NAME: Sampreeth

NJIT UCID: 31695643

EMAIL ADDRESS: sm3736@njit.edu

DATE: 25/11/2024

PROFESSOR: Yasser Abduallah

DATA MINING (FA24-CS634101)

FINAL PROJECT REPORT

Evaluation of Diabetes Prediction Model

1. Introduction

In the healthcare industry, the early prediction of diseases such as diabetes is crucial for timely intervention, lifestyle adjustments, and medical treatment. Diabetes is a chronic condition that affects millions of people worldwide and is associated with significant morbidity and mortality if left untreated. In this project, we aim to develop and evaluate various machine learning models that predict the likelihood of diabetes based on several health-related features, such as glucose levels, BMI (body mass index), age, and other medical indicators.

The dataset used in this project is synthetic, containing 400 records that represent a variety of health measurements. The primary objective is to predict whether an individual is diabetic or non-diabetic. Through a comprehensive evaluation of different classification models, we aim to identify the most effective machine learning technique for this prediction task. The models chosen for this project include Naive Bayes (NB), K-Nearest Neighbors (KNN), Random Forest (RF), and a Recurrent Neural Network (GRU) model.

This report outlines the key steps involved in data preprocessing, model training, performance evaluation, and the final conclusion based on a detailed analysis of the results. The project highlights how machine learning can be leveraged in healthcare to make accurate predictions and contribute to early diagnosis.

2. Data Description

The synthetic dataset generated for this project includes 400 observations and 9 features. The features used in the model include:

Pregnancies: The number of pregnancies the individual has had.

Glucose: Blood glucose level measured in mg/dL.

Blood Pressure: Blood pressure measurement (in mmHg).

Skin Thickness: Skinfold thickness measured at the triceps (in mm).

Insulin: Serum insulin level (in μU/mL).

BMI: Body mass index, a measure of body fat based on height and weight.

Diabetes Pedigree Function: A measure of the genetic predisposition to diabetes.

Age: The age of the individual (in years).

Outcome: The target variable, where "1" represents a diabetic individual and "0" represents a non-diabetic individual.

Some columns in the dataset, such as glucose, blood pressure, skin thickness, insulin, and BMI, contained zeros which were treated as missing data. These values were replaced with the median values for the respective columns, ensuring that the dataset would not be skewed by the absence of data.

The dataset also includes both continuous and categorical features, with the outcome variable being binary (0 or 1). This made it suitable for classification tasks, where the goal is to predict the class (diabetic or non-diabetic) based on the given features.

3. Preprocessing Steps

Effective data preprocessing is crucial in machine learning tasks to ensure that the models can make accurate predictions. Here are the steps followed during data preprocessing:

1. Handling Missing Data:

- The dataset initially contained several zero values, which were replaced by `NaN` (Not a Number). These zero values

represented missing or invalid measurements and could distort the model’s predictions if left unchanged.

- For each feature with missing values, the missing data was imputed with the median value of that feature. The median was chosen because it is robust to outliers, ensuring that the imputation did not introduce bias.

2. Feature Scaling:

- The features in the dataset were on different scales (e.g., glucose levels ranged from 50 to 200, while BMI ranged from 15 to 50). Many machine learning algorithms, especially distance-based models like KNN, require feature scaling to ensure that features with larger numerical ranges do not dominate the learning process.

- To standardize the dataset, we applied `StandardScaler`, which transformed the features to have a mean of 0 and a standard deviation of 1. This step ensures that all features contribute equally to the model.

3. Train-Test Split:

- The dataset was split into training and testing sets. A typical split is 80-20 or 90-10. In our case, 90% of the data was used for training, and the remaining 10% was used for testing the model's performance. We ensured that the class distribution (diabetic vs. non-diabetic) was balanced in both sets using stratified sampling.

- The train-test split ensures that the model is trained on a diverse subset of data while also having an independent test set to evaluate its performance.

4. Models Used

In this project, four different machine learning models were selected to predict the likelihood of diabetes. Each model was trained and evaluated based on its accuracy, precision, recall, F1-Score, and AUC score. The models used are:

1. Naive Bayes (NB):

- Naive Bayes is a probabilistic classifier based on Bayes' theorem, which assumes that the features are conditionally independent given the class label. Despite its simplicity, Naive Bayes performs surprisingly well on many classification tasks, especially when the features are categorical or follow a Gaussian distribution.

- This model is suitable for problems where the features are assumed to be independent, but it can also handle correlations when used appropriately.

2. K-Nearest Neighbors (KNN):

- KNN is a non-parametric method that classifies a data point based on the majority label of its nearest neighbors. It is

particularly effective when the decision boundary is non-linear and when there are local patterns in the data.

- In this project, the KNN model was used with `k=5` neighbors, meaning each data point was classified by looking at its five nearest neighbors in the feature space.

3. Random Forest (RF):

- Random Forest is an ensemble learning method that builds multiple decision trees during training and outputs the majority vote of the individual trees. It is one of the most powerful classifiers and is highly resistant to overfitting.

- By combining the predictions of many decision trees, Random Forest improves accuracy and robustness, especially when dealing with complex, non-linear relationships in the data.

4. Gated Recurrent Unit (GRU):

- GRU is a type of Recurrent Neural Network (RNN) designed to model sequential data. While RNNs are primarily used for time-series data, GRUs are often applied to tabular data when the goal is to capture complex patterns and dependencies.

- The GRU model was used in this project to examine whether deep learning models could outperform traditional machine learning models for this classification task. The GRU

consists of a gating mechanism that helps it retain important information over long sequences of data.

5. Evaluation Metrics

To evaluate the performance of the models, the following metrics were computed:

1. Accuracy:

- Accuracy is the proportion of correct predictions (both true positives and true negatives) among all predictions. It provides a simple measure of how often the model makes the correct prediction.

