



MACHINE LEARNING AND NEURAL NETWORK

Diabetes Prediction



Table of Contents

Introduction	2
Dataset Description	3
Data Preprocessing	3
Model Training	8
Model Evaluation	13
Conclusion	15
References	16

List of Figures

Figure 1 Importing Libraries	3
Figure 2 Checking Data is Cleaning	4
Figure 3 Histogram of Age Distribution	5
Figure 4 Scatterplot of Age vs BMI	5
Figure 5 Boxplot of Blood Glucose Levels by Diabetes Status	6
Figure 6 Categorical Feature Encoding	7
Figure 7 Dataset Split	7
Figure 8 Training of Logistic Regression	8
Figure 9 Classification Report of Logistic Regression	9
Figure 10 Training Decision Tree Classifier	9
Figure 11 Classification Report of Decision Tree	10
Figure 12 Training Random Forest model	11
Figure 13 Classification Report of Random Forest	11
Figure 14 Training Voting Classifier	12
Figure 15 Classification Report of Voting Classifier	12
Figure 16 ROC Curves for each Model	13
Figure 17 Comparison of Actual and Predicted Values	14

Introduction

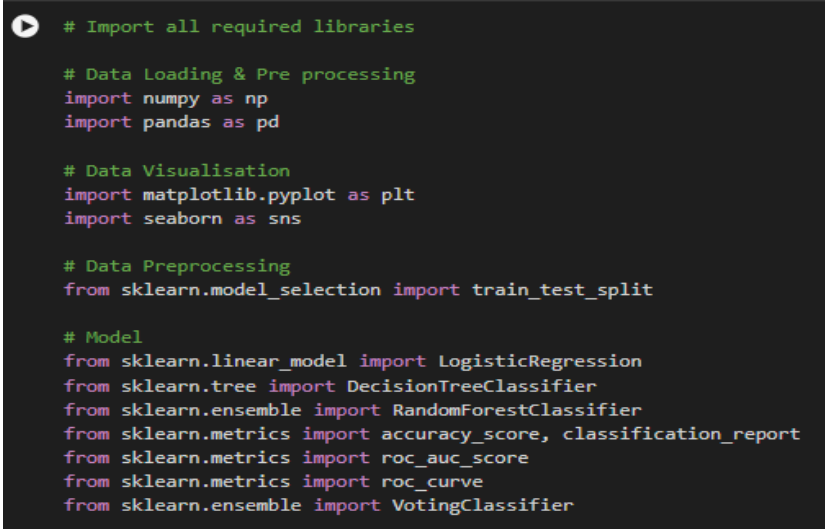
Diabetes is a serious chronic condition of blood sugar which can cause serious health problems, such as damage to the kidneys, nerves, heart etc. Early detection and prediction in case of the diabetes are extremely necessary since the wider prevalence of diabetes is on the rise around the globe. In recent years, interest in machine learning (ML) techniques has surged in the medical field because they can deal with large datasets and find it more difficult to infer patterns than the more classical approaches. Predictive tasks are particularly well suited to feature algorithms of machine learning (Dharmarathne et al., 2024). Using these models, they can analyze historical health data including age, glucose levels, and BMI blood pressure to predict what the odds are of someone developing diabetes in the future. Researchers use the diabetes dataset to apply different ML algorithms and aim to build accurate and efficient models that would help healthcare professionals to early diagnose diabetes and thus achieve better patient outcomes and more effective preventative strategies.

Dataset Description

The Diabetes Prediction Dataset available on Kaggle is a complete data set to predict the chance for an individual to develop diabetes with the aid of some health attribute-related information. Demographic, medical history and physical measurements are among the features it contains that are typically linked to diabetes risk. The dataset is of 9 attributes and each row represents an individual's data, consisting of 768 records (Mustafa, 2023). **Key features include Pregnancies (the number of pregnancies), BloodPressure (diastolic blood pressure), Insulin (serum insulin levels), BMI (body mass index), DiabetesPedigreeFunction, SkinThickness, Glucose, Age, and a target variable Outcome (whether the person has diabetes: 1 for yes, 0 for no).** In terms of modelling using machine learning algorithms, this dataset has a good value in terms of data as it uses real-world data on health that can be used to predict diabetes.

