DIABETES PREDICTION SYSTEM

Phase 4: Development Part 2

Indroduction:

In this phase we will discuss about the 2nd part development of AI Based diabetes Prediction. In this part we will continue the remaining of last phase. In the last phase we discussed about the data collection and cleaning

thedata . In this phase we will continue diabetes prediction by

- Selecting the machine learning algorithm
- Training the model
- Evaluating the performance

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1

7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0

Given Data :

Data Collection link:https://www.kaggle.com/datasets/mathchi/diabetes-data-set

LOGISTIC REGRESSION:

```
y = dataset_new['Outcome']

X = dataset_new.drop('Outcome', axis=1)

# Splitting X and Y

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.20, random_state = 42, stratify = dataset_new['Outcome'] )

from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(X_train, Y_train)

y_predict = model.predict(X_test)

/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,

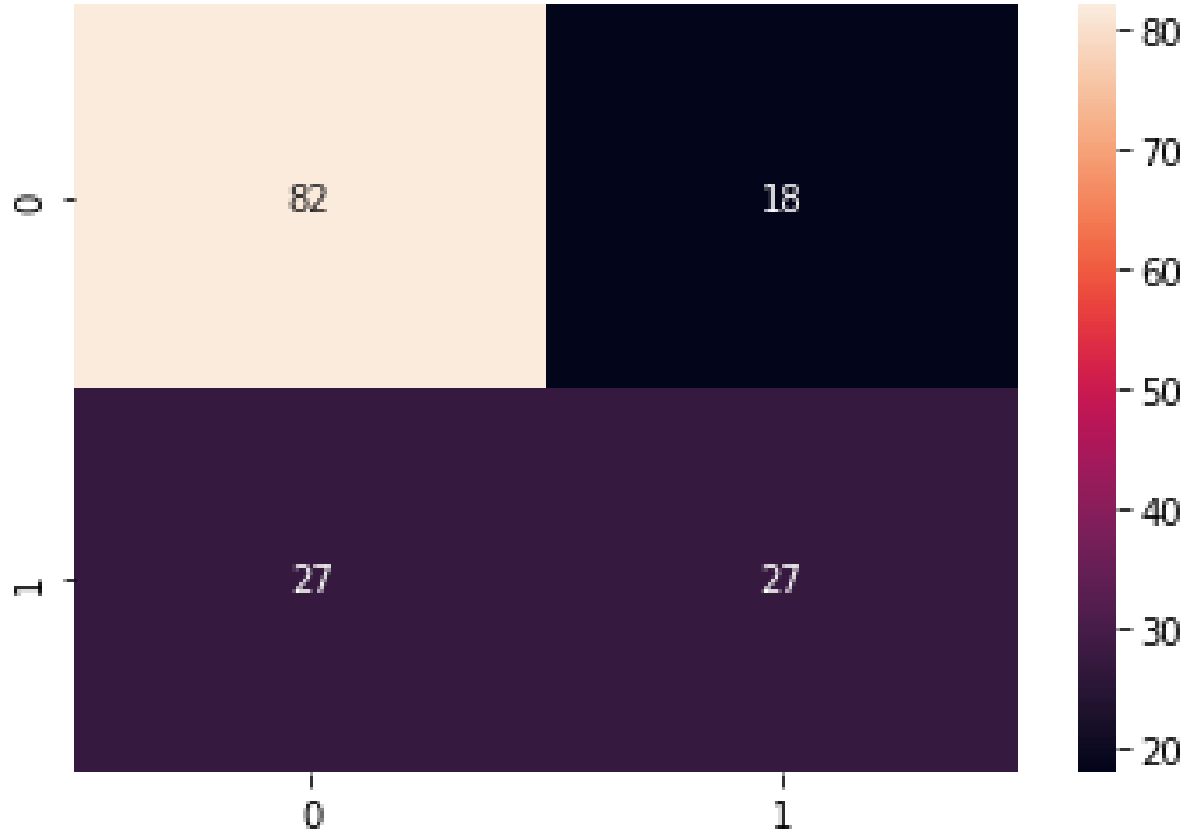
y_predict

array([1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0])

Confusion matrix

from sklearn.metrics import confusion_matrix