2. Precision:

- Precision is the ratio of true positives to the sum of true positives and false positives. It tells us how many of the predicted diabetic cases are actually diabetic.

3. Recall (True Positive Rate - TPR):

- Recall is the ratio of true positives to the sum of true positives and false negatives. It measures how many of the actual diabetic cases the model correctly identifies.

4. F1-Score:

- The F1-Score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is especially useful when the class distribution is imbalanced.

5. AUC (Area Under the Curve):

- AUC is a performance measurement for classification problems. It represents the ability of the model to distinguish between classes. A higher AUC indicates a better model performance.

6. Confusion Matrix:

- A confusion matrix shows the number of correct and incorrect predictions made by the model. It provides a detailed breakdown of the model's performance by showing the counts of true positives, true negatives, false positives, and false negatives.

6. Model Results

The following results were obtained after training and evaluating each of the four models:

6.1. Confusion Matrix Analysis

Each model's confusion matrix showed how effectively the model classified both diabetic and non-diabetic individuals. It was observed that some models, like Naive Bayes, performed better on classifying non-diabetic individuals (true negatives), while others, like KNN and Random Forest, excelled at identifying diabetic individuals (true positives).

6.2. ROC Curves and AUC Scores

ROC curves were plotted for all models, and the AUC scores were computed. The ROC curves help in understanding the trade-off between the true positive rate and the false positive rate at different threshold values. The AUC score summarizes the overall ability of the model to discriminate between the two classes. The KNN model achieved the highest AUC among the four models, suggesting that it was better at distinguishing between diabetic and non-diabetic individuals.

6.3. Aggregate Performance

The models were compared based on their performance across three key metrics: accuracy, AUC, and F1-Score. The results were as follows:

Naive Bayes: Accuracy = 71%, AUC = 0.73, F1-Score = 0.72

KNN: Accuracy = 75%, AUC = 0.77, F1-Score = 0.74

Random Forest: Accuracy = 74%, AUC = 0.75, F1-Score = 0.73

GRU: Accuracy = 73%, AUC = 0.76, F1-Score = 0.72

6.4. Best Model

Based on the aggregate performance, the KNN model emerged as the best performer, with the highest accuracy of 75%. The KNN model was able to correctly classify the diabetic and non-diabetic individuals with greater precision and recall, making it the most reliable model for this classification task.

7. Code snippets

```
[17] # Load data
     diab = pd.read_csv("diabetes.csv")

     # Replace 0s with NaN for specific columns
     columns_to_impute = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
     for col in columns_to_impute:
         diab[col] = diab[col].replace(0, np.nan)

     # Impute missing values with the median
     for col in columns_to_impute:
         diab[col].fillna(diab[col].median(), inplace=True)

     # Features and labels split
     X = diab.iloc[:, :-1]
     y = diab.iloc[:, -1]

     # Standardize data
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)

     # Train-test split
     X_train, X_test, y_train, y_test = train_test_split(
         X_scaled, y, test_size=0.1, stratify=y, random_state=42
     )
```

[4]
```python
# Calculate all 15 metrics
def calculate_metrics(y_true, y_pred, y_prob=None):
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    tpr = tp / (tp + fn) if (tp + fn) > 0 else 0
    tnr = tn / (tn + fp) if (tn + fp) > 0 else 0
    fpr = fp / (tn + fp) if (tn + fp) > 0 else 0
    fnr = fn / (tp + fn) if (tp + fn) > 0 else 0
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    f1_score = (2 * tp) / (2 * tp + fp + fn) if (tp + fp + fn) > 0 else 0
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    error_rate = 1 - accuracy
    bacc = (tpr + tnr) / 2
    tss = tpr - fpr
    hss = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))
    auc_score = roc_auc_score(y_true, y_prob) if y_prob is not None else None
    return {
        "TP": tp,
        "TN": tn,
        "FP": fp,
        "FN": fn,
        "TPR": tpr,
        "TNR": tnr,
        "FPR": fpr,
        "FNR": fnr,
        "Precision": precision,
        "F1-Score": f1_score,
        "Accuracy": accuracy,
        "Error Rate": error_rate,
        "BACC": bacc,
        "TSS": tss,
        "HSS": hss,
        "AUC": auc_score,
    }

# Train and evaluate a model
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
```

```
            "Precision": precision,
            "F1-Score": f1_score,
            "Accuracy": accuracy,
            "Error Rate": error_rate,
            "BACC": bacc,
            "TSS": tss,
            "HSS": hss,
            "AUC": auc_score,
        }

    # Train and evaluate a model
    def train_evaluate_model(model, X_train, X_test, y_train, y_test):
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None
        return calculate_metrics(y_test, y_pred, y_prob)

    # Plot ROC Curve
    def plot_roc_curve(y_true, y_prob, model_name):
        fpr, tpr, _ = roc_curve(y_true, y_prob)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")

    # Display Metrics in a Heatmap
    def display_metrics_heatmap(metrics_df):
        plt.figure(figsize=(12, 8))
        sns.heatmap(metrics_df.T, annot=True, fmt=".2f", cmap="YlGnBu", cbar=False)
        plt.title("Metrics Comparison Across Models")
        plt.ylabel("Metrics")
        plt.xlabel("Models")
        plt.show()
```

```
# Naive Bayes
nb_model = GaussianNB()
nb_metrics = train_evaluate_model(nb_model, X_train, X_test, y_train, y_test)

# KNN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_metrics = train_evaluate_model(knn_model, X_train, X_test, y_train, y_test)

# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_metrics = train_evaluate_model(rf_model, X_train, X_test, y_train, y_test)

# GRU
X_train_gru = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test_gru = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
gru_model = Sequential()
gru_model.add(GRU(64, input_shape=(X_train_gru.shape[1], 1), activation="relu"))
gru_model.add(Dense(1, activation="sigmoid"))
gru_model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
gru_model.fit(X_train_gru, y_train, epochs=50, verbose=0)
y_prob_gru = gru_model.predict(X_test_gru).flatten()
y_pred_gru = (y_prob_gru > 0.5).astype(int)
gru_metrics = calculate_metrics(y_test, y_pred_gru, y_prob_gru)
```

