Abstract

The emerging increase of diabetes in across the world, that recently affects around million people at worldwide, of which extra than one-third go undetected in near the beginning stage , a strong need for supporting the health check decision-making process is generated in diabetes.

In this Project, I am building a system that can predict whether a person has diabetes or not with the help of Machine Learning. This project is done in Python. In this project, Support Vector Machine model has been used for the prediction. This project will portray how data related to diabetes can be leveraged to predict if a person has diabetes or not.

More specifically, this Project will also tells how machine learning can be utilized to predict other diseases.

Table of Contents

Introduction

Diabetes is a disease that occurs when the blood glucose level becomes high, which ultimately leads to other health problems such as heart diseases, kidney disease, etc. Diabetes is caused mainly due to the consumption of highly processed food, bad consumption habits, etc. According to WHO, the number of people with diabetes has been increased over the years.

Prerequisites

- Python 3.7
- Scikit Learn, Numpy 1.19.5, Pandas 1.1.5
- Colaboratory
- Basic understanding of supervised machine learning methods: specifically classification.

Data Collection

I have taken this dataset from Kaggle (originally from the **National Institute of Diabetes and Digestive and Kidney Diseases**). The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of **PIMA Indian heritage** .

Implementation

```
# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('/content/diabetes.csv')
```

Data Analysis

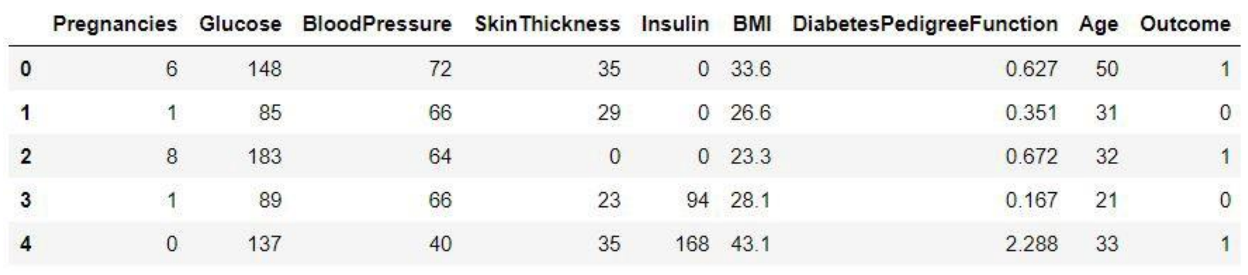
The datasets consists of several medical predictor variables and one target variable, `Outcome`. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Attribute Information:

1. Pregnancies: *Number of times pregnant*
2. Glucose: *Plasma glucose concentration a 2 hours in an oral glucose tolerance test*
3. BloodPressure: *Diastolic blood pressure (mm Hg)*
4. SkinThickness: *Triceps skin fold thickness (mm)*
5. Insulin: *2-Hour serum insulin (mu U/ml)*
6. BMI: Body *mass index (weight in kg/(height in m)^2)*
7. DiabetesPedigreeFunction: *Diabetes pedigree function*
8. Age: *Age (years)*
9. Outcome: *Class variable (0 or 1)*

Implementation

```
# printing the first 5 rows of the dataset
diabetes_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

