Predictive Model for diagnosing Diabetes

Nikhil Rajashekhar Pujar
Roll No. -20103100
NIT Jalandhar
Jalandhar,Punjab
nikhilrp.cs.20@nitj.ac.in

Palla Santosh Kumar
Roll No. – 20103104
NIT Jalandhar
Jalandhar,Punjab
pallask.cs.20@nitj.ac.in

Prateek Singh
Roll No. – 20103109
NIT Jalandhar
Jalandhar,Punjab
prateeks.cs.20@nitj.ac.in

• ***Abstract—***

The aim of this project is to build a predictive model for the detection of diabetes based on dependent variables using algorithms **such as Decision Tree, Naïve Bayes, Random Forest, KNN Classification and SVM**.

The motivation behind building this project is to develop an **accurate prediction model** that uses medical parameters to arrive at a conclusion whether a patient has diabetes or not.

I. Introduction

1 out of every 11 Indians has been formally diagnosed with diabetes and many more patients go undiagnosed. The model built in this project could be used to verify lab test results.

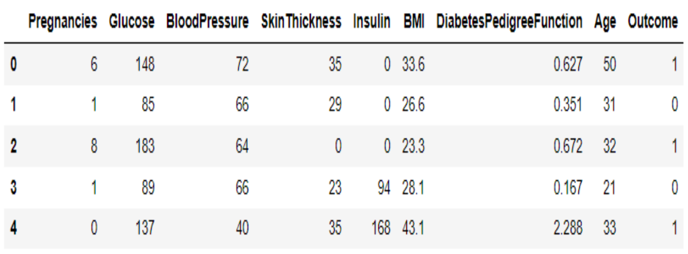
Diabetes is a disease caused by high glucose levels. Diabetes, if ignored may cause some major issues such as cardiovascular issues, renal issues, high blood pressure, retinal damage. Diabetes can be controlled if it is predicted earlier and this is the goal of this project. Early prediction of Diabetes may save a patient's life.

II. The dependent variables are :

- ***Pregnancies***
- ***Glucose***
- ***Blood Pressure***
- ***Skin Thickness***
- ***Insulin***
- ***BMI***
- ***Diabetes Pedigree Function***
- ***Age***

III. Dataset used

Pima Indian diabetes dataset has been used. It has been downloaded from Kaggle.



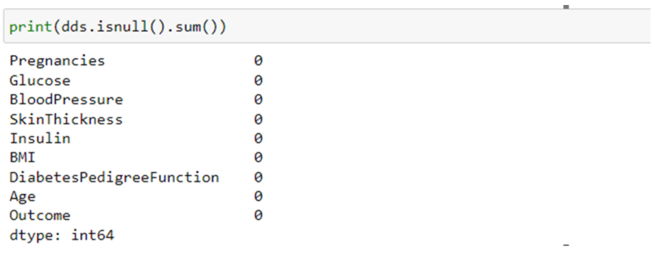
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

IV. Libraries used

- Numpy – used for performing numerical operations efficiently on large arrays
- Pandas – used for data manipulation through objects like Series and DataFrames
- Matplotlib – used for plotting graphs and daa visulaization
- Seaborn - used for plotting graphs and daa visulaization
- Sklearn – contains various classifiers such as as Decision Tree, Naïve Bayes, Random Forest, KNN Classification and SVM

V. Pre-processing

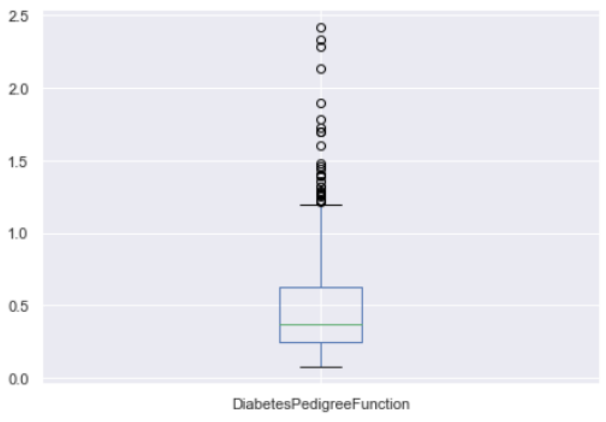
- Before applying the algorithm to make predictions, we must ensure the dataset is clean.
- First, we check if there are any null values*:*



```
print(dds.isnull().sum())
```