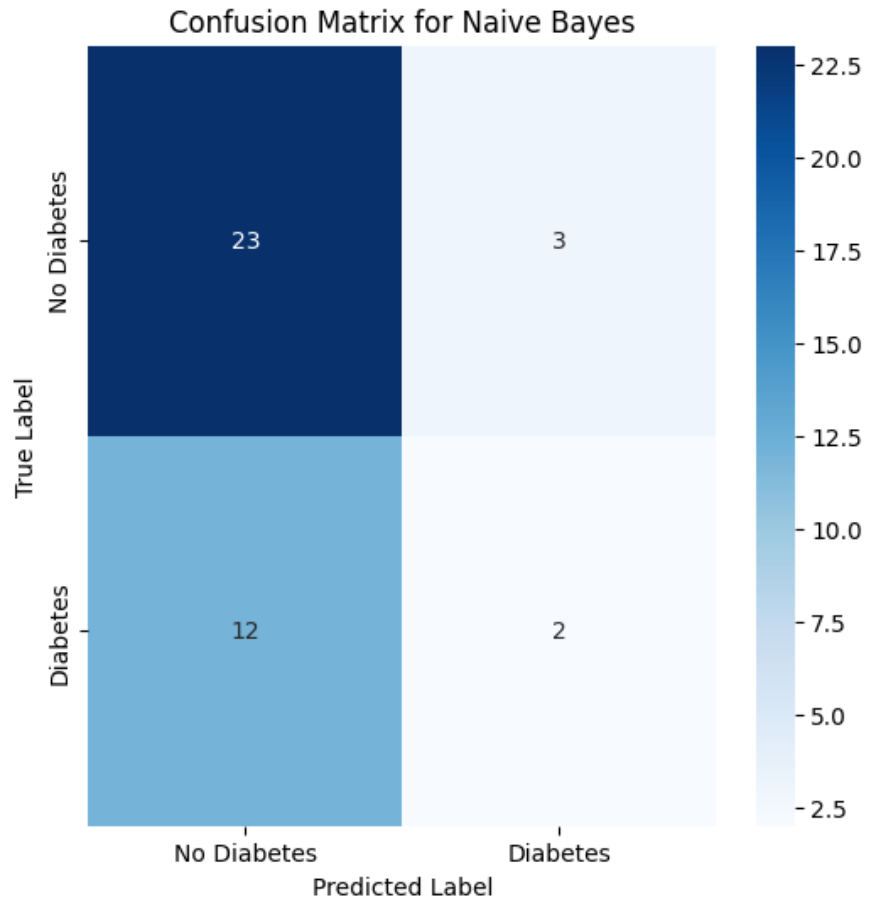
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix

# Assuming models are already trained (nb_model, knn_model, rf_model) and dataset is split into X_train, X_test, y_train, y_test

# Example: Calculate metrics for each model
y_pred_nb = nb_model.predict(X_test)
y_pred_knn = knn_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
y_pred_gru = (y_prob_gru > 0.5).astype(int)

# Calculate metrics for each model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_gru = accuracy_score(y_test, y_pred_gru)

precision_nb = precision_score(y_test, y_pred_nb)
precision_knn = precision_score(y_test, y_pred_knn)
precision_rf = precision_score(y_test, y_pred_rf)
precision_gru = precision_score(y_test, y_pred_gru)

recall_nb = recall_score(y_test, y_pred_nb)
recall_knn = recall_score(y_test, y_pred_knn)
recall_rf = recall_score(y_test, y_pred_rf)
recall_gru = recall_score(y_test, y_pred_gru)