```
cm = confusion_matrix(Y_test, y_predict)
cm
array([[82, 18],
       [27, 27]])
# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cm), annot=True)
<AxesSubplot:>
```




```
from sklearn.metrics import accuracy_score
accuracy =accuracy_score(Y_test, y_predict)
accuracy
```

0.7077922077922078

Example: Let's check whether the person have diabetes or not using some random values

```
y_predict =
model.predict([[1,148,72,35,79.799,33.6,0.627,50]])

print(y_predict)

if y_predict==1:

    print("Diabetic")

else:

print("Non Diabetic")
```

[1]

Diabetic

/opt/conda/lib/python3.7/site-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names

"X does not have valid feature names, but"

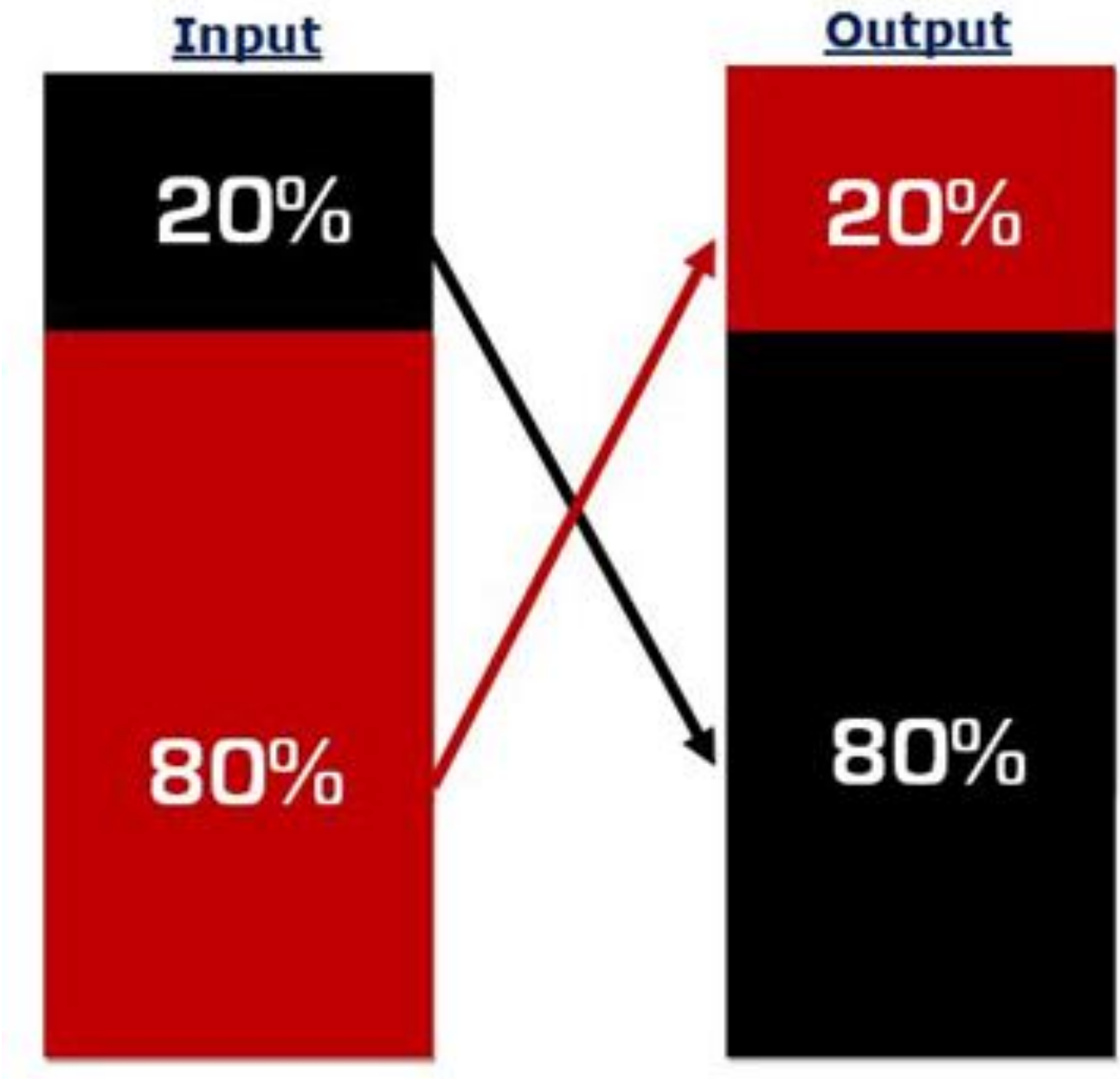
Model Preparation:

Now we are going to split our dataset into Train and Test sets. Usually, the training and test sets are divided in a ratio

of 80:20, respectively. Why is it 80:20 ratio? Well, the answer lies in the Pareto Principle. I am not going dive into the details and explain the whole idea of Pareto Principle, however I will give the basic idea of it:

"80 percent of the output from a given situation or system is determined by 20 percent of the input."

It is worth to mention, that it is not compulsory to use 80:20 ratio, since it can be 70:30 as well. However, if we want to achieve a better results in model training we are going to split our dataset, where 80% of the data will be used for training and other 20% for testing.



Simply speaking, if we use 80% of the data during testing, we will get a poor accuracy, while if we use 20% of the data during testing, we will get a better results. As was mentioned before, we can use any ratio we want, but we will stick to the standard ratio for now.

Below are some useful links where you can find out more information about Pareto Principle and its application in Machine Learning:

https://en.wikipedia.org/wiki/Pareto_principle#Mathematical_notes

https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d

https://eazybi.com/blog/the-80-20-rule

Before splitting data into training and test sets, it is necessary to identify our target and features columns. In this case, Target column will be "Result", while other 8 columns will be Features, based on which our model will try to predict the target.

