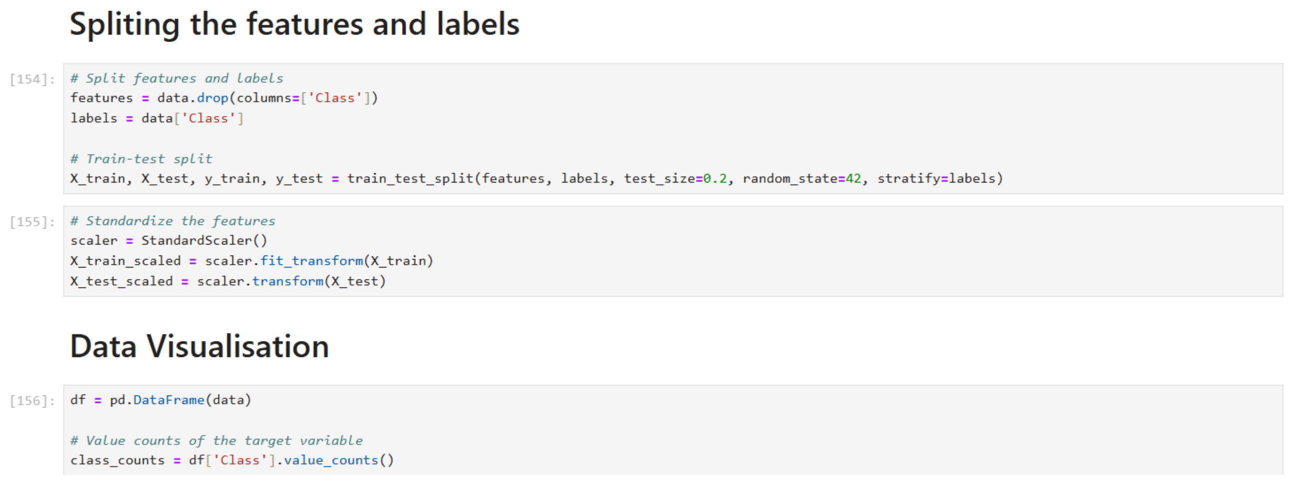
```
[148]: import pandas as pd
       import numpy as np
       import matplotlib.pyplot as plt
       import seaborn as sns
       from sklearn.neighbors import KNeighborsClassifier
       from sklearn.ensemble import RandomForestClassifier
       from sklearn.model_selection import GridSearchCV, train_test_split
       from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, brier_score_loss
       from keras.models import Sequential
       from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
       from sklearn.preprocessing import LabelEncoder
       from sklearn.preprocessing import StandardScaler
       from keras.layers import LSTM
       from sklearn.metrics import roc_curve, auc
       import tensorflow as tf
```

Load The Data set

```
[149]: # Load dataset
       data = pd.read_csv('raisin_varieties.csv')

       data.head()
```



```
Area               0
Perimeter          0
Major_Axis_Length  0
Minor_Axis_Length  0
Eccentricity       0
Convex_Area        0
Extent             0
Class              0
dtype: int64
```



Spliting the features and labels

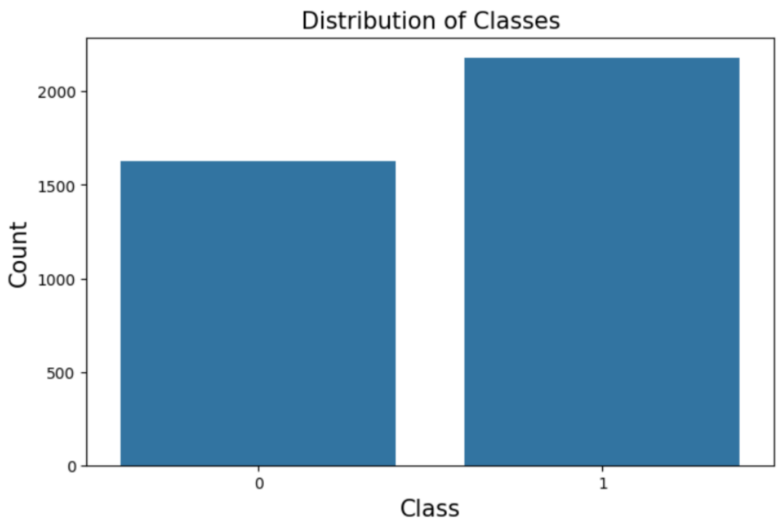
```
[154]: # Split features and labels
       features = data.drop(columns=['Class'])
       labels = data['Class']

       # Train-test split
       X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42, stratify=labels)
```