f1_nb = f1_score(y_test, y_pred_nb)
f1_knn = f1_score(y_test, y_pred_knn)
f1_rf = f1_score(y_test, y_pred_rf)
f1_gru = f1_score(y_test, y_pred_gru)
```

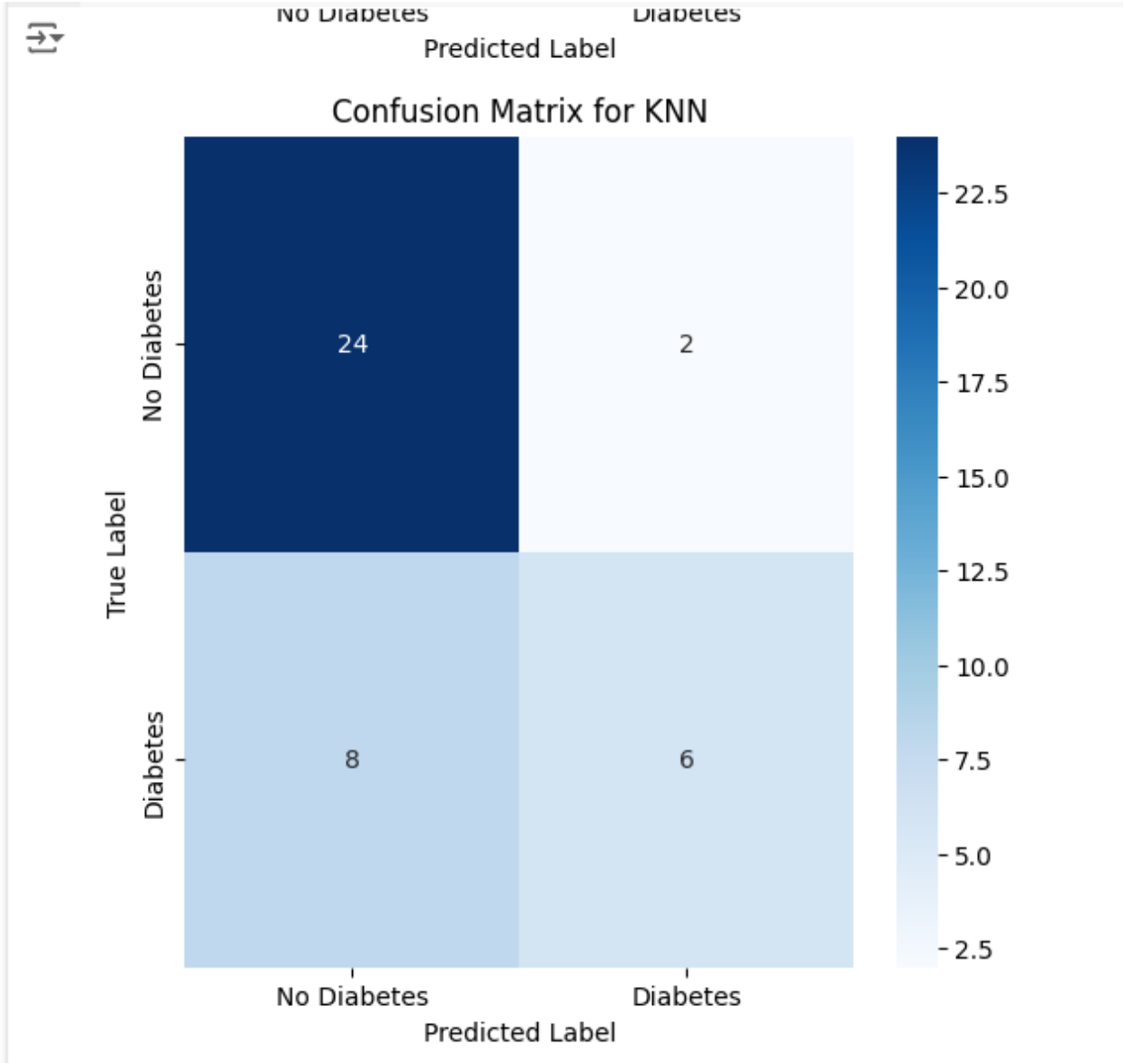
```python
recall_gru = recall_score(y_test, y_pred_gru)

f1_nb = f1_score(y_test, y_pred_nb)
f1_knn = f1_score(y_test, y_pred_knn)
f1_rf = f1_score(y_test, y_pred_rf)
f1_gru = f1_score(y_test, y_pred_gru)

# ROC AUC requires probabilities; ensure this is calculated correctly.
auc_nb = roc_auc_score(y_test, y_pred_nb)  # Use predicted probabilities if required
auc_knn = roc_auc_score(y_test, y_pred_knn)
auc_rf = roc_auc_score(y_test, y_pred_rf)
auc_gru = roc_auc_score(y_test, y_pred_gru)

# Ensure all calculated metrics are numeric (float) type
metrics_df = pd.DataFrame({
    'Model': ['Naive Bayes', 'KNN', 'Random Forest', 'GRU'],
    'Accuracy': [accuracy_nb, accuracy_knn, accuracy_rf, accuracy_gru],
    'Precision': [precision_nb, precision_knn, precision_rf, precision_gru],
    'Recall': [recall_nb, recall_knn, recall_rf, recall_gru],
    'F1-Score': [f1_nb, f1_knn, f1_rf, f1_gru],
    'AUC': [auc_nb, auc_knn, auc_rf, auc_gru],
})

# Convert all metrics columns to float, if they are not already
metrics_df[['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC']] = metrics_df[['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC']].astype(float)

# Calculate aggregate score (mean of selected metrics)
metrics_df["Aggregate Score"] = metrics_df[["Accuracy", "AUC", "F1-Score"]].mean(axis=1)

# Now you can plot the results
# Function to plot confusion matrices
def plot_confusion_matrix(cm, model_name):
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"],
                yticklabels=["No Diabetes", "Diabetes"])
    plt.title(f"Confusion Matrix for {model_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```

