NAME: Hem Thumar
NJIT UCID: ht296
Email Address: ht296@njit.edu
24rd November 2024
Professor: Yasser Abduallah
CS 634 - Data Mining

Final Project Report

Using Naive Bayes, RF and LSTM To Predict Heart disease

2.1 Abstract

Heart disease remains one of the leading causes of mortality worldwide, making early and accurate diagnosis essential for effective intervention and management. This study explores the use of machine learning and deep learning techniques to predict the likelihood of heart disease based on diagnostic data. Specifically, we implement and compare three prominent algorithms: Long Short-Term Memory (LSTM) networks, Random Forest (RF) classifiers, and Naive Bayes for heart disease prediction. The LSTM model, known for its ability to handle sequential data, is employed to capture temporal dependencies in patient medical histories. The Random Forest classifier, an ensemble method, is used to provide robust predictions by leveraging multiple decision trees. Meanwhile, the Naive Bayes classifier, based on probabilistic theory, is applied for its simplicity and efficiency in handling large datasets. The performance of these models is evaluated using standard metrics such as accuracy, precision, recall, F1-score, and other relevant evaluation metrics to assess their predictive capabilities. This research contributes to the integration of artificial intelligence in healthcare, providing reliable computational tools for early heart disease detection and decision support.

2.2 Introduction

Heart disease is one of the leading causes of morbidity and mortality worldwide, with significant public health implications. Early and accurate diagnosis of heart disease is critical to managing the condition, preventing complications, and improving patient outcomes. Traditional diagnostic methods, such as clinical evaluations and imaging, are often time-consuming, expensive, and subject to human error, underscoring the need for automated systems that leverage computational methods to improve accuracy and efficiency.

Recent advances in machine learning (ML) and deep learning (DL) have made it possible to develop robust predictive models for various diseases, including heart disease. This study integrates traditional machine learning techniques—Random Forest (RF) and Naive Bayes—with a deep learning approach, Long Short-Term Memory (LSTM), to predict the likelihood of heart disease based on diagnostic measurements. While RF and Naive Bayes are known for their ability to handle structured data efficiently, LSTM networks are particularly well-suited for modeling temporal dependencies and sequential patterns in time-series or medical history data.

The primary objective of this research is to evaluate and compare the predictive performance of these algorithms for heart disease prediction, helping to identify the most effective approach for analyzing diagnostic data. By assessing the strengths and limitations of each model, this study contributes to the growing role of artificial intelligence in healthcare, particularly in supporting early diagnosis and intervention for heart disease.

2.3 Key Concepts and Principles

1.Heart Disease: Background

1. **Definition**: Heart disease refers to a variety of conditions that affect the heart, including coronary artery disease, heart failure, and arrhythmias.
2. **Public Health Challenge**: Heart disease is a leading cause of death globally, and its complications—such as heart attack, stroke, and sudden cardiac arrest—can have life-threatening consequences.
3. **Diagnostic Importance**: Early and accurate detection of heart disease can significantly reduce the risk of severe complications and improve patient quality of life.

2.The Role of Automated Systems

1. **Traditional Diagnostic Limitations**:
    1. Time-consuming and expensive procedures such as electrocardiograms (ECG), echocardiograms, and stress tests.
    2. Potential for human error in interpretation and decision-making.
2. **Automated Approaches**: Machine learning (ML) and deep learning (DL) techniques offer the advantage of reducing human dependency, automating analysis, and improving diagnostic accuracy.

3.Machine Learning Techniques

1.Random Forest (RF):

1. **Definition**: An ensemble learning method that combines multiple decision trees to make predictions based on majority voting.
2. **Advantages**: RF handles non-linear relationships well, reduces overfitting, and is robust in classifying large datasets. It also provides feature importance, which is valuable for interpretability in medical contexts.
3. **Application**: Suitable for structured data such as patient demographics, medical history, and clinical measurements.

2.Naive Bayes:

1. **Definition**: A probabilistic classifier based on Bayes' theorem, which assumes that features are conditionally independent given the class label.
2. **Advantages**: Naive Bayes is simple, computationally efficient, and works well with small to medium-sized datasets. It performs particularly well when the independence assumption is approximately correct.
3. **Application**: Effective for quick classification tasks and can handle categorical and numerical features with ease.

