Muhammad Ahmer Bilal Jan 30965
Syed Ahmed Ali Shah 22943
Malik Haider 27709

Diabetes Prediction System Report

1. Introduction

Diabetes is a chronic health condition that affects how your body turns food into energy. The purpose of this project is to develop a machine learning model to predict the likelihood of a person having diabetes based on various health metrics. Early discovery of diabetes can greatly enhance the handling and care for the disease thereby minimizing the dangers of severe health conditions.

2. Requirements

Functional Requirements

1. Data Preprocessing: This involves removal of dups , missing values, anomalies and outliers before normalizing since it may cause misleading values during analysis.
2. Model Training: Processing of the available information is done before being used for development and training of a machine learning tool.

3. Model Evaluation: Evaluate the model's performance using appropriate metrics such as accuracy, ROC AUC score, and crossvalidation.
4. Prediction: Predict diabetes on new, unseen data

3. Technologies Used

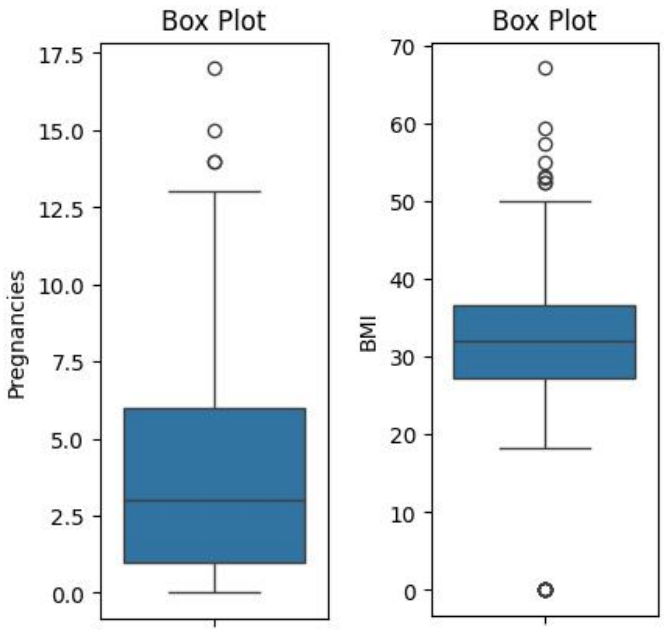
1. Pandas: For data manipulation and analysis using data structures like DataFrames, pandas is used.
2. Numpy: Numerical computation is done by numpy
3. Matplotlib: Visualization is done using matplotlib which has abilities to create static or animated graphs.
4. ScikitLearn is a very large software library used for machine learning. ScikitLearn includes tools for model selection, training, evaluation and parameter tuning.

4. Methodology

1. **Data Collection:** The dataset used is https://www.kaggle.com/datasets/mathchi/diabetesdataset, containing various health metrics of individuals.

2. **Data Cleaning:**
Checked for duplicates and missing values. There were none.
There were no missing values but there were a lot of Zeros, with insulin and skin thickness having alarming values of zero values i.e 374 and 227 respectively. These zero values maybe because of data entry mistakes. We then found correlation between columns. As shown in the below correlation matrix heatmap:

Here we can see that insulin and skin thickness have weakest correlation between them i.e 0.1. So we decided to drop these two columns.
After that we made some boxplots to check if there are any outliers. The boxplots are shown below



As visible in the above boxplots we have some outliers, there must be more in other columns. So, we Handled outliers using the Interquartile Range (IQR) method.

Calculating Quartiles and IQR:

The analysis begins with calculating the 25th percentile (Q1) and the 75th percentile (Q3) of the dataset. These quartiles divide the data into four equal parts.

The quartiles, Q1 and Q3, divide the dataset into four equal parts in the following way:

1. **Q1 (First Quartile)**: This value represents the 25th percentile of the dataset. It divides the dataset into two parts: the lower 25% and the upper 75% of the data.

2. **Q2 (Second Quartile)**: This value is also known as the median. It represents the 50th percentile of the dataset and divides the data into two equal parts: the lower 50% and the upper 50%.

3. **Q3 (Third Quartile)**: This value represents the 75th percentile of the dataset. It divides the dataset into two parts: the lower 75% and the upper 25% of the data.

The Interquartile Range (IQR) is then computed by subtracting Q1 from Q3. The IQR represents the middle 50% of the data and is used to identify outliers.

Determining Outlier Thresholds:

The next step involves determining the thresholds for detecting outliers using the IQR. The upper limit is set as Q3 plus 1.5 times the IQR, and the lower limit is set as Q1 minus 1.5 times the IQR.