Data Preprocessing

The code begins by importing necessary libraries: NumPy, Pandas, Matplotlib, Seaborn, and various modules from scikit-learn for model building, evaluation, and visualization.



```
# Import all required libraries

# Data Loading & Pre processing
import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Data Preprocessing
from sklearn.model_selection import train_test_split

# Model
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.ensemble import VotingClassifier
```

Figure 1 Importing Libraries

Pandas is utilized to read the CSV file into a DataFrame named df. The head() and tail() functions are used to display the first and last five rows of the data. The function info() provides a summary of the DataFrame, including non-null counts and data types, while isnull().sum() and sns.heatmap(df.isnull()) check for missing values and visually represent them.

```
# Check Data Is Cleaned or Not
df.isnull().sum()

0
gender      0
age         0
hypertension 0
heart_disease 0
smoking_history 0
bmi         0
HbA1c_level 0
blood_glucose_level 0
diabetes     0
dtype: int64
```

Figure 2 Checking Data is Cleaning

The code presented above is used to check dataset contains null values or not. `Df.isnull().sum()` function is utilized to check null values in every feature of the dataset. The output shows that there is no null value available in the dataset.

Exploratory Data Analysis (EDA):

The code then conducts extensive EDA using Seaborn and Matplotlib to visualize the data distributions and relationships between features:

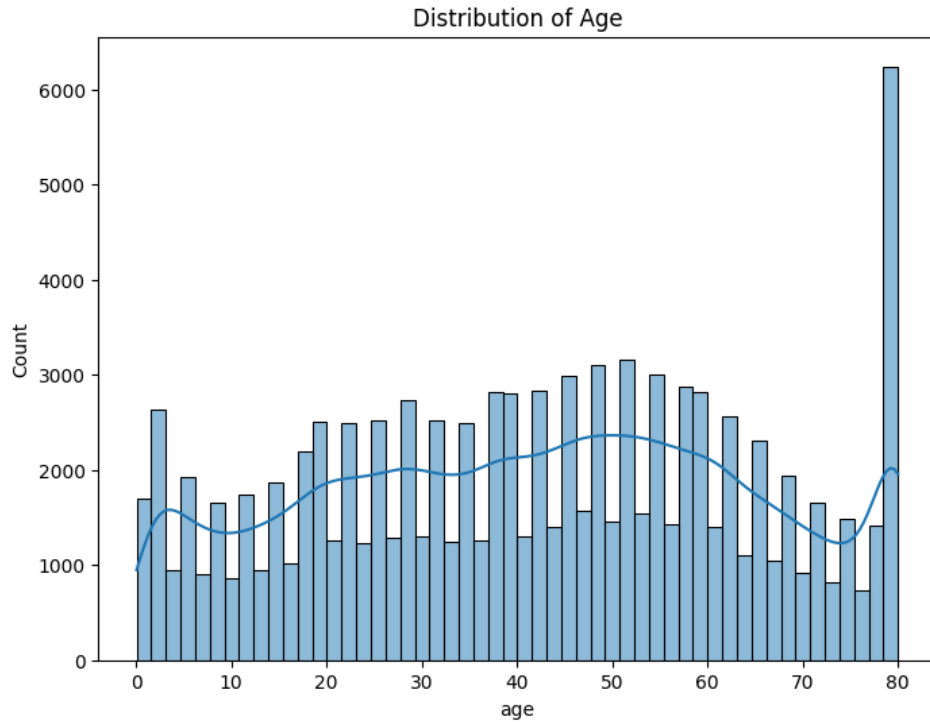


Figure 3 Histogram of Age Distribution

This image provides a histogram of the distribution of age of people suffering from diabetes. It is analyzed that most of the people who suffer from diabetes are from the age group of 80.

