NAME: Emily Pham
NJIT UCID: etp6
Email Address: etp6@njit.edu
April 6th, 2025
Professor: Yasser Abduallah
CS 634-854 Data Mining

CS634 Midterm Project Report

# Multimodal Classification Using Random Forest, K-Nearest Neighbor, and Long Short-Term Memory Algorithms on Breast Cancer Dataset

---

## ABSTRACT

This project implements and evaluates the individual performance of three different classification algorithms–Random Forest, K-Nearest Neighbor, and Long Short-Term Memory--using the *Breast Cancer Wisconsin (Diagnostic)* dataset from the UCI Machine Learning Repository. The objective of this project is to evaluate and compare performance between each of the three algorithms. All three algorithms utilize 10-fold cross-validation, and each fold's metrics (Accuracy, Precision, Recall, F1, TSS, and HSS) are recorded and averaged. Results are saved in tabular format and presented for user visualization in a final summary.

*Implementation and Code Usage*

---

## INTRODUCTION

Classification is a fundamental task in the field of data mining. It refers to the process of organizing data into categories based upon various factors such as file type, contents, and other metadata. In this project, three different models are used to classify tumors as either malignant or benign using the *Breast Cancer Wisconsin (Diagnostic)* dataset. The three machine learning algorithms used in this project are as follows:

1. **Random Forest** is an ensemble decision tree-based model that is primarily used for making predictions. It is known for its improved accuracy and reduced overfitting.
2. **K-Nearest Neighbor (KNN)** is a distance-based model that classifies data points based on proximity to other existing and labeled data points.
3. **Long Short-Term Memory (LSTM)** is a type of recurrent neural network (RNN) that excels at classifying sequential data, such as test or time series, through its ability to remember and retain information over long periods of time

The goal of this project is to observe how these varied algorithms perform on the same dataset using identical evaluation conditions. All metrics are manually computed from TP, TN, FP, and FN values derived from confusion matrices.

## CORE CONCEPTS AND PRINCIPLES

**Positive vs. Negative Classification**

In binary classification, predictions are made in terms of positive and negative classes. This can be thought of as identifying whether something is "true" or "false," or "present" or "absent".

- **Positive Class**: The outcome or characteristic of interest, typically representing the event/target that the user is aiming to detect (e.g., malignant tumor).
- **Negative Class**: The absence of the characteristic or outcome of interest, or the opposite of the positive class (e.g., benign tumor).

Model predictions are compared against actual values to determine whether each instance falls into TP, TN, FP, or FN.

**Confusion Matrix**

The **confusion matrix** (also known as the error matrix) is a summary table that is used to evaluate the performance of a classification algorithm. It compares the actual target values with those predicted by the model. Four essential values were extracted from each confusion matrix for use in the computation of all metrics, as follows:

- **True Positive (TP)**: The proportion of cases correctly predicted as positive (e.g., malignant tumors correctly identified).
- **True Negative (TN)**: The proportion of cases correctly predicted as negative (e.g., benign tumors correctly identified.)
- **False Positive (FP)**: The proportion of negative cases incorrectly predicted as positive (e.g., benign tumors wrongly classified as malignant).
- **False Negative (FN)**: The proportion of positive cases incorrectly predicted as negative (e.g., malignant tumors missed as benign).

**Metric Calculation**

Each model manually computes the following metrics using only TP, TN, FP, and FN:

**Basic Performance Metrics:**

- **Accuracy (Acc)**: Proportion of all predictions, positive or negative, that were correct.

$$Acc = \frac{(TP+TN)}{(TP+FP+FN+TN)} = \frac{(TP+TN)}{(P+N)}$$

- **Error Rate (Err)**: Proportion of all predictions, positive or negative, that were incorrect.

$$Err = \frac{(FP+FN)}{(TP+FP+FN+TN)} = \frac{(FP+FN)}{(P+N)}$$

**Positive and Negative Class Metrics:**

- **Precision (P)**: Proportion of predicted positives that are actually true positives.

$$P = \frac{TP}{(TP+FP)}$$

- **True Positive Rate (TPR)**: Proportion of true positives that are correctly predicted. This is also known as **recall (R)** or **sensitivity**.

