Diabetes Prediction model

A MACHINE LEARNING ANALYSIS

SUBMITTED TO

DR SAKTHI BALAN

GROUP MEMBERS:

Bhavit Jain (18UCS089)
Anmol Jhanwar (18UCC138)
Brajesh Agarwal (18UCC015)
Geetansh Moondra (18UCC046)

INDEX

Project Objective

Based on certain diagnostic tests used in the dataset, the purpose of the dataset is to forecast whether or not a patient has diabetes. The databases consist of several variables for medical prediction and one target variable, the outcome. Predictor variables include the number of births, their BMI, insulin level, age, and so on that the patient has had.

The Dataset Name and Origin

Name : Diabetes Data Set
Origin : UCI Machine Learning Repository
Dataset URL :
https://raw.githubusercontent.com/MrUnchained1303/IDS_Diabetes/main/diabetes.csv

Dataset Specification

Abstract: This diabetes dataset is from AIM '94

Data Set Characteristics:	Multivariate, Time-Series	**Number of Instances:**	N/A	**Area:**	Life
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	20	**Date Donated**	N/A
Associated Tasks:	N/A	**Missing Values?**	N/A	**Number of Web Hits:**	524331

Dataset Description

Diabetes files are made up of 4 fields per document. Each area is divided by a tab and a new line divides each record.

Format:

1. Date : MM-DD-YYYY
2. Time : XX:YY
3. Value
4. Code

Importing Libraries

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Descriptive Statistics

```
[3]  url = 'https://raw.githubusercontent.com/MrUnchained1303/IDS_Diabetes/main/diabetes.csv'
```

```
[4]  # Importing dataset
     dataset = pd.read_csv(url)
```

```
# Preview data
dataset.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

```
[6] # Dataset dimensions - (rows, columns)
    dataset.shape
```

```
(768, 9)
```

```
[7] # Features data-type
    dataset.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
```

Statistical Summary:

```
[8]  # Statistical summary
     dataset.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

```
[9]  # Count of null values
     dataset.isnull().sum()
```

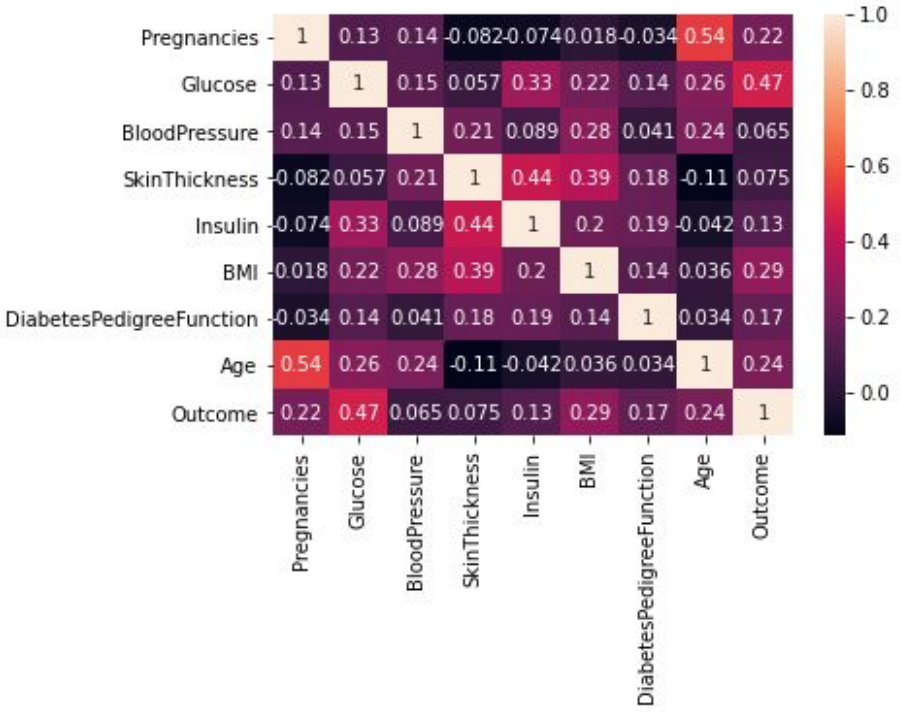
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Basic Observation :

1. The dataset has a total of 768 records and 9 functions.
2. Each function may have either an integer or a float form of data.
3. Some characteristics, such as glucose, blood pressure, insulin, BMI, have zero values that reflect missing information.
4. In the dataset, zero NaN values are present..
5. 1 represents positive for diabetes in the result section and 0 represents negative for diabetes.