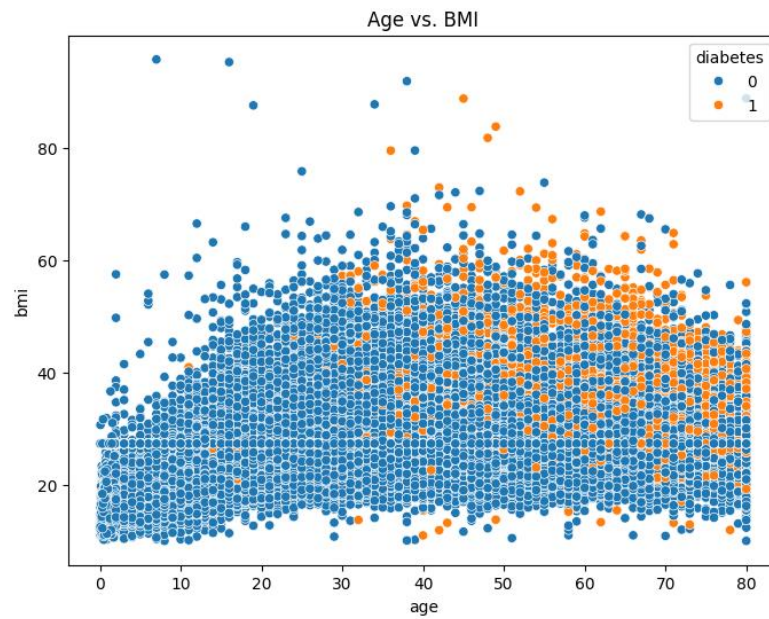


Figure 4 Scatterplot of Age vs BMI

This scatter plot displays how age may relate to Body Mass Index (BMI) for those individuals categorised by diabetes status. Diabetics are marked with orange dots and those without the disease with blue dots. Figure 4 shows that BMI tends to expand across a broader range as age increases, and there is an obvious clustering of overlapping (higher) BMIs in older age groups (40-70 years). Amongst individuals aged 40 and above with higher BMI, there is a large number of diabetic cases. This suggests there may be a correlation whereby a larger BMI and increasing age may increase the risks for developing diabetes (Massari et al., 2022).

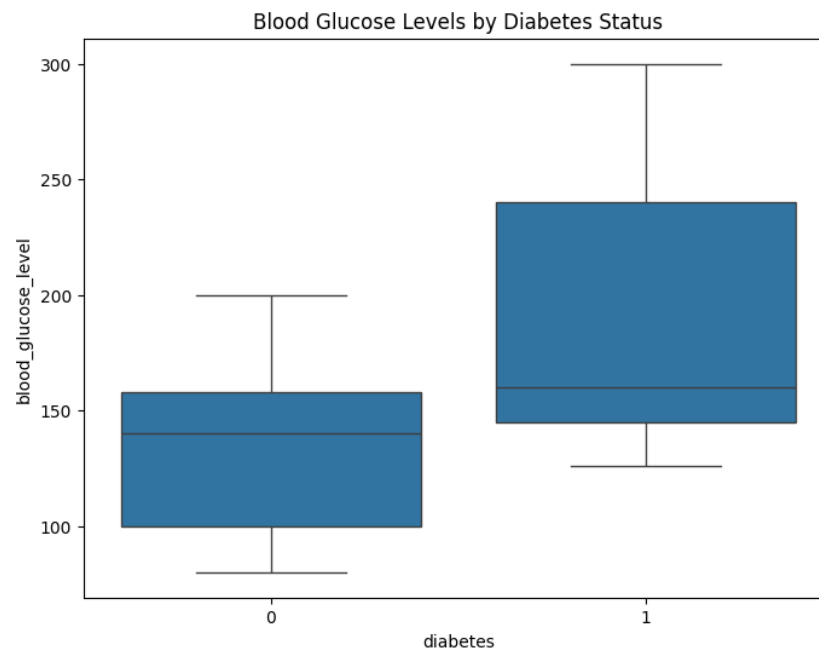


Figure 5 Boxplot of Blood Glucose Levels by Diabetes Status

The box plot illustrates blood glucose levels categorized by diabetes status (0: non-diabetic, 1: diabetic). The mean blood glucose level for the non-diabetic group is 150, and the interquartile range (IQR) is around 120–175. Median blood glucose for diabetics is around 200 IQR from about 140 to over 250. Blood glucose levels in diabetics vary more widely than in nondiabetics. The whiskers show the complete range of data and suggest that, in general, diabetics have dramatically more elevated and dispersed blood glucose levels compared to non-diabetics.