$$TPR = \frac{TP}{(TP+FN)}$$

- **True Negative Rate (TNR)**: Proportion of true negatives that are correctly predicted.

$$TNR = \frac{TN}{(TN+FP)}$$

- **False Positive Rate (FPR)**: Proportion of true negatives that are incorrectly predicted.

$$FPR = \frac{FP}{(FP+TN)}$$

- **False Negative Rate (FNR)**: Proportion of true positives that are incorrectly predicted.

$$FNR = \frac{FN}{(FN+TP)}$$

**Balanced and Skill Metrics:**

- **F1 Measure (F1)**: Harmonic mean of precision and recall.

$$F1 = \frac{(2\times TP)}{(2\times TP+FP+FN)} = \frac{2\times(P\times R)}{(P+R)}$$

- **Balanced Accuracy (BACC)**: Measures the average sensitivity (TPR) and specificity (FPR).

$$BACC = \frac{TPR+TNR}{2} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

- **True Skill Statistic (TSS)**: Measures the difference between the recall minus the probability of false detection. This determines model skill by accounting for correct predictions in both classes.

$$TSS = \frac{TP}{TP+FN} - \frac{FP}{FP+TN}$$

- **Heidke Skill Score (HSS)**: Measures the fractional prediction over random prediction. This compares the accuracy of classification to random chance.

$$HSS = \frac{2(TP\times TN - FP\times FN)}{(TP+FN)\times(FN+TN)+(TP+FP)\times(FP+T)}$$

**Probability-Based Metrics:**

- **Brier Score (BS)**: Measures the mean squared error between expected probabilities and predicted probabilities.

$$BS = \frac{1}{m}\sum_{n=1}^{m}\left(yn - \widehat{y}_n\right)^2$$

*... where predicted probability is denoted by* $\widehat{y}_n$

- **Brier Skill Score (BSS)**: Measures the likelihood of an event to occur.

$$BSS = \frac{BS}{\frac{1}{m}\sum_{n=1}^{m}(y_n - \bar{y})^2}$$

*... where* $\bar{y}$ *is the mean value of all the observed probabilities* $= \frac{1}{m}\sum_{n=1}^{m} y_n$

**K-Fold Cross-Validation**

Classifier performance was evaluated using the 10-Fold Cross-Validation method, which is suitable for classifiers that predict labels (positive or negative). In this project, the dataset is split into 10 parts. Each algorithm trains on 9 parts and tests on the remaining 1 part, which is then repeated 10 times. Metrics from each fold are saved and averaged.

## RESULTS AND EVALUATIONS

Average performance metrics across all folds were obtained and saved in a tabular format for comparison. Based on the average results, **Random Forest** is observed to be the most effective model for predicting binary classification of the Breast Cancer Wisconsin (Diagnostic) dataset. This is due to its consistently high precision, recall, and low error rate, especially when compared to KNN and LSTM, making it both accurate and reliable across all folds.

## CONCLUSION

In conclusion, this project is able to successfully implement three classification models and evaluate their performance on a medical diagnostic dataset. It was determined that the Random Forest model was the most reliable and accurate. However, it can be noted that the KNN model was able to provide a good baseline, while the LSTM model demonstrated how deep learning could be adapted, though it was unable to outperform the others.

By using manual calculations with values derived from the confusion matrix, along with implementing 10-fold cross-validation uniformly across all three models, the evaluation is deemed consistent and objective. All results are saved in tabular format for visualization and are reproducible via the included Jupyter notebook.

# 1) CSV FILES

The *breast_cancer_data.csv* is a CSV file which displays the dataset used within this project for classification. The following screenshot is a snippet of the entire file.

