Diabetes

Subria Islam

Option: 1(ANN neural network)

—

ANN Binary Classification and Logistic Regression

Description of the work

Description of Dataset:

This diabetes dataset is collected from the National Institute of Diabetes and Digestive and Kidney Diseases. It has 768 instances. It has 9 attributes(columns) and all are in numerical value.

All attributes (columns) are:
Pregnancies: Number of times a person has been pregnant.
Glucose: Blood sugar level measured 2 hours after consuming a glucose drink.
BloodPressure: Diastolic blood pressure (mm Hg)
SkinThickness: Thickness of a fold of skin on the triceps (mm).
Insulin: Blood insulin level measured 2 hours after consuming glucose (mu U/ml).
BMI: Body mass index, a measure of body weight relative to height (weight in kg/(height in m)^2).
DiabetesPedigreeFunction: A function that predicts the likelihood of diabetes based on family history.
Age: Age (years).
Outcome: A binary class variable, where 0 typically means no diabetes, and 1 means diabetes is present.

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	79.7995	33.6	0.627	50	1
1	1	85	66	29	79.7995	26.6	0.351	31	0
2	8	183	64	20.5365	79.7995	23.3	0.672	32	1

Goal:

Based on the diagnostic measurements, make predictions about whether a patient is likely to have diabetes or not.

Data Preparation for the training

Dataset columns:

In the dataset, all columns don't have any null values.

```
Number of null values in each column:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

But some columns have a missing value which is mentioned as zero. But Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI values don't be zero.

```
Number of zero values in each column:
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

Data preparation:

zero values of Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI were replaced with the mean value of the column.

df - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	79.7995	33.6	0.627	50	1
1	1	85	66	29	79.7995	26.6	0.351	31	0
2	8	183	64	20.5365	79.7995	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
5	5	116	74	20.5365	79.7995	25.6	0.201	30	0
6	3	78	50	32	88	31	0.248	26	1
7	10	115	69.1055	20.5365	79.7995	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	20.5365	79.7995	31.9926	0.232	54	1
10	4	110	92	20.5365	79.7995	37.6	0.191	30	0

Dividing the data into X and y and creating X and y dataframes.

X - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	79.7995	33.6	0.627	50
1	1	85	66	29	79.7995	26.6	0.351	31
2	8	183	64	20.5365	79.7995	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
5	5	116	74	20.5365	79.7995	25.6	0.201	30
6	3	78	50	32	88	31	0.248	26
7	10	115	69.1055	20.5365	79.7995	35.3	0.134	29
8	2	197	70	45	543	30.5	0.158	53
9	8	125	96	20.5365	79.7995	31.9926	0.232	54
10	4	110	92	20.5365	79.7995	37.6	0.191	30

y - Series

0	1
1	0
2	1
3	0
4	1
5	0
6	1
7	0
8	1
9	1
10	0

For training and testing purposes all data were split. 80% for training and 20% for tasting. The datasets X and y are divided into four dataframes: X_train, X_test, y_train and y_test.

Screenshot of training data:

X_train - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
603	7	150	78	29	126	35.2	0.692	54
118	4	97	60	23	79.7995	28.2	0.443	22
247	0	165	90	33	680	52.3	0.427	23
157	1	109	56	21	135	25.2	0.833	23
468	8	120	69.1055	20.5365	79.7995	30	0.183	38
193	11	135	69.1055	20.5365	79.7995	52.3	0.578	40
306	10	161	68	23	132	25.5	0.326	47
319	6	194	78	20.5365	79.7995	23.5	0.129	59
97	1	71	48	18	76	20.4	0.323	22
530	2	122	60	18	106	29.8	0.717	22

y_train - Series

Index	Outcome
603	1
118	0
247	0
157	0
468	1
193	1
306	1
319	1
97	0
530	0

Screenshot of X_test and y_test:

X_test - DataFrame

Index	regnancie	Glucose	BloodPressure	SkinThickness	Insulin	BMI	etesPedigreeFunc	Age
661	1	199	76	43	79.7995	42.9	1.394	22
122	2	107	74	30	100	33.6	0.404	23
113	4	76	62	20.5365	79.7995	34	0.391	25
14	5	166	72	19	175	25.8	0.587	51
529	0	111	65	20.5365	79.7995	24.6	0.66	31
103	1	81	72	18	40	26.6	0.283	24
338	9	152	78	34	171	34.2	0.893	33
588	3	176	86	27	156	33.3	1.154	52
395	2	127	58	24	275	27.7	1.6	25
204	6	103	72	32	190	37.7	0.324	55

y_test - Series

Index	Outcome
661	1
122	0
113	0
14	1
529	0
103	0
338	1
588	1
395	0
204	0

StandardScaler is employed to achieve normalized and similarly distributed features, optimizing the performance of machine learning algorithms, especially beneficial for neural networks. Both 'X_train' and 'X_test' are Scaled by using 'StandardScaler' from scikit-learn.