3.Deep Learning Technique

1. **Long Short-Term Memory (LSTM)**:
    1. **Definition**: A type of recurrent neural network (RNN) designed to model long-term dependencies and sequential data.
    2. **Advantages**: LSTM excels in modeling temporal or sequential relationships, making it suitable for patient medical histories or time-series data such as trends in blood pressure, heart rate, and ECG measurements over time.
    3. **Application**: Ideal for sequential data or cases where understanding the progression of disease over time is important for diagnosis.

4. Study Objectives

1. **Predictive Performance Evaluation**:

    To compare the performance of RF, Naive Bayes, and LSTM based on accuracy, precision, recall, F1-score, and other relevant metrics.

2. **Suitability for Heart Disease Prediction**:

    To assess the strengths and weaknesses of traditional ML methods and deep learning approaches in the context of heart disease diagnosis.

3. **Contribution to Healthcare AI**:

    To identify the most suitable algorithm for early detection of heart disease, thereby enhancing healthcare decision-making and patient outcomes.

5.Theoretical Framework

1.Supervised Learning:

All three algorithms (RF, Naive Bayes, and LSTM) are trained on labeled data (i.e., diagnostic features and outcomes), making them supervised learning algorithms.

2.Feature Importance:

RF provides feature importance as part of its ensemble method, which helps in understanding which features (e.g., cholesterol levels, age, blood pressure) contribute most to the diagnosis.

3.Sequential Dependencies:

LSTM models are designed to capture the temporal dependencies in medical history or time-series data, which can be critical for detecting patterns of heart disease progression.

6.Broader Implications

1.Advancing Predictive Healthcare:

Integrating ML and DL methods into the healthcare workflow can enhance diagnostic speed, reduce errors, and ensure consistent, reliable results, particularly for heart disease prediction.

2.AI in Medicine:

This study represents the growing role of artificial intelligence in clinical settings, offering the potential for more accurate, automated, and personalized healthcare solutions.

7.Challenges and Considerations

1.Data Quality and Size:

Machine learning models, particularly deep learning models like LSTM, require large and high-quality datasets for optimal performance. The availability of comprehensive datasets for heart disease prediction is crucial.

2.Model Interpretability:

While RF and Naive Bayes are relatively easy to interpret, LSTM models are often considered "black boxes," which can make it challenging to understand the reasons behind specific predictions, a key factor in healthcare applications.

3.Computational Resources:

LSTM and other deep learning techniques require significantly more computational power and resources compared to traditional ML methods like RF and Naive Bayes.

8.Expected Outcomes

1.Identification of the Most Effective Model:

1. The research aims to identify which algorithm—RF, Naive Bayes, or LSTM—performs best in terms of predictive accuracy for heart disease detection.

2.Insights into Trade-offs:

1. The study will provide insights into the trade-offs between computational complexity, interpretability, and diagnostic accuracy for heart disease prediction.

3.Support for Early Diagnosis:

1. The findings will contribute to improving early diagnosis and intervention strategies for heart disease, potentially reducing healthcare costs and improving patient outcomes.

9.Dataset

- The dataset is downloaded from kegal, the link is given below for referances

https://www.kaggle.com/datasets/priyanka841/heart-disease-prediction-uci

2.4 Source code

```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

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

```
heart_data = pd.read_csv(r"heart.csv")
heart_data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1

```
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

2.4.2 Loading Data And Preprocessing

```
# Feature and target extraction
features = heart_data.iloc[:, :-1].values
labels = heart_data.iloc[:, -1].values
```

2.4.3 Separating The Dataset into Features and Output lable

```
# Function to compute performance metrics
def evaluate_model_performance(true_labels, predicted_labels):
    conf_matrix = confusion_matrix(true_labels, predicted_labels)
    true_pos, false_neg, false_pos, true_neg = conf_matrix.ravel()
    acc = accuracy_score(true_labels, predicted_labels)
    prec = precision_score(true_labels, predicted_labels, zero_division=0)
    rec = recall_score(true_labels, predicted_labels)
    f1 = f1_score(true_labels, predicted_labels)

    false_pos_rate = false_pos / (false_pos + true_neg) if (false_pos + true_neg) > 0 else 0
    false_neg_rate = false_neg / (false_neg + true_pos) if (false_neg + true_pos) > 0 else 0
    tss_value = rec - false_pos_rate
    hss_value = (2 * (true_pos * true_neg - false_pos * false_neg)) / (
        (true_pos + false_neg) * (false_neg + true_neg) + (true_pos + false_pos) * (false_pos + true_neg)
    ) if (true_pos + false_neg + false_pos + true_neg) > 0 else 0

    return {
        "Accuracy": acc,
        "Precision": prec,
        "Recall": rec,
        "F1-Score": f1,
        "TSS": tss_value,
        "HSS": hss_value,
        "True_Positive": true_pos,
        "True_Negative": true_neg,
        "False_Positive": false_pos,
        "False_Negative": false_neg,
        "FPR": false_pos_rate,
        "FNR": false_neg_rate,
    }
```

