NAME: Kaviya Sree Ravikumar Meenakshi
NJIT UCID: kr549
Email Address: kr549@njit.edu
Date: 11.24.2024
Professor: Yasser Abduallah
Course: FA24 - CS634101 (Data Mining)

New Jersey Institute of Technology

FINAL PROJECT REPORT

A Comprehensive Analysis of Model Performance Metrics

TABLE OF CONTENTS

ABSTRACT

The objective of this study is to evaluate and compare the performance of four diverse classification algorithms — Naïve Bayes, Decision Tree, Random Forest, and Long Short-Term Memory (LSTM) networks — using a comprehensive set of performance metrics. The dataset was preprocessed and analyzed to address potential class imbalance and prepare it for modeling. A 10-fold cross-validation approach was employed to ensure robust evaluation of each algorithm's performance. The project emphasized understanding the trade-offs between these metrics and visualized key results. Random Forest and LSTM had the edge over other algorithms after evaluating the metrics on a wholistic approach. Thus, the findings highlight the importance of using metrics to choose the best algorithms for real-world tasks.

INTRODUCTION

Classification is a very significant task in machine learning, where the goal is to assign data into known categories. This project deals with binary classification (i.e. two target classes). It is widely applied in various fields, including healthcare, finance, and marketing. In this project, the focus is on using machine learning algorithms to classify breast cancer data into benign or malignant cases. A total of four algorithms are used to evaluate over 10 metrics—Naïve Bayes, Decision Tree, Random Forest and Long Short-Term Memory (LSTM) networks. Let's understand these algorithms a little better.

(1) Naïve Bayes

Naïve Bayes is a probabilistic algorithm based on Bayes' theorem, assuming the independence of features. Though simple, it works well for classification tasks and does really well when the independence assumption is approximately true. It is computationally efficient and often forms a baseline for classification problems.

(2) Decision Tree

A Decision Tree is a tree-structured classifier that splits data into branches based on feature values. Each internal node denotes a feature, each branch denotes a decision rule, and each leaf node denotes an outcome. Decision trees are interpretable, flexible, yet prone to overfitting on training data, especially without proper pruning.

(3) Random Forest

Random Forest is an ensemble learning algorithm that builds multiple Decision Trees during training and combines their outputs to improve classification performance. By averaging predictions over trees, it reduces overfitting and boosts generalization. Random Forest is generally robust and works well with high-dimensional data.

(4) Long Short-Term Memory (LSTM)

The LSTM is a special type of RNN constructed to deal with sequential data. Traditionally, LSTMs were used for problems involving time-series, while recently they have been applied to classification problems on structured data by treating the feature set as a sequence. LSTMs can really uncover long-term dependencies in data, which is very useful in difficult tasks, though computationally intensive, which is a big contrast with traditional models.

To ensure a comprehensive evaluation, we employ k-fold Cross-Validation with k = 10. It slices the dataset into ten equal parts, taking nine folds for training while testing one fold iteratively, making sure each data point gets a chance to be in the test set at least once. This approach reduces the risk of overfitting, provides a reliable estimate of model performance, and is especially valuable for datasets with limited samples. In this way, the evaluation results for each fold are averaged out into unbiased and robust metrics of performance.

Further details about the evaluation metrics computed in this project are provided in the next section.

CORE CONCEPTS AND PRINCIPLES

Although this project involves the application of various algorithms, the primary focus is on computing and understanding the evaluation metrics. Let us take a closer look at these metrics.

(1) No. of positive examples (P): The total number of positive class instances in the dataset.

(2) No. of negative examples (N): The total number of negative class instances in the dataset.

(3) True Positive Rate (TPR): It measures the proportion of correctly identified positive instances. TPR is also known as recall or sensitivity.

(4) True Negative Rate (TNR): The proportion of correctly identified negative instances. TNR is also known as specificity.

(5) False Positive Rate (FPR): The proportion of negative instances incorrectly classified as positive.

(6) False Negative Rate (FNR): The proportion of positive instances incorrectly classified as negative.

(7) Recall (Sensitivity): Equivalent to TPR, it measures the model's ability to correctly detect positive cases.

(8) Precision: The proportion of correctly identified positive instances out of all predicted positives.

(9) F1 Score: The harmonic mean of precision and recall, balancing both metrics.

(10) Accuracy: The overall correctness of the model, calculated as the ratio of correctly classified instances to the total instances.

(11) Error Rate: The complement of accuracy, representing the proportion of incorrectly classified instances.

(12) Balanced Accuracy: The average of TPR and TNR, useful in cases of class imbalance.

(13) True Skill Statistic (TSS): A measure of a classifier's performance that considers both TPR and TNR.

(14) Heidke Skill Score (HSS): A metric that compares the model's accuracy to random chance.

(15) Brier Score (BS): Measures the accuracy of probabilistic predictions, with lower scores indicating better calibration.

(16) Brier Skill Score (BSS): Compares the BS of the model to a baseline, with higher scores indicating better performance.

(17) Area Under Curve (AUC): The area under the ROC curve, representing the model's ability to distinguish between classes.

(18) ROC Curve: A graphical representation of the trade-off between True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds, used to evaluate model performance.

Interpretation of these metrics:

Table 1: Interpretation of Evaluation Metrics

Metric	**Interpretation**
True Positive Rate (TPR)	Better when closer to 1
True Negative Rate (TNR)	Better when closer to 1
False Positive Rate (FPR)	Lower is better, closer to 0
False Negative Rate (FNR)	Lower is better, closer to 0
Recall (Sensitivity)	Better when closer to 1
Precision	Better when closer to 1
F1 Score	Better when closer to 1
Accuracy	Better when closer to 1
Error Rate	Lower is better, closer to 0
Balanced Accuracy	Better when closer to 1
True Skill Statistic (TSS)	Better when closer to 1
Heidke Skill Score (HSS)	Better when closer to 1
Brier Score (BS)	Lower is better
Brier Skill Score (BSS)	Higher is better
Area Under Curve (AUC)	Better when closer to 1
ROC Curve	A curve closer to the top-left is better

PROJECT WORKFLOW

The project follows a very simple workflow in order to comparative different evaluation metrics across different algorithms. A structured overview is outlined below:

(1) Data Selection and Preprocessing

The breast cancer dataset from sklearn was selected for this project due to its suitability for binary classification tasks. Initial exploration of the dataset was conducted using .describe(), .info() and .head() to understand the data structure and characteristics. The class distribution of the target variable was visualized to detect class imbalance and feature scaling was applied to ensure the data was ready for modeling.

(2) Model Selection

Four classification algorithms were chosen for this study: Naive Bayes, Decision Tree, Random Forest and LSTM. The data was prepared to meet the specific requirements of each algorithm as it must be fairly and effectively implemented on the same dataset.

(3) Implementation of k-Fold Cross-Validation

To evaluate model performance, k-fold cross-validation was implemented with k=10, dividing the data into ten subsets for iterative training and validation. This method ensured robust performance evaluation.

(4) Evaluation Metric Computation

Same mentioned in the previous section, a wide range of evaluation metrics was computed for each algorithm, including the number of positive and negative examples, True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), Recall, Precision, F1 Score, Accuracy, and Error Rate. Additional metrics like Balanced Accuracy, True Skill Statistics (TSS), Heidke Skill Score (HSS), Brier Score (BS), Brier Skill Score (BSS), and Area Under the Curve (AUC) were also calculated. To visualize performance, ROC curves were plotted to demonstrate the trade-off between TPR and FPR.

(5) Result Analysis and Interpretation

The results for each model were analyzed based on the computed evaluation metrics, providing insights into the strengths and weaknesses of Naive Bayes, Decision Tree, Random Forest, and LSTM. The impact of class imbalance on model performance is also discussed in detail in the results and evaluation section.

DATASET DESCRIPTION

The dataset used for this project is the **Breast Cancer Dataset**, which can be found under the sklearn library. It is one of the popular datasets used for binary classification, comprising tumor features obtained from digitized images, where the target variable describes whether the tumors are malignant or benign. The dataset was **directly imported from the sklearn.datasets module**, no explicit loading of dataset from local machine was required. The dataset contained 30 numerical features; hence it was very apt for classification models, which use structured input data. This convenient integration allowed seamless preprocessing and analysis, saving time and ensuring consistency in data handling.

