CS634 – Data Mining Final Project Report

Name: Rutvik Manojbhai Sonani

NJIT UCID: Rs2693

Email Address: Rs2693@njit.edu

Date: 11-15-2025

Professor: Dr. Yasser Abdullah

Course: CS 634 101 Data Mining

Title

Comparative analysis using supervised and deep learning algorithms for binary classification on a dataset.

Objective

The objective of project is to test and compare three different machine learning models Random Forest, Long Short-Term Memory (LSTM) and Support Vector Machine (SVM) on a binary classification task using a dataset. The goal is to determine which of the three learning algorithms performs the best when it comes to discerning the relationships among all written variables such as physical strength, training level, special abilities, and so on. In the project we will train each model, calculate all required performance metrics, and then determine which algorithm yields the best representation of accuracy and reliability with this type of data.

Introduction

Data mining is a process of finding useful and meaningful pattern from large set of data. One of the fundamental challenges in this regard is the binary classification wherein the system is required to select one out of the two possible categories. The current study experiments on the data set through binary classification methods to determine the probability of the character being “good” or “not good”.

To see the different methods' results on the same data, three algorithms from three different families were deployed:

1. Random Forest - a decision tree-based model where multiple trees are created and the final prediction is made on the basis of voting among the tree outputs.
2. Long Short-Term Memory (LSTM) - a deep learning approach that can master complex interactions of the features.
3. Support Vector Machine (SVM) – a machine learning model that separates two classes by drawing the best possible line or boundary between them. It chooses this boundary in such a way that the distance between the line and the nearest data points of each class is as large as possible, which helps the model make accurate predictions.

Each of these algorithms was trained and evaluated on the same dataset. Afterwards, the evaluation of the models was done using the predetermined metrics and the performance was ascertained. The objective of this project is to reveal the proficiency level of these algorithms in the context of data mining and further announce the one that provides the most trustworthy results for the classification challenge.

Important Concepts, and Principles

In this section the fundamental and concept of 3 important algorithm is being explained which is used in this project.

1. **Random Forest** - a decision tree-based model where multiple trees are created and the final prediction is made on the basis of voting among the tree outputs.

   Step of Random Forest -

- **Random Sampling:** Each tree is trained on a random subset of the training data (bootstrapping), ensuring diversity among trees and reducing overfitting risk.
- **Random Feature Selection:** At each split, a random subset of features is considered, further diversifying the trees and preventing any single feature from dominating the model.
- **Limiting Tree Complexity:** Hyperparameters such as maximum tree depth, minimum samples per leaf, and minimum samples required to split a node help restrict model complexity to avoid overfitting.
- **Aggregation of Multiple Trees:** By combining predictions from many different trees (majority voting or averaging), Random Forest reduces variance and balances out individual trees' overfitting tendencies.
- **Regularization and Validation:** Techniques like cross-validation and tuning parameters such as number of trees and features considered improve the generalization ability of the model.

2. **Long Short-Term Memory (LSTM)** - is a specialized type of Recurrent Neural Network (RNN) designed to capture long-term dependencies in sequential data, effectively addressing the vanishing gradient problem that traditional RNNs face. Unlike regular RNNs, LSTMs introduce a memory cell along with gates (input, forget, and output gates) that regulate the flow of information, allowing the network to selectively remember or forget information over extended time steps.

   The key steps in the operation of a Long Short-Term Memory (LSTM) unit are:

- **Forget Gate:** Decide what information from the previous cell state should be discarded. This step uses a sigmoid function to produce values between 0 and 1, where 0 means "completely forget" and 1 means "completely keep" for each piece of data from the prior state.
- **Input Gate:** Determine which new information to add to the cell state. It involves two parts: a sigmoid layer that decides which values to update, and a layer that creates candidate values to potentially add to the state.
- **Update Cell State:** Combine the old cell state (after forgetting) with the newly selected candidate values from the input gate to form an updated cell state.

- **Output Gate:** Finally, this gate controls the output of information from the current memory cell to the subsequent step, considering the current input and the updated memory.

3. **Support Vector Machine (SVM):**
   SVM is a supervised learning algorithm that tries to separate different classes by drawing the best possible boundary called a hyperplane. It chooses this boundary in such a way that the gap between the boundary and the nearest points from each class is as large as possible. This helps the model make clear and accurate decisions.