| Diagnosis | Radius_Mean | Texture_Mean | Perimeter_Mean | Area_Mean | Smoothness_Mean | Compactness_Mean | Concavity_Mean | Concave_Points_Mean | Symmetry_Mean | Fractal_Dimension_Mean | Radius_Std_Dev | Texture_Std_Dev | Perimeter_Std_Dev |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 17.99 | 10.38 | 122.8 | 1001.0 | 0.1184 | 0.2776 | 0.3001 | 0.1471 | 0.2419 | 0.07871 | 1.095 | 0.9053 | 8.589 |
| 1 | 20.57 | 17.77 | 132.9 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | 0.05667 | 0.5435 | 0.7339 | 3.398 |
| 1 | 19.69 | 21.25 | 130.0 | 1203.0 | 0.1096 | 0.1599 | 0.1974 | 0.1279 | 0.2069 | 0.05999 | 0.7456 | 0.7869 | 4.585 |
| 1 | 11.42 | 20.38 | 77.58 | 386.1 | 0.1425 | 0.2839 | 0.2414 | 0.1052 | 0.2597 | 0.09744 | 0.4956 | 1.156 | 3.445 |
| 1 | 20.29 | 14.34 | 135.1 | 1297.0 | 0.1003 | 0.1328 | 0.198 | 0.1043 | 0.1809 | 0.05883 | 0.7572 | 0.7813 | 5.438 |
| 1 | 12.45 | 15.7 | 82.57 | 477.1 | 0.1278 | 0.17 | 0.1578 | 0.08089 | 0.2087 | 0.07613 | 0.3345 | 0.8902 | 2.217 |
| 1 | 18.25 | 19.98 | 119.6 | 1040.0 | 0.09463 | 0.109 | 0.1127 | 0.074 | 0.1794 | 0.05742 | 0.4467 | 0.7732 | 3.18 |
| 1 | 13.71 | 20.83 | 90.2 | 577.9 | 0.1189 | 0.1645 | 0.09366 | 0.05985 | 0.2196 | 0.07451 | 0.5835 | 1.377 | 3.856 |
| 1 | 13.0 | 21.82 | 87.5 | 519.8 | 0.1273 | 0.1932 | 0.1859 | 0.09353 | 0.235 | 0.07389 | 0.3063 | 1.002 | 2.406 |
| 1 | 12.46 | 24.04 | 83.97 | 475.9 | 0.1186 | 0.2396 | 0.2273 | 0.08543 | 0.203 | 0.08243 | 0.2976 | 1.599 | 2.039 |
| 1 | 16.02 | 23.24 | 102.7 | 797.8 | 0.08206 | 0.06669 | 0.03299 | 0.03323 | 0.1528 | 0.05697 | 0.3795 | 1.187 | 2.466 |
| 1 | 15.78 | 17.89 | 103.6 | 781.0 | 0.0971 | 0.1292 | 0.09954 | 0.06606 | 0.1842 | 0.06082 | 0.5058 | 0.9849 | 3.564 |
| 1 | 19.17 | 24.8 | 132.4 | 1123.0 | 0.0974 | 0.2458 | 0.2065 | 0.1118 | 0.2397 | 0.078 | 0.9555 | 3.568 | 11.07 |
| 1 | 15.85 | 23.95 | 103.7 | 782.7 | 0.08401 | 0.1002 | 0.09938 | 0.05364 | 0.1847 | 0.05338 | 0.4033 | 1.078 | 2.903 |
| 1 | 13.73 | 22.61 | 93.6 | 578.3 | 0.1131 | 0.2293 | 0.2128 | 0.08025 | 0.2069 | 0.07682 | 0.2121 | 1.169 | 2.061 |
| 1 | 14.54 | 27.54 | 96.73 | 658.8 | 0.1139 | 0.1595 | 0.1639 | 0.07364 | 0.2303 | 0.07077 | 0.37 | 1.033 | 2.879 |
| 1 | 14.68 | 20.13 | 94.74 | 684.5 | 0.09867 | 0.072 | 0.07395 | 0.05259 | 0.1586 | 0.05922 | 0.4727 | 1.24 | 3.195 |
| 1 | 16.13 | 20.68 | 108.1 | 798.8 | 0.117 | 0.2022 | 0.1722 | 0.1028 | 0.2164 | 0.07356 | 0.5692 | 1.073 | 3.854 |
| 1 | 19.81 | 22.15 | 130.0 | 1260.0 | 0.09831 | 0.1027 | 0.1479 | 0.09498 | 0.1582 | 0.05395 | 0.7582 | 1.017 | 5.865 |

*Screenshot 1.1 Extract from the 'breast_cancer_data.csv' CSV File*

## 2) CODE

The following screenshots are from the Python source code file.