```
# Initialize results dictionary with all expected keys
results_dict = {
    'Model_Name': [],
    'Fold_Number': [],
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1-Score': [],
    'True_Positive': [],
    'True_Negative': [],
    'False_Positive': [],
    'False_Negative': [],
    'FPR': [],
    'FNR': [],
    'TSS': [],
    'HSS': []
}
```

2.4.3 matrix calculation

```python
# StratifiedKFold for cross-validation
cv_folds = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Function to train Random Forest
def train_random_forest(features, labels, cv_folds, results_dict, fold_counter=1):
    """Train Random Forest model using StratifiedKFold cross-validation."""
    rf_model = RandomForestClassifier(random_state=42, n_estimators=200, max_depth=10)

    for train_indices, test_indices in cv_folds.split(features, labels):
        train_X, test_X = features[train_indices], features[test_indices]
        train_y, test_y = labels[train_indices], labels[test_indices]

        rf_model.fit(train_X, train_y)
        rf_predictions = rf_model.predict(test_X)

        rf_metrics = evaluate_model_performance(test_y, rf_predictions)
        for key, value in rf_metrics.items():
            results_dict[key].append(value)
        results_dict['Model_Name'].append('Random Forest')
        results_dict['Fold_Number'].append(fold_counter)
        fold_counter += 1

    return rf_model
```

2.4.4 Random forest algorithm

```python
# Function to train LSTM model
def train_lstm(features, labels):
    """Train LSTM model."""
    normalizer = StandardScaler()
    normalized_features = normalizer.fit_transform(features)

    train_X_lstm, test_X_lstm, train_y_lstm, test_y_lstm = train_test_split(normalized_features, labels, test_size=0.2, random_state=42, stratify=labels)
    train_X_lstm = train_X_lstm.reshape(train_X_lstm.shape[0], train_X_lstm.shape[1], 1)
    test_X_lstm = test_X_lstm.reshape(test_X_lstm.shape[0], test_X_lstm.shape[1], 1)

    lstm_model = Sequential([
        LSTM(128, activation='relu', input_shape=(train_X_lstm.shape[1], 1)),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    lstm_model.fit(train_X_lstm, train_y_lstm, epochs=100, batch_size=32, validation_split=0.2, verbose=0)

    lstm_predictions = (lstm_model.predict(test_X_lstm) > 0.5).astype(int)
    lstm_metrics = evaluate_model_performance(test_y_lstm, lstm_predictions)

    return lstm_model, lstm_metrics
```

2.4.5 LSTM

```python
# Function to train Naive Bayes model
def train_naive_bayes(features, labels):
    """Train Naive Bayes model."""
    normalizer = StandardScaler()
    normalized_features = normalizer.fit_transform(features)

    train_X_lstm, test_X_lstm, train_y_lstm, test_y_lstm = train_test_split(normalized_features, labels, test_size=0.2, random_state=42, stratify=labels)
    bayes_model = GaussianNB()
    bayes_model.fit(train_X_lstm, train_y_lstm)
    bayes_predictions = bayes_model.predict(test_X_lstm)

    bayes_metrics = evaluate_model_performance(test_y_lstm, bayes_predictions)

    return bayes_model, bayes_metrics
```

2.4.6 Naive Bayes

```python
# Main function to run models and evaluate performance
def main():
    """Main function to execute the models and gather results."""
    # Train Random Forest
    fold_counter = 1  # Initialize fold_counter
    rf_model = train_random_forest(features, labels, cv_folds, results_dict, fold_counter)

    # Train LSTM model
    lstm_model, lstm_metrics = train_lstm(features, labels)

    # Train Naive Bayes model
    bayes_model, bayes_metrics = train_naive_bayes(features, labels)

    # Add LSTM and Naive Bayes results to results_dict
    for key, value in lstm_metrics.items():
        results_dict[key].append(value)
    results_dict['Model_Name'].append('LSTM')
    results_dict['Fold_Number'].append('N/A')  # LSTM doesn't use CV

    for key, value in bayes_metrics.items():
        results_dict[key].append(value)
    results_dict['Model_Name'].append('Naive Bayes')
    results_dict['Fold_Number'].append('N/A')