   Steps of Support Vector Machine (SVM)
   • **Finding Important Points (Support Vectors):** SVM first looks at the points that lie very close to the boundary. These special points, called *support vectors*, decide where the separating line should be placed.
   • **Drawing the Best Boundary (Hyperplane):** Using the support vectors, SVM draws a line or plane that separates the classes. The aim is to draw the boundary that gives the maximum gap between the two classes.
   • **Maximizing the Margin:** The bigger the margin, the better the model can separate the classes. So SVM adjusts the boundary to make this margin as wide as possible.
   • **Handling Non-Linear Data (Kernel Trick):** If a straight line cannot separate the data, SVM uses a *kernel*, which transforms the data into a higher dimension. This helps create a curved or complex boundary that separates the classes properly.
   • **Making Predictions:** Once the boundary is learned, SVM decides the class of a new point based on which side of the boundary it lies.
   • **Regularization and Parameter Tuning:** Settings like C that controls mistakes and gamma that controls curve smoothness are adjusted to reduce overfitting and improve accuracy.

Project Structure and Workflow

1. About Datasets:
   This project uses a structured dataset containing medical and demographic information of individuals, with the goal of predicting

whether a person is diabetic or not. Each record includes several health-related measurements that are commonly used in medical diagnosis, such as glucose level, blood pressure, body mass index, insulin levels, and more. The target column indicates whether the patient has diabetes (1) or does not have diabetes (0).
Below is the description of each field in the dataset:

Description of Each Field in the Diabetes Dataset

- **Pregnancies**: Number of times the patient has been pregnant.
- **Glucose**: Plasma glucose concentration measured two hours into an oral glucose tolerance test.
- **BloodPressure**: Diastolic blood pressure measured in millimeters of mercury (mm Hg).
- **SkinThickness**: Thickness of the triceps skin fold measured in millimeters, used as an indicator of body fat.
- **Insulin**: Serum insulin concentration measured two hours into an oral glucose tolerance test.
- **BMI (Body Mass Index)**: A measure of body fat calculated from weight and height (kg/m²).
- **DiabetesPedigreeFunction**: A score that indicates the likelihood of diabetes based on family history and genetic factors.
- **Age**: Age of the patient in years.
- **Outcome**: The target variable indicating diabetes diagnosis **1** for diabetic and **0** for non-diabetic.

2. **Environment and Installing required libraries and modules:**
   To run the project file, we need some python packages: scikit-learn, numpy, pandas, tensorflow, Matplotlib.pyplot, Seaborn.

   - Pandas is used to handle datasets, load data, clean data, and prepare it for subsequent analysis.
   - Numpy is essential for efficient numerical computations, data manipulation, and creating array structures that are crucial for machine learning models.
   - Matplotlib.pyplot is a versatile tool used to create both basic and advanced graphs. It excels in visualizing data distributions and model results.

- Seaborn, built on top of Matplotlib, offers a high-level API that simplifies the process of creating visually appealing and statistically informative plots with minimal code and a focus on beautiful styling.
- Scikit-learn provides tools for implementing and evaluating machine learning models, such as Support Vector Machine (SVM) and Random Forest.
- TensorFlow is specifically required to build and train the Long Short-Term Memory (LSTM) deep learning model for classification tasks.

If you are trying to run the main.py file, then followed bellowed the instruction.

1. First Install Python if you do not have To run python file, we need to install python in our system.

- Download the installer from python.org and add Python to your PATH during installation.

2. Set Up A virtual Environment –
   to establish a virtual environment, locate the folder containing all the necessary files.
   Then, open the terminal within that folder and proceed with the following steps.
   python -m venv venv
   than .venv\Scripts\activate
3. Install Required Python Packages For your file, install these libraries:
   • pandas
   • numpy
   • scikit-learn
   • Matplotlib
   • Seaborn
   • Tensorflow

4. Prepare Your Data Files Make sure your dataset CSV files is in the same directory as your script or update the script with their full paths.
5. Run the Python Script Run your main file in your terminal. python main.py

3. Code Structure and Project Workflow Overview

This project is structured to enable users to perform binary classification using algorithms including Support Vector Machine (SVM) Random Forest, and Long Short-Term Memory (LSTM) on the dataset.

Workflow:

- Dataset Loading & Selection: In this section, the code shows the available dataset, loads it using pandas, and performs data analysis to give an overview of features and target distribution.

```
diabetes_data = pd.read_csv('diabetes.csv')

print("Dataset loaded. Shape:", diabetes_data.shape)
print("\nFirst 5 rows:")
print(diabetes_data.head())

print("\nDataset Info:")
print(diabetes_data.info())
```

```
Dataset loaded. Shape: (768, 9)

First 5 rows:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

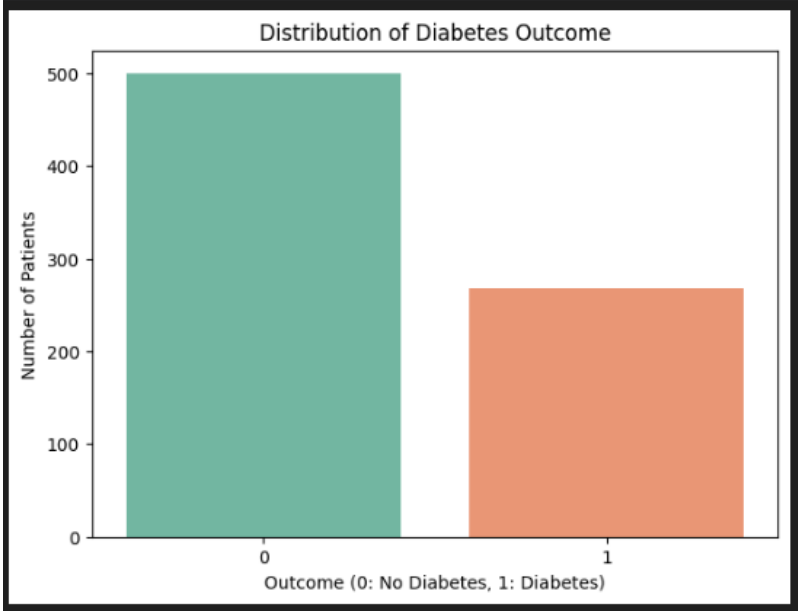
- Preprocessing:
  It handles missing and duplicates values using pandas built-in functions.

```python
def replace_zeros_with_median(df):
    columns_to_impute = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
    for col in columns_to_impute:
        median_val = df[col][df[col] != 0].median()
        df[col] = df[col].replace(0, median_val)
    return df

diabetes_data = replace_zeros_with_median(diabetes_data)

print("Zeros replaced with median values.")
print(diabetes_data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].describe())
```

```
Zeros replaced with median values.
          Glucose  BloodPressure  SkinThickness     Insulin         BMI
count  768.000000     768.000000     768.000000  768.000000  768.000000
mean   121.656250      72.386719      29.108073  140.671875   32.455208
std     30.438286      12.096642       8.791221   86.383060    6.875177
min     44.000000      24.000000       7.000000   14.000000   18.200000
25%     99.750000      64.000000      25.000000  121.500000   27.500000
50%    117.000000      72.000000      29.000000  125.000000   32.300000
75%    140.250000      80.000000      32.000000  127.250000   36.600000
max    199.000000     122.000000      99.000000  846.000000   67.100000
```

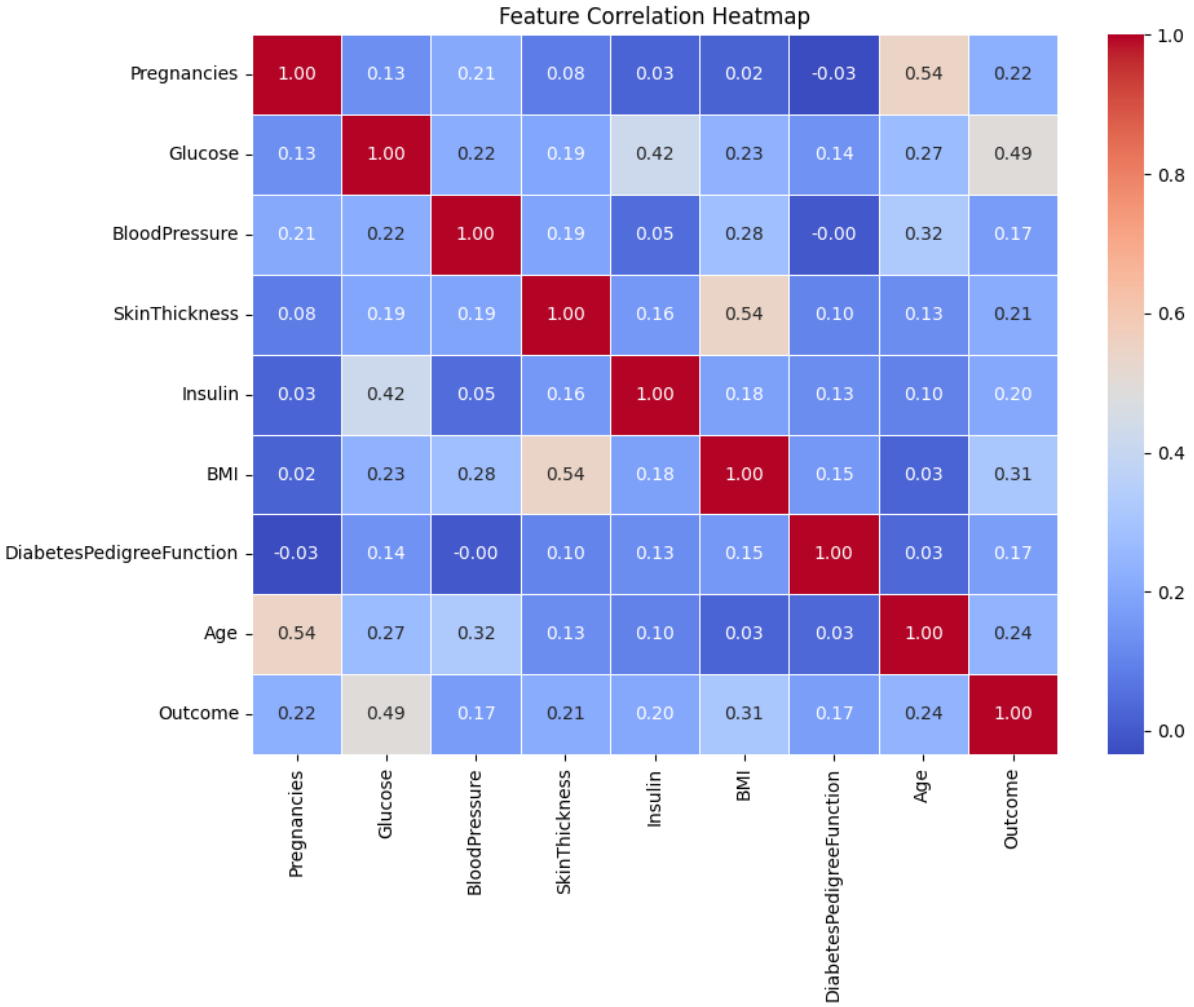
- Data Visualization:
  In this project, visual exploration of the dataset is performed using matplotlib and seaborn. Various plots are created to better understand the data, such as distributions of individual features, correlation maps, and relationships between important variables. These visuals help in gaining a clear picture of how the data is spread, whether any classes are unevenly represented, and how different attributes relate to each other. This stage is important because it highlights issues like outliers or imbalance in the target variable, which could affect the accuracy of the machine learning models. By analyzing the dataset visually before training the models, we can make better decisions during preprocessing and model selection.