Screenshot of X_train and X_test after scaling:

X_train - NumPy object array

	0	1	2	3	4	5	6	7
0	0.908329	0.93644	0.45816	0.235913	0.0592793	0.36865	0.677404	1.69956
1	0.0364468	-0.816286	-1.03864	-0.382505	-0.425194	-0.632929	-0.070497	-0.965692
2	-1.12606	1.43249	1.45603	0.648191	5.86869	2.81536	-0.118555	-0.882403
3	-0.835435	-0.419442	-1.37126	-0.588644	0.153656	-1.06218	1.10091	-0.882403
4	1.19896	-0.055669	-0.28147	-0.636421	-0.425194	-0.37538	-0.851438	0.366933
5	2.07084	0.440386	-0.28147	-0.636421	-0.425194	2.81536	0.334992	0.533511
6	1.78021	1.30021	-0.373396	-0.382505	0.122197	-1.01925	-0.42192	1.11653
7	0.617702	2.39153	0.45816	-0.636421	-0.425194	-1.30542	-1.01363	2.116
8	-0.835435	-1.67611	-2.03651	-0.897853	-0.465036	-1.74897	-0.430931	-0.965692
9	-0.544808	0.0104716	-1.03864	-0.897853	-0.150447	-0.403997	0.752494	-0.965692
10	1.78021	1.89548	-0.207084	-0.636421	-0.425194	0.354341	-0.800376	0.283644

X_test - NumPy object array

	0	1	2	3	4	5	6	7
0	-0.835435	2.55689	0.291849	1.67889	-0.425194	1.47039	2.78594	-0.965692
1	-0.544808	-0.485583	0.125538	0.338982	-0.213365	0.139717	-0.187638	-0.882403
2	0.0364468	-1.51076	-0.872329	-0.636421	-0.425194	0.196951	-0.226685	-0.715825
3	0.327074	1.46557	-0.0407731	-0.794783	0.573108	-0.976327	0.362024	1.44969
4	-1.12606	-0.353302	-0.622862	-0.636421	-0.425194	-1.14803	0.581288	-0.21609
5	-0.835435	-1.34541	-0.0407731	-0.897853	-0.842543	-0.861861	-0.551076	-0.799114
6	1.48958	1.00258	0.45816	0.751261	0.531163	0.225567	1.28113	-0.0495122
7	-0.254181	1.79627	1.12341	0.0297734	0.373868	0.0967927	2.06508	1.53298
8	-0.544808	0.175823	-1.20495	-0.279435	1.62174	-0.70447	3.40469	-0.715825
9	0.617702	-0.617864	-0.0407731	0.545121	0.730403	0.726356	-0.427928	1.78285
10	-0.254181	1.201	0.291849	0.9574	1.30715	-0.146448	1.15498	-0.465958

Neural networks training

First, a Sequential model is initialized, which is a liner stack of layers. An input layer with 8 neurons is added to the neural network, expecting input data with a shape defined by the number of features in X, initializing weights normally, and using the Rectified Linear Unit (ReLU) activation function. Then a Hidden layer with 8 neurons are added and an activation function: Relu is used. Also, an output layer with 1 neuron is added, using the sigmoid activation function.

A summary of the model architecture is displayed, including the layers, their output shapes, and the number of trainable parameters.

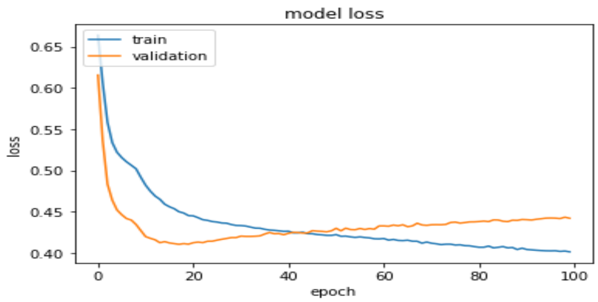
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 8)                 72

 dense_4 (Dense)             (None, 8)                 72

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 153 (612.00 Byte)
Trainable params: 153 (612.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

Then the model is compiled with binary cross-entropy loss, Adam optimizer, and metrics for evaluation. Also, the model is trained on the training data with specified epochs, batch_size, and validation data.

The 'history.history.keys()' method provides keys representing training and validation metrics, including loss and accuracy, tracked during the model training process.

```
dict_keys(['loss', 'binary_crossentropy', 'accuracy', 'val_loss', 'val_binary_crossentropy', 'val_accuracy'])
```

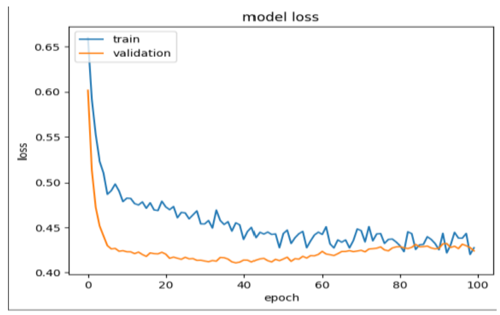
A plot is generated to display the training and validation loss values across epochs using Matplotlib for visualization purposes.



It’s an overfitting model. Overfitting in models, excelling in training but failing with new data. To avoid overfitting, Dropout layers