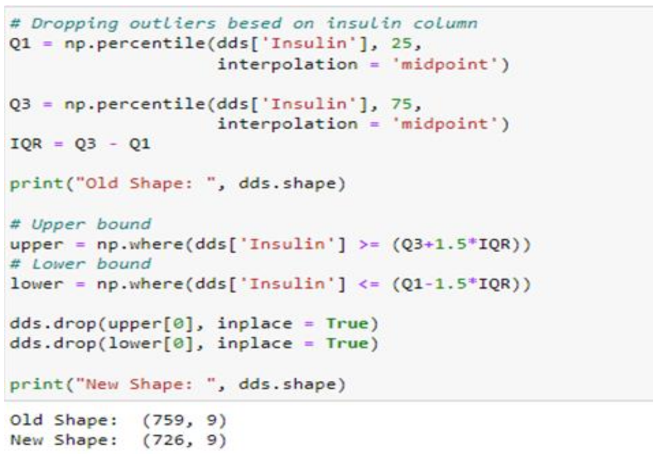
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

- Since there are no null values, we can proceed further.
- Identifying outliers using boxplot



- A boxplot gives a summary of minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It can tell you about your outliers and what their values are. It can also tell if the data is symmetrical, or if the data is tightly grouped, and if the data is skewed.
- In the above plot, the outliers are represented by circles.

- *Dropping outliers*
  - Define upper bound=Q3+1.5*Inter Quartile Range
  - Define lower bound=Q1-1.5*Inter Quartile Range
  - Drop all rows which do not fall in these ranges
  - Age and Insulin columns have been considered because maximum outliers were observed in these 2 columns.



```
# Dropping outliers besed on insulin column
Q1 = np.percentile(dds['Insulin'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(dds['Insulin'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", dds.shape)

# Upper bound
upper = np.where(dds['Insulin'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(dds['Insulin'] <= (Q1-1.5*IQR))

dds.drop(upper[0], inplace = True)
dds.drop(lower[0], inplace = True)

print("New Shape: ", dds.shape)
```

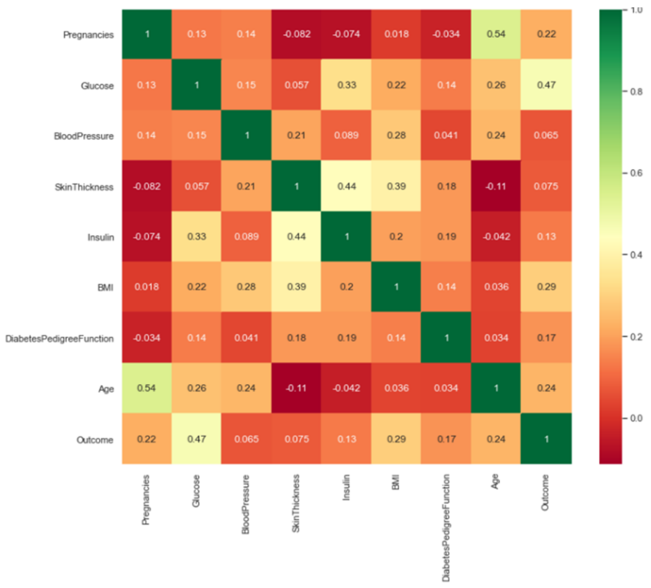
```
Old Shape:  (759, 9)
New Shape:  (726, 9)
```

- *Scaling the dataset*
  - Since every independent variable has a different range, it is necessary to standardise the dataset in order to obtain more accurate results.
  - Using the StandardScaler(), all attributes are scaled independently to values with **mean=0** and **variance=1**
  - Formula used:

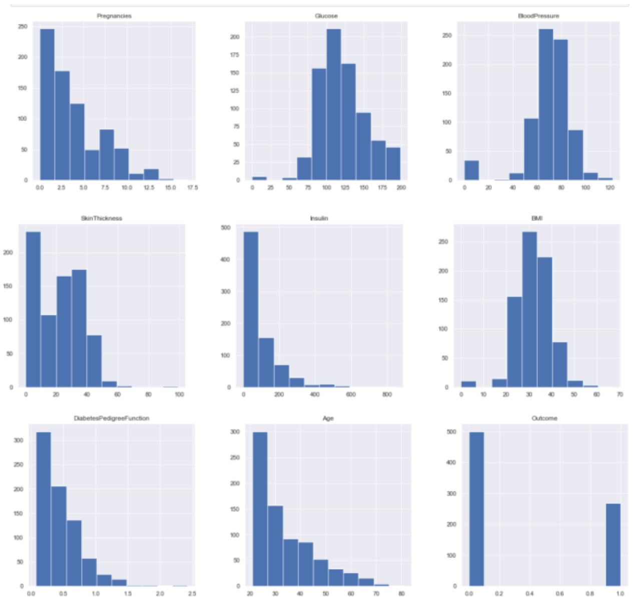
$$x_{\text{scaled}} = \frac{x - mean}{sd}$$

- Fit() calculates the scaling features of each attribute
- Transform() applies the learnt values of the scaling features to the dataset
- Fit_transform() performs both these steps in a single function

- *Identifying correlation between features*
  - A heatmap can easily identify multicollinearity. Presence of multicollinearity will hinder the accuracy of the model.