*Imports and Dataset Loading*

The program first imports all necessary libraries and tools. It then reads in the dataset from the CSV file and separates it into input features (X) and output labels (y) for use by all three models. It also prepares the dataset for cross-validation.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam

kf = KFold(n_splits=10, shuffle=True, random_state=42)
data = pd.read_csv("breast_cancer_data.csv")
X_df = data.drop(columns=["Diagnosis"])
y_df = data["Diagnosis"]
X_np = X_df.values
y_np = y_df.values
```

*Screenshot 2.1. Imports and Dataset Loading*

*Random Forest Model*

The program implements the Random Forest classification model using 10-fold cross-validation. It trains the model, calculates manual performance metrics from confusion matrix values, and stores all metric results in an external CSV file within the project's native folder.

```
def run_knn():
    print("\nRunning K-Nearest Neighbor (KNN)...")
    results = []
    for i, (train_idx, test_idx) in enumerate(kf.split(X_df), start=1):
        X_train, X_test = X_df.iloc[train_idx], X_df.iloc[test_idx]
        y_train, y_test = y_df.iloc[train_idx], y_df.iloc[test_idx]
        model = KNeighborsClassifier(n_neighbors=5)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        prob = model.predict_proba(X_test)[:, 1]
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results.append(compute_metrics(i, tp, tn, fp, fn, prob, y_test))
    df = pd.DataFrame(results)
    df.loc["Average"] = df.mean(numeric_only=True)
    print("\nKNN Results:\n", df)
    df.to_csv("knn_results.csv", index=False)
```

*Screenshot 2.2. Random Forest Algorithm*

*K-Nearest Neighbor (KNN) Model*

The program implements the K-Nearest Neighbor algorithm using the same 10-fold cross-validation approach. It trains the model, calculates manual performance metrics from confusion matrix values, and stores all metric results in an external CSV file within the project's native folder.

```
def run_knn():
    print("\nRunning K-Nearest Neighbor (KNN)...")
    results = []
    for i, (train_idx, test_idx) in enumerate(kf.split(X_df), start=1):
        X_train, X_test = X_df.iloc[train_idx], X_df.iloc[test_idx]
        y_train, y_test = y_df.iloc[train_idx], y_df.iloc[test_idx]
        model = KNeighborsClassifier(n_neighbors=5)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        prob = model.predict_proba(X_test)[:, 1]
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results.append(compute_metrics(i, tp, tn, fp, fn, prob, y_test))
    df = pd.DataFrame(results)
    df.loc["Average"] = df.mean(numeric_only=True)
    print("\nKNN Results:\n", df)
    df.to_csv("knn_results.csv", index=False)
```

*Screenshot 2.3. KNN Model*

*Long-Short Term Memory (LSTM) Model*

The project implements the LSTM algorithm using the same 10-fold cross-validation approach. It trains the model, calculates manual performance metrics from confusion matrix values, and stores all metric results in an external CSV file within the project's native folder.

```
def run_lstm():
    print("\nRunning Long Short-Term Memory (LSTM)...")
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_np)
    X_lstm = X_scaled.reshape((X_scaled.shape[0], X_scaled.shape[1], 1))
    results = []
    for i, (train_idx, test_idx) in enumerate(kf.split(X_lstm), start=1):
        X_train, X_test = X_lstm[train_idx], X_lstm[test_idx]
        y_train, y_test = y_np[train_idx], y_np[test_idx]
        model = Sequential([
            Input(shape=(X_train.shape[1], 1)),
            LSTM(32),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
        model.fit(X_train, y_train, epochs=15, batch_size=16, verbose=0)
        prob = model.predict(X_test).flatten()
        y_pred = (prob > 0.5).astype("int32")
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results.append(compute_metrics(i, tp, tn, fp, fn, prob, y_test))
    df = pd.DataFrame(results)
    df.loc["Average"] = df.mean(numeric_only=True)
    print("\nLSTM Results:\n", df)
    df.to_csv("lstm_results.csv", index=False)
```

*Screenshot 2.4. LSTM Model*

*Manual Metric Computations*

The program implements a function to manually perform calculations of metrics when called.