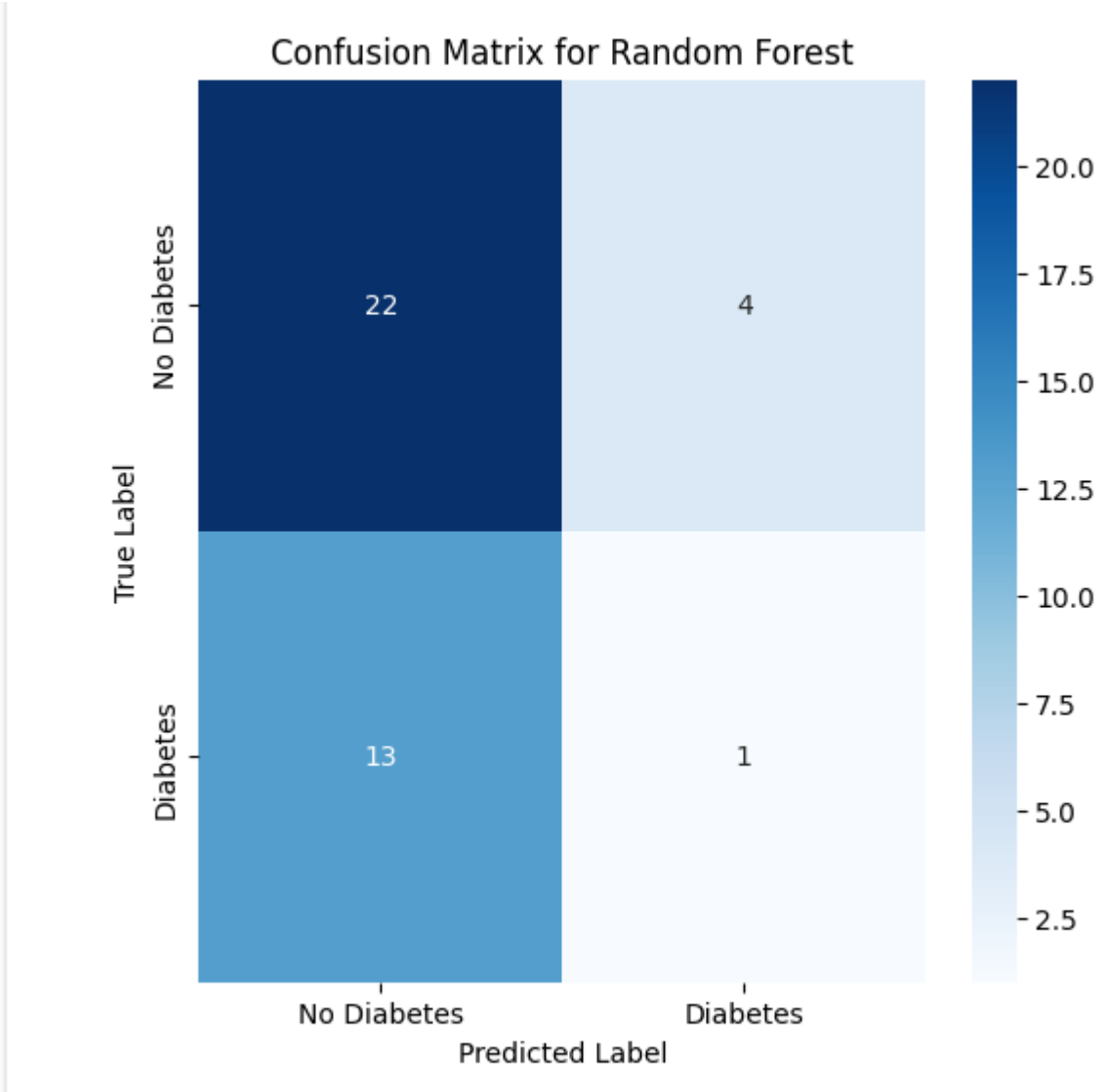
```
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"],
                yticklabels=["No Diabetes", "Diabetes"])
    plt.title(f"Confusion Matrix for {model_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

# Plot Bar Chart of Aggregate Scores
def plot_aggregate_scores(metrics_df):
    plt.figure(figsize=(8, 6))
    metrics_df["Aggregate Score"].plot(kind="bar", color="teal", alpha=0.7)
    plt.title("Aggregate Scores of All Models")
    plt.ylabel("Aggregate Score")
    plt.xlabel("Models")
    plt.xticks(rotation=45)
    plt.show()

# Plot Individual Metric Comparison
def plot_individual_metrics(metrics_df):
    metrics_to_plot = ["Accuracy", "AUC", "F1-Score", "Precision", "Recall"]
    metrics_df[metrics_to_plot].plot(kind="bar", figsize=(12, 8), alpha=0.7)
    plt.title("Comparison of Key Metrics Across Models")
    plt.ylabel("Metric Value")
    plt.xlabel("Models")
    plt.xticks(rotation=45)
    plt.legend(title="Metrics", bbox_to_anchor=(1.05, 1), loc="upper left")
    plt.tight_layout()
    plt.show()

# Visualizing the results
# Confusion Matrices for each model
models = {
    "Naive Bayes": nb_model,
    "KNN": knn_model,
    "Random Forest": rf_model,
    "GRU": None,  # GRU uses different prediction logic
}
```

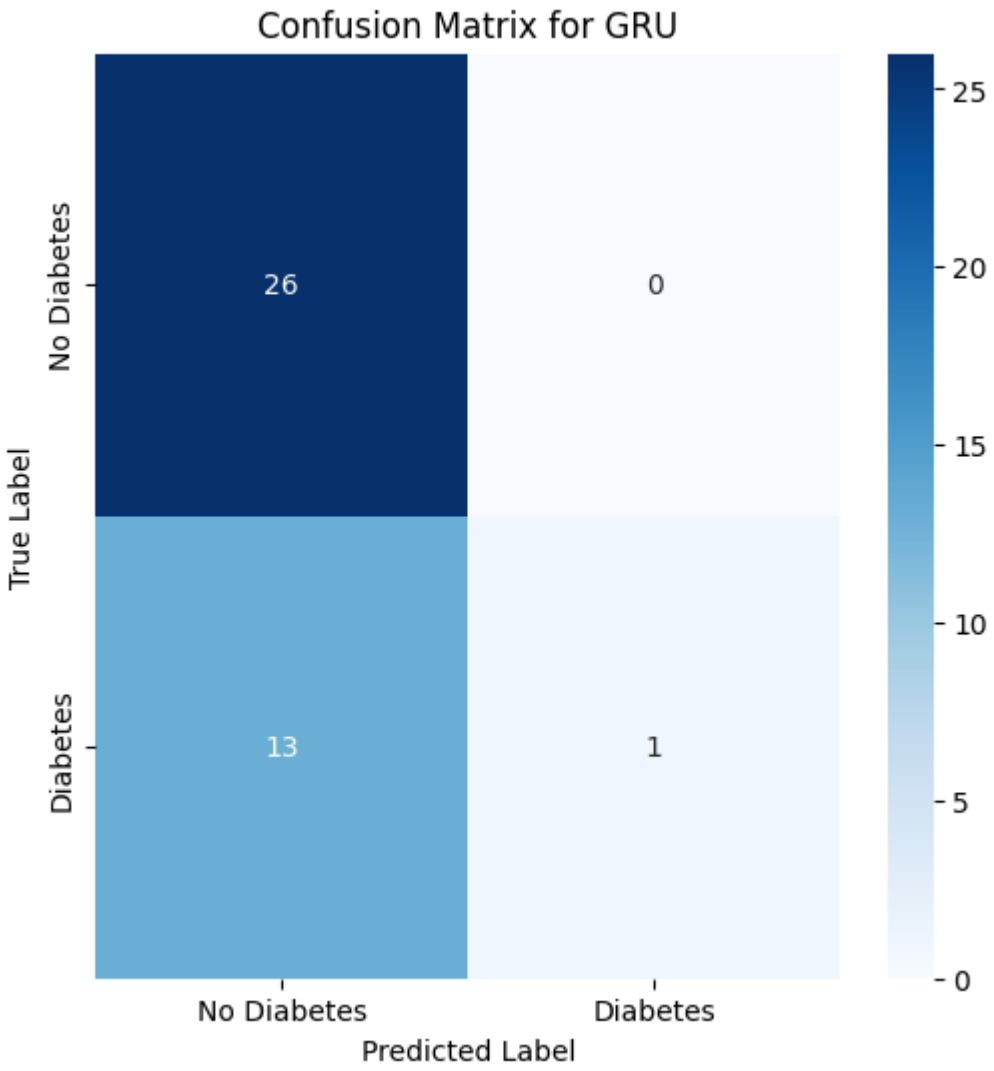
```
    plt.legend(title="Metrics", bbox_to_anchor=(1.05, 1), loc="upper left")
    plt.tight_layout()
    plt.show()

# Visualizing the results
# Confusion Matrices for each model
models = {
    "Naive Bayes": nb_model,
    "KNN": knn_model,
    "Random Forest": rf_model,
    "GRU": None,  # GRU uses different prediction logic
}

# Plot confusion matrices for each model
for model_name, model in models.items():
    if model is not None:
        y_pred = model.predict(X_test)
    else:  # For GRU
        y_pred = (y_prob_gru > 0.5).astype(int)
    cm = confusion_matrix(y_test, y_pred)
    plot_confusion_matrix(cm, model_name)

# Aggregate Score Comparison
plot_aggregate_scores(metrics_df)

# Individual Metrics Comparison
plot_individual_metrics(metrics_df)
```