```
# number of rows and Columns in this dataset
```

```
diabetes_dataset.shape
```

Data Cleaning

The next phase of the machine learning work flow is data cleaning. Considered to be one of the crucial steps of the workflow, because it can make or break the model. There is a saying in machine learning **“Better data beats fancier algorithms”**, which suggests better data gives you better resulting models.

There are several factors to consider in the data cleaning process.

1. Duplicate or irrelevant observations.
2. Bad labeling of data, same category occurring multiple times.
3. Missing or null data points.
4. Unexpected outliers.

Missing or Null Data points

We can find any missing or null data points of the data set (if there is any) using the following pandas function.

We can observe that there are no data points missing in the data set. If there were any, we should deal with them accordingly.

Unexpected Outliers

When analyzing the histogram we can identify that there are some outliers in some columns. We will further analyze those outliers and determine what we can do about them.

Blood pressure: By observing the data we can see that there are 0 values for blood pressure. And it is evident that the readings of the data set seem wrong because a living person cannot have a diastolic blood pressure of zero. By observing the data we can see 35 counts where the value is 0.

Plasma glucose levels: Even after fasting glucose levels would not be as low as zero. Therefore zero is an invalid reading. By observing the data we can see 5 counts where the value is 0.

Skin Fold Thickness: For normal people, skin fold thickness can't be less than 10 mm better yet zero. Total count where value is 0: 227.

BMI: Should not be 0 or close to zero unless the person is really underweight which could be life-threatening.

Insulin: In a rare situation a person can have zero insulin but by observing the data, we can find that there is a total of 374 counts.

Here are several ways to handle invalid data values :

1. *Ignore/remove these cases:* This is not actually possible in most cases because that would mean losing valuable information. And in this case "skin thickness" and "insulin" columns mean to have a lot of invalid points. But it might work for "BMI", "glucose "and "blood pressure" data points.

2. *Put average/mean values*: This might work for some data sets, but in our case putting a mean value to the blood pressure column would send a wrong signal to the model.

3. *Avoid using features:* It is possible to not use the features with a lot of invalid values for the model. This may work for “skin thickness” but it's hard to predict that.

By the end of the data cleaning process, we have come to the conclusion that this given data set is incomplete. Since this is a demonstration for machine learning we will proceed with the given data with some minor adjustments. We will remove the rows which the “BloodPressure”, “BMI” and “Glucose” are zero.

Feature Engineering

Feature engineering is the process of transforming the gathered data into features that better represent the problem that we are trying to solve to the model, to improve its performance and accuracy.

Feature engineering creates more input features from the existing features and also combines several features to produce more intuitive features to feed to the model.

“ Feature engineering enables us to highlight the important features and facilitate to bring domain expertise on the problem to the table. It also allows avoiding overfitting the model despite providing many input features”.

The domain of the problem we are trying to tackle requires lots of related features. Since the data set is already provided, and by examining the data we can’t further create or dismiss any data at this point. In the data set, we have the following features.

‘Pregnancies’, ‘Glucose’, ‘Blood Pressure’, ‘Skin Thickness’, ‘Insulin’, ‘BMI’, ‘Diabetes Pedigree Function’, ‘Age’

By a crude observation, we can say that the ‘Skin Thickness’ is not an indicator of diabetes. But we can’t deny the fact that it is unusable at this point.

Therefore we will use all the features available. We separate the data set into features and the response that we are going to predict. We will assign the features to the **X variable** and the response to the **Y variable**.

Generally feature engineering is performed before selecting the model. However, for this tutorial we follow a different approach. Initially, we will be utilizing all the features provided in the data set to the model, we will revisit features engineering to discuss feature importance on the selected model.

Data Standardization

Data standardization is the process of converting data to a common format to enable users to process and analyze it. Most organizations utilize data from a number of sources; this can include data warehouses, lakes, cloud storage, and databases. However, data from disparate sources can be problematic if it isn't uniform, leading to difficulties down the line (e.g., when you use that data to produce dashboards and visualizations, etc.).

Data standardization is crucial for many reasons. First of all, it helps you establish clear, consistently defined elements and attributes, providing a comprehensive catalog of your data. Whatever insights you're trying to get or problems you're attempting to solve, properly understanding your data is a crucial starting point.

Getting there involves converting that data into a uniform format, with logical and consistent definitions. These definitions will form your metadata — the labels that identify the what, how, why, who, when, and where of your data. That's the basis of your data standardization process.

From an accuracy perspective, standardizing the way we label data will improve access to the most relevant and current information. This will help make your analytics and reporting easier. Security-wise, mindful cataloging forms the basis of a powerful authentication and authorization approach, which will apply security restrictions to data items and data users as appropriate.

Implementation

```
input_data =(4,110,92,0,0,37.6,0.191,30)

#changing the input_data to numpy array
input_data_as_numpy_array =np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data= scaler.transform(input_data_reshaped)
print(std_data)
```

Result
```
[[ 0.04601433 -0.34096773  1.18359575 -1.28821221 -0.69289057
0.71168975


  -0.84827977 -0.27575966]]
```

Model Selection

Model selection or algorithm selection phase is the most exciting and the heart of machine learning. It is the phase where we select the model which performs best for the data set at hand.

First, we will be calculating the **“Classification Accuracy (Testing Accuracy)”** of a given set of classification models with their default parameters to determine which model performs better with the diabetes data set.

We will import the necessary libraries for the notebook. We import **Support Vector Classifier.**

We will initialize the classifier models with their default parameters and add them to the Model.

Evaluation Methods

It is a general practice to avoid training and testing on the same data. The reasons are that the goal of the model is to predict **out-of-sample data**, and the model could be overly complex leading to **overfitting**. To avoid the aforementioned problems, there is one precaution(Train/Test Split).

We will import *“train_test_split”* for **train/test split** and *“accuracy_score”* is to evaluate the accuracy of the model in the train/test split method.

Train/Test Split

This method split the data set into two portions: a **training set** and a **testing set**. The **training set** is used to train the model. And the **testing set** is used to test the model, and evaluate the accuracy.

Train/Test Split with Scikit Learn :

Next, we can split the features and responses into train and test portions. We stratify (a process where each response class should be represented with equal proportions in each of the portions) the samples.

Implementation

```
X_train, X_test, Y_train,Y_test =train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)

print(X.shape,X_train.shape,X_test.shape)
```

Output

```
(768, 8) (614, 8) (154, 8)
```

Model Evaluation(Result)

Calculating the accuracy of model using the *“accuracy_score”*.

Logistic Regression model has managed to achieve a classification accuracy of 77.27 %.

<u>Implementation</u>

```
prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0]==0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

<u>Output</u>

```
[0]
The person is not diabetic
```