Feature Engineering:

```

▶ # Handel object type columns

# Convert 'gender' to numerical values
df['gender'] = df['gender'].map({'Female': 0, 'Male': 1, 'Other': 2})

# Convert 'smoking_history' to numerical values using one-hot encoding
smoking_history_dummies = pd.get_dummies(df['smoking_history'], prefix='smoking_history')
df = pd.concat([df, smoking_history_dummies], axis=1)
df = df.drop('smoking_history', axis=1)

```

Figure 6 Categorical Feature Encoding

Categorical Feature Encoding: The code handles categorical features 'gender' and 'smoking_history'. 'gender' is mapped to numerical values (Female: 0, Male: 1, Other:2). 'smoking_history' is one-hot encoded, creating new binary columns for every category. The original 'smoking_history' column is then dropped.

Data Splitting:

The dataset is split into training and testing sets with the use of the `train_test_split` function from scikit-learn library. This ensures that the model's performance is evaluated on unseen data.

```

[138] # Split target and festure columns
      X = df.drop('diabetes', axis=1)      # Features
      y = df['diabetes']                  # Target

[139] # Split training and testing dataset
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[140] print(X_train.shape)
      print(X_test.shape)
      print(y_train.shape)
      print(y_test.shape)

(80000, 13)
(20000, 13)
(80000,)
(20000,)

```

Figure 7 Dataset Split

This code can be used to train the data that can be used in the model. It then separates the dataset into features (X) and target (y) columns. As a result, our dataset consists of all the input features or attributes such as age, BMI, and glucose level in the X variable and y, which is the target variable or diabetes column (which indicates that the individual has diabetes or not). Secondly, given for

`train_test_split` function from Scikit Learn is utilized to split the data into testing and training sets. Analysts use 80% of the training set for training the model and 20% of the training set for testing the model (Mujumdar and V Vaidehi, 2019).

Model Training

In this task, 4 models are trained on the diabetes prediction dataset. These models are:

Logistic Regression:

This is a simple linear model used for binary classification tasks. Its goal is to discover how to split the space in such a way that the two classes (diabetes or non-diabetes) are as close to balance as possible. After that model is fit to the training data and performance is evaluated using accuracy and a classification report.

```
# 1. Logistic Regression
logreg_model = LogisticRegression(max_iter=1000)

# Fit training data to model
logreg_model.fit(X_train, y_train)

# Make prediction using trained model on testing data
logreg_pred = logreg_model.predict(X_test)

# Model Evaluation
logreg_accuracy = accuracy_score(y_test, logreg_pred)
print(f"Logistic Regression Accuracy: {logreg_accuracy}")

Logistic Regression Accuracy: 0.9592

# Classification Report
print(classification_report(y_test, logreg_pred))
```

Figure 8 Training of Logistic Regression

This training code gives giving Logistic Regression model to train on the diabetes prediction dataset. A logistic regression with `max_iter=1000` is initialized just as this to guarantee convergence. Lastly, the `fit()` method is utilized to train the model on the training dataset (`X_train`, `y_train`). The `predict()` method is utilized to make the predictions after training on the test set (`X_test`). The analyst calculate the accuracy score for the model's performance (Preet, 2022). A detailed metrics report such as F1 score, precision, recall, and support is created for both classes and returned in the form of a classification report.

	precision	recall	f1-score	support
0	0.97	0.99	0.98	18292
1	0.87	0.62	0.72	1708
accuracy			0.96	20000
macro avg	0.92	0.80	0.85	20000
weighted avg	0.96	0.96	0.96	20000

Figure 9 Classification Report of Logistic Regression

Logistic Regression model performance is shown in the classification report. The results on class 0 (non-diabetic) show precision of 0.97, recall of 0.99 and F1 score of 0.98, which is strong. While the F1-score, precision, and recall are all lower, precision is 0.87, recall is 0.62, and the F1-score is 0.72 for class 1 (diabetic). The final accuracy is 96%, and its weighted average F1-score is 0.96.

