**AI BASED DIABETICS PREDICTION SYSTEMS: PHASE 3**

**DEVELOPEMENT PART 1**

**Process and Data:**

Predicting diabetes using a Random Forest algorithm involves several steps, including data collection, preprocessing, model building, and evaluation. Here's an overview of the process:

* **Data Collection:** Gather a dataset that contains relevant features such as age, gender, BMI, blood pressure, and other health indicators. You can use public health datasets or collect your own data.
* **Data Preprocessing:** Prepare the data for analysis. This typically involves:

Handling missing values (imputation).

Encoding categorical variables (e.g., one-hot encoding).

Scaling or normalizing numerical features.

Splitting the data into training and testing sets for model evaluation.

* **Feature Selection:** You can use techniques like feature importance to identify the most relevant features for diabetes prediction.
* **Random Forest Model**: Random Forest is an ensemble learning method. To build a Random Forest model:

Choose the number of trees (n\_estimators) for the forest.

Train each tree on a random subset of the data with replacement (bagging).

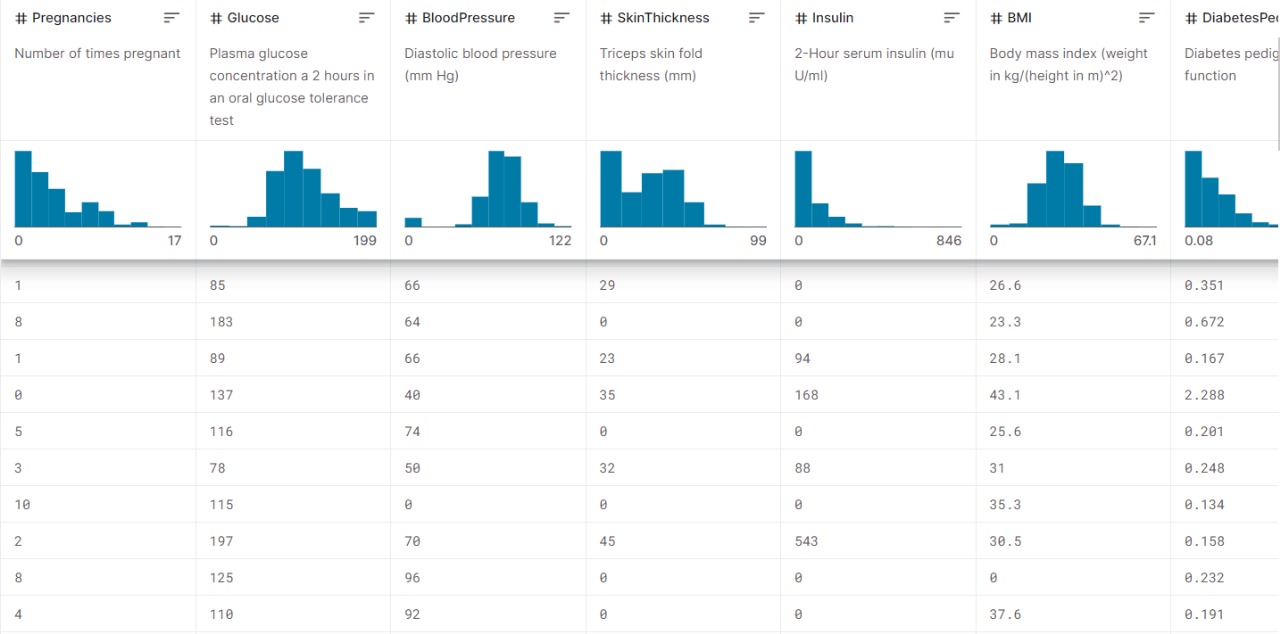
At each split in a tree, choose a random subset of features to split on.