    # Convert results to DataFrame
    results_df = pd.DataFrame(results_dict)

    # Print the results
    print(results_df)

    # Calculate average metrics for each model and compare
    average_metrics = results_df.groupby('Model_Name').mean(numeric_only=True)
```

2.4.7 main function

```python
    # Plot Confusion Matrices
    def plot_confusion_matrix(y_test, y_pred, title):
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
        plt.title(f'{title} - Confusion Matrix')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.show()

    # Plot for each model
    rf_predictions = rf_model.predict(features)
    plot_confusion_matrix(labels, rf_predictions, "Random Forest")

    bayes_predictions = bayes_model.predict(features)
    plot_confusion_matrix(labels, bayes_predictions, "Naive Bayes")

    lstm_predictions = (lstm_model.predict(features.reshape(features.shape[0], features.shape[1], 1)) > 0.5).astype(int)
    plot_confusion_matrix(labels, lstm_predictions, "LSTM")
```

2.4.8 graphical representation of all the algorithms

```python
# Compare models based on average metrics
comparison = {
    "Model": [],
    "Average Accuracy": [],
    "Average F1-Score": [],
    "Average TSS": [],
    "Average HSS": []
}

for model in average_metrics.index:
    comparison["Model"].append(model)
    comparison["Average Accuracy"].append(average_metrics.loc[model, "Accuracy"])
    comparison["Average F1-Score"].append(average_metrics.loc[model, "F1-Score"])
    comparison["Average TSS"].append(average_metrics.loc[model, "TSS"])
    comparison["Average HSS"].append(average_metrics.loc[model, "HSS"])

comparison_df = pd.DataFrame(comparison)

# Print model comparison
print(comparison_df)
```

```python
# Identify the best model based on Average Accuracy
best_model_row = comparison_df.loc[comparison_df['Average Accuracy'].idxmax()]
best_model_name = best_model_row['Model']
best_model_accuracy = best_model_row['Average Accuracy']

# Add a line to print which model is best
best_model_summary = f"\nThe best model for this dataset based on Average Accuracy is: {best_model_name} with an Accuracy of {best_model_accuracy:.4f}."
print(best_model_summary)
```

2.4.9 comparison of all the algorithms

```python
    print("Models saved successfully!")

if __name__ == "__main__":
    main()
```

2.4.10 calling main function

```python
# Save models
models_dir = "models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