Here's how the dataset looks like:

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  mean compactness  mean concavity  mean concave points
mean symmetry  mean fractal dimension  radius error  texture error  \
0        17.99         10.38          122.80     1001.0          0.11840           0.27760          0.3001              0.14710
0.2419                 0.07871        1.0950        0.9053
1        20.57         17.77          132.90     1326.0          0.08474           0.07864          0.0869              0.07017
0.1812                 0.05667        0.5435        0.7339
2        19.69         21.25          130.00     1203.0          0.10960           0.15990          0.1974              0.12790
0.2069                 0.05999        0.7456        0.7869
3        11.42         20.38           77.58      386.1          0.14250           0.28390          0.2414              0.10520
0.2597                 0.09744        0.4956        1.1560
4        20.29         14.34          135.10     1297.0          0.10030           0.13280          0.1980              0.10430
0.1809                 0.05883        0.7572        0.7813

   perimeter error  area error  smoothness error  compactness error  concavity error  concave points error  symmetry error  fra
ctal dimension error  worst radius  worst texture  worst perimeter  \
0            8.589      153.40          0.006399            0.04904          0.05373               0.01587         0.03003
0.006193        25.38          17.33           184.60
1            3.398       74.08          0.005225            0.01308          0.01860               0.01340         0.01389
0.003532        24.99          23.41           158.80
2            4.585       94.03          0.006150            0.04006          0.03832               0.02058         0.02250
0.004571        23.57          25.53           152.50
3            3.445       27.23          0.009110            0.07458          0.05661               0.01867         0.05963
0.009208        14.91          26.50            98.87
4            5.438       94.44          0.011490            0.02461          0.05688               0.01885         0.01756
0.005115        22.54          16.67           152.20

   worst area  worst smoothness  worst compactness  worst concavity  worst concave points  worst symmetry  worst fractal dimens
ion  target
0      2019.0            0.1622             0.6656           0.7119                0.2654          0.4601                  0.11
890       0
1      1956.0            0.1238             0.1866           0.2416                0.1860          0.2750                  0.08
902       0
2      1709.0            0.1444             0.4245           0.4504                0.2430          0.3613                  0.08
758       0
3       567.7            0.2098             0.8663           0.6869                0.2575          0.6638                  0.17
300       0
4      1575.0            0.1374             0.2050           0.4000                0.1625          0.2364                  0.07
678       0
```

HOW TO EXECUTE THE CODE

(1) Perquisites

Although this program was coded using `Python 3.9.13`, this program can be run on systems with `Python 3.9` or higher. The program also requires the following basic libraries to be installed.

Table 2: Prerequisites

Library	Description	Command to install
numpy	A library used for numerical operations.	`pip install numpy`
pandas	A library used for data manipulation and analysis.	`pip install pandas`
matplotlib	A library used for sed for plotting and visualization.	`pip install matplotlib`
seaborn	A higher-level data visualization library based on Matplotlib.	`pip install seaborn`
scikit-learn	A library used for datasets, models and metrics.	`pip install scikit-learn`
keras	A library used for deep learning (with TensorFlow backend).	`pip install keras`
tensorflow	A backend library for keras, necessary for building and training LSTM models.	`pip install tensorflow`

(2) Set up the environment

Download either the .ipynb file or .py file from GitHub. There is no explicit dataset file as the dataset is used from python's built-in library. If you want to run .ipynb file use google colab or jupyter notebook. In case you prefer the .py file, it can be run using the command prompt. You will mostly not require any additional libraries as they might be already installed in your system. In case there is any error with the packages, please use Table 2 to install them.

CODE

Import the necessary libraries.

Import necessary libraries

In [17]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.metrics import brier_score_loss, roc_curve, auc
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Input, LSTM, Dense
from keras.optimizers import Adam
```

Function to compute all evaluation metrics

Metrics Computation Function

In [18]:
```python
def compute_metrics(TP, TN, FP, FN):

    # The number of positive examples
    P = TP + FN
    # The number of negative examples
    N = TN + FP
    # True positive rate
    TPR = TP / P if P != 0 else 0 # same as recall/sensitivity
    # True negative rate
    TNR = TN / N if N != 0 else 0
    # False positive rate
    FPR = FP / N if N != 0 else 0
    # False negative rate
    FNR = FN / P if P != 0 else 0
    # Recall or sensitivity
    Recall = TPR
    # Precision
    Precision = TP / (TP + FP) if (TP + FP) != 0 else 0
    # F1 Score
    F1 = 2 * (Precision * Recall) / (Precision + Recall) if (Precision + Recall) != 0 else 0
    # Accuracy
    Accuracy = (TP + TN) / (P + N) if (P + N) != 0 else 0
    # Error rate
    Error = (FP + FN) / (P + N) if (P + N) != 0 else 0
    # Balanced accuracy
    Balanced_Accuracy = (TPR + TNR) / 2
    # True skill statistics
    TSS = TPR - FPR
    # Heidke skill score
    HSS = (2 * (TP * TN - FP * FN)) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) if ((TP + FN) * (FN + TN) + (TP + FP) * (FP
```

```python
    # Store metrics in a dictionary
    metrics = {
        'No. of positive examples (P)': P,
        'No. of negative examples (N)': N,
        'True Positive Rate (TPR)': TPR,
        'True Negative Rate (TNR)': TNR,
        'False Positive Rate (FPR)': FPR,
        'False Negative Rate (FNR)': FNR,
        'Recall (Sensitivity)': Recall,
        'Precision': Precision,
        'F1 Score': F1,
        'Accuracy': Accuracy,
        'Error Rate': Error,
        'Balanced Accuracy': Balanced_Accuracy,
        'TSS': TSS,
        'HSS': HSS,
    }

    return metrics
```

```python
# Computing average metrics across different values of k
def average_metrics(metrics_list):
    average_metrics = {}
    for key in metrics_list[0].keys():
        average_metrics[key] = sum(d[key] for d in metrics_list) / len(metrics_list)
    return average_metrics
```

Function to plot ROC Curves (individually and combined)

ROC Curve Plot Function

In [19]:
```python
# Computing average ROC data
def avg_roc(fpr_list, tpr_list):

    all_fpr = np.concatenate(fpr_list)
    all_tpr = np.concatenate(tpr_list)
    sorted_fpr, sorted_tpr = zip(*sorted(zip(all_fpr, all_tpr)))
    mean_fpr = np.linspace(0, 1, 100)
    mean_tpr = np.interp(mean_fpr, sorted_fpr, sorted_tpr)
    mean_fpr = np.insert(mean_fpr, 0, 0)
    mean_tpr = np.insert(mean_tpr, 0, 0)

    return mean_fpr, mean_tpr

# Individual ROC curve plots
def plot_roc(fpr_list, tpr_list, auc_list, model_name):

    avg_fpr, avg_tpr = avg_roc(fpr_list, tpr_list)
    avg_auc = np.mean(auc_list)
    plt.plot(avg_fpr, avg_tpr, color = 'purple', label=f"{model_name} (AUC = {avg_auc:.3f})")
    plt.plot([0, 1], [0, 1], color='black', linestyle='--', label="Random Guess")
    plt.legend(loc="lower right")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve - {model_name}")
    plt.grid()
    plt.show()
```

```python
# Combined ROC curve plot
def plot_combined_roc(nb_fpr, nb_tpr, nb_auc, dt_fpr, dt_tpr, dt_auc, rf_fpr, rf_tpr, rf_auc, lstm_fpr, lstm_tpr, lstm_auc):

    plt.figure(figsize=(10, 8))

    # Naive Bayes
    nb_fpr_avg, nb_tpr_avg = avg_roc(nb_fpr, nb_tpr)
    plt.plot(nb_fpr_avg, nb_tpr_avg, label=f"Naive Bayes (AUC = {np.mean(nb_auc):.2f})", color="blue")

    # Decision Tree
    dt_fpr_avg, dt_tpr_avg = avg_roc(dt_fpr, dt_tpr)
    plt.plot(dt_fpr_avg, dt_tpr_avg, label=f"Decision Tree (AUC = {np.mean(dt_auc):.2f})", color="green")

    # Random Forest
    rf_fpr_avg, rf_tpr_avg = avg_roc(rf_fpr, rf_tpr)
    plt.plot(rf_fpr_avg, rf_tpr_avg, label=f"Random Forest (AUC = {np.mean(rf_auc):.2f})", color="orange")

    # LSTM
    lstm_fpr_avg, lstm_tpr_avg = avg_roc(lstm_fpr, lstm_tpr)
    plt.plot(lstm_fpr_avg, lstm_tpr_avg, label=f"LSTM (AUC = {np.mean(lstm_auc):.2f})", color="red")

    # Adding the random guess line
    plt.plot([0, 1], [0, 1], color='black', linestyle='--', label="Random Guess")

    plt.legend(loc="lower right")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Combined ROC Curve")
    plt.grid()
    plt.show()
```

Function to train and evaluate Naïve Bayes algorithm using k-fold cross validation (k = 10).