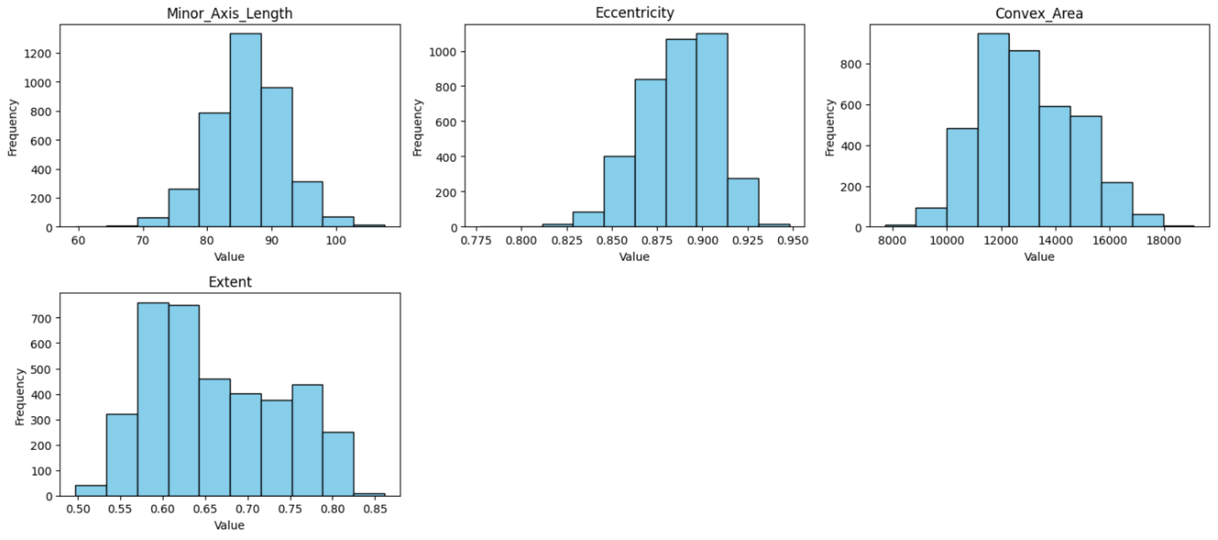
```
[155]: # Standardize the features
       scaler = StandardScaler()
       X_train_scaled = scaler.fit_transform(X_train)
       X_test_scaled = scaler.transform(X_test)
```