```python
def compute_metrics(fold, tp, tn, fp, fn, prob, y_test):
    total = tp + tn + fp + fn
    accuracy = (tp + tn) / total
    error_rate = (fp + fn) / total
    precision = tp / (tp + fp) if (tp + fp) else 0
    recall = tp / (tp + fn) if (tp + fn) else 0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) else 0
    tpr = recall
    tnr = tn / (tn + fp) if (tn + fp) else 0
    fpr = fp / (fp + tn) if (fp + tn) else 0
    fnr = fn / (fn + tp) if (fn + tp) else 0
    bacc = (tpr + tnr) / 2
    tss = tpr + tnr - 1
    hss = 2 * (tp * tn - fn * fp) / ((tp + fn)*(fn + tn) + (tp + fp)*(fp + tn)) if ((tp + fn)*(fn + tn) + (tp + fp)*(fp + tn)) else 0
    brier_score = np.mean((prob - np.array(y_test)) ** 2)
    bss = 1 - (brier_score / 0.25)

    return {
        "Fold": fold, "Total": total, "Accuracy": accuracy, "Error Rate": error_rate,
        "Precision": precision, "TPR": tpr, "TNR": tnr, "FPR": fpr, "FNR": fnr,
        "F1": f1, "BACC": bacc, "TSS": tss, "HSS": hss,
        "BS": brier_score, "BSS": bss,
        "TP": tp, "TN": tn, "FP": fp, "FN": fn
    }
```

*Screenshot 2.5. Manual Metric Computations*

*Summary of Results*

The program provides the user with a final summary of averages and prints it in a tabular format.

```python
def summarize_results():
    rf = pd.read_csv("rf_results.csv").tail(1).copy()
    knn = pd.read_csv("knn_results.csv").tail(1).copy()
    lstm = pd.read_csv("lstm_results.csv").tail(1).copy()

    rf["Model"] = "RF"
    knn["Model"] = "KNN"
    lstm["Model"] = "LSTM"

    summary = pd.concat([rf, knn, lstm], ignore_index=True)
    summary = summary[["Model"] + [col for col in summary.columns if col != "Model"]]
    print("\nFinal Summary of Metrics:\n")
    print(summary.round(4))

    best_model = summary.sort_values("Accuracy", ascending=False).iloc[0]
    print(f"\nBased on the average results, {best_model['Model']} is better at predicting binary classification of the Breast Cancer Wisconsin (Diagnostic) dataset.")
    print("This is because it achieved the highest average accuracy, supported by balanced and skillful metric scores.")
```

*Screenshot 2.6. Summary of Averages*

## 3. OUTPUT

The following screenshots are to demonstrate outputs of the program in the Terminal.

### *Running Random Forest*

```
Running Random Forest...

Random Forest Results:
         Fold  Total  Accuracy  Error Rate  Precision       TPR       TNR       FPR       FNR        F1      BACC       TSS       HSS        BS       BSS    TP    TN   FP   FN
0         1.0   57.0  0.964912    0.035088   0.941176  0.941176  0.975000  0.025000  0.058824  0.941176  0.958088  0.916176  0.916176  0.026832  0.892674  16.0  39.0  1.0  1.0
1         2.0   57.0  0.964912    0.035088   1.000000  0.923077  1.000000  0.000000  0.076923  0.960000  0.961538  0.923077  0.928839  0.025774  0.896905  24.0  31.0  0.0  2.0
2         3.0   57.0  0.982456    0.017544   1.000000  0.950000  1.000000  0.000000  0.050000  0.974359  0.975000  0.950000  0.961039  0.021858  0.912568  19.0  37.0  0.0  1.0
3         4.0   57.0  0.964912    0.035088   0.941176  0.941176  0.975000  0.025000  0.058824  0.941176  0.958088  0.916176  0.916176  0.019733  0.921067  16.0  39.0  1.0  1.0
4         5.0   57.0  0.964912    0.035088   0.944444  0.944444  0.974359  0.025641  0.055556  0.944444  0.959402  0.918803  0.918803  0.034711  0.861158  17.0  38.0  1.0  1.0
5         6.0   57.0  0.947368    0.052632   0.958333  0.920000  0.968750  0.031250  0.080000  0.938776  0.944375  0.888750  0.892655  0.046467  0.814133  23.0  31.0  1.0  2.0
6         7.0   57.0  0.964912    0.035088   0.941176  0.941176  0.975000  0.025000  0.058824  0.941176  0.958088  0.916176  0.916176  0.026800  0.892800  16.0  39.0  1.0  1.0
7         8.0   57.0  0.947368    0.052632   0.960000  0.923077  0.967742  0.032258  0.076923  0.941176  0.945409  0.890819  0.893591  0.029674  0.881305  24.0  30.0  1.0  2.0
8         9.0   57.0  0.964912    0.035088   1.000000  0.925926  1.000000  0.000000  0.074074  0.961538  0.962963  0.925926  0.929368  0.042846  0.828618  25.0  30.0  0.0  2.0
9        10.0   56.0  0.964286    0.035714   0.947368  0.947368  0.972973  0.027027  0.052632  0.947368  0.960171  0.920341  0.920341  0.041143  0.835429  18.0  36.0  1.0  1.0
Average   5.5   56.9  0.963095    0.036905   0.963368  0.935742  0.980882  0.019118  0.064258  0.949119  0.958312  0.916625  0.919317  0.031584  0.873666  19.8  35.0  0.7  1.4
```

*Screenshot 3.1. Random Forest Output*

### *Running K-Nearest Neighbor*