Model 1 - Naive Bayes

In [20]:
```
# Naive Bayes implementation
def evaluate_naive_bayes(X, y, kfolds=10):
    print("\n\n(1) Naive Bayes\n")

    kf = KFold(n_splits=kfolds, shuffle=True, random_state=42)
    metrics_list = []
    fpr_list, tpr_list, auc_list = [], [], []

    model = GaussianNB()

    # Train-test split
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Training the model
        model.fit(X_train, y_train)

        # Probabilities of positive class
        y_pred_prob = model.predict_proba(X_test)[:, 1]

        # Computing and storing ROC curve metrics
        fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        fpr_list.append(fpr)
        tpr_list.append(tpr)
        auc_list.append(roc_auc)
```

```
        # Prediction
        y_pred = model.predict(X_test)

        # Computation of BS and BSS
        bs = brier_score_loss(y_test, y_pred_prob)
        ref_prob = np.mean(y_test)
        bs_ref = brier_score_loss(y_test, np.full_like(y_test, ref_prob))
        bss = 1 - (bs / bs_ref)

        # Extracting TP, TN, FP, FN from Confusion matrix
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        metrics = compute_metrics(tp, tn, fp, fn)

        # Adding metrics to the list
        metrics['BS'] = bs
        metrics['BSS'] = bss
        metrics['AUC'] = roc_auc
        metrics_list.append(metrics)
```

```
        # Reducing column name lengths for formatting
        new_column_names = {
            "No. of positive examples (P)": "P",
            "No. of negative examples (N)": "N",
            "True Positive Rate (TPR)": "TPR",
            "True Negative Rate (TNR)": "TNR",
            "False Positive Rate (FPR)": "FPR",
            "False Negative Rate (FNR)": "FNR",
            "Recall (Sensitivity)": "Recall",
            "Precision": "Precision",
            "F1 Score": "F1 score",
            "Accuracy": "Accuracy",
            "Error Rate": "Error rate",
            "Balanced Accuracy": "BACC",
            "TSS": "TSS",
            "HSS": "HSS",
            "BS": "BS",
            "BSS": "BSS",
            "AUC": "AUC",
        }

    # Printing all metrics for each value of kfold
    metrics_df = pd.DataFrame(metrics_list)
    metrics_df = metrics_df.round(2)
    metrics_df.rename(columns=new_column_names, inplace=True)
    metrics_df.index = range(1, len(metrics_df) + 1)
    metrics_df.index.name = "k"
    pd.set_option("display.max_columns", None)
    pd.set_option("display.width", 200)
    pd.set_option("display.max_rows", None)
    print(metrics_df)

    return metrics_list, fpr_list, tpr_list, auc_list
```

Function to train and evaluate Decision Tree algorithm using k-fold cross validation (k = 10)

Model 2 - Decision Tree

In [30]:
```python
# Decision Tree Implementation
def evaluate_decision_tree(X, y, kfolds=10):
    print("\n\n(2) Decision Tree\n")

    kf = KFold(n_splits=kfolds, shuffle=True, random_state=42)
    metrics_list = []
    fpr_list, tpr_list, auc_list = [], [], []

    model = DecisionTreeClassifier(random_state=42)

    # Train-test split
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Training the model
        model.fit(X_train, y_train)

        # Probabilities of positive class
        y_pred_prob = model.predict_proba(X_test)[:, 1]

        # Computing and storing ROC curve metrics
        fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        fpr_list.append(fpr)
        tpr_list.append(tpr)
        auc_list.append(roc_auc)

        # Prediction
        y_pred = model.predict(X_test)
```

```python
        # Computation of BS and BSS
        bs = brier_score_loss(y_test, y_pred_prob)
        ref_prob = np.mean(y_test)
        bs_ref = brier_score_loss(y_test, np.full_like(y_test, ref_prob))
        bss = 1 - (bs / bs_ref)

        # Extracting TP, TN, FP, FN from Confusion matrix
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        metrics = compute_metrics(tp, tn, fp, fn)

        # Adding metrics to the list
        metrics['BS'] = bs
        metrics['BSS'] = bss
        metrics['AUC'] = roc_auc
        metrics_list.append(metrics)

        # Reducing column name lengths for formatting
        new_column_names = {
            "No. of positive examples (P)": "P",
            "No. of negative examples (N)": "N",
            "True Positive Rate (TPR)": "TPR",
            "True Negative Rate (TNR)": "TNR",
            "False Positive Rate (FPR)": "FPR",
            "False Negative Rate (FNR)": "FNR",
            "Recall (Sensitivity)": "Recall",
            "Precision": "Precision",
            "F1 Score": "F1 score",
            "Accuracy": "Accuracy",
            "Error Rate": "Error rate",
            "Balanced Accuracy": "BACC",
            "TSS": "TSS",
            "HSS": "HSS",
```

```python
            "BS": "BS",
            "BSS": "BSS",
            "AUC": "AUC",
        }

    # Printing all metrics for each value of kfold
    metrics_df = pd.DataFrame(metrics_list)
    metrics_df = metrics_df.round(2)
    metrics_df.rename(columns=new_column_names, inplace=True)
    metrics_df.index = range(1, len(metrics_df) + 1)
    metrics_df.index.name = "k"
    pd.set_option("display.max_columns", None)
    pd.set_option("display.width", 200)
    pd.set_option("display.max_rows", None)
    print(metrics_df)

    return metrics_list, fpr_list, tpr_list, auc_list
```

Function to train and evaluate Random Forest algorithm using k-fold cross validation (k = 10)

Model 3 - Random Forest

In [31]:
```
# Random Forest Implementation
def evaluate_random_forest(X, y, kfolds=10):
    print("\n\n(3) Random Forest\n")

    kf = KFold(n_splits=kfolds, shuffle=True, random_state=42)
    metrics_list = []
    fpr_list, tpr_list, auc_list = [], [], []
    model = RandomForestClassifier(random_state=42)

    # Train-test split
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Training the model
        model.fit(X_train, y_train)

        # Probabilities of positive class
        y_pred_prob = model.predict_proba(X_test)[:, 1]  # Probabilities for the positive class

        # Computing and storing ROC curve metrics
        fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        fpr_list.append(fpr)
        tpr_list.append(tpr)
        auc_list.append(roc_auc)

        # Prediction
        y_pred = model.predict(X_test)
        # Computation of BS and BSS
        bs = brier_score_loss(y_test, y_pred_prob)
        ref_prob = np.mean(y_test)
        bs_ref = brier_score_loss(y_test, np.full_like(y_test, ref_prob))
        bss = 1 - (bs / bs_ref)


        # Extracting TP, TN, FP, FN from Confusion matrix
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        metrics = compute_metrics(tp, tn, fp, fn)

        # Add metrics to the list
        metrics['BS'] = bs
        metrics['BSS'] = bss
        metrics['AUC'] = roc_auc
        metrics_list.append(metrics)

        # Reducing column name lengths for formatting
        new_column_names = {
            "No. of positive examples (P)": "P",
            "No. of negative examples (N)": "N",
            "True Positive Rate (TPR)": "TPR",
            "True Negative Rate (TNR)": "TNR",
            "False Positive Rate (FPR)": "FPR",
            "False Negative Rate (FNR)": "FNR",
            "Recall (Sensitivity)": "Recall",
            "Precision": "Precision",
            "F1 Score": "F1 score",
            "Accuracy": "Accuracy",
            "Error Rate": "Error rate",
            "Balanced Accuracy": "BACC",
            "TSS": "TSS",
            "HSS": "HSS",
            "BS": "BS",
            "BSS": "BSS",
            "AUC": "AUC",
        }

    # Printing all metrics for each value of kfold
    metrics_df = pd.DataFrame(metrics_list)
    metrics_df = metrics_df.round(2)
    metrics_df.rename(columns=new_column_names, inplace=True)
    metrics_df.index = range(1, len(metrics_df) + 1)
    metrics_df.index.name = "k"
    pd.set_option("display.max_columns", None)
    pd.set_option("display.width", 200)
    pd.set_option("display.max_rows", None)
    print(metrics_df)

    return metrics_list, fpr_list, tpr_list, auc_list
```

Function to train and evaluate LSTM algorithm using k-fold cross validation (k = 10). The LSTM model's architecture contains one input layer, 64 hidden layers (with ReLU activation function) and one output layer (with sigmoid activation function).