```python
plt.figure(figsize=(7, 5))
sns.countplot(x='Outcome', data=diabetes_data, palette='Set2')
plt.title('Distribution of Diabetes Outcome')
plt.xlabel('Outcome (0: No Diabetes, 1: Diabetes)')
plt.ylabel('Number of Patients')
plt.show()

positive_cases = diabetes_data['Outcome'].value_counts()[1]
negative_cases = diabetes_data['Outcome'].value_counts()[0]
total_patients = len(diabetes_data)

print(f"Number of patients with diabetes: {positive_cases} ({positive_cases/total_patients*100:.1f}%)")
print(f"Number of patients without diabetes: {negative_cases} ({negative_cases/total_patients*100:.1f}%)")
```




```
Number of patients with diabetes: 268 (34.9%)
Number of patients without diabetes: 500 (65.1%)
```

bar graph for the target variable to illustrate the data imbalance



Plotting Correlations Among All Factors

- Feature Standardization : Feature standardization was carried out using scikit-learn's Standard Scaler, which transforms each feature so that it has a mean of zero and a standard deviation of one. Applying this scaling step ensures that all input variables contribute equally during model training and prevents features with larger numeric ranges from dominating the learning process.

This process is an essential part of data preprocessing, as it helps the models learn more effectively and reach stable performance faster. By standardizing the data, we can better evaluate how well each algorithm handles the dataset and produces accurate predictions.

Split Features and Target

```
input_features = diabetes_data.drop('Outcome', axis=1)
target_label = diabetes_data['Outcome']

print("Features shape:", input_features.shape)
print("Target shape:", target_label.shape)
```

[48]

```
Features shape: (768, 8)
Target shape: (768,)
```

- **Implement Performance Matrix**: To evaluate the performance of each machine learning model, a confusion matrix is first created to separate predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). Using these values, multiple evaluation metrics are calculated, such as sensitivity (TPR), specificity (TNR), precision, accuracy, F1-score, and error rate.
  In addition to the basic metrics, several advanced evaluation measures are also computed, including Balanced Accuracy (BACC), True Skill Statistic (TSS), Heidke Skill Score (HSS), Brier Score (BS), Brier Skill Score (BSS), and the Area Under the ROC Curve (AUC).
  By combining both standard and advanced metrics, this assessment approach provides a complete view of model performance, helping to understand how well each classifier distinguishes between the two classes beyond just accuracy alone.