Data Visualisation

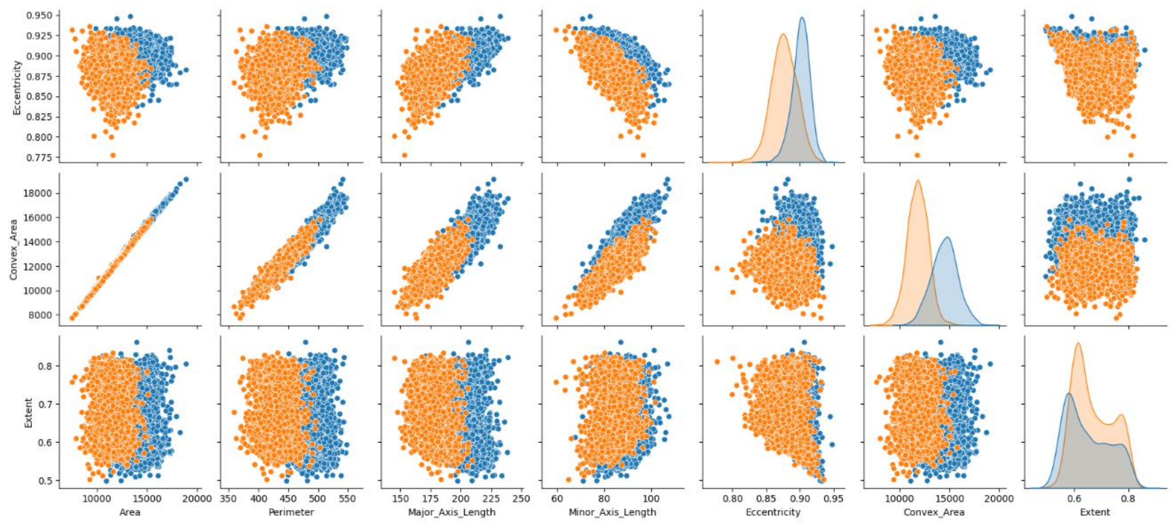
```
[156]: df = pd.DataFrame(data)

       # Value counts of the target variable
       class_counts = df['Class'].value_counts()
```

```
plt.figure(figsize=(8, 5))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.xlabel('Class', fontsize=15)
plt.ylabel('Count', fontsize=15)
plt.title('Distribution of Classes', fontsize=15)
plt.show()
```



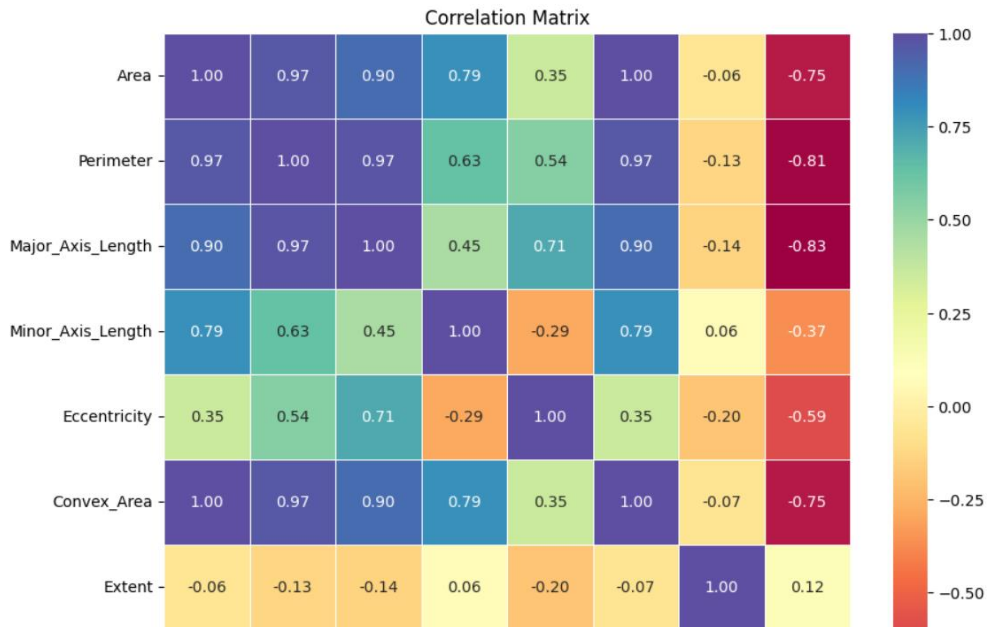


[158]:
```
# Generate pairplot
sns.pairplot(data, hue='Class')  # Exclude the target variable 'Class' for pairwise comparison
plt.show()
```



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

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

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



```
    metrics = [TP, TN, FP, FN, TPR, TNR, FPR, FNR, Precision, F1_measure, Accuracy, Error_rate, BACC, TSS, HSS]
    return metrics
```

[162]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score, brier_score_loss