Model 4 - Long Short-Term Memory (LSTM)

In [32]:
```
# LSTM Model Implementation
def evaluate_lstm(X, y, kfolds=10):
    print("\n\n\n(4) LSTM")

    kf = KFold(n_splits=kfolds, shuffle=True, random_state=42)
    metrics_list = []
    fpr_list, tpr_list, auc_list = [], [], []

    print("\nLSTM Training Results: ")

    # Train-test split
    for i, (train_index, test_index) in enumerate(kf.split(X)):
        print(f"\n\t------- Fold k = {i + 1} -------")
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Reshaping the data for LSTM
        X_train_lstm = X_train.values.reshape((X_train.shape[0], 1, X_train.shape[1]))
        X_test_lstm = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))

        # Building the LSTM model
        model = Sequential()
        model.add(Input(shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
        model.add(LSTM(units=64, activation='relu'))
        model.add(Dense(units=1, activation='sigmoid'))
        model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

        # Train the model and print the loss and accuracy
        history = model.fit(X_train_lstm, y_train, epochs=10, batch_size=32, verbose=0)
        loss = history.history['loss']
        accuracy = history.history['accuracy']
        for i, (l, acc) in enumerate(zip(loss, accuracy)):
            print(f"Epoch {i+1}: Loss = {l:.4f}, Accuracy = {acc:.4f}")

        # Predicting probabilities
        y_pred_prob = model.predict(X_test_lstm)

        # Computing and storing ROC curve metrics
        fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
        roc_auc = auc(fpr, tpr)
        fpr_list.append(fpr)
        tpr_list.append(tpr)
        auc_list.append(roc_auc)

        # Prediction
        y_pred = (model.predict(X_test_lstm) > 0.5).astype(int)

        # Computation of BS and BSS
        bs = brier_score_loss(y_test, y_pred_prob)
        ref_prob = np.mean(y_test)
        bs_ref = brier_score_loss(y_test, np.full_like(y_test, ref_prob))
        bss = 1 - (bs / bs_ref)


        # Extracting TP, TN, FP, FN from Confusion matrix
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        metrics = compute_metrics(tp, tn, fp, fn)
```

```
        # Add metrics to the list
        metrics['BS'] = bs
        metrics['BSS'] = bss
        metrics['AUC'] = roc_auc  # Add AUC to the metrics dictionary
        metrics_list.append(metrics)

        # Reducing column name lengths for formatting
        new_column_names = {
            "No. of positive examples (P)": "P",
            "No. of negative examples (N)": "N",
            "True Positive Rate (TPR)": "TPR",
            "True Negative Rate (TNR)": "TNR",
            "False Positive Rate (FPR)": "FPR",
            "False Negative Rate (FNR)": "FNR",
            "Recall (Sensitivity)": "Recall",
            "Precision": "Precision",
            "F1 Score": "F1 score",
            "Accuracy": "Accuracy",
            "Error Rate": "Error rate",
            "Balanced Accuracy": "BACC",
            "TSS": "TSS",
            "HSS": "HSS",
            "BS": "BS",
            "BSS": "BSS",
            "AUC": "AUC",
        }
```

```
    # Printing all metrics for each value of kfold
    print("\n")
    metrics_df = pd.DataFrame(metrics_list)
    metrics_df = metrics_df.round(2)
    metrics_df.rename(columns=new_column_names, inplace=True)
    metrics_df.index = range(1, len(metrics_df) + 1)
    metrics_df.index.name = "k"
    pd.set_option("display.max_columns", None)
    pd.set_option("display.width", 200)
    pd.set_option("display.max_rows", None)
    print(metrics_df)

    return metrics_list, fpr_list, tpr_list, auc_list
```

This is the main section of the code where all the functions are called to perform their respective tasks.

Driver Function

```
In [33]: print("------------------------------------------------------------------------------------------------------------------------")
print("                      Final Project - A Comprehensive Analysis of Model Performance Metrics                     ")
print("------------------------------------------------------------------------------------------------------------------------\n")
print("\nTABLE OF CONTENTS:\n\n[I] Introduction\n[II] Data Preprocessing and Exploration\n[III] Evaluation metrics for all models
print("\n\n[I] INTRODUCTION\n")
print("I implemented four different classification algorithms on the Breast Cancer dataset—Naive Bayes, Decision Tree, Random For
print("\n\n[II] DATA PREPROCESSING AND EXPLORATION")

# Load the dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Dataset sample
print("\n--> Dataset Sample:\n")
df = pd.concat([X, y], axis=1)
print(df.head())

# General information about the dataset
print("\n--> Dataset General Information:\n")
print( X.info())

# Statistical summary of the dataset
print("\n\n--> Dataset Statistical Information:\n\n", X.describe())
```

```python
# Count of target labels
class_counts = y.value_counts()
print("\n\n--> Target Variable Split:\n", class_counts)


# Target Variable Distribution (Class Imbalance) Plot
print("\n\n--> Target Variable Distribution Plot")
plt.figure(figsize=(8, 6))
sns.barplot(x=class_counts.index, y=class_counts.values, palette="viridis")
plt.title("Target Variable Distribution (Class Imbalance)")
plt.xlabel("Class")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=["Class 0 (Malignant)", "Class 1 (Benign)"])
plt.show()
print("\nSince there is a class imbalance in the dataset, instead of relying only on accuracy, we need to focus on metrics like p

# Correlation matrix
print("\n\n--> Correlation Matrix")
fig, axis = plt.subplots(figsize=(15, 15))
correlation_matrix = X.corr()
sns.heatmap(correlation_matrix, annot=True, linewidths=.5, fmt='.2f', ax=axis)
plt.show()

print("\n\n\n[III] EVALUATION METRICS FOR ALL MODELS USING K-FOLD CROSS VALIDATION (K = 10)")
# Evaluating each model
naive_bayes_metrics, nb_fpr, nb_tpr, nb_auc = evaluate_naive_bayes(X_scaled, y)
decision_tree_metrics, dt_fpr, dt_tpr, dt_auc = evaluate_decision_tree(X_scaled, y)
random_forest_metrics, rf_fpr, rf_tpr, rf_auc = evaluate_random_forest(X_scaled, y)
lstm_metrics, lstm_fpr, lstm_tpr, lstm_auc = evaluate_lstm(X_scaled, y)
```

```python
# Calculating average metrics for all models
naive_bayes_avg_metrics = average_metrics(naive_bayes_metrics)
decision_tree_avg_metrics = average_metrics(decision_tree_metrics)
random_forest_avg_metrics = average_metrics(random_forest_metrics)
lstm_avg_metrics = average_metrics(lstm_metrics)

# Displaying summary for evaluation metrics across models
metrics_table = pd.DataFrame({
    "Naive Bayes": naive_bayes_avg_metrics,
    "Decision Tree": decision_tree_avg_metrics,
    "Random Forest": random_forest_avg_metrics,
    "LSTM": lstm_avg_metrics,
})

print("\n\n[IV] COMPARISON OF EVALUATION METRICS FOR ALL FOUR ALGORITHMS\n")
print(metrics_table.round(4).to_string())


# Plotting ROC for each model
print("\n\n[V] ROC CURVES FOR ALL MODELS\n")
print("(1) Naive Bayes")
plot_roc(nb_fpr, nb_tpr, nb_auc, "Naive Bayes")
print("(2) Decision Tree")
plot_roc(dt_fpr, dt_tpr, dt_auc, "Decision Tree")
print("(3) Random Forest")
plot_roc(rf_fpr, rf_tpr, rf_auc, "Random Forest")
print("(4) LSTM")
plot_roc(lstm_fpr, lstm_tpr, lstm_auc, "LSTM")
```

```python
# Plotting the combined ROC curve for all models
print("(5) Combined ROC Curve")
plot_combined_roc(nb_fpr, nb_tpr, nb_auc, dt_fpr, dt_tpr, dt_auc, rf_fpr, rf_tpr, rf_auc, lstm_fpr, lstm_tpr, lstm_auc)

print("\n\n\nThank you for running the program! Goodbye!\n")
print("------------------------------------------------------------------------------------------------------------------")
```