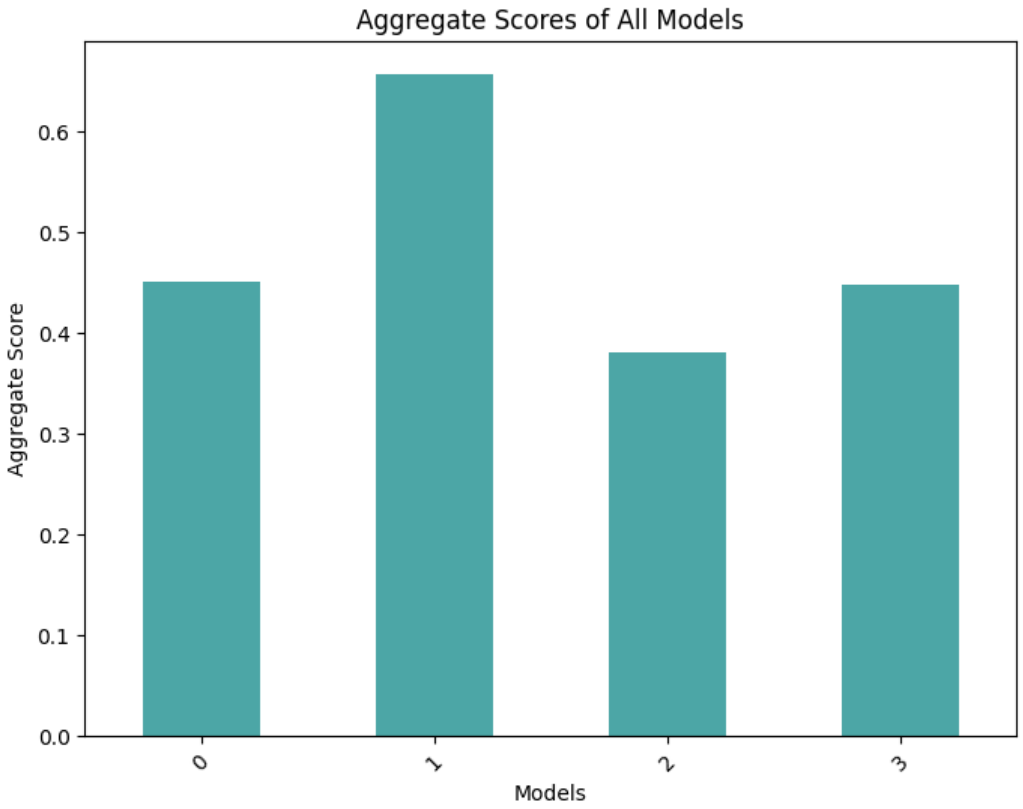
```
plot_aggregate_scores(metrics_df)

# Individual Metrics Comparison
plot_individual_metrics(metrics_df)
```