```
Running K-Nearest Neighbor (KNN)...

KNN Results:
         Fold  Total  Accuracy  Error Rate  Precision       TPR       TNR       FPR       FNR        F1      BACC       TSS       HSS        BS       BSS    TP    TN   FP   FN
0         1.0   57.0  0.964912    0.035088   0.941176  0.941176  0.975000  0.025000  0.058824  0.941176  0.958088  0.916176  0.916176  0.021053  0.915789  16.0  39.0  1.0  1.0
1         2.0   57.0  0.947368    0.052632   1.000000  0.884615  1.000000  0.000000  0.115385  0.938776  0.942308  0.884615  0.892924  0.041404  0.834386  23.0  31.0  0.0  3.0
2         3.0   57.0  0.964912    0.035088   0.909091  1.000000  0.945946  0.054054  0.000000  0.952381  0.972973  0.945946  0.924703  0.023860  0.904561  20.0  35.0  2.0  0.0
3         4.0   57.0  1.000000    0.000000   1.000000  1.000000  1.000000  0.000000  0.000000  1.000000  1.000000  1.000000  1.000000  0.015439  0.938246  17.0  40.0  0.0  0.0
4         5.0   57.0  0.894737    0.105263   0.875000  0.777778  0.948718  0.051282  0.222222  0.823529  0.863248  0.726496  0.748899  0.101053  0.595789  14.0  37.0  2.0  4.0
5         6.0   57.0  0.894737    0.105263   0.880000  0.880000  0.906250  0.093750  0.120000  0.880000  0.893125  0.786250  0.786250  0.075789  0.696842  22.0  29.0  3.0  3.0
6         7.0   57.0  0.964912    0.035088   0.894737  1.000000  0.950000  0.050000  0.000000  0.944444  0.975000  0.950000  0.918919  0.023860  0.904561  17.0  38.0  2.0  0.0
7         8.0   57.0  0.929825    0.070175   0.958333  0.884615  0.967742  0.032258  0.115385  0.920000  0.926179  0.852357  0.857678  0.068772  0.724912  23.0  30.0  1.0  3.0
8         9.0   57.0  0.912281    0.087719   0.958333  0.851852  0.966667  0.033333  0.148148  0.901961  0.909259  0.818519  0.823091  0.072982  0.708070  23.0  29.0  1.0  4.0
9        10.0   56.0  0.892857    0.107143   0.882353  0.789474  0.945946  0.054054  0.210526  0.833333  0.867710  0.735420  0.754745  0.082143  0.671429  15.0  35.0  2.0  4.0
Average   5.5   56.9  0.936654    0.063346   0.929902  0.900951  0.960627  0.039373  0.099049  0.913560  0.930789  0.861578  0.862338  0.052635  0.789459  19.0  34.3  1.4  2.2
```

*Screenshot 3.2. KNN Output*

### *Running Long Short-Term Memory*