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

    if LSTM_flag == 1:
        # Convert data to numpy array
        Xtrain, Xtest, ytrain, ytest = map(np.array, [X_train, X_test, y_train, y_test])
        # Reshape data
        shape = Xtrain.shape
        Xtrain_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])
        model_roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
        metrics.extend(calc_metrics(matrix))
```

KNN Classifier Algorithm Implementation

[167]:
```
#Define KNN parameters for grid search
knn_parameters = {"n_neighbors": [2, 3, 4, 6, 8, 10, 12, 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(X_train_scaled, y_train)
# Print the best parameters found by GridSearchCV
print("\nBest Parameters for KNN based on GridSearchCV: ", knn_cv.best_params_)
best_n_neighbors = knn_cv.best_params_['n_neighbors']
```

```
Best Parameters for KNN based on GridSearchCV:  {'n_neighbors': 15}
```

LSTM Model Implementation & Using 10 fold Cross Validation

[168]:
```
from sklearn.model_selection import StratifiedKFold

features = df.drop(columns=['Class'])  # Drop the target column to keep only the features
labels = df['Class']  # Target variable should be the 'Class' column


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

# 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']
```

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

# 10 Iterations of 10-fold cross-validation
for iter_num, (train_index, test_index) in enumerate(cv_stratified.split(features, labels), 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)
    # LSTM model
    lstm_model = Sequential()
    lstm_model.add(LSTM(64, activation='relu', return_sequences=False))
    lstm_model.add(Dense(1, activation='sigmoid'))
    # Compile model
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


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

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

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

    # Display metrics for all algorithms in each iteration
    print('\nIteration {}: \n'.format(iter_num))
```

```
Brier_score          0.10    0.06    0.08
AUC                  0.93    0.97    0.97
Acc_by_package_fn    0.87    0.92    0.90


24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Iteration 10:


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

                      KNN      RF    LSTM
TP                 408.00  411.00  422.00
TN                 253.00  288.00  256.00
FP                  73.00   38.00   70.00
FN                  28.00   25.00   14.00
TPR                  0.94    0.94    0.97
TNR                  0.78    0.88    0.79
FPR                  0.22    0.12    0.21
FNR                  0.06    0.06    0.03
Precision            0.85    0.92    0.86
F1_measure           0.89    0.93    0.91
Accuracy             0.87    0.92    0.89
Error_rate           0.13    0.08    0.11
BACC                 0.86    0.91    0.88
TSS                  0.71    0.83    0.75
HSS                  0.72    0.83    0.77
Brier_score          0.10    0.06    0.09
AUC                  0.93    0.97    0.96
Acc_by_package_fn    0.87    0.92    0.89
```