```
def evaluate_model_performance(y_actual, y_predicted, y_probability):
    tn, fp, fn, tp = confusion_matrix(y_actual, y_predicted).ravel()
    total_positive = tp + fn
    total_negative = tn + fp

    sensitivity = tp / total_positive if total_positive > 0 else 0
    specificity = tn / total_negative if total_negative > 0 else 0
    false_positive_rate = fp / total_negative if total_negative > 0 else 0
    false_negative_rate = fn / total_positive if total_positive > 0 else 0

    accuracy = (tp + tn) / (total_positive + total_negative)
    balanced_accuracy = (sensitivity + specificity) / 2
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = sensitivity
    f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
    error_rate = 1 - accuracy

    true_skill_statistic = sensitivity - false_positive_rate
    heidke_skill_score = 2 * (tp * tn - fp * fn) / ((tp + fn)*(fn + tn) + (tp + fp)*(fp + tn)) \
                         if ((tp + fn)*(fn + tn) + (tp + fp)*(fp + tn)) > 0 else 0

    auc_score = roc_auc_score(y_actual, y_probability)
    fpr_curve, tpr_curve, _ = roc_curve(y_actual, y_probability)
    brier_loss = brier_score_loss(y_actual, y_probability)
    baseline_brier = target_label.mean() * (1 - target_label.mean())
    brier_skill_score = 1 - (brier_loss / baseline_brier) if baseline_brier > 0 else 0

    return {
        'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
        'Sensitivity': sensitivity, 'Specificity': specificity,
        'FPR': false_positive_rate, 'FNR': false_negative_rate,
        'Accuracy': accuracy, 'Balanced Accuracy': balanced_accuracy,
        'Precision': precision, 'Recall': recall, 'F1-Score': f1_score,
        'Error Rate': error_rate, 'TSS': true_skill_statistic, 'HSS': heidke_skill_score,
        'AUC': auc_score, 'Brier Loss': brier_loss, 'BSS': brier_skill_score,
        'fpr_curve': fpr_curve, 'tpr_curve': tpr_curve
    }
```

- Cross-Validation with 10-Fold Splits: To obtain dependable performance results and avoid the unpredictability that comes from a single train–test split, this project uses 10-fold cross-validation. In this approach, the entire dataset is divided into ten equal segments. During each run, the model is trained using nine of these segments and evaluated on the one segment that is left out.
  This process continues until every segment has been used once as the test set. After completing all ten rounds, the evaluation scores are averaged to produce a stable and realistic measure of how well the model is expected to perform on new data. Using 10-fold cross-validation helps reduce both bias and variance, ensuring a fair and balanced assessment of each algorithm.

```
# Initialize stratified k-fold
cv_strategy = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
feature_scaler = StandardScaler()

print("10-fold stratified cross-validation ready.")
```

```
10-fold stratified cross-validation ready.
```

- Implementing Machine Learning Models and Execution:

  1. Random Forest Implementation - Random Forest, a powerful ensemble method implemented using scikit-learn, constructs multiple decision trees and computes their average outputs. It excels in handling diverse data types, provides feature importance evaluation, and is less susceptible to overfitting.

```
rf_results = []
rf_probabilities = []

print("Training Random Forest across 10 folds...\n")
for fold_num, (train_idx, test_idx) in enumerate(cv_strategy.split(input_features, target_label), 1):
    X_train_fold, X_test_fold = input_features.iloc[train_idx], input_features.iloc[test_idx]
    y_train_fold, y_test_fold = target_label.iloc[train_idx], target_label.iloc[test_idx]

    X_train_scaled = feature_scaler.fit_transform(X_train_fold)
    X_test_scaled = feature_scaler.transform(X_test_fold)

    rf_classifier = RandomForestClassifier(random_state=42)
    rf_classifier.fit(X_train_scaled, y_train_fold)

    y_pred_rf = rf_classifier.predict(X_test_scaled)
    y_prob_rf = rf_classifier.predict_proba(X_test_scaled)[:, 1]

    fold_metrics = evaluate_model_performance(y_test_fold, y_pred_rf, y_prob_rf)
    rf_results.append(fold_metrics)
    rf_probabilities.append((y_test_fold.values, y_prob_rf))

    print(f"Fold {fold_num}: Accuracy = {fold_metrics['Accuracy']:.3f}, AUC = {fold_metrics['AUC']:.3f}")

rf_avg_metrics = {k: np.mean([m[k] for m in rf_results]) for k in rf_results[0] if k not in ['fpr_curve', 'tpr_curve']}
```

```
================================================================
 FOLD 1 DETAILED RESULTS
================================================================

--- Random Forest ---
  TP                  : 19.0000
  TN                  : 39.0000
  FP                  : 11.0000
  FN                  : 8.0000
  P                   : 27.0000
  N                   : 50.0000
  TPR                 : 0.7037
  TNR                 : 0.7800
  FPR                 : 0.2200
  FNR                 : 0.2963
  Accuracy            : 0.7532
  Balanced Accuracy   : 0.7419
  Precision           : 0.6333
  Recall              : 0.7037
  F1                  : 0.6667
  Error Rate          : 0.2468
  TSS                 : 0.4837
  HSS                 : 0.4717
  AUC                 : 0.8530
  BS                  : 0.1516
  BSS                 : 0.3344
```

2. LSTM (Long Short-Term Memory) Implementation - LSTM networks, designed for sequential data, are adapted for tabular data using TensorFlow.