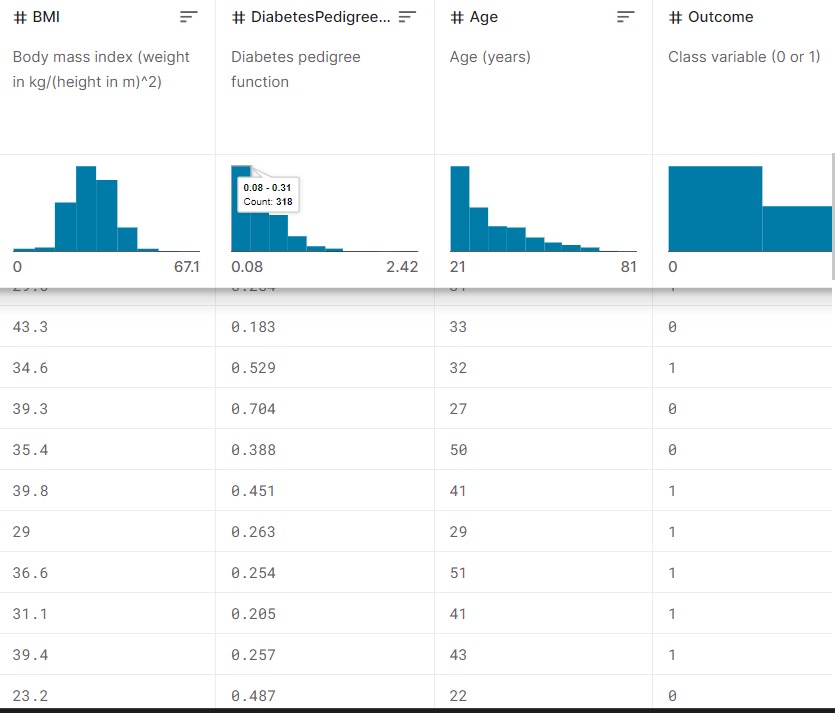
Aggregate the predictions from all trees to make the final prediction.

* **Model Training:** Fit the Random Forest model to the training data using the fit method.
* **Model Evaluation:** Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC AUC on the test dataset. This helps you assess how well the model predicts diabetes.
* **Hyperparameter** **Tuning** : You can fine-tune hyperparameters of the Random Forest, such as the number of trees, maximum depth, and minimum samples per leaf, to optimize performance.
* **Deployment**: If the model performs well, you can deploy it in a healthcare setting for diabetes prediction.
* **Monitoring and Maintenance**: Continuously monitor the model's performance and update it as necessary, as healthcare data and patient populations change.
* **Interpretability**: Random Forest models can provide feature importance scores, which help in understanding which features are most influential in predicting diabetes.

**AI Based Diabetes Prediction Systems Dataset link:**

<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>





**Python Implementation:**

The dataset is originally collected and circulated by “National Institute of Diabetes and Digestive and Kidney Diseases” which is available at Kaggle in the name of Pima Indians Diabetes Database. The main objective is to predict whether a patient has diabetics or not, based on the diagnostic measurements gathered in the database.

We’ll start with importing Pandas and NumPy into our python environment and loading a .csv dataset into a pandas dataframe named df. To see the first five records from the dataset we use pandas df.head() function. We’ll also use seaborn and matplotlib for visualization. Each and every examples shown in this article are verified on a Jupyter notebook.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#importing dataset

df = pd.read\_csv('../input/pima-indians-diabetes-database/diabetes.csv')

df.head()

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

The dataset contains 768 observable with eight feature variables and one target variable. Before starting to analyze the data and draw any conclusions, it is essential to understand the presence of missing values in any dataset. To do so the simplest way is to use df.info() function which will provide us the column names with the number of non-null values in each column.

df.dtypes

Pregnancies int64

Glucose int64

BloodPressure int64

SkinThickness int64

Insulin int64

BMI float64

DiabetesPedigreeFunction float64

Age int64

Outcome int64

dtype: object

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

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

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

According to the output we don’t observe any null values. But there are five features such as Glucose, BloodPressure, SkinThickness, Insulin and BMI contains zero values which is not possible in the medical history. We will consider these values as missing values. We’ll replace the zero values to NaN and then impute them with their mean value.

df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

# making a list of columns with total number of missing values

print('Column'+ '\t\t\t\t Total missing Values'+'\t\t\t\t % of missing values')

#print("\n")

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

Column Total missing Values % of missing values

Pregnancies 0 0.00

Glucose 5 0.65

BloodPressure 35 4.56

SkinThickness 227 29.56

Insulin 374 48.70

BMI 11 1.43

DiabetesPedigreeFunction 0 0.00

Age 0 0.00

Outcome 0 0.00

df['Glucose'].fillna(df['Glucose'].mean(), inplace=True)

df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace=True)

df['SkinThickness'].fillna(df['SkinThickness'].mean(), inplace=True)

df['Insulin'].fillna(df['Insulin'].mean(), inplace=True)

df['BMI'].fillna(df['BMI'].mean(), inplace=True)

# making a list of columns with total number of missing values

print('Column'+ '\t\t\t\t Total missing Values'+'\t\t\t\t % of missing values')

#print("\n")

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

Column Total missing Values % of missing values

Pregnancies 0 0.00

Glucose 0 0.00

BloodPressure 0 0.00

SkinThickness 0 0.00

Insulin 0 0.00

BMI 0 0.00

DiabetesPedigreeFunction 0 0.00

Age 0 0.00

Outcome 0 0.00