joblib.dump(rf_model, os.path.join(models_dir, "random_forest_model.pkl"))
joblib.dump(bayes_model, os.path.join(models_dir, "naive_bayes_model.pkl"))
lstm_model.save(os.path.join(models_dir, "lstm_model.h5"))
```

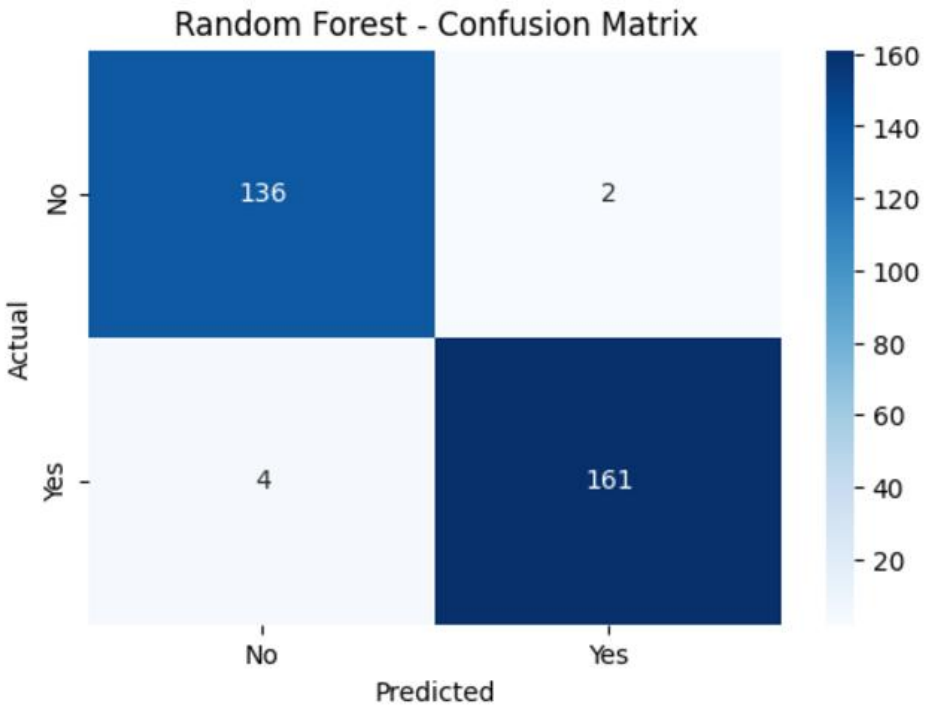
2.4.11 saving models

2.5 OUTPUT

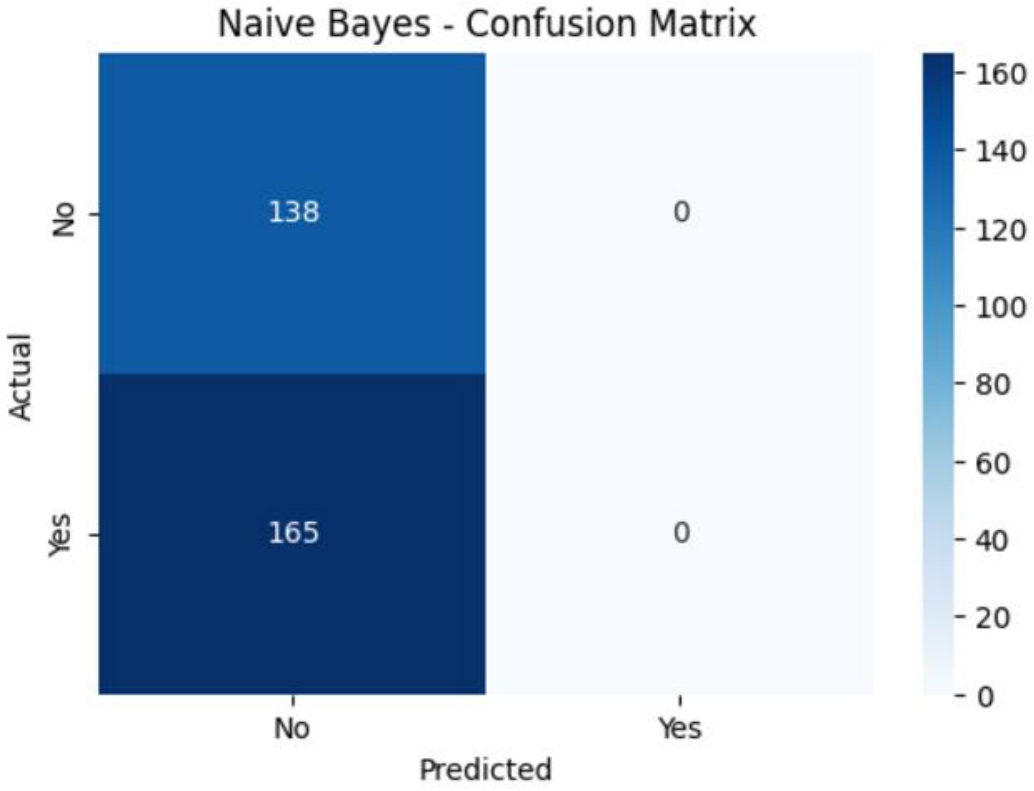
```
C:\Users\DELL\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
       Model_Name Fold_Number  Accuracy  Precision    Recall  F1-Score  \
0   Random Forest           1  0.838710   0.875000  0.823529  0.848485
1   Random Forest           2  0.935484   0.894737  1.000000  0.944444
2   Random Forest           3  0.741935   0.846154  0.647059  0.733333
3   Random Forest           4  0.866667   0.882353  0.882353  0.882353
4   Random Forest           5  0.733333   0.764706  0.764706  0.764706
5   Random Forest           6  0.766667   0.714286  0.937500  0.810811
6   Random Forest           7  0.833333   0.789474  0.937500  0.857143
7   Random Forest           8  0.800000   0.812500  0.812500  0.812500
8   Random Forest           9  0.866667   0.800000  1.000000  0.888889
9   Random Forest          10  0.800000   0.857143  0.750000  0.800000
10           LSTM         N/A  0.737705   0.729730  0.818182  0.771429
11    Naive Bayes         N/A  0.819672   0.789474  0.909091  0.845070