Decision Tree (DT) Classifier:

It creates an input feature tree-based model to decide its outputs of decisions based on the input feature. Splitting data by giving thresholds to features and trying to separate classes, creates subsets of data. The analyst computed its performance in the same way as logistic regression, by generating a classification report.

```
# 2. Decision Tree Classifier
dt_model = DecisionTreeClassifier()

# Fit training data to model
dt_model.fit(X_train, y_train)

# DecisionTreeClassifier
DecisionTreeClassifier()

# Make prediction using trained model on testing data
dt_pred = dt_model.predict(X_test)

# Model Evaluation
dt_accuracy = accuracy_score(y_test, dt_pred)
print(f"Decision Tree Accuracy: {dt_accuracy}")

Decision Tree Accuracy: 0.9527
```

Figure 10 Training Decision Tree Classifier

The diabetes prediction dataset was fed to a Decision Tree Classifier to train on it. Training begins with **DecisionTreeClassifier()**, which constructs a tree of data classification based on values of different features. The model is trained on the training dataset using `fit()` method. When trained, predictions are made on the test set (`X_test`) using the `predict` technique. The accuracy score is

calculated from the model and is printed (Ganie et al., 2023). Also, there is a classification report that gives details on precision, support, recall, and F1-score for diabetic (class 1) and non-diabetic (class 0).

	precision	recall	f1-score	support
0	0.97	0.97	0.97	18292
1	0.72	0.73	0.72	1708
accuracy			0.95	20000
macro avg	0.85	0.85	0.85	20000
weighted avg	0.95	0.95	0.95	20000

Figure 11 Classification Report of Decision Tree

The DT model performs very well with a Precision of 0.97, Recall of 0.97 and an F1 score of 0.97 for class 0 (non-diabetic). For class 1 (diabetic), precision is 0.72, recall 0.73, F1 score 0.72, performance is worse. It achieves an overall accuracy of 95% a balanced macro average F1-score of 0.85 and a weighted average F1-score of 0.95.

Random Forest (RF) Classifier:

Random forest is an effective and useful ensemble method, using several decision trees to get better prediction with the result of ensemble being an average of individual trees results. In general speaking, a multi-decision tree is highly robust than a single-decision tree and is less vulnerable to overfitting.

```

# 3. Random Forest Classifier
rf_model = RandomForestClassifier()

# Fit training data to model
rf_model.fit(X_train, y_train)

RandomForestClassifier()

# Make prediction using trained model on testing data
rf_pred = rf_model.predict(X_test)

# Model Evaluation
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest Accuracy: {rf_accuracy}")

Random Forest Accuracy: 0.97015

# Classification Report
print(classification_report(y_test, rf_pred))

```

Figure 12 Training Random Forest model

Using this code, a RF Classifier is trained on the diabetes prediction dataset. The model is initialized by using **RandomForestClassifier()** which creates multiple decision trees and concatenates the decisions of the outcome to defeat overfitting. `fit()` method is utilized for training of the model using the given training dataset. Analysts use the `predict()` method to predict the test set (`X_test`) after training. Furthermore, a classification report is printed to present an extensive evaluation of the model performance.

	precision	recall	f1-score	support
0	0.97	1.00	0.98	18292
1	0.94	0.69	0.80	1708
accuracy			0.97	20000
macro avg	0.96	0.84	0.89	20000
weighted avg	0.97	0.97	0.97	20000

Figure 13 Classification Report of Random Forest

The RF model shows good performance for class 0 (non diabetic) with a high precision (0.97), recall (1.00), F1-score (0.98). The Moderate performance is measured for class 1 (diabetic) as precision is 0.94, recall is 0.69, and F1 score is 0.80. Model achieved 97% overall accuracy with 0.89 macro balanced average F1 score, which shows strong model overall accuracy.