RESULTS AND EVALUATION

(1) Output screenshots

```
------------------------------------------------------------------------------------------------------------------------
                        Final Project - A Comprehensive Analysis of Model Performance Metrics
------------------------------------------------------------------------------------------------------------------------


TABLE OF CONTENTS:

[I] Introduction
[II] Data Preprocessing and Exploration
[III] Evaluation metrics for all models using KFold Cross Validation (k = 10)
[IV] Comparison of Evaluation Metrics for all four algorithms
[v] ROC Curves for all models


[I] INTRODUCTION

I implemented four different classification algorithms on the Breast Cancer dataset—Naive Bayes, Decision Tree, Random Forest,
and LSTM—to evaluate their performance metrics. This dataset predicts whether a tumor is malignant (1) or benign (0) based on 3
0 features.The metrics analyzed include the number of positive and negative examples, true positive rate (TPR), true negative r
ate (TNR), false positive rate (FPR), false negative rate (FNR), recall (sensitivity), precision, F1 score, accuracy, error rat
e, balanced accuracy, true skill statistic (TSS), Heidke skill score (HSS), Brier score, Brier skill score (BSS), area under th
e curve (AUC), and the ROC curve. The project utilizes k-fold cross-validation with k=10 to ensure robust evaluation.
```

```
[II] DATA PREPROCESSING AND EXPLORATION

--> Dataset Sample:

   mean radius  mean texture  mean perimeter  mean area  mean smoothness  mean compactness  mean concavity  mean concave points
mean symmetry  mean fractal dimension  radius error  texture error  \
0        17.99         10.38          122.80     1001.0          0.11840           0.27760          0.3001              0.14710
0.2419                 0.07871        1.0950         0.9053
1        20.57         17.77          132.90     1326.0          0.08474           0.07864          0.0869              0.07017
0.1812                 0.05667        0.5435         0.7339
2        19.69         21.25          130.00     1203.0          0.10960           0.15990          0.1974              0.12790
0.2069                 0.05999        0.7456         0.7869
3        11.42         20.38           77.58      386.1          0.14250           0.28390          0.2414              0.10520
0.2597                 0.09744        0.4956         1.1560
4        20.29         14.34          135.10     1297.0          0.10030           0.13280          0.1980              0.10430
0.1809                 0.05883        0.7572         0.7813

   perimeter error  area error  smoothness error  compactness error  concavity error  concave points error  symmetry error  fra
ctal dimension error  worst radius  worst texture  worst perimeter  \
0            8.589      153.40          0.006399            0.04904          0.05373               0.01587         0.03003
0.006193         25.38          17.33           184.60
1            3.398       74.08          0.005225            0.01308          0.01860               0.01340         0.01389
0.003532         24.99          23.41           158.80
2            4.585       94.03          0.006150            0.04006          0.03832               0.02058         0.02250
0.004571         23.57          25.53           152.50
3            3.445       27.23          0.009110            0.07458          0.05661               0.01867         0.05963
0.009208         14.91          26.50            98.87
4            5.438       94.44          0.011490            0.02461          0.05688               0.01885         0.01756
0.005115         22.54          16.67           152.20
```

```
   worst area  worst smoothness  worst compactness  worst concavity  worst concave points  worst symmetry  worst fractal dimens
ion  0
0      2019.0            0.1622             0.6656           0.7119                0.2654          0.4601                  0.11
890  0
1      1956.0            0.1238             0.1866           0.2416                0.1860          0.2750                  0.08
902  0
2      1709.0            0.1444             0.4245           0.4504                0.2430          0.3613                  0.08
758  0
3       567.7            0.2098             0.8663           0.6869                0.2575          0.6638                  0.17
300  0
4      1575.0            0.1374             0.2050           0.4000                0.1625          0.2364                  0.07
678  0
```

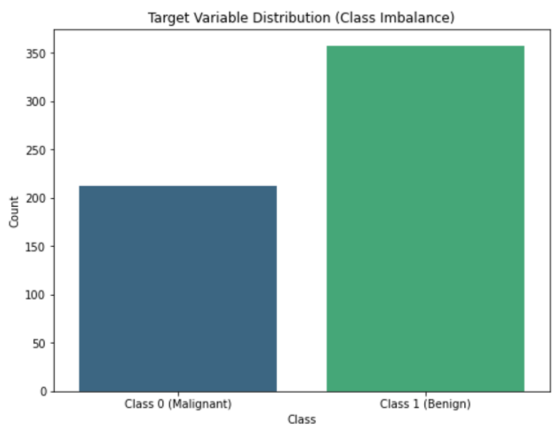
```
--> Dataset General Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
```

```
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
None


--> Dataset Statistical Information:

        mean radius  mean texture  mean perimeter    mean area  mean smoothness  mean compactness  mean concavity  mean concave
points  mean symmetry  mean fractal dimension  radius error  \
count   569.000000    569.000000      569.000000   569.000000       569.000000        569.000000      569.000000          569.
000000     569.000000              569.000000    569.000000
mean     14.127292     19.289649       91.969033   654.889104         0.096360          0.104341        0.088799            0.
048919       0.181162                0.062798      0.405172
std       3.524049      4.301036       24.298981   351.914129         0.014064          0.052813        0.079720            0.
038803       0.027414                0.007060      0.277313
min       6.981000      9.710000       43.790000   143.500000         0.052630          0.019380        0.000000            0.
000000       0.106000                0.049960      0.111500
25%      11.700000     16.170000       75.170000   420.300000         0.086370          0.064920        0.029560            0.
020310       0.161900                0.057700      0.232400
50%      13.370000     18.840000       86.240000   551.100000         0.095870          0.092630        0.061540            0.
033500       0.179200                0.061540      0.324200
75%      15.780000     21.800000      104.100000   782.700000         0.105300          0.130400        0.130700            0.
074000       0.195700                0.066120      0.478900
max      28.110000     39.280000      188.500000  2501.000000         0.163400          0.345400        0.426800            0.
201200       0.304000                0.097440      2.873000
```