Performance Comparision between all models

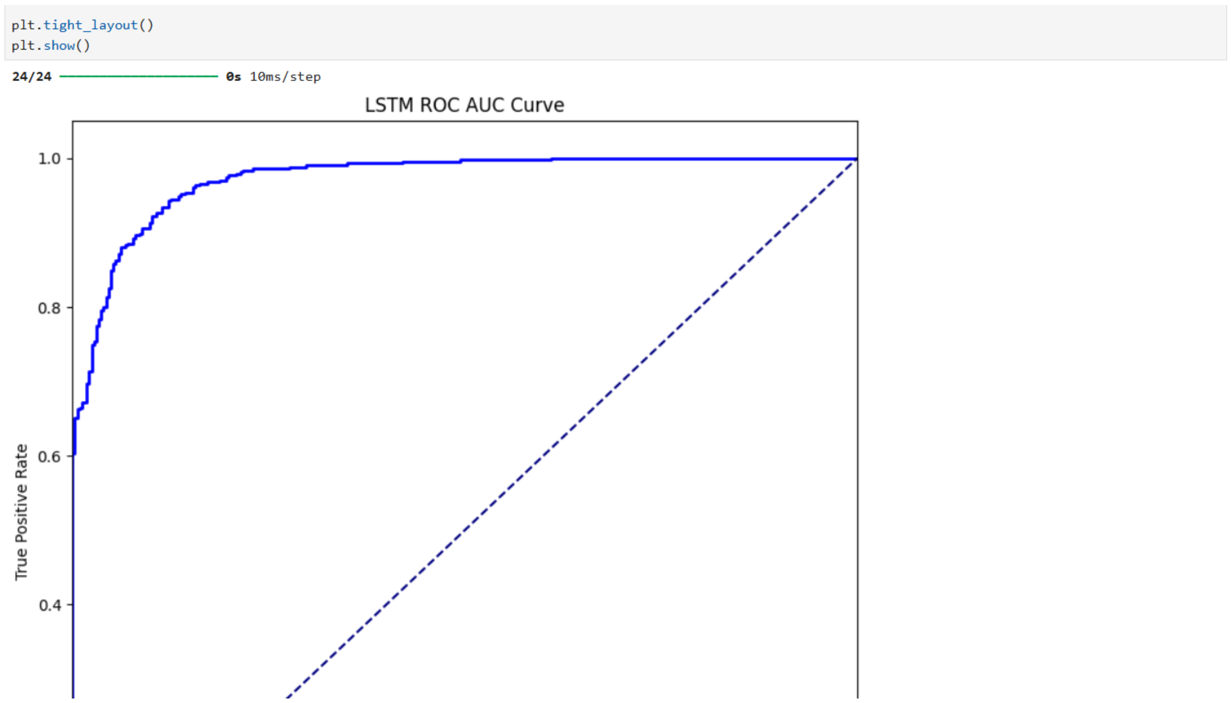
[171]:

```python
# Analyze the results to determine which algorithm performs better
discussion = ""

# Summarize performance for each algorithm
knn_summary = f"KNN has a high True Positive Rate (TPR: {avg_performance_df.loc['TPR', 'KNN']:.2f}) and balanced metrics, but its True Negative Rate (TNR
rf_summary = f"RF demonstrates the best overall performance, with the highest Accuracy ({avg_performance_df.loc['Accuracy', 'RF']:.2f}), TPR ({avg_perfor
lstm_summary = f"LSTM has lower metrics overall, with a TPR of {avg_performance_df.loc['TPR', 'LSTM']:.2f} and Accuracy of {avg_performance_df.loc['Accur

# Compare algorithms
comparison = f"""
1. KNN is simple and achieves reliable results with an Accuracy of {avg_performance_df.loc['Accuracy', 'KNN']:.2f} and F1 Score of {avg_performance_df.lo
2. RF is the best performer with the highest overall metrics, including AUC ({avg_performance_df.loc['AUC', 'RF']:.2f}) and Brier Score ({avg_performance_
3. LSTM, while promising, performs poorly here, with significantly lower Accuracy ({avg_performance_df.loc['Accuracy', 'LSTM']:.2f}) and higher Error Rat
"""

# Justification for RF's performance
justification = f"""
RF's superior performance can be attributed to its ability to combine the outputs of multiple decision trees, effectively capturing complex patterns in t
"""

# Compile discussion
discussion += f"{knn_summary}\n\n"
discussion += f"{rf_summary}\n\n"
discussion += f"{lstm_summary}\n\n"
discussion += "Comparison of Algorithms:"
discussion += comparison
discussion += "\nJustification for RF's Performance:"
discussion += justification

# Print discussion
print(discussion)
```

Visualization of The LSTM ROC Curve

[174]:

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

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

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

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


# Predict probabilities for test set
lstm_probs = lstm_model.predict(X_test).ravel()  # Assuming lstm_model is already trained

lstm_fpr, lstm_tpr, _ = roc_curve(y_test, lstm_probs)
lstm_roc_auc = auc(lstm_fpr, lstm_tpr)

# Plot ROC AUC curves
```



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

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

Other

The source code (.py file) and data sets (.csv files) will be attached to the zip file.

Link to Git Repository

https://github.com/kartikeyypatel/FinalTerm_DataMining_Project/