**Logo

Description automatically generated**

**IT 7143 Cloud Analytical Technology**

**Final Project Report**

**Group : 5**

**Bhavitha Redrouthu**

**Mythili jayaraman**

**vidhi dave**

**viswa narendra reddy**

**INDEX**

|  |  |
| --- | --- |
| **CONTENTS** | **Page No** |
| **Abstract** | **3** |
| **Introduction** | **3** |
| **Preliminary Analysis** | **4** |
| **Data Preprocessing** | **5** |
| **Modeling and Results** | **7** |
| **Conclusion** | **19** |

**Abstract**

This project explores the use of machine learning algorithms to predict the likelihood of diabetes in patients based on a set of features. The dataset consists of information on patients, including their gender, age, history of hypertension, heart disease, smoking, BMI, HbAlc level, and blood glucose level. The target variable is whether the patient has diabetes or not.

# The project begins by preprocessing the data to handle missing values and to prepare it for use in training machine learning models. The dataset is split into training and testing sets, and three different machine learning algorithms are used to build models: Logistic Regression, Random Forest Classifier, Decision Tree, and Deep Neural Network model (Multilayer Perception).

The performance of the models is evaluated using metrics such as accuracy, precision, recall, and F1 score. The results of the project demonstrate that all three models are effective in predicting the likelihood of diabetes in patients, with the Random Forest Classifier achieving the highest accuracy score.

The project concludes by discussing the implications of these results. Overall, the project demonstrates the usefulness of machine learning algorithms in healthcare and highlights the potential for further development and refinement of these methods to improve patient outcomes.

**Introduction**

Diabetes is a chronic metabolic disorder that affects millions of people worldwide. Early detection and effective management of diabetes are critical in reducing the risk of complications and improving patient outcomes. Machine learning algorithms have shown great potential in predicting the likelihood of diabetes in patients based on various factors, including age, gender, body mass index (BMI), and blood glucose levels. In this project, we explore the use of machine learning algorithms to predict the likelihood of diabetes in patients using a diabetes dataset. The dataset contains information on patients' gender, age, hypertension, heart disease, smoke history, BMI, HbAlc level, blood glucose level, and whether they have diabetes or not. We use three different machine learning algorithms: Logistic regression, Random Forest Classifier, Decision Tree, and Deep Neural Network model (Multilayer Perception) to build models and predict diabetes likelihood in patients. The project evaluates the performance of these models using various metrics such as accuracy, precision, recall, and F1 score. The results of the project demonstrate the effectiveness of machine learning algorithms in predicting the likelihood of diabetes in patients, highlighting their potential use in healthcare to improve patient outcomes. This project contributes to the ongoing research in machine learning applications in healthcare and provides insight into possible future directions for research in this area.

**Preliminary Analysis**

Diabetes dataset has 100000 records and total 9 columns. The data consists of the following columns

1. Gender – indicates patient gender.
2. Age – Indicates the age of the patient.
3. Hypertension – Patient Blood pressure record
4. Heart\_disease – Indicates if patient has any heart related issues or not.
5. Smoking\_history – Patient smoking history
6. Bmi – Body Mass Index
7. HbA1c\_level – Blood glucose level test result
8. Blood\_glucose\_level – Patient blood glucose level indicator
9. Diabetes – indicates if patient has diabetes or not.

For this dataset, diabetes will be the target.

Datatype information:

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 gender 100000 non-null object

1 age 100000 non-null float64

2 hypertension 100000 non-null int64

3 heart\_disease 100000 non-null int64

4 smoking\_history 100000 non-null object

5 bmi 100000 non-null float64

6 HbA1c\_level 100000 non-null float64

7 blood\_glucose\_level 100000 non-null int64

8 diabetes 100000 non-null int64

dtypes: float64(3), int64(4), object (2)

Dataset indicates no missing values. Gender and Smoking\_history are categorical, remaining other columns are numerical.

Graphical user interface, application, Word

Description automatically generated

Histogram has been used to plot the columns data distribution as it helps in exploratory data analysis, Dataset is fairly distributed with some outliers which can be clipped during preprocessing. Data needs to be standardized as there is difference among the range of numerical columns.

Dataset indicates no missing values. Gender and Smoking history are categorical columns which will be transformed into numerical values during preprocessing.

**Data Preprocessing**

For the Data Preprocessing steps, we do convert categorical variables into numerical codes by using the astype and cat.codes methods. The data is then split into training and testing sets, and linear regression is used to predict blood glucose levels using the preprocessed data. The mean squared error and R2 score are calculated to evaluate the performance of the model on the test set.