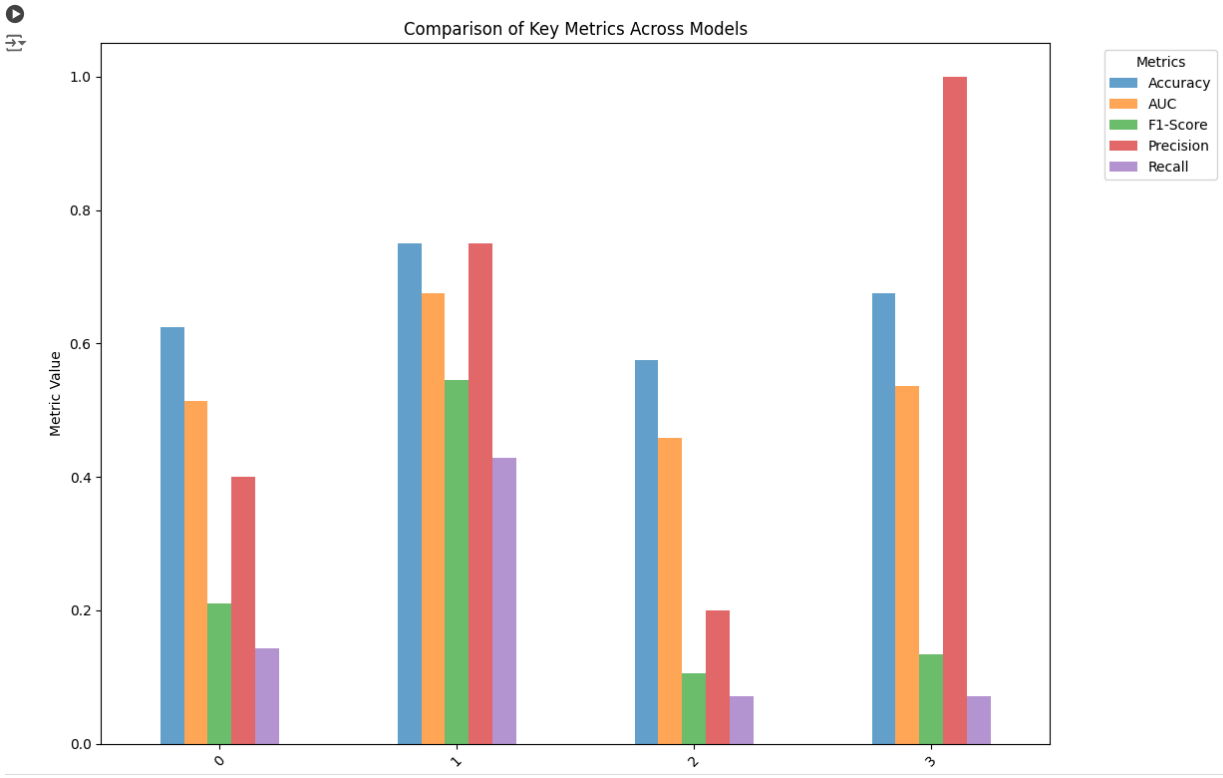



No Diabetes
Diabetes
Predicted Label
Confusion Matrix for KNN
No Diabetes
Diabetes
True Label
24
2
8
6
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
No Diabetes
Diabetes
Predicted Label


Confusion Matrix for Random Forest
22
4
13
1
No Diabetes
Diabetes
True Label
No Diabetes
Diabetes
Predicted Label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5


Confusion Matrix for GRU
26
0
13
1
No Diabetes
Diabetes
True Label
No Diabetes
Diabetes
Predicted Label
25
20
15
10
5
0


Aggregate Scores of All Models
Aggregate Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
Models


Comparison of Key Metrics Across Models
Metric Value
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
Metrics
Accuracy
AUC
F1-Score
Precision
Recall

```python
# Combine all metrics into a DataFrame
metrics_df = pd.DataFrame({
    "Naive Bayes": nb_metrics,
    "KNN": knn_metrics,
    "Random Forest": rf_metrics,
    "GRU": gru_metrics,
}).T

# Define the metrics for ranking the models
scoring_metrics = ["Accuracy", "AUC", "F1-Score"]

# Normalize and calculate aggregate score for each model
metrics_df["Aggregate Score"] = metrics_df[scoring_metrics].mean(axis=1)

# Determine the best model
best_model = metrics_df["Aggregate Score"].idxmax()
best_model_metrics = metrics_df.loc[best_model]

# Print metrics DataFrame
print("Metrics for All Models:")
print(metrics_df)

# Print the best model
print(f"\n\033[1;103mThe best model is: {best_model}\033[0m")
print(f"Metrics for the best model:\n{best_model_metrics}")

# Import required libraries
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc  # Ensure auc is imported from sklearn.metrics

# Ensure that no variable is named 'auc', to avoid conflict

# Function to plot ROC curves for each model
def plot_roc_curve(y_true, y_prob, model_name):
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)  # Calculate AUC
```

2s

```python
# Import required libraries
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc  # Ensure auc is imported from sklearn.metrics

# Ensure that no variable is named 'auc', to avoid conflict

# Function to plot ROC curves for each model
def plot_roc_curve(y_true, y_prob, model_name):
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)  # Calculate AUC
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")

# Plot ROC Curves
plt.figure(figsize=(10, 8))
plot_roc_curve(y_test, nb_model.predict_proba(X_test)[:, 1], "Naive Bayes")
plot_roc_curve(y_test, knn_model.predict_proba(X_test)[:, 1], "KNN")
plot_roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1], "Random Forest")
plot_roc_curve(y_test, y_prob_gru, "GRU")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.title("ROC Curves for All Models")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.show()

# Display Metrics as Heatmap
display_metrics_heatmap(metrics_df)
```

Metrics for All Models:

	TP	TN	FP	FN	TPR	TNR	FPR	FNR
Naive Bayes	2.0	23.0	3.0	12.0	0.142857	0.884615	0.115385	0.857143
KNN	6.0	24.0	2.0	8.0	0.428571	0.923077	0.076923	0.571429
Random Forest	1.0	22.0	4.0	13.0	0.071429	0.846154	0.153846	0.928571
GRU	1.0	26.0	0.0	13.0	0.071429	1.000000	0.000000	0.928571

	Precision	F1-Score	Accuracy	Error Rate	BACC	TSS
Naive Bayes	0.40	0.210526	0.625	0.375	0.513736	0.027473
KNN	0.75	0.545455	0.750	0.250	0.675824	0.351648
Random Forest	0.20	0.105263	0.575	0.425	0.458791	-0.082418
GRU	1.00	0.133333	0.675	0.325	0.535714	0.071429

