ML Mini Project

Diabetes Prediction Project

Rahul Guglani(20103116)

Samneet Singh(20103128)

Saarthak Markandey(20103125)

Submitted to: Dr. Jagdeep Kaur Ma’am

15.04.2022

4TH Sem CSE Mini Project

Machine Learning

Introduction

Glucose is a type of sugar that is used as fuel by the body. When you eat, your body converts food into glucose. The glucose then goes into your bloodstream and is carried throughout the body to provide energy to all of your cells. In order for glucose to move from your bloodstream into your cells, you need insulin. Insulin carries the glucose , or sugar, in your bloodstream into your cells. Insulin is a hormone made by the pancreas, an organ in upper part of abdomen.

If your body has a problem making or using insulin, the glucose in your

bloodstream cannot get into your cells. As a result, glucose stays in the blood

(high blood sugar) and the cells do not get enough glucose. A diagnosis of prediabetes or diabetes is made when glucose stays at higher-than-normal levels

(also called hyperglycemia). According to the World Health Organization more than 420 million people worldwide live with diabetes. Diabetes is a major cause of blindness, amputation, kidney failure and cardiovascular disease.

Motivation

Diabetes is an increasingly growing health issue due to our inactive lifestyle. If

it is detected in time then through proper medical treatment, adverse effects can be prevented. To help in early detection, technology can be used very reliably and efficiently. Using machine learning I have built a predictive model that can predict whether the patient has diabetes or not and I appreciate the essence of machine learning in solving some of the problems that plague humanity.

Decision Tree Algorithm

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. It is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node(terminal node) holds a class label.

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Feature Importance

Feature Importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.

Feature Importance scores play an important role in a predictive modeling project, including providing insight into the data, insight into the model, and the basis for dimensionality reduction and feature selection that can improve the efficiency and effectiveness of a predictive model on the problem.

Confusion Matrix

It is tabular summary of the number of correct and incorrect predictions made by a classifier. It is used to measure the performance of a classification model.

There are some terms in the confusion matrix:

1. True Positive-It is an outcome where the model correctly predicts the positive class.

2. True Negative-It is an outcome where the model correctly predicts the negative class.

3. False Positive-It is an outcome where the model incorrectly predicts the positive class.

4. False Negative-It is an outcome where the model incorrectly predicts the negative class.

Classification Report

A classification report is used to measure the quality of predictions from a

classification algorithm. The classification report visualizer displays the precision,recall,F1and support scores for the model.

1. Precision- Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class, it is defined as the ratio of true positives to the sum of a true positive and false positive.

Precision = TP/ (TP + FP)

2. Recall- Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

Recall = TP/(TP+FN)

3. F1 score- F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Therefore, this score takes both false positives and false negatives into account. It gives a better measure of the incorrectly classified cases than the accuracy metric. So, F1 score is more useful than accuracy especially if you have an uneven class distribution.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

4. Support- Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

Software Required

1. Anaconda Navigator(Jupyter Notebook)

2. Windows 10

3. Language-python3

Hardware Required

1. Processor : Intel core i5 10th Gen

2. Random Memory : 8.00 GB

Prerequisites

You'll need to install the following libraries with pip:

pip install numpy pandas sklearn matplotlib seaborn warnings

You'll need to install Jupyter Lab to run your code. Get to your command prompt and run the following command:

C:\Users\DataFlair>jupyter lab

You'll see a new browser window open up; create a new console and use it to run your code. To run multiple lines of code at once, press Shift+Enter.

Methodology

Follow the steps for detecting the diabetes:

1. Necessary libraries are imported and then read the data into a DataFrame of first few records.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        from sklearn import metrics
        from sklearn.metrics import classification_report,confusion_matrix
        #Ignore warnings
        warnings.filterwarnings("ignore")
        # Seaborn visualization set up
        %matplotlib inline
        sns.set_style('darkgrid')
```

```
In [2]: # Reading the dataset
        data=pd.read_csv("diabetes.csv")
        data.head()
```

Out[2]:

	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

2. Return the tuple representing the dimensionality of the DataFrame using shape function and get the full summary of the DataFrame using info() function .

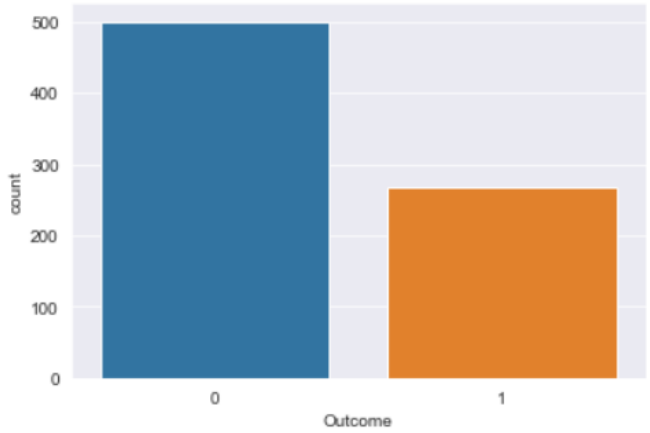
In this dataset there are total 768 rows and 9 columns. 7 columns have int64 data type and 2 columns have float64 data type. Memory usage is 54.1KB.