    True_Positive  True_Negative  False_Positive  False_Negative       FPR  \
0              12             14               3               2  0.176471
1              12             17               0               2  0.000000
2              12             11               6               2  0.352941
3              11             15               2               2  0.117647
4               9             13               4               4  0.235294
5               8             15               1               6  0.062500
6              10             15               1               4  0.062500
7              11             13               3               3  0.187500
8              10             16               0               4  0.000000
9              12             12               4               2  0.250000
10             18             27               6              10  0.181818
11             20             30               3               8  0.090909
```

	FNR	TSS	HSS
0	0.142857	0.647059	0.676409
1	0.142857	1.000000	0.868085
2	0.142857	0.294118	0.491803
3	0.153846	0.764706	0.728507
4	0.307692	0.529412	0.457014
5	0.428571	0.875000	0.520548
6	0.285714	0.875000	0.660633
7	0.214286	0.625000	0.598214
8	0.285714	1.000000	0.727273
9	0.142857	0.500000	0.601770
10	0.357143	0.636364	0.466083
11	0.285714	0.818182	0.631925

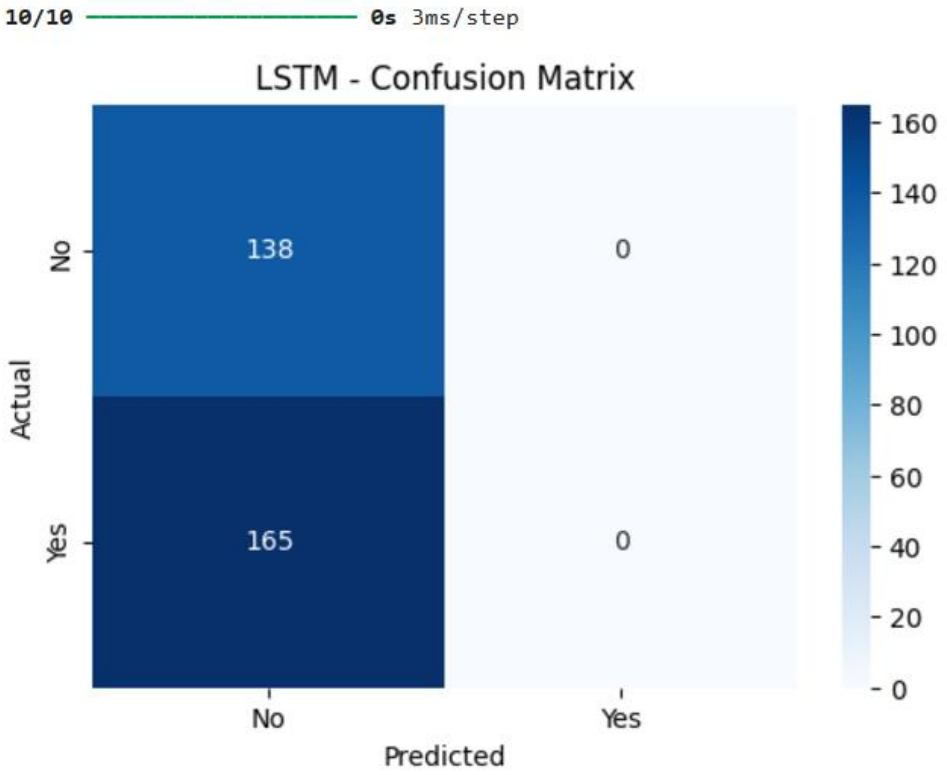
2.5.1 matrix calculation



2.5.2 confusion metrix for random forest



2.5.3 confusion metrix for Naive Bayes



2.5.4 confusion metrix for LSTM

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
           Model  Average Accuracy  Average F1-Score  Average TSS  Average HSS
0           LSTM          0.737705          0.771429     0.636364     0.466083
1    Naive Bayes          0.819672          0.845070     0.818182     0.631925
2  Random Forest          0.818280          0.834266     0.711029     0.633026

The best model for this dataset based on Average Accuracy is: Naive Bayes with an Accuracy of 0.8197.
Models saved successfully!
```

2.5.5 matrix comparison based on average of accuracy

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

2.7 Link to Git Repository:

https://github.com/HemThumar23/DM_finalproject