Correlation Heatmap:

```
[14] # Heatmap
     sns.heatmap(dataset.corr(), annot = True)
     plt.show()
```



Inference from heatmap

We can see that the correlation heat map has a high relationship between the outcome and [Glucose, BMI, Era, Insulin]. We can choose these characteristics to take feedback from the consumer and forecast the effect.

Data Preprocessing

In data preprocessing, we have replaced NaN and zero values with mean values.

Algorithm Application

Logistic regression:

Logistic regression is a classification algorithm used to assign a discrete set of categories to observations. Logistic regression transforms the output using the logistic sigmoid function to return a probability value that can then be mapped to two or three distinct groups, unlike linear regression that produces constant number values.

It is possible to forecast different things by logistic regression:

Logistic Regression will assist us in predicting whether the student has passed or failed or not. Predictions of discrete logistic regression are . Probability scores underlying the classifications of the model can also be viewed.

```
[24] # Logistic Regression Algorithm
     from sklearn.linear_model import LogisticRegression
     logreg = LogisticRegression(random_state = 42)
     logreg.fit(X_train, Y_train)

     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
```

K-Nearest Neighbor(KNN):

K-Nearest Neighbor is one of Machine Learning's most simple and efficient classification algorithms. It belongs to the supervised field of learning and sees extreme use in pattern recognition, data mining and detection of intrusion.

In real-life situations, it is commonly true because it is non-parametric, which means it does not make any underlying conclusions regarding data delivery (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).

Any prior information (also called training data) is provided to us, which classifies coordinates into attribute-defined classes.

In a case where only two classes are available (e.g. Red/Blue), K can be held as an odd number, so that a simple majority can be calculated. With enhanced K, by different classifications, we get clearer, more defined distinctions. The accuracy of the classifier above also improves if we increase the number of data points in the training set.

Plot N - Neighbors:

```
[26] # Plotting a graph for n_neighbors
     from sklearn import metrics
     from sklearn.neighbors import KNeighborsClassifier

     X_axis = list(range(1, 31))
     acc = pd.Series()
     x = range(1,31)

     for i in list(range(1, 31)):
         knn_model = KNeighborsClassifier(n_neighbors = i)
         knn_model.fit(X_train, Y_train)
         prediction = knn_model.predict(X_test)
         acc = acc.append(pd.Series(metrics.accuracy_score(prediction, Y_test)))
     plt.plot(X_axis, acc)
     plt.xticks(x)
     plt.title("Finding best value for n_estimators")
     plt.xlabel("n_estimators")
     plt.ylabel("Accuracy")
     plt.grid()
     plt.show()
     print('Highest value: ',acc.values.max())
```


```
Highest value:  0.7857142857142857
```

Implementation of Algorithm:

```
[27] # K nearest neighbors Algorithm
     from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors = 24, metric = 'minkowski', p = 2)
     knn.fit(X_train, Y_train)

     KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                          metric_params=None, n_jobs=None, n_neighbors=24, p=2,
                          weights='uniform')
```

Support Vector Machine:

Support-vector machines are supervised learning models that use related learning algorithms, analyze data used for classification and regression analysis. A SVM training algorithm creates a model that assigns new examples to one or the other subclass, each classified as belonging to one or the other of two categories, making it a linear linear conditional classifier that is non-probabilistic.

An SVM model is a representation, mapped in such a way that a simple distance is as wide as possible, of the examples as space points. New cases, based on the side of the gap they land on, are then tracked and calculated to belong to a break.

```
[28] # Support Vector Classifier Algorithm
     from sklearn.svm import SVC
     svc = SVC(kernel = 'linear', random_state = 42)
     svc.fit(X_train, Y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=42, shrinking=True, tol=0.001,
    verbose=False)
```

Naive Bayes:

The Naive Bayes algorithm is an algorithm for supervised learning based on the Bayes theorem and used to overcome classification problems. It is mostly used in text classification, which requires a high-dimensional training dataset. One of the easiest and most powerful classification algorithms is the Naïve Bayes Classifier, which helps to construct fast machine learning models that can predict easily.

```
[29] # Naive Bayes Algorithm
     from sklearn.naive_bayes import GaussianNB
     nb = GaussianNB()
     nb.fit(X_train, Y_train)

     GaussianNB(priors=None, var_smoothing=1e-09)
```

Decision Tree:

Decision trees are the most powerful and popular tool for classification and prediction. A Decision tree is a tree-like flowchart where each internal node represents a query for an attribute, each branch represents a test outcome, and each leaf node (terminal node) has a name for the class.

```
[30] # Decision tree Algorithm
     from sklearn.tree import DecisionTreeClassifier
     dectree = DecisionTreeClassifier(criterion = 'entropy', random_state = 42)
     dectree.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

Making Prediction on Test Dataset:

```
[31] # Making predictions on test dataset
     Y_pred_logreg = logreg.predict(X_test)
     Y_pred_knn = knn.predict(X_test)
     Y_pred_svc = svc.predict(X_test)
     Y_pred_nb = nb.predict(X_test)
     Y_pred_dectree = dectree.predict(X_test)
```

Model Evaluation

Accuracy:

The percentage of precise forecasts is known as the precision of the test data. It can be conveniently calculated by dividing the number of correct predictions by the number of total predictions.

```
[32] # Evaluating using accuracy_score metric
     from sklearn.metrics import accuracy_score
     accuracy_logreg = accuracy_score(Y_test, Y_pred_logreg)
     accuracy_knn = accuracy_score(Y_test, Y_pred_knn)
     accuracy_svc = accuracy_score(Y_test, Y_pred_svc)
     accuracy_nb = accuracy_score(Y_test, Y_pred_nb)
     accuracy_dectree = accuracy_score(Y_test, Y_pred_dectree)
```

Accuracy check for different Algorithms

```
[33] # Accuracy on test set
     print("Logistic Regression: " + str(accuracy_logreg * 100))
     print("K Nearest neighbors: " + str(accuracy_knn * 100))
     print("Support Vector Classifier: " + str(accuracy_svc * 100))
     print("Naive Bayes: " + str(accuracy_nb * 100))
     print("Decision tree: " + str(accuracy_dectree * 100))

     Logistic Regression: 72.07792207792207
     K Nearest neighbors: 78.57142857142857
     Support Vector Classifier: 73.37662337662337
     Naive Bayes: 71.42857142857143
     Decision tree: 68.18181818181817
```

From the above comparison, we can observe that K Nearest neighbors gets the highest accuracy of 78.57%

Confusion Matrix

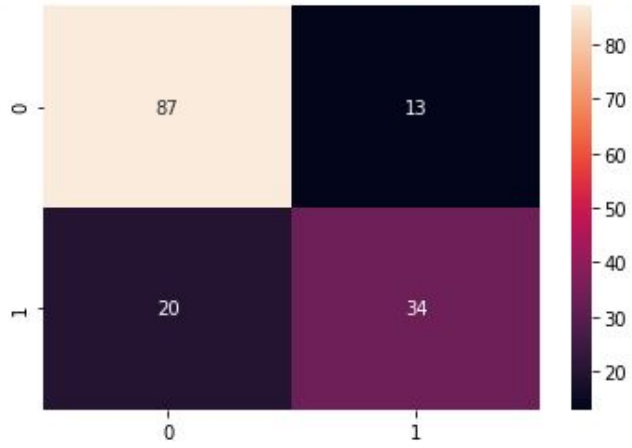
```
[35] # Confusion matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(Y_test, Y_pred_knn)
     cm
```

```
array([[87, 13],
       [20, 34]])
```

Heatmap of Confusion matrix

```
[36] # Heatmap of Confusion matrix
     sns.heatmap(pd.DataFrame(cm), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f465aebdef0>
```



Classification Report

```
[37] # Classification report
     from sklearn.metrics import classification_report
     print(classification_report(Y_test, Y_pred_knn))
```

```
                  precision    recall  f1-score   support

              0.0       0.81      0.87      0.84       100
              1.0       0.72      0.63      0.67        54

         accuracy                           0.79       154
        macro avg       0.77      0.75      0.76       154
     weighted avg       0.78      0.79      0.78       154
```

Conclusions

The feature-to-feature correlations and outcome-to-feature correlations were taken into account in this. Between characteristics: greater correlations means that one of them can be dropped. However the high correlation between a function and the result means that the function is important and contains a lot of data.

As we can see all of the models performed well. Among all the classifiers we used in the project, the K- Nearest Neighbour performed best. We can also see the various f-score values in the KNN classification report.

References

- sklearn.metrics.classification_report — scikit-learn 0.23.2 documentation
- Understanding Confusion Matrix. When we get the data, after data... | by Sarang Narkhede
- seaborn.heatmap — seaborn 0.11.0 documentation
- sklearn.metrics.accuracy_score — scikit-learn 0.23.2 documentation
- Machine Learning Classifiers. What is classification? | by Sidath Asiri