```
In [4]: data.shape
Out[4]: (768, 9)

In [5]: data.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
```

3. Check the distribution of the target variable i.e., Outcome.

```
In [6]: sns.countplot(x='Outcome',data=data)
Out[6]: <AxesSubplot:xlabel='Outcome', ylabel='count'>
```



From the plot above, the data contains more cases without diabetes (0) than those with diabetes (1).

4. Check for any missing values in the dataset. Then split the dataset before we train it. After successfully splitting the dataset, we train it using train_test_split.

```
In [7]: data.isnull().sum()
Out[7]: Pregnancies                 0
        Glucose                     0
        BloodPressure               0
        SkinThickness               0
        Insulin                     0
        BMI                         0
        DiabetesPedigreeFunction    0
        Age                         0
        Outcome                     0
        dtype: int64

In [8]: #Spitting the dataset
        x=data.drop('Outcome',axis=1)
        y=data['Outcome']

In [9]: #Train the dataset
        from sklearn.model_selection import train_test_split
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=101)
```

As shown above, there are no missing values in the dataset. The dataset had already been cleaned. After splitting X will contain all the

independent variables while y will have the dependent variable (Outcome).

5. Check the zero values in the dataset by printing for each variable. Now use 'mean' method that will computes the mean of the column and imputes the values that have zero with the mean.

```
In [10]: # Check columns with zero values
         print("Total number of rows:{0}",format(len(data)))
         print("Number of rows missing Pregnancies:{0}", format(len(data.loc[data['Pregnancies']==0])))
         print("Number of rows missing Glucose:{0}", format(len(data.loc[data['Glucose']==0])))
         print("Number of rows missing BloodPressure : {0}", format(len(data.loc[data['BloodPressure']==0])))
         print("Number of rows missing SkinThickness : {0}", format(len(data.loc[data['SkinThickness']==0])))
         print("Number of rows missing Insulin : {0}", format(len(data.loc[data['Insulin']==0])))
         print("Number of rows missing BMI : {0}", format(len(data.loc[data['BMI']==0])))
         print("Number of rows missing DiabetesPedigreeFunction : {0}",format(len(data.loc[data['DiabetesPedigreeFunction']==0])))
         print("Number of rows missing Age : {0}", format(len(data.loc[data['Age']])))

         Total number of rows:{0} 768
         Number of rows missing Pregnancies:{0} 111
         Number of rows missing Glucose:{0} 5
         Number of rows missing BloodPressure : {0} 35
         Number of rows missing SkinThickness : {0} 227
         Number of rows missing Insulin : {0} 374
         Number of rows missing BMI : {0} 11
         Number of rows missing DiabetesPedigreeFunction : {0} 0
         Number of rows missing Age : {0} 768

In [11]: # Imputing zeros values in the dataset
         from sklearn.impute import SimpleImputer
         fill_values=SimpleImputer(missing_values=0,strategy='mean')
         x_train=fill_values.fit_transform(x_train)
         x_test=fill_values.fit_transform(x_test)
```

There are a total of 768 zero values in the X dataset (independent variables) and we have also printed the zero values for each column.

6. Build the model using Decision Tree. Predict the labels of the data passed as argument based upon the learned or trained data obtained from the model. After that get the accuracy score, confusion matrix and classification report of the model.

```
In [12]: #Building the model using Decision Tree
         from sklearn.tree import DecisionTreeClassifier
         dtree=DecisionTreeClassifier()
         dtree.fit(x_train,y_train)

Out[12]: DecisionTreeClassifier()

In [13]: #Predict the labels of the data values on the basis of trained model
         predictions=dtree.predict(x_test)
         predictions

Out[13]: array([1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
                1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
                1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
                0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
                0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1], dtype=int64)

In [14]: #Getting the accuracy score for the DecisionTree
         print("Accuracy score = ",format(metrics.accuracy_score(y_test,predictions)))

         Accuracy score =  0.7445887445887446

In [15]: #Getting the confusion matrix and classification report
         print(confusion_matrix(y_test,predictions))
         print(classification_report(y_test,predictions))

         [[123  27]
          [ 32  49]]
                       precision    recall  f1-score   support

                    0       0.79      0.82      0.81       150
                    1       0.64      0.60      0.62        81

             accuracy                           0.74       231
            macro avg       0.72      0.71      0.72       231
         weighted avg       0.74      0.74      0.74       231
```

As shown above accuracy score of the model which is built by using Decision Tree is 0.7446.

7. Build another model using RandomForest. Predict the labels of the data passed as argument based upon the learned or trained data obtained from the model. After that get the accuracy score, confusion matrix and classification report of the model.

In [16]:
```
#Building the model using RandomForest
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=200)
rfc.fit(x_train,y_train)
```

Out[16]:
```
RandomForestClassifier(n_estimators=200)
```

In [17]:
```
#Predict the labels of the data values on the basis of the trained model
predictions=rfc.predict(x_test)
predictions
```

Out[17]:
```
array([1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1], dtype=int64)
```

In [18]:
```
#Getting the accuracy score for RandomForest
print("Accuracy Score = ",format(metrics.accuracy_score(y_test,predictions)))
```

```
Accuracy Score =  0.7748917748917749
```

In [19]:
```
#Getting the confusion matrix and classification report
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[125  25]
 [ 27  54]]
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       150
           1       0.68      0.67      0.68        81

    accuracy                           0.77       231
   macro avg       0.75      0.75      0.75       231
weighted avg       0.77      0.77      0.77       231
```

As shown above, the accuracy score of the model which is built by using

Random Forest is 0.7749.

8. Find out the Feature Importance and display it along with Feature names.