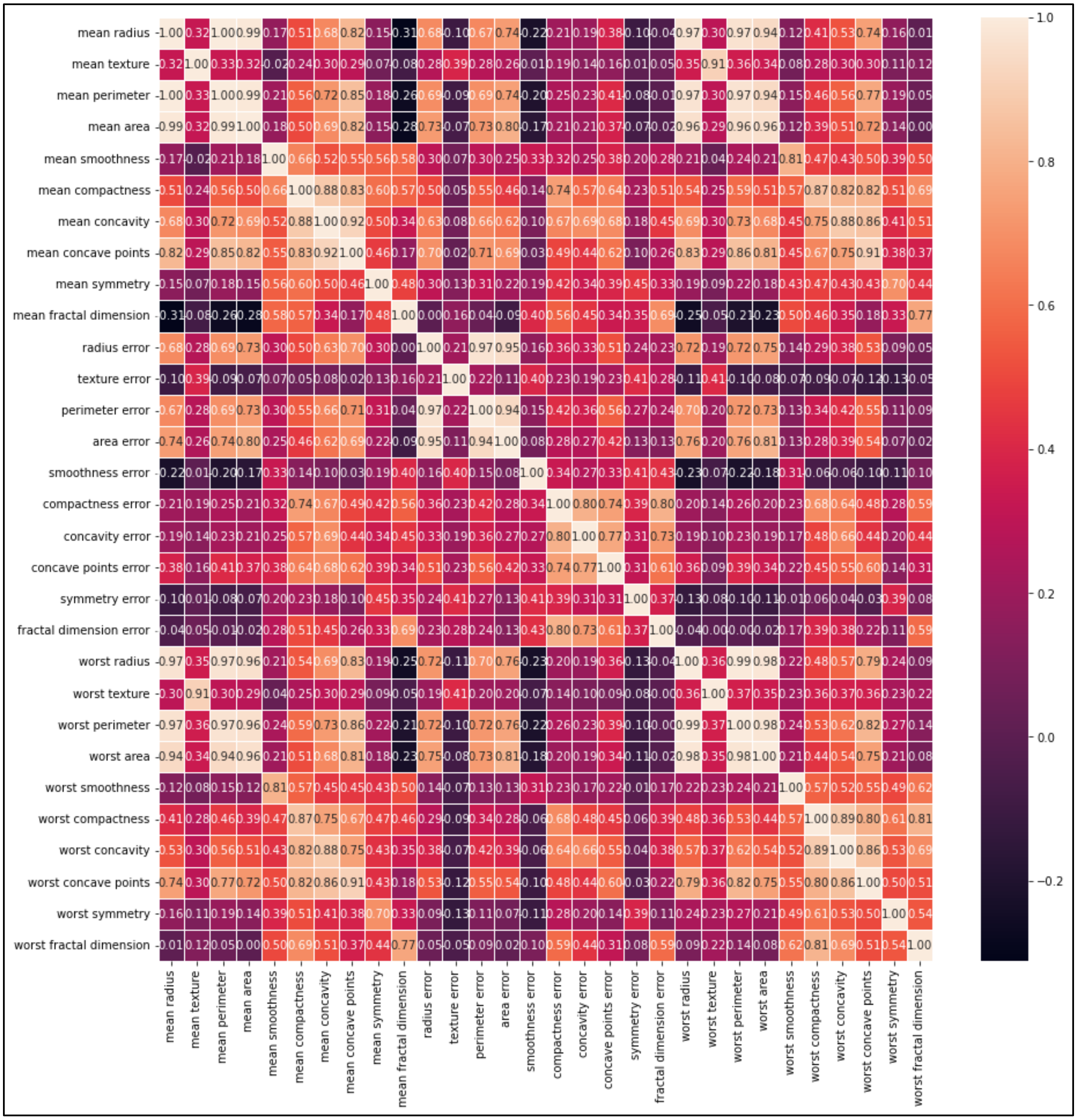
```
       texture error  perimeter error  area error  smoothness error  compactness error  concavity error  concave points error
symmetry error  fractal dimension error  worst radius  worst texture  \
count     569.000000       569.000000  569.000000        569.000000         569.000000       569.000000            569.000000
569.000000               569.000000    569.000000     569.000000
mean        1.216853         2.866059   40.337079          0.007041           0.025478         0.031894              0.011796
0.020542                 0.003795     16.269190      25.677223
std         0.551648         2.021855   45.491006          0.003003           0.017908         0.030186              0.006170
0.008266                 0.002646      4.833242       6.146258
min         0.360200         0.757000    6.802000          0.001713           0.002252         0.000000              0.000000
0.007882                 0.000895      7.930000      12.020000
25%         0.833900         1.606000   17.850000          0.005169           0.013080         0.015090              0.007638
0.015160                 0.002248     13.010000      21.080000
50%         1.108000         2.287000   24.530000          0.006380           0.020450         0.025890              0.010930
0.018730                 0.003187     14.970000      25.410000
75%         1.474000         3.357000   45.190000          0.008146           0.032450         0.042050              0.014710
0.023480                 0.004558     18.790000      29.720000
max         4.885000        21.980000  542.200000          0.031130           0.135400         0.396000              0.052790
0.078950                 0.029840     36.040000      49.540000
```

```
        worst perimeter   worst area  worst smoothness  worst compactness  worst concavity  worst concave points  worst symmetry  
worst fractal dimension  
count        569.000000   569.000000        569.000000         569.000000       569.000000            569.000000      569.000000  
569.000000  
mean         107.261213   880.583128          0.132369           0.254265         0.272188              0.114606        0.290076  
0.083946  
std           33.602542   569.356993          0.022832           0.157336         0.208624              0.065732        0.061867  
0.018061  
min           50.410000   185.200000          0.071170           0.027290         0.000000              0.000000        0.156500  
0.055040  
25%           84.110000   515.300000          0.116600           0.147200         0.114500              0.064930        0.250400  
0.071460  
50%           97.660000   686.500000          0.131300           0.211900         0.226700              0.099930        0.282200  
0.080040  
75%          125.400000  1084.000000          0.146000           0.339100         0.382900              0.161400        0.317900  
0.092080  
max          251.200000  4254.000000          0.222600           1.058000         1.252000              0.291000        0.663800  
0.207500  


--> Target Variable Split:
 1    357
0    212
dtype: int64
```

```
--> Target Variable Distribution Plot
```




```
Since there is a class imbalance in the dataset, instead of relying only on accuracy, we need to focus on metrics like precisio
n, recall, F1-score and AUC-ROC


--> Correlation Matrix
```


mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

[III] EVALUATION METRICS FOR ALL MODELS USING K-FOLD CROSS VALIDATION (K = 10)

(1) Naive Bayes

k	P	N	TPR	TNR	FPR	FNR	Recall	Precision	F1 score	Accuracy	Error rate	BACC	TSS	HSS	BS	BSS	AUC
1	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.04	0.95	0.99
2	31	26	1.00	0.92	0.08	0.00	1.00	0.94	0.97	0.96	0.04	0.96	0.92	0.93	0.03	0.94	1.00
3	37	20	0.89	0.85	0.15	0.11	0.89	0.92	0.90	0.88	0.12	0.87	0.74	0.73	0.12	0.81	0.98
4	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.04	0.95	0.99
5	39	18	0.95	0.89	0.11	0.05	0.95	0.95	0.95	0.93	0.07	0.92	0.84	0.84	0.07	0.90	0.99
6	32	25	0.94	0.96	0.04	0.06	0.94	0.97	0.95	0.95	0.05	0.95	0.90	0.89	0.05	0.91	0.99
7	40	17	0.88	0.88	0.12	0.12	0.88	0.95	0.91	0.88	0.12	0.88	0.76	0.72	0.11	0.84	0.98
8	31	26	1.00	0.85	0.15	0.00	1.00	0.89	0.94	0.93	0.07	0.92	0.85	0.86	0.06	0.89	1.00
9	30	27	0.97	0.93	0.07	0.03	0.97	0.94	0.95	0.95	0.05	0.95	0.89	0.89	0.05	0.90	0.97
10	37	19	0.95	0.74	0.26	0.05	0.95	0.88	0.91	0.88	0.12	0.84	0.68	0.71	0.10	0.85	0.97

(2) Decision Tree

k	P	N	TPR	TNR	FPR	FNR	Recall	Precision	F1 score	Accuracy	Error rate	BACC	TSS	HSS	BS	BSS	AUC
1	40	17	0.95	0.88	0.12	0.05	0.95	0.95	0.95	0.93	0.07	0.92	0.83	0.83	0.07	0.90	0.92
2	31	26	0.97	0.96	0.04	0.03	0.97	0.97	0.97	0.96	0.04	0.96	0.93	0.93	0.04	0.94	0.96
3	37	20	0.92	0.90	0.10	0.08	0.92	0.94	0.93	0.91	0.09	0.91	0.82	0.81	0.09	0.86	0.91
4	40	17	0.92	0.82	0.18	0.08	0.92	0.92	0.92	0.89	0.11	0.87	0.75	0.75	0.11	0.85	0.87
5	39	18	0.97	0.83	0.17	0.03	0.97	0.93	0.95	0.93	0.07	0.90	0.81	0.83	0.07	0.90	0.90
6	32	25	0.84	0.88	0.12	0.16	0.84	0.90	0.87	0.86	0.14	0.86	0.72	0.72	0.14	0.75	0.86
7	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.04	0.95	0.96
8	31	26	0.97	1.00	0.00	0.03	0.97	1.00	0.98	0.98	0.02	0.98	0.97	0.96	0.02	0.97	0.98
9	30	27	0.97	0.93	0.07	0.03	0.97	0.94	0.95	0.95	0.05	0.95	0.89	0.89	0.05	0.90	0.95
10	37	19	0.97	0.95	0.05	0.03	0.97	0.97	0.97	0.96	0.04	0.96	0.92	0.92	0.04	0.95	0.96

(3) Random Forest

k	P	N	TPR	TNR	FPR	FNR	Recall	Precision	F1 score	Accuracy	Error rate	BACC	TSS	HSS	BS	BSS	AUC
1	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.03	0.96	0.99
2	31	26	1.00	0.92	0.08	0.00	1.00	0.94	0.97	0.96	0.04	0.96	0.92	0.93	0.03	0.95	1.00
3	37	20	1.00	0.95	0.05	0.00	1.00	0.97	0.99	0.98	0.02	0.98	0.95	0.96	0.02	0.97	1.00
4	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.02	0.97	1.00
5	39	18	0.97	0.94	0.06	0.03	0.97	0.97	0.97	0.96	0.04	0.96	0.92	0.92	0.03	0.95	1.00
6	32	25	0.97	0.92	0.08	0.03	0.97	0.94	0.95	0.95	0.05	0.94	0.89	0.89	0.05	0.92	0.98
7	40	17	0.98	0.94	0.06	0.02	0.98	0.98	0.98	0.96	0.04	0.96	0.92	0.92	0.03	0.96	1.00
8	31	26	0.97	0.92	0.08	0.03	0.97	0.94	0.95	0.95	0.05	0.95	0.89	0.89	0.03	0.95	1.00
9	30	27	1.00	0.93	0.07	0.00	1.00	0.94	0.97	0.96	0.04	0.96	0.93	0.93	0.04	0.92	0.98
10	37	19	0.97	0.95	0.05	0.03	0.97	0.97	0.97	0.96	0.04	0.96	0.92	0.92	0.04	0.94	0.99