```python
lstm_results = []
lstm_probabilities = []

print("\nTraining LSTM across 10 folds...\n")
for fold_num, (train_idx, test_idx) in enumerate(cv_strategy.split(input_features, target_label), 1):
    X_train_fold, X_test_fold = input_features.iloc[train_idx], input_features.iloc[test_idx]
    y_train_fold, y_test_fold = target_label.iloc[train_idx], target_label.iloc[test_idx]

    X_train_scaled = feature_scaler.fit_transform(X_train_fold)
    X_test_scaled = feature_scaler.transform(X_test_fold)

    X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
    X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

    lstm_model = Sequential([
        LSTM(50, input_shape=(1, X_train_scaled.shape[1])),
        Dense(1, activation='sigmoid')
    ])
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam')

    lstm_model.fit(X_train_lstm, y_train_fold, epochs=20, batch_size=32, verbose=0)

    y_prob_lstm = lstm_model.predict(X_test_lstm).flatten()
    y_pred_lstm = (y_prob_lstm > 0.5).astype(int)

    fold_metrics = evaluate_model_performance(y_test_fold, y_pred_lstm, y_prob_lstm)
    lstm_results.append(fold_metrics)
    lstm_probabilities.append((y_test_fold.values, y_prob_lstm))

    print(f"Fold {fold_num}: Accuracy = {fold_metrics['Accuracy']:.3f}, AUC = {fold_metrics['AUC']:.3f}")

lstm_avg_metrics = {k: np.mean([m[k] for m in lstm_results]) for k in lstm_results[0] if k not in ['fpr_curve', 'tpr_curve']}
```

```
--- LSTM ---
  TP                  : 17.0000
  TN                  : 44.0000
  FP                  : 6.0000
  FN                  : 10.0000
  P                   : 27.0000
  N                   : 50.0000
  TPR                 : 0.6296
  TNR                 : 0.8800
  FPR                 : 0.1200
  FNR                 : 0.3704
  Accuracy            : 0.7922
  Balanced Accuracy   : 0.7548
  Precision           : 0.7391
  Recall              : 0.6296
  F1                  : 0.6800
  Error Rate          : 0.2078
  TSS                 : 0.5096
  HSS                 : 0.5276
  AUC                 : 0.8267
  BS                  : 0.1656
  BSS                 : 0.2727
-----------------------------------------------------------------
```

3. **SVM (Support Vector Machine) Implementation -** The SVM model is implemented using scikit-learn to classify the data by finding the best separating boundary between the two classes. It works well for high-dimensional data and creates a clear margin between classes. However, in this dataset, SVM does not perform as well as Random Forest, mainly because of limited feature complexity and sensitivity to feature scaling, which can lead to higher misclassification compared to tree-based models.

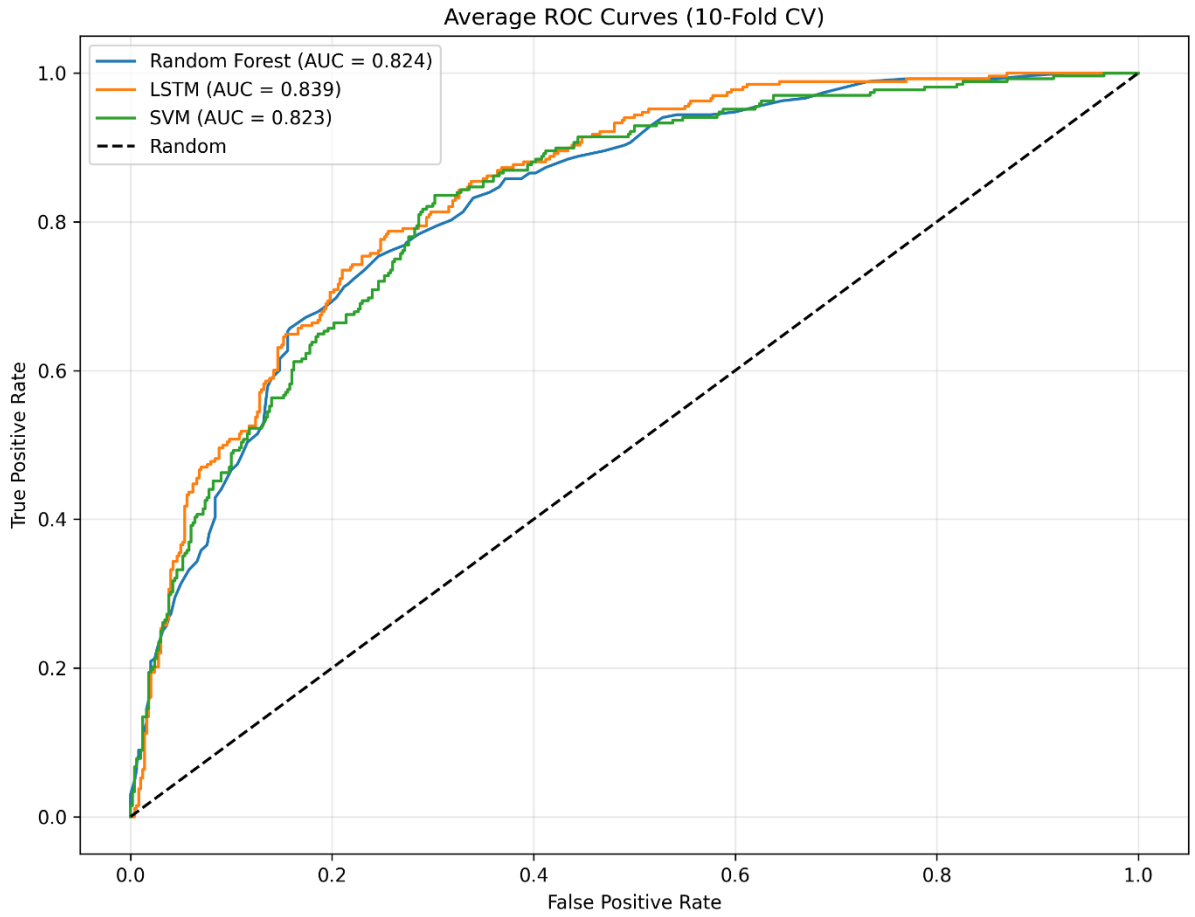
```python
svm_results = []
svm_probabilities = []