```
X = df.drop(columns = 'Result', axis = 1)  # Our features

y = df['Result']  # Our target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, stratify = y, random_state = 2)

#prepare for training
```

The code line above was used to split the features and target columns into train and test sets randomly. We can display the output for features as follows:

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

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

Features contain 768 rows and 8 columns, while 614 rows was used for training, and 154 wors were used for testing.

Now its time to define the model we are going to use. Since we are solving a classification problem, we will use Logistic Regression model. There is a possibility to use other machine learning algorithms, for example: "K-Nearest Neighbors", "Random Forest", "Decision Tree", etc. However, since the dataset is pretty small, we will use them somewhere else.

model = LogisticRegression()

Model Training

Letsuse .fit() function to feed our model with the training sets for both: features and target.

model.fit(X_train, y_train)

LogisticRegression()

Perfect, now we will find the accuracy and precision scores for training and test data.

Finding Accuracy for training data

```
X_train_prediction = model.predict(X_train)

training_data_accuracy = accuracy_score(X_train_prediction, y_train)

training_data_precision = precision_score(X_train_prediction, y_train)

print('Accuracy on training data : ', training_data_accuracy)

print('Precision on training data : ', training_data_precision)
```

```
Accuracy on training data :  0.7801302931596091

Precision on training data :  0.5420560747663551
```

```
# Finding Accuracy for testing data

X_test_prediction = model.predict(X_test)

testing_data_accuracy = accuracy_score(X_test_prediction, y_test)

testing_data_precision = precision_score(X_test_prediction, y_test)

print('Accuracy on testing data : ', testing_data_accuracy)

print('Precision on testing data : ', testing_data_precision)
```

```
Accuracy on testing data :  0.7532467532467533

Precision on testing data :  0.5
```

CONCLUSION:

In summary, the second part of our diabetes prediction project, centered around machine learning and logistic regression, represents a significant step forward in the realm of healthcare

and disease management. By harnessing the power of data and predictive modeling, we are paving the way for more accurate and timely diagnosis of diabetes, which can lead to better patient outcomes and a higher quality of life. The potential for improving healthcare through this approach is promising, and we look forward to further developments and applications in the field**.**