Text

Description automatically generated

Next, we apply the Isolation Forest algorithm to detect anomalies or outliers in a given dataset, which is a technique that is commonly used for detecting fraudulent or malicious activity in data. The algorithm is initialized with a contamination rate of 0.05, which represents the proportion of outliers in the dataset. The fit method is used to train the algorithm on the dataset, and the predict method is used to generate a binary output where -1 indicates an outlier, and 1 indicates an inlier. Finally, the number of outliers detected is printed by counting the number of -1 values in the outliers array using the np.count\_nonzero method.

Text

Description automatically generated

Then we encode the categorical variables `gender` and `smoking\_history` using the `LabelEncoder` method, which assigns numerical codes to each category. The numerical features `age`, `bmi`, `HbA1c\_level`, and `blood\_glucose\_level` are then scaled using the `MinMaxScaler` method, which transforms the features to be between 0 and 1 to ensure that all the features contribute equally to the model training process. The encoded and scaled data is then ready for further preprocessing or for use in machine learning models.

Graphical user interface, text, application

Description automatically generated

Next, we split the original dataset into training and testing sets using the `train\_test\_split` method with a test size of 0.2 and a random state of 42. It then splits the training and testing sets further into new sets using the same method with a test size of 0.5 and the same random state of 42. The shapes of the resulting dataset splits are then printed, which show the number of rows and columns in each of the four sets: `X\_train`, `X\_test`, `y\_train`, and `y\_test`.

Graphical user interface, text, application, email

Description automatically generated

**Modeling and Results :**

**Random Forest Classifier :**

Graphical user interface, text, application

Description automatically generated

The above code performs a grid search over a parameter grid for a random forest classifier using scikit-learn's GridSearchCV.

GridSearchCV takes in several arguments including the model to be used, the parameter grid to be searched over, the evaluation metric, the number of cross-validation folds, the number of parallel jobs to run, and the level of detail in the output messages.

Once GridSearchCV is set up, the fit method is used to perform the grid search and find the best hyperparameters for the model.

After the search is complete, the best hyperparameters and the corresponding best score can be accessed using the best\_params\_ and best\_score\_ attributes, respectively.

The best parameters and best score results are as follows :

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

The above code makes predictions on the test set X\_test using the predict method of the GridSearchCV object grid\_search\_rf.

y\_pred will contain the predicted class labels for the test set, based on the best hyperparameters found during the grid search.

The code then displays the first 20 predicted values using indexing (y\_pred[0:20]).

Lastly, we computed the accuracy score for the random forest classifier model on the test data.

**Decision Tree:**

**Text

Description automatically generated**

Grid Search Cross Validation to find the best hyperparameters for a Decision Tree classifier, first required libraries like DecisionTreeClassifier from sklearn.tree, GridSearchCV from sklearn.model\_selection, and accuracy\_score and f1\_score from sklearn.metrics are imported.

parameter grid has been setup with a range of values for max\_depth, max\_features, min\_samples\_split, and min\_samples\_leaf to search for the best combination of hyperparameters. After initializing Decision Tree Classifier object, grid Search Cross Validation object has been setup with the parameter grid, a 5-fold cross-validation, and 'accuracy' as the scoring metric. Lastly Grid Search Cross Validation object has been fit to the training data (X\_train, y\_train), it performs a search over the hyperparameters specified in the parameter grid and selects the combination of hyperparameters that results in the highest cross-validation accuracy score.

**Graphical user interface, text, application, email

Description automatically generated**

**K Nearest Neighbors Classifier :**

Graphical user interface, text

Description automatically generated with medium confidence

The above code performs a grid search over a parameter grid for a k-nearest neighbor classifier using scikit-learn's GridSearchCV.

GridSearchCV takes in several arguments including the model to be used, the parameter grid to be searched over, the evaluation metric, the number of cross-validation folds, the number of parallel jobs to run, and the level of detail in the output messages.

Once GridSearchCV is set up, the fit method is used to perform the grid search and find the best hyperparameters for the model.

After the search is complete, the best hyperparameters and the corresponding best score can be accessed using the best\_params\_ and best\_score\_ attributes, respectively.

Lastly, we predicted the accuracy score of K – Nearest Neighbor by importing accuracy\_score from sklearn metrics and the accuracy is 0.95.