```
LSTM Results:
         Fold  Total  Accuracy  Error Rate  Precision       TPR       TNR       FPR       FNR        F1      BACC       TSS       HSS        BS       BSS    TP    TN   FP   FN
0         1.0   57.0  0.947368    0.052632   0.888889  0.941176  0.950000  0.050000  0.058824  0.914286  0.945588  0.891176  0.876356  0.059145  0.763419  16.0  38.0  2.0  1.0
1         2.0   57.0  0.947368    0.052632   0.925926  0.961538  0.935484  0.064516  0.038462  0.943396  0.948511  0.897022  0.894249  0.035223  0.859108  25.0  29.0  2.0  1.0
2         3.0   57.0  0.929825    0.070175   0.863636  0.950000  0.918919  0.081081  0.050000  0.904762  0.934459  0.868919  0.849406  0.058187  0.767254  19.0  34.0  3.0  1.0
3         4.0   57.0  0.947368    0.052632   0.888889  0.941176  0.950000  0.050000  0.058824  0.914286  0.945588  0.891176  0.876356  0.035263  0.858946  16.0  38.0  2.0  1.0
4         5.0   57.0  0.947368    0.052632   0.941176  0.888889  0.974359  0.025641  0.111111  0.914286  0.931624  0.863248  0.876356  0.047657  0.809374  16.0  38.0  1.0  2.0
5         6.0   57.0  0.912281    0.087719   1.000000  0.800000  1.000000  0.000000  0.200000  0.888889  0.900000  0.800000  0.817891  0.058873  0.764509  20.0  32.0  0.0  5.0
6         7.0   57.0  0.947368    0.052632   0.888889  0.941176  0.950000  0.050000  0.058824  0.914286  0.945588  0.891176  0.876356  0.044175  0.823298  16.0  38.0  2.0  1.0
7         8.0   57.0  0.964912    0.035088   1.000000  0.923077  1.000000  0.000000  0.076923  0.960000  0.961538  0.923077  0.928839  0.039768  0.840929  24.0  31.0  0.0  2.0
8         9.0   57.0  0.912281    0.087719   0.923077  0.888889  0.933333  0.066667  0.111111  0.905660  0.911111  0.822222  0.823748  0.058466  0.766135  24.0  28.0  2.0  3.0
9        10.0   56.0  0.946429    0.053571   0.900000  0.947368  0.945946  0.054054  0.052632  0.923077  0.946657  0.893314  0.882022  0.045703  0.817188  18.0  35.0  2.0  1.0
Average   5.5   56.9  0.940257    0.059743   0.922048  0.918329  0.955804  0.044196  0.081671  0.918293  0.937067  0.874133  0.870158  0.048246  0.807016  19.4  34.1  1.6  1.8
```

*Screenshot 3.3. LSTM Output*

### *Final Summary of Metrics & Print Statement*

```
Final Summary of Metrics:

   Model  Fold  Total  Accuracy  Error Rate  Precision     TPR     TNR     FPR     FNR      F1    BACC     TSS     HSS      BS     BSS    TP    TN   FP   FN
0     RF   5.5   56.9    0.9631      0.0369     0.9634  0.9357  0.9809  0.0191  0.0643  0.9491  0.9583  0.9166  0.9193  0.0316  0.8737  19.8  35.0  0.7  1.4
1    KNN   5.5   56.9    0.9367      0.0633     0.9299  0.9010  0.9606  0.0394  0.0990  0.9136  0.9308  0.8616  0.8623  0.0526  0.7895  19.0  34.3  1.4  2.2
2   LSTM   5.5   56.9    0.9403      0.0597     0.9220  0.9183  0.9558  0.0442  0.0817  0.9183  0.9371  0.8741  0.8702  0.0482  0.8070  19.4  34.1  1.6  1.8
```

*Screenshot 3.4. Final Summary*

```
Based on the average results, RF is better at predicting binary classification of the Breast Cancer Wisconsin (Diagnostic) dataset.
This is because it achieved the highest average accuracy, supported by balanced and skillful metric scores.
```

*Screenshot 3.5. Final Print Statement*

*Submission Files*

- `finaltermproj_notebook.ipynb` – Jupyter notebook with all code and results
- `breast_cancer_data.csv` – Dataset used for classification
- `rf_results.csv`, `knn_results.csv`, `lstm_results.csv` – Per-fold and average metrics
- `finaltermproj.py` – Python source code

*Link to the GitHub repository:*
https://github.com/etp6/CS634-Final-Term-Project