Voting Classifier (Ensemble Model):

Finally, to improve model performance, a Voting Classifier, which combines predictions of the first three models like Random Forest Logistic Regression, and Decision Tree, is used.

```
# Create a voting classifier
voting_clf = VotingClassifier(estimators=[('lr', logreg_model), ('dt', dt_model), ('rf', rf_model)], voting='soft')

# Train the voting classifier
voting_clf.fit(X_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(

VotingClassifier
├── lr
│   └── LogisticRegression
├── dt
│   └── DecisionTreeClassifier
└── rf
    └── RandomForestClassifier

# Make predictions
voting_pred = voting_clf.predict(X_test)

# Evaluate the voting classifier
voting_accuracy = accuracy_score(y_test, voting_pred)
print(f"Voting Classifier Accuracy: {voting_accuracy}")

Voting Classifier Accuracy: 0.967
```

Figure 14 Training Voting Classifier

This code trains a Voting Classifier that merges the predictions of three individual models: LR, RF, and DT. Using a "soft" voting strategy, the VotingClassifier averages the probabilities from the models to produce a predicted class. As with any sklearn pipeline analyst first initialized the voting classifier and then trained it on the training data (X_train, y_train) with the fit() method. Using predict() the test set (X_test) is predicted. To evaluate the classifier's performance the classification report that contains precision, support, recall, and F1 score for both classes is printed (Tasin et al., 2022).

	precision	recall	f1-score	support
0	0.97	0.99	0.98	18292
1	0.88	0.71	0.79	1708
accuracy			0.97	20000
macro avg	0.93	0.85	0.88	20000
weighted avg	0.97	0.97	0.97	20000

Figure 15 Classification Report of Voting Classifier

In general, the Voting Classifier has an accuracy of 97%. This performs well for class 0 (non-diabetic) with precision and recall of 0.97 and 0.99, respectively. The results for class 1 (diabetic) are precision= 0.88 and recall= 0.72, so performance is moderate. An F1-score of 0.88 for the macro average is obtained, indicating a good balance between classes.

Model Evaluation

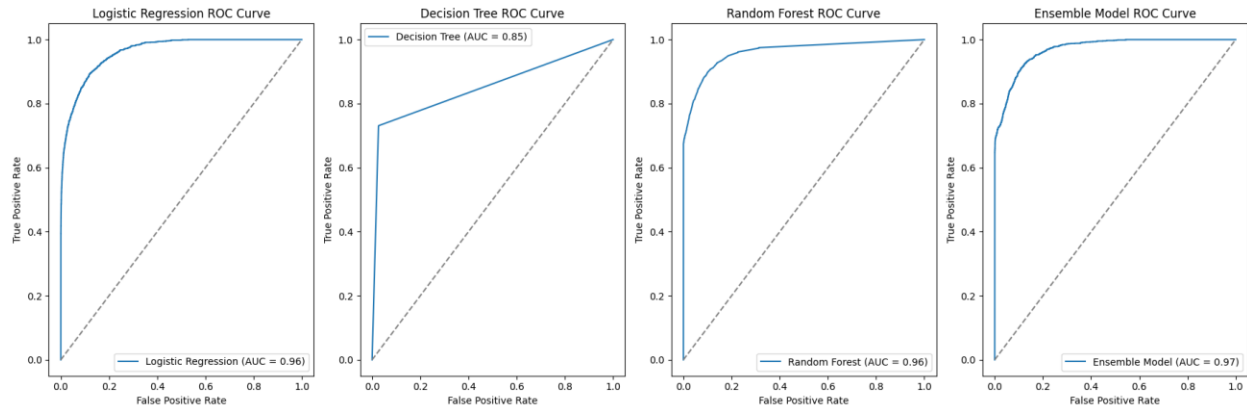


Figure 16 ROC Curves for each Model

The image presents the Receiver Operating Characteristic curves for four different ML models include LR, DT, RF, and Ensemble Model (Hasan et al., 2020). It is a graphical representation of the diagnostic ability of a binary classifier because its discrimination threshold is different and it is referred to as ROC curve.

- The Logistic Regression model shows an AUC of 0.96.
- The Decision Tree model shows a much lower performance with an AUC of 0.85.
- The Logistic Regression is matched by the Random Forest model and has an AUC of 0.96.
- The Ensemble Model achieves a slightly better AUC of 0.97 than all other models.