**Deep Neural Network Model - Multilayer Perception:**

This model defines and trains a **deep neural network with multilayer perceptron (MLP)** using TensorFlow and Keras. The input data is split into batches using TensorFlow's `Dataset` class, and a batch size of 512 is set. The MLP consists of 9 fully connected (dense) layers, each with a ReLU activation function except for the output layer, which uses a sigmoid activation function. The `binary\_crossentropy` loss function is used, along with the `adam` optimizer and several metrics, including `accuracy` and `auc\_precision\_recall`. An early stopping callback is defined to stop training if the validation loss does not improve for five epochs. The `fit` method is called to train the model using the training set, and the training history is stored in the `hist4` variable.

Text

Description automatically generated

The section evaluates the MLP model's performance on the test set by calling the `evaluate` method on the model and passing in the test set as arguments. The loss, test accuracy, and test AUC are computed and stored in the `loss`, `test\_accuracy`, and `test\_auc` variables, respectively. The test accuracy and AUC are then printed using `f-strings` with 4 decimal places.

Graphical user interface, text, application, email

Description automatically generated

The next section calls two custom plot functions plot\_model\_loss and plot\_model\_accuracy, which are likely defined elsewhere in the code or in another module. The plot\_model\_loss function is used to plot the training and validation loss over epochs, and the plot\_model\_accuracy function is used to plot the training and validation accuracy over epochs. The functions likely take the hist4 object as input and display the plots.

Graphical user interface, application

Description automatically generated

The code prints a summary of the MLP model's architecture using the `summary` method. The summary provides a table with details of each layer, including the layer type, output shape, and number of parameters. The output shape of each layer is determined by the input shape of the previous layer and the layer's configuration. The summary also provides the total number of parameters in the model, which is the sum of the parameters in all the layers.

Graphical user interface, application, table

Description automatically generated

Then the next section creates a DataFrame object d1 from the history attribute of the hist4 object, which contains the training history of the MLP model. The plot method is then called on d1 to display a line plot of the training history. The figure size of the plot is set to (6,6). The plot likely shows the training and validation loss and accuracy over epochs.

Graphical user interface, application, table

Description automatically generated

This section calculates and prints a classification report for evaluating the MLP model's performance on the test set. The model predictions are obtained by calling the `predict` method on the test set batches and setting a threshold of 0.5. The predictions are then converted to integers using the `astype` method. The `classification\_report` function from scikit-learn is then used to compute and print the precision, recall, F1 score, and support for each class, as well as the weighted average of these metrics. The report provides a detailed summary of the model's performance on the test set.

Text

Description automatically generated

**Finding The Top 10 Combinations With The Highest Probabilities Of Getting Diabetes :**

Text

Description automatically generated

Here we used a logistic regression classifier to predict the probabilities of getting diabetes for various combinations of demographic and behavioral features.

\*Firstly we defined age groups, sexes, and smoking habits as arrays.

\*Created a list of all possible combinations of age, sex, and smoking habit.

\*Calculated the mean feature values of the training set.

\*Added the mean feature values to the list of combinations.

\*Used the logistic classifier to predict the probability of getting diabetes for each combination.

\*Sorted the list of combinations and probabilities in descending order.

Then lastly we printed the top 10 combinations with the highest probabilities of getting diabetes, along with their corresponding demographic and behavioral features.

Chart

Description automatically generated with low confidence

\*From the above, it appears that individuals who are male, smokers, and 80 years of age have a higher likelihood of developing diabetes.

\*Individuals who are make , non-smoker and 60 years of age have less chance of developing diabetes.

**Conclusion :**

Graphical user interface, text, application, email

Description automatically generated

Here, we are evaluating the performance of several machine learning models on a dataset by calculating their accuracy scores.

\*We define a dictionary called accuracy\_scores and populate it with the accuracy scores of five different models. The models evaluated are Logistic Regression, Random Forest Classifier, Decision Tree, K-Nearest Neighbors Classifier, and a DNN Model (Multilayer Perception).

\*Then we used the max() function to find the model with the highest accuracy score and store its name in the best\_model variable. Then, retrieve the accuracy score of the best model and store it in the best\_model\_accuracy variable.

\*Then print the accuracy scores of all models in the accuracy\_scores dictionary and then finally print statement indicating which model has the highest accuracy score.

By calculating and comparing their accuracy scores, we can quickly identify which model performs the best and may be the most suitable for use in practice.

A picture containing graphical user interface

Description automatically generated

Looking from the above accuracy score , we can say that Random Forest Classifier and Decision Tree have the highest accuracy score of 97.17%.

We can conclude that Random Forest or Decision Tree as the best model we have on this data set and can be used for the deployment purpose.