(4) LSTM

LSTM Training Results:

```
        ------- Fold k = 1 -------
Epoch 1: Loss = 0.6273, Accuracy = 0.8262
Epoch 2: Loss = 0.4740, Accuracy = 0.9336
Epoch 3: Loss = 0.3425, Accuracy = 0.9434
Epoch 4: Loss = 0.2494, Accuracy = 0.9512
Epoch 5: Loss = 0.1945, Accuracy = 0.9531
Epoch 6: Loss = 0.1629, Accuracy = 0.9590
Epoch 7: Loss = 0.1416, Accuracy = 0.9629
Epoch 8: Loss = 0.1253, Accuracy = 0.9688
Epoch 9: Loss = 0.1143, Accuracy = 0.9727
Epoch 10: Loss = 0.1050, Accuracy = 0.9785
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 0s/step

        ------- Fold k = 2 -------
Epoch 1: Loss = 0.5676, Accuracy = 0.8867
Epoch 2: Loss = 0.4046, Accuracy = 0.9395
Epoch 3: Loss = 0.2885, Accuracy = 0.9355
Epoch 4: Loss = 0.2193, Accuracy = 0.9414
Epoch 5: Loss = 0.1785, Accuracy = 0.9453
Epoch 6: Loss = 0.1525, Accuracy = 0.9570
Epoch 7: Loss = 0.1346, Accuracy = 0.9609
Epoch 8: Loss = 0.1211, Accuracy = 0.9688
Epoch 9: Loss = 0.1106, Accuracy = 0.9688
Epoch 10: Loss = 0.1025, Accuracy = 0.9707
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 0s/step
```

```
        ------- Fold k = 3 -------
Epoch 1: Loss = 0.6002, Accuracy = 0.8945
Epoch 2: Loss = 0.4483, Accuracy = 0.9395
Epoch 3: Loss = 0.3208, Accuracy = 0.9453
Epoch 4: Loss = 0.2350, Accuracy = 0.9512
Epoch 5: Loss = 0.1865, Accuracy = 0.9531
Epoch 6: Loss = 0.1575, Accuracy = 0.9531
Epoch 7: Loss = 0.1379, Accuracy = 0.9590
Epoch 8: Loss = 0.1244, Accuracy = 0.9629
Epoch 9: Loss = 0.1144, Accuracy = 0.9707
Epoch 10: Loss = 0.1053, Accuracy = 0.9707
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step

        ------- Fold k = 4 -------
Epoch 1: Loss = 0.6235, Accuracy = 0.8555
Epoch 2: Loss = 0.4750, Accuracy = 0.9395
Epoch 3: Loss = 0.3411, Accuracy = 0.9512
Epoch 4: Loss = 0.2468, Accuracy = 0.9512
Epoch 5: Loss = 0.1904, Accuracy = 0.9570
Epoch 6: Loss = 0.1571, Accuracy = 0.9590
Epoch 7: Loss = 0.1359, Accuracy = 0.9648
Epoch 8: Loss = 0.1218, Accuracy = 0.9668
Epoch 9: Loss = 0.1107, Accuracy = 0.9727
Epoch 10: Loss = 0.1026, Accuracy = 0.9785
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
```

```
        ------- Fold k = 5 -------
Epoch 1: Loss = 0.5901, Accuracy = 0.9180
Epoch 2: Loss = 0.4293, Accuracy = 0.9297
Epoch 3: Loss = 0.3030, Accuracy = 0.9336
Epoch 4: Loss = 0.2269, Accuracy = 0.9395
Epoch 5: Loss = 0.1809, Accuracy = 0.9453
Epoch 6: Loss = 0.1538, Accuracy = 0.9512
Epoch 7: Loss = 0.1338, Accuracy = 0.9551
Epoch 8: Loss = 0.1200, Accuracy = 0.9629
Epoch 9: Loss = 0.1092, Accuracy = 0.9668
Epoch 10: Loss = 0.1000, Accuracy = 0.9707
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step

        ------- Fold k = 6 -------
Epoch 1: Loss = 0.6246, Accuracy = 0.7891
Epoch 2: Loss = 0.4564, Accuracy = 0.9199
Epoch 3: Loss = 0.3228, Accuracy = 0.9277
Epoch 4: Loss = 0.2346, Accuracy = 0.9414
Epoch 5: Loss = 0.1846, Accuracy = 0.9473
Epoch 6: Loss = 0.1531, Accuracy = 0.9492
Epoch 7: Loss = 0.1330, Accuracy = 0.9570
Epoch 8: Loss = 0.1189, Accuracy = 0.9629
Epoch 9: Loss = 0.1077, Accuracy = 0.9668
Epoch 10: Loss = 0.0986, Accuracy = 0.9707
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 1000us/step
```

```
        ------- Fold k = 7 -------
Epoch 1: Loss = 0.6001, Accuracy = 0.8535
Epoch 2: Loss = 0.4387, Accuracy = 0.9355
Epoch 3: Loss = 0.3127, Accuracy = 0.9395
Epoch 4: Loss = 0.2291, Accuracy = 0.9434
Epoch 5: Loss = 0.1822, Accuracy = 0.9512
Epoch 6: Loss = 0.1535, Accuracy = 0.9570
Epoch 7: Loss = 0.1346, Accuracy = 0.9629
Epoch 8: Loss = 0.1210, Accuracy = 0.9688
Epoch 9: Loss = 0.1109, Accuracy = 0.9727
Epoch 10: Loss = 0.1029, Accuracy = 0.9746

2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 2ms/step

        ------- Fold k = 8 -------
Epoch 1: Loss = 0.6098, Accuracy = 0.8926
Epoch 2: Loss = 0.4583, Accuracy = 0.9473
Epoch 3: Loss = 0.3285, Accuracy = 0.9453
Epoch 4: Loss = 0.2381, Accuracy = 0.9473
Epoch 5: Loss = 0.1851, Accuracy = 0.9512
Epoch 6: Loss = 0.1531, Accuracy = 0.9531
Epoch 7: Loss = 0.1334, Accuracy = 0.9551
Epoch 8: Loss = 0.1194, Accuracy = 0.9648
Epoch 9: Loss = 0.1094, Accuracy = 0.9785
Epoch 10: Loss = 0.1011, Accuracy = 0.9785
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
```

```
        ------- Fold k = 9 -------
Epoch 1: Loss = 0.6612, Accuracy = 0.6562
Epoch 2: Loss = 0.5050, Accuracy = 0.9531
Epoch 3: Loss = 0.3670, Accuracy = 0.9453
Epoch 4: Loss = 0.2625, Accuracy = 0.9473
Epoch 5: Loss = 0.1984, Accuracy = 0.9551
Epoch 6: Loss = 0.1601, Accuracy = 0.9590
Epoch 7: Loss = 0.1367, Accuracy = 0.9609
Epoch 8: Loss = 0.1203, Accuracy = 0.9648
Epoch 9: Loss = 0.1078, Accuracy = 0.9746
Epoch 10: Loss = 0.0986, Accuracy = 0.9746
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 2ms/step

        ------- Fold k = 10 -------
Epoch 1: Loss = 0.6209, Accuracy = 0.8304
Epoch 2: Loss = 0.4605, Accuracy = 0.9220
Epoch 3: Loss = 0.3315, Accuracy = 0.9318
Epoch 4: Loss = 0.2480, Accuracy = 0.9415
Epoch 5: Loss = 0.1971, Accuracy = 0.9474
Epoch 6: Loss = 0.1653, Accuracy = 0.9571
Epoch 7: Loss = 0.1450, Accuracy = 0.9591
Epoch 8: Loss = 0.1291, Accuracy = 0.9610
Epoch 9: Loss = 0.1171, Accuracy = 0.9688
Epoch 10: Loss = 0.1078, Accuracy = 0.9708
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
```