Overall, all other models, except the Decision Tree, have great predictive power, and are fairly close to optimal tradeoff in true positive vs false positive, with better than slight outperformance of the Ensemble Model.

```

import random
random_indices = random.sample(range(len(X_test)), 10)
for index in random_indices:
    prediction = rf_model.predict(X_test.iloc[[index]])
    actual_value = y_test.iloc[index]
    print(f"Index: {index}")
    print(f"Actual Value: {actual_value}")
    print(f"Predicted Value: {prediction[0]}")
    print("-" * 20)

Index: 14397
Actual Value: 1
Predicted Value: 1
-----
Index: 655
Actual Value: 0
Predicted Value: 0
-----
Index: 15707
Actual Value: 0
Predicted Value: 0
-----
Index: 10663
Actual Value: 0
Predicted Value: 0
-----
Index: 6152
Actual Value: 0
Predicted Value: 0
-----
Index: 13198
Actual Value: 0
Predicted Value: 0
-----
Index: 11449
Actual Value: 0
Predicted Value: 0
-----
Index: 5070
Actual Value: 0
Predicted Value: 0
-----
Index: 5309
Actual Value: 0
Predicted Value: 0
-----
Index: 484
Actual Value: 0
Predicted Value: 0

```

Figure 17 Comparison of Actual and Predicted Values

Analysts use `random.sample()` from code to select 10 random samples from the test dataset(`X_test`), and for each sample use a Random Forest Classifier(`rf_model`) to predict the diabetes status. Analysts compare the predicted value with the actual value(`y_test`) of the target variable. Both the index and the predicted and actual value values are printed for each sample. The output shows for Index 14397 actual and predicted values are the same.

Conclusion

A comprehensive health dataset was used to train several machine learning models to predict the likelihood of diabetes in this research. In the models above Random Forest, Logistic Regression, Decision Tree, and Voting Classifier, all perform well on predictive ability, and the best overall accuracy is 97% using the Voting Classifier. In terms of evaluation metrics, including precision, recall and F1-score, models were shown to be able to properly discriminate between diabetics and non-diabetics with varying degrees of accuracy when distinguishing diabetics. Taken together, these results suggest that machine learning could be used to effectively support early diabetes diagnosis to help healthcare professionals plan more efficient prevention strategies.

References

- Dharmarathne, G., Jayasinghe, T.N., Bogahawaththa, M., D.P.P. Meddage and Rathnayake, U. (2024). A novel machine learning approach for diagnosing diabetes with a self-explainable interface. *Healthcare Analytics*, [online] 5, pp.100301–100301. doi:<https://doi.org/10.1016/j.health.2024.100301>.
- Ganie, S.M., Dutta, K., Malik, M.B., Mallik, S. and Qin, H. (2023). An ensemble learning approach for diabetes prediction using boosting techniques. *Frontiers in Genetics*, [online] 14. doi:<https://doi.org/10.3389/fgene.2023.1252159>.
- Hasan, M.K., Alam, M.A., Das, D., Hossain, E. and Hasan, M. (2020). Diabetes Prediction Using Ensembling of Different Machine Learning Classifiers. *IEEE Access*, [online] 8, pp.76516–76531. doi:<https://doi.org/10.1109/access.2020.2989857>.
- Massari, H.E., Sabouri, Z., Mhammedi, S. and Gherabi, N. (2022). Diabetes Prediction Using Machine Learning Algorithms and Ontology. *Journal of ICT Standardization*. [online] doi:<https://doi.org/10.13052/jicts2245-800x.10212>.
- Mujumdar, A. and V Vaidehi (2019). Diabetes Prediction using Machine Learning Algorithms. *Procedia Computer Science*, [online] 165, pp.292–299. doi:<https://doi.org/10.1016/j.procs.2020.01.047>.
- Mustafa, M. (2023). *Diabetes prediction dataset*. [online] Kaggle.com. Available at: <https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset> [Accessed 25 Nov. 2024].
- Preet, A. (2022). *Diabetes Prediction Using Machine Learning*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2022/01/diabetes-prediction-using-machine-learning/> [Accessed 25 Nov. 2024].
- Tasin, I., Nabil, T.U., Islam, S. and Khan, R. (2022). Diabetes prediction using machine learning and explainable AI techniques. *Healthcare Technology Letters*, [online] 10(1-2), pp.1–10. doi:<https://doi.org/10.1049/htl2.12039>.