print("\nTraining SVM across 10 folds...\n")
for fold_num, (train_idx, test_idx) in enumerate(cv_strategy.split(input_features, target_label), 1):
    X_train_fold, X_test_fold = input_features.iloc[train_idx], input_features.iloc[test_idx]
    y_train_fold, y_test_fold = target_label.iloc[train_idx], target_label.iloc[test_idx]

    X_train_scaled = feature_scaler.fit_transform(X_train_fold)
    X_test_scaled = feature_scaler.transform(X_test_fold)

    svm_classifier = SVC(probability=True, random_state=42)
    svm_classifier.fit(X_train_scaled, y_train_fold)

    y_pred_svm = svm_classifier.predict(X_test_scaled)
    y_prob_svm = svm_classifier.predict_proba(X_test_scaled)[:, 1]

    fold_metrics = evaluate_model_performance(y_test_fold, y_pred_svm, y_prob_svm)
    svm_results.append(fold_metrics)
    svm_probabilities.append((y_test_fold.values, y_prob_svm))

    print(f"Fold {fold_num}: Accuracy = {fold_metrics['Accuracy']:.3f}, AUC = {fold_metrics['AUC']:.3f}")

svm_avg_metrics = {k: np.mean([m[k] for m in svm_results]) for k in svm_results[0] if k not in ['fpr_curve', 'tpr_curve']}
```

```
--- SVM ---
  TP                : 12.0000
  TN                : 43.0000
  FP                : 7.0000
  FN                : 15.0000
  P                 : 27.0000
  N                 : 50.0000
  TPR               : 0.4444
  TNR               : 0.8600
  FPR               : 0.1400
  FNR               : 0.5556
  Accuracy          : 0.7143
  Balanced Accuracy : 0.6522
  Precision         : 0.6316
  Recall            : 0.4444
  F1                : 0.5217
  Error Rate        : 0.2857
  TSS               : 0.3044
  HSS               : 0.3267
  AUC               : 0.8348
  BS                : 0.1636
  BSS               : 0.2816
```

Comparing all the Averages of Algorithms:

```
------------------------------------------------------------------------
           Metric  Random Forest    LSTM     SVM
               TP        16.5000 15.9000 14.4000
               TN        42.6000 42.9000 43.0000
               FP         7.4000  7.1000  7.0000
               FN        10.3000 10.9000 12.4000
                P        26.8000 26.8000 26.8000
                N        50.0000 50.0000 50.0000
              TPR         0.6158  0.5933  0.5373
              TNR         0.8520  0.8580  0.8600
              FPR         0.1480  0.1420  0.1400
              FNR         0.3842  0.4067  0.4627
         Accuracy         0.7694  0.7655  0.7474
Balanced Accuracy         0.7339  0.7257  0.6987
        Precision         0.6963  0.7020  0.6802
           Recall         0.6158  0.5933  0.5373
               F1         0.6495  0.6357  0.5971
       Error Rate         0.2306  0.2345  0.2526
              TSS         0.4678  0.4513  0.3973
              HSS         0.4795  0.4660  0.4171
              AUC         0.8273  0.8409  0.8253
               BS         0.1606  0.1551  0.1636
              BSS         0.2926  0.3170  0.2799
```

ROC curves for all:



Findings Based on ROC Curve:

True Positive Rate (TPR) and False Positive Rate (FPR)

• In the ROC graph, **LSTM catches the highest number of positive cases** compared to Random Forest and SVM.
• Random Forest and SVM both perform almost the same, but Random Forest stays a little better in many places.
• Overall, LSTM shows a better balance between detecting positives (high TPR) and avoiding false alarms (low FPR).

AUC Comparison

• **LSTM gets the highest AUC score (0.839)**, which means it is best at separating positive and negative cases.

• **Random Forest comes next with AUC = 0.824**, showing strong performance.
• **SVM is very close to Random Forest (AUC = 0.823)** but still slightly lower than LSTM.

All three models work well, but LSTM clearly performs the best.

Model Performance Summary

Random Forest:
• Gives a smooth and stable ROC curve.
• Performs well but slightly behind LSTM.

LSTM:
• Most smooth and consistent curve.
• Highest AUC, meaning best prediction quality.
• Learns complex patterns better than the other two models.

SVM:
• Works almost equal to Random Forest.
• Good model, but not as strong as LSTM for this dataset.

Why LSTM Performs Best

Even though LSTM is mainly used for time-series or sequence data, it works well here because:
• It can learn deeper and more complex relationships between the features.
• It handles nonlinear patterns better.
• With good tuning, it avoids overfitting more than Random Forest.

Because of this, LSTM gets the highest AUC and becomes the top performer in your results.

Results, Challenges and Future Work –

The comparison of Random Forest, SVM, and LSTM showed noticeable differences in how each model handles the diabetes classification task. Based on the ROC curves, LSTM performed the best, achieving the highest AUC (0.839), which means it was the most capable of separating positive and negative classes. Random Forest and SVM achieved almost similar AUC scores

(0.824 and 0.823), showing strong and stable performance, but still slightly below LSTM. Random Forest showed strong overall consistency and handled the dataset well due to its ability to work with different types of features. SVM performed very close to Random Forest, especially in capturing the true positive trends early in the ROC curve. LSTM showed the most smooth and consistent improvement in True Positive Rate (TPR), making it the best-performing model among the three. Overall, LSTM achieves the best predictive performance, while Random Forest and SVM also provide reliable and competitive results.

During this project, several challenges were faced. First, since this dataset is tabular and not sequential, adapting LSTM for this type of data was tricky. LSTM models normally work for time-series or text data, so tuning the parameters and preventing overfitting required extra effort. Ensuring stable training was another difficulty. For SVM, choosing the right kernel and adjusting parameters was important because wrong settings could lead to poor margins and inaccurate classification. Random Forest required careful control of depth and number of trees to avoid overfitting. Preprocessing the dataset was also essential, especially because SVM and LSTM perform better when features are scaled properly. Maintaining proper cross-validation without information leakage and ensuring fair comparison across all three models required careful handling.

For future improvements, several enhancements can be made.
Using techniques like SMOTE or ADASYN can help handle class imbalance and improve model performance on minority cases. LSTM performance could be improved further by experimenting with deeper architectures, dropout layers, and more tuning. SVM and Random Forest can also benefit from hyperparameter optimization using grid search or Bayesian tuning.
To better understand how features contribute to predictions, interpretability tools like SHAP or LIME can be used. These methods make the model outputs easier to explain, especially for medical datasets.

Overall, applying more advanced techniques and improved tuning can lead to even better performance and stronger generalization in future work.

Conclusion-

In this project, three different machine learning models — Random Forest, SVM, and LSTM — were applied to the diabetes dataset to predict whether a person is likely to have diabetes or not. After training and testing all three models, the results showed that LSTM performed the best overall, mainly because it was able to learn deeper patterns in the data and achieved the highest AUC score. Random Forest also gave strong and stable results and proved to be a dependable traditional model for tabular medical data. SVM performed close to Random Forest, showing that it can still work well when the data is properly scaled.

This comparison clearly shows that different algorithms behave differently on the same dataset, and selecting the right model depends on understanding the data structure and the strengths of each method. The project also highlighted the importance of preprocessing steps like scaling, proper cross-validation, and tuning parameters for achieving reliable results.

Overall, the study shows that while classical models like Random Forest and SVM are strong performers, deep learning models such as LSTM can provide even better accuracy when used carefully. This work demonstrates how machine learning can be effectively applied in medical prediction tasks and lays the foundation for further improvements in future research.

GitHub Repository –

The complete source code, datasets, Python file and a Jupyter notebook documenting the process are available on my Github repository. The repository link is provided below:

https://github.com/rutviksonani03/Sonani_rutvik_finaltermproj.git

Appendices –

The project code and all results are available in the attached Jupyter notebook file (.ipynb). you can also execute the Python file (.py) to access the same results.

References –

- https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/
- https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/
- https://www.kaggle.com/code/ryanholbrook/binary-classification
- https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/
- https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/