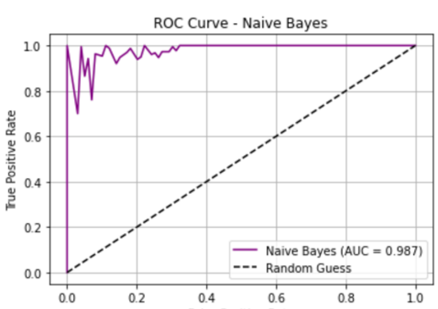
k	P	N	TPR	TNR	FPR	FNR	Recall	Precision	F1 score	Accuracy	Error rate	BACC	TSS	HSS	BS	BSS	AUC
1	40	17	0.95	0.94	0.06	0.05	0.95	0.97	0.96	0.95	0.05	0.95	0.89	0.88	0.03	0.96	0.99
2	31	26	1.00	1.00	0.00	0.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.02	0.97	1.00
3	37	20	0.97	1.00	0.00	0.03	0.97	1.00	0.99	0.98	0.02	0.99	0.97	0.96	0.02	0.97	1.00
4	40	17	0.98	1.00	0.00	0.02	0.98	1.00	0.99	0.98	0.02	0.99	0.98	0.96	0.02	0.97	1.00
5	39	18	0.97	0.94	0.06	0.03	0.97	0.97	0.97	0.96	0.04	0.96	0.92	0.92	0.03	0.96	0.99
6	32	25	1.00	0.92	0.08	0.00	1.00	0.94	0.97	0.96	0.04	0.96	0.92	0.93	0.04	0.93	0.98
7	40	17	1.00	0.94	0.06	0.00	1.00	0.98	0.99	0.98	0.02	0.97	0.94	0.96	0.02	0.98	1.00
8	31	26	1.00	0.88	0.12	0.00	1.00	0.91	0.95	0.95	0.05	0.94	0.88	0.89	0.03	0.95	1.00
9	30	27	0.97	0.93	0.07	0.03	0.97	0.94	0.95	0.95	0.05	0.95	0.89	0.89	0.04	0.92	0.98
10	37	19	0.97	0.89	0.11	0.03	0.97	0.95	0.96	0.95	0.05	0.93	0.87	0.88	0.03	0.95	0.99

[IV] COMPARISON OF EVALUATION METRICS FOR ALL FOUR ALGORITHMS

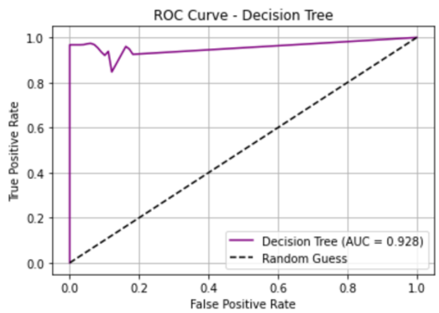
	Naive Bayes	Decision Tree	Random Forest	LSTM
No. of positive examples (P)	35.7000	35.7000	35.7000	35.7000
No. of negative examples (N)	21.2000	21.2000	21.2000	21.2000
True Positive Rate (TPR)	0.9516	0.9462	0.9809	0.9812
True Negative Rate (TNR)	0.8896	0.9095	0.9357	0.9452
False Positive Rate (FPR)	0.1104	0.0905	0.0643	0.0548
False Negative Rate (FNR)	0.0484	0.0538	0.0191	0.0188
Recall (Sensitivity)	0.9516	0.9462	0.9809	0.9812
Precision	0.9365	0.9497	0.9600	0.9660
F1 Score	0.9432	0.9478	0.9702	0.9732
Accuracy	0.9279	0.9350	0.9631	0.9666
Error Rate	0.0721	0.0650	0.0369	0.0334
Balanced Accuracy	0.9206	0.9279	0.9583	0.9632
TSS	0.8411	0.8557	0.9166	0.9264
HSS	0.8408	0.8565	0.9193	0.9267
BS	0.0672	0.0650	0.0316	0.0283
BSS	0.8935	0.8961	0.9483	0.9536
AUC	0.9867	0.9279	0.9927	0.9936

[V] ROC CURVES FOR ALL MODELS

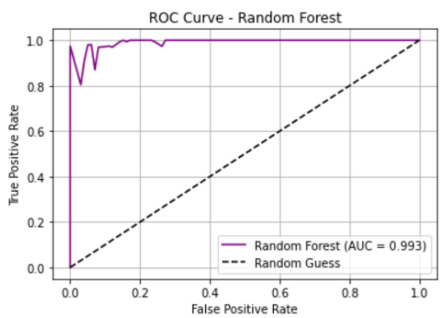
(1) Naive Bayes



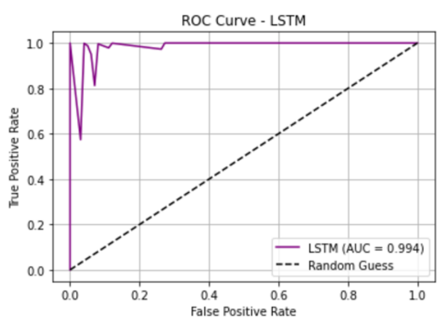
(2) Decision Tree



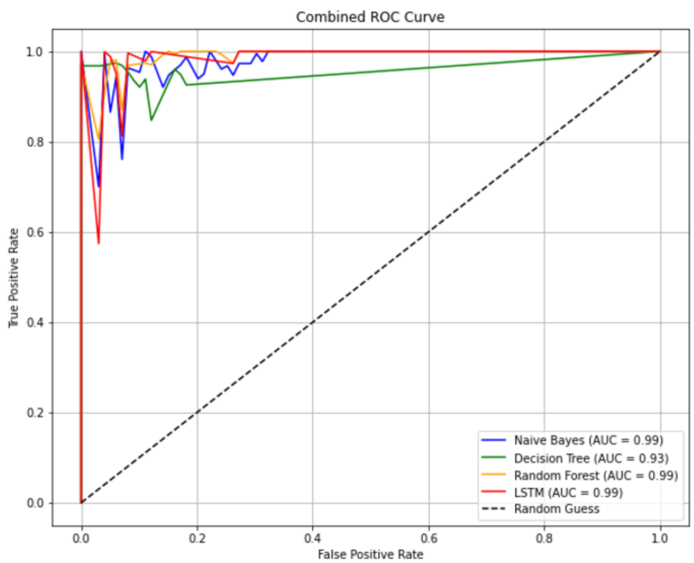
(3) Random Forest



(4) LSTM



(5) Combined ROC Curve



Thank you for running the program! Goodbye!

(2) Inference

The results from the four classification algorithms—Naive Bayes, Decision Tree, Random Forest, and LSTM—demonstrate varying levels of performance across multiple evaluation metrics. **Since there is some imbalance in the target variable, the decision of best algorithm cannot be determined by looking at accuracy or any other one evaluation metric. Thus, it is important to look at the results with a wholistic approach.**

All models show high True Positive Rates (TPR) and Recall values, ranging from 0.9462 (Decision Tree) to 0.9812 (LSTM), indicating their strong ability to correctly identify positive cases. The Random Forest and LSTM models excel in terms of precision, F1 score, and accuracy, with the Random Forest achieving the highest F1 score (0.9702) and accuracy (0.9631). In contrast, the Naive Bayes model, while still strong, shows slightly lower precision (0.9365) and accuracy (0.9279).

The models also exhibit low error rates, with the LSTM achieving the lowest error rate (0.0334). When evaluating class imbalance, all models show good balanced accuracy, with Random Forest and LSTM performing the best, reflecting their robustness in handling both positive and negative examples. The Area Under the Curve (AUC) is highest for Random Forest (0.9927) and LSTM

(0.9936), highlighting their superior ability to discriminate between classes. The ROC curve demonstrates that all models have high True Positive Rates (TPR) with relatively low False Positive Rates (FPR), indicating good performance in distinguishing between the classes. The Random Forest and LSTM models have the highest AUC values (0.99), suggesting superior classification ability, while the Decision Tree model, with an AUC of 0.93, performs slightly lower in comparison. Overall, **LSTM and Random Forest emerge as the top performers**, offering a good trade-off between sensitivity, specificity, overall accuracy and other parameters.

CONCLUSION

In conclusion, this project effectively evaluated the performance of four different classification algorithms—Naive Bayes, Decision Tree, Random Forest and LSTM—on the breast cancer dataset. The results showed that all models performed well in terms of accuracy, precision, recall, and other evaluation metrics, with Random Forest and LSTM demonstrating the highest performance. The use of k-fold cross-validation ensured a fair comparison between models. Through this project, I was able to understand the different metrics and how to interpret them.

SOURCE CODE AND REPOSITORY

The source code .py file and Jupyter Notebook .ipynb file are attached to the zip file.

Link to GitHub repository:
https://github.com/kr549/ravikumarmeenakshi_kaviyasree_finaltermproj.git