These thresholds help identify data points that are significantly higher or lower than the majority of the data.

Identifying Outliers:

Data points that fall outside the calculated upper and lower limits are identified as outliers. This identification process checks if each data point falls below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR.

Filtering Outliers:

`(data < (Q1 - 1.5 * IQR))` checks if each value in the dataset is less than (Q1 - 1.5 * IQR).

`(data > (Q3 + 1.5 * IQR))` checks if each value in the dataset is greater than (Q3 + 1.5 * IQR).

`((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR)))` combines these conditions using the OR operator `|`, meaning it selects values that meet either condition. These are potential outliers.

Removing Rows with Outliers:

`.any(axis=1)` checks if any of the conditions (whether the value is less than Q1 - 1.5 * IQR or greater than Q3 + 1.5 * IQR) are true for each row of the dataset. If any condition is true for a row, it means that row contains an outlier.

`~` negates the result, so it selects rows where none of the conditions are true, meaning there are no outliers.

`data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]` creates a new dataset (`data_out`) by selecting rows from the original dataset (`data`) where there are no outliers.

Shapes of the Datasets:

`data.shape` gives the dimensions (number of rows and columns) of the original dataset.

`data_out.shape` gives the dimensions of the dataset after removing outliers (`data_out`).

In simple words, this finds outliers in the dataset by checking if values are too small (below Q1 - 1.5 * IQR) or too large (above Q3 + 1.5 * IQR). Then, it creates a new dataset (`data_out`) without those outliers. Finally, it prints the dimensions of both the original dataset and the dataset without outliers.

There are two more techniques of removing outliers:

- Trimming
- Capping

The technique we have used is the IQR method it removes entire rows containing outliers, while trimming removes a portion of data from the tails, and capping adjusts extreme values to be within a specified range without removing them. Each technique has its advantages and may be preferred depending on the specific characteristics of the dataset and the analysis goals.

After handling outliers we replaced the 0 values with missing Nan values. Nan indicates that the values are not a number and they are missing. After replacing zeros with missing values we used knn method to replace them with the mean of 5 nearest neighbors.

3. **Data Splitting:** Split the data into training and testing sets using an 8020 ratio.

4. **Model Selection:** The performance of 3 models was evaluated using mean crossvalidation scores. Logistic Regression outperformed both Random Forest and Decision Tree classifiers.

5. **Model Development:** Developed a Logistic Regression model.

6. **Hyperparameter Tuning:** Grid search was performed to tune hyperparameters for Logistic Regression, resulting in improved performance. There are a lot of parameters that can be used but most of them are set to default and are not changed. Few that can be changed are explained below

C : It is the regularization strength parameter. Regularization strength in the context of machine learning, particularly in models like Logistic Regression, refers to a technique used to prevent overfitting. We tried different values like 0.01, 0.1, 1, 10, 100. But value of 10 gave best results.

Penalty: It is the regularization method that will be used. In our case The model will try both regularization methods to find the optimal one. In our case model returned that l2 gave the best results

Solver: It is the algorithm used. We used 'liblinear' which is a solver suitable for small

datasets and supports both L1 and L2 regularization

7. **Model Evaluation:** Evaluated the model's performance using accuracy score

1. Results

CrossValidation Scores:
Random Forest Classifier: Mean crossvalidation score: 0.768
Logistic Regression: Mean crossvalidation score: 0.778
Decision Tree Classifier: Mean crossvalidation score: 0.705

Performance of Logistic Regression after hyperparameter tuning:
Accuracy: 0.829

6. Future Work

Potential enhancements and improvements include:

1. Feature Engineering: Create new features that might have a stronger correlation with the target variable.
2. Additional Algorithms: Explore other machine learning algorithms such as Gradient Boosting, Support Vector Machines, or Neural Networks for potentially better performance.
3. Data Augmentation: Gather more data to improve the robustness and generalizability of the model.
4. Deployment: Develop a userfriendly interface for the model and deploy it as a web or mobile application for broader accessibility.

7. Summary

This study demonstrates the application of machine learning techniques in predicting diabetes using health metrics. The models developed, particularly Logistic Regression with optimized parameters, show promising results in accurately predicting diabetes. Further refinement and validation of these models could potentially enhance their utility in clinical settings for early detection and management of diabetes.

8. References

https://www.youtube.com/watch?v=ay_OcblJasE&t=383s
https://medium.com/@sampurn10chouksey/disease-prediction-using-machine-learning-part-1-c4c7ed741123
https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html
https://www.geeksforgeeks.org/python-imputation-using-the-knnimputer/