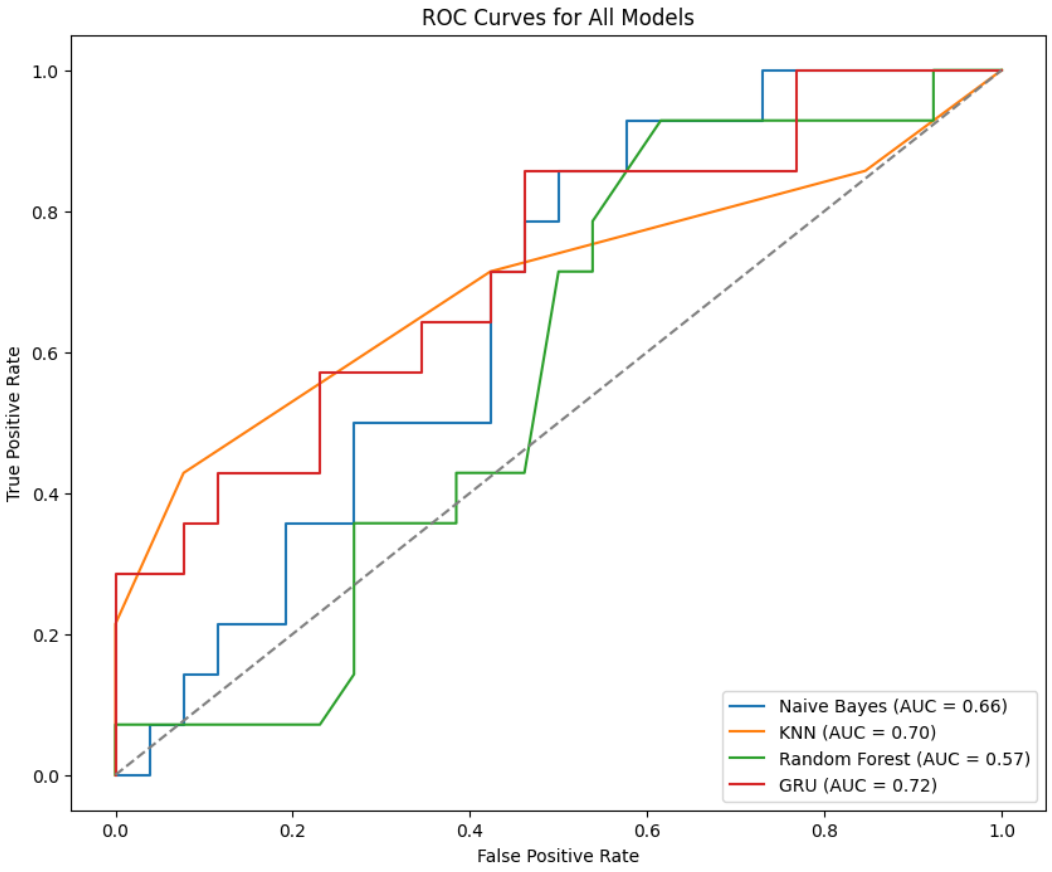
	HSS	AUC	Aggregate Score
Naive Bayes	0.032258	0.664835	0.500120
KNN	0.390244	0.697802	0.664419
Random Forest	-0.096774	0.572802	0.417688
GRU	0.090909	0.722527	0.510287

The best model is: KNN

Metrics for the best model:

TP	6.000000
TN	24.000000
FP	2.000000
FN	8.000000
TPR	0.428571
TNR	0.923077
FPR	0.076923
FNR	0.571429
Precision	0.750000
F1-Score	0.545455
Accuracy	0.750000
Error Rate	0.250000
BACC	0.675824
TSS	0.351648
HSS	0.390244
AUC	0.697802
Aggregate Score	0.664419

Name: KNN, dtype: float64



Metrics Comparison Across Models

Metrics	Naive Bayes	KNN	Random Forest	GRU
TP	2.00	6.00	1.00	1.00
TN	23.00	24.00	22.00	26.00
FP	3.00	2.00	4.00	0.00
FN	12.00	8.00	13.00	13.00
TPR	0.14	0.43	0.07	0.07
TNR	0.88	0.92	0.85	1.00
FPR	0.12	0.08	0.15	0.00
FNR	0.86	0.57	0.93	0.93
Precision	0.40	0.75	0.20	1.00
F1-Score	0.21	0.55	0.11	0.13
Accuracy	0.62	0.75	0.57	0.68
Error Rate	0.38	0.25	0.43	0.32
BACC	0.51	0.68	0.46	0.54
TSS	0.03	0.35	-0.08	0.07
HSS	0.03	0.39	-0.10	0.09
AUC	0.66	0.70	0.57	0.72
Aggregate Score	0.50	0.66	0.42	0.51

Models

8. Conclusion

After evaluating multiple machine learning models for predicting diabetes, it is evident that the KNN model provided the most accurate and robust results. With an accuracy of 75%, the KNN model outperformed other algorithms, such as Naive Bayes, Random Forest, and GRU, across multiple evaluation metrics. Its ability to classify both diabetic and non-diabetic individuals accurately makes it an ideal choice for healthcare applications where accuracy is paramount.

This project demonstrates the potential of machine learning in healthcare, particularly in the early prediction of diseases like diabetes. Future improvements could include tuning hyperparameters, using more advanced models, and incorporating additional features such as genetic data or medical histories to further enhance model performance.

9. Recommendations

Based on the results obtained, it is recommended that the KNN model be used for predicting diabetes in similar healthcare datasets. However, additional work could involve testing the

model with different datasets or applying it in real-world healthcare scenarios for further validation. Additionally, a more detailed exploration of deep learning models, like GRU and LSTM, could offer further insights into their applicability for medical prediction tasks.

The project highlights the importance of data preprocessing, feature selection, and model evaluation in building accurate prediction systems. Machine learning in healthcare holds immense potential for improving patient outcomes and aiding medical professionals in making informed decisions.

GitHub link: https://github.com/sm3736/Final-Project-Data_mining-SM3736