```
In [20]: #Getting feature importances
         rfc.feature_importances_

Out[20]: array([0.0736704 , 0.25754566, 0.08067697, 0.07740157, 0.09149771,
                0.15788287, 0.12503758, 0.13628724])
```

```
In [21]: #Display feature_names and their respective Importance
         df=pd.DataFrame({"Feature_name ":x.columns,"Importances":rfc.feature_importances_})
         df
```

Out[21]:

	Feature_name	Importances
0	Pregnancies	0.073670
1	Glucose	0.257546
2	BloodPressure	0.080677
3	SkinThickness	0.077402
4	Insulin	0.091498
5	BMI	0.157883
6	DiabetesPedigreeFunction	0.125038
7	Age	0.136287

9. Sort the Feature_names in descending order of their Importances.

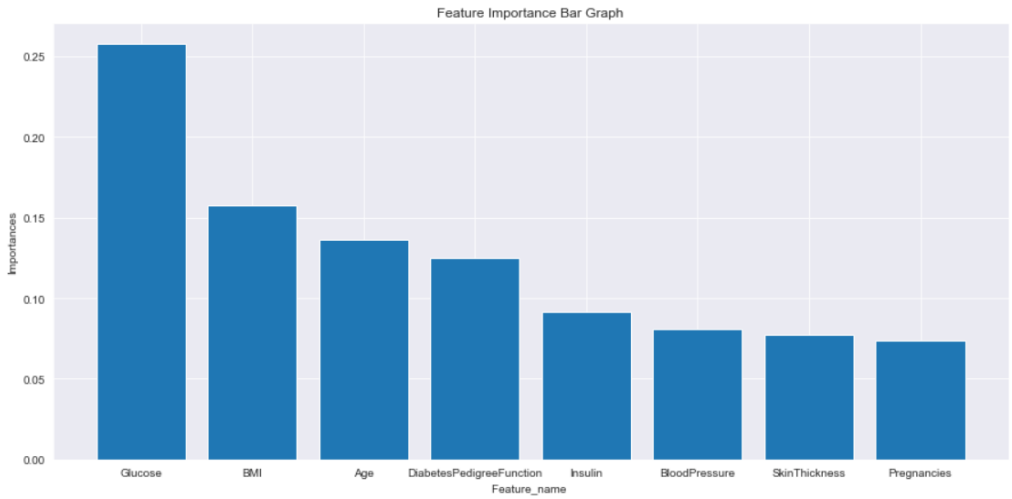
```
In [22]: df.sort_values(by =['Importances'],inplace = True,ascending = False)
         df
```

Out[22]:

	Feature_name	Importances
1	Glucose	0.257546
5	BMI	0.157883
7	Age	0.136287
6	DiabetesPedigreeFunction	0.125038
4	Insulin	0.091498
2	BloodPressure	0.080677
3	SkinThickness	0.077402
0	Pregnancies	0.073670

10. Plot the bar graph of Feature Importance.

```
In [29]: plt.figure(figsize=(15,7))
         plt.bar(df['Feature_name '],df['Importances'])
         plt.xlabel('Feature_name ')
         plt.ylabel('Importances')
         plt.title('Feature Importance Bar Graph')
         plt.show()
```



As shown in the graph, the most important feature in this diabetes prediction is Glucose followed by BMI.

11. Display the prediction probabilities of absence or presence of diabetes Respectively.

```
In [31]: print('prediction Probabilities')
         rfc.predict_proba(x_test)

         prediction Probabilities

Out[31]: array([[0.44 , 0.56 ],
                [0.15 , 0.85 ],
                [0.845, 0.155],
                [0.595, 0.405],
                [0.945, 0.055],
                [0.365, 0.635],
                [0.145, 0.855],
                [0.855, 0.145],
                [0.49 , 0.51 ],
                [0.78 , 0.22 ],
                [0.98 , 0.02 ],
                [0.805, 0.195],
                [0.65 , 0.35 ],
                [0.565, 0.435],
                [0.725, 0.275],
                [0.29 , 0.71 ],
                [0.17 , 0.83 ],
                [0.97 , 0.03 ]
```

The patient at index 0 has 44% chance of absence of diabetes, while the patient at index 1 has a 85% predicted chance of having diabetes.

Github Link

https://github.com/SaarthakMarkandey/MLProject_Diabetes_Prediction