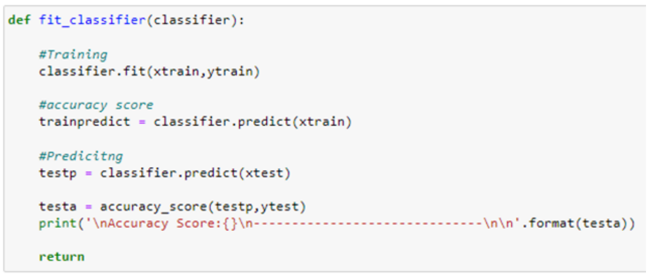


VI. An Overview of the Data



VII. The Heart of the Model

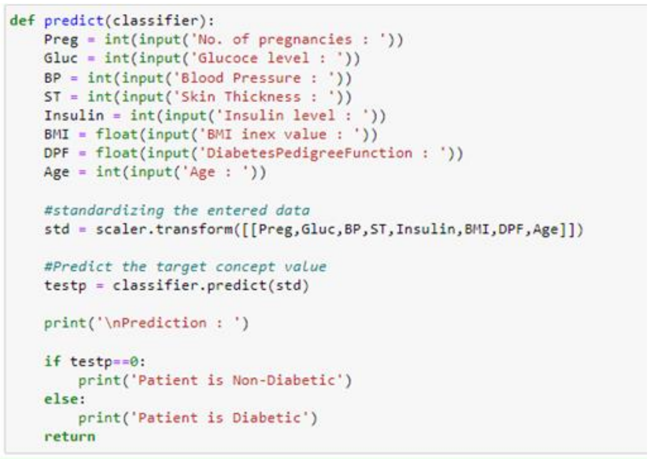
- A common function fit_classifier() has been designed which will take the type of classifier as input, fit the said classifier on the data and return the accuracy and score of the model.



```
def fit_classifier(classifier):

    #Training
    classifier.fit(xtrain,ytrain)

    #accuracy score
    trainpredict = classifier.predict(xtrain)

    #Predicitng
    testp = classifier.predict(xtest)

    testa = accuracy_score(testp,ytest)
    print('\nAccuracy Score:{}\n-----------------------------\n\n'.format(testa))

    return
```

VIII. Instance Based Predicting System

- A function predict() has been created that classifies individual examples given to the classifier on the fly.
- Input is taken from the user, the data entered is scaled and this scaled data is fed into the classifier.



```
def predict(classifier):
    Preg = int(input('No. of pregnancies : '))
    Gluc = int(input('Glucoce level : '))
    BP = int(input('Blood Pressure : '))
    ST = int(input('Skin Thickness : '))
    Insulin = int(input('Insulin level : '))
    BMI = float(input('BMI inex value : '))
    DPF = float(input('DiabetesPedigreeFunction : '))
    Age = int(input('Age : '))

    #standardizing the entered data
    std = scaler.transform([[Preg,Gluc,BP,ST,Insulin,BMI,DPF,Age]])

    #Predict the target concept value
    testp = classifier.predict(std)

    print('\nPrediction : ')

    if testp==0:
        print('Patient is Non-Diabetic')
    else:
        print('Patient is Diabetic')
    return
```

IX. An Overview of the Algorithms Used

- Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. It finds a hyperplane in an N-dimensional space that classifies the data points based on which part of the plane it lies in.
- Decision Tree – Each node represents a test on and attribute and each branch of the node represents a possible value of the attribute. The node is classified down the tree till it reaches a leaf node where a classification is provided to it.
- Random forest - It builds decision trees on different samples of the data and takes their majority vote for classification and average in case of regression.
- Naïve Bayes - Naive Bayes is called naive because it assumes that each input variable is independent. It finds the MAP hypothesis and gives the classification according to that.
- KNN Classification – The classification of the nearest k neighbours is assigned to the new instance

X. Accuracy of the Various Algorithms Used

- SVM

  Accuracy Score: 0.7602
- Decision Tree

  Accuracy Score: 0.6780
- Random Forest

  Accuracy Score: 0.7739
- Naïve Bayes

  Accuracy Score: 0.7465
- KNN Classifier

  Accuracy Score: 0.7602

XI. Conclusion

- Random Forest classifier has the highest accuracy score, so it should be used for future classification. An accurate model has been conceived and can be used for early detection of diabetes.
- The model's predictions are quite accurate and can help many people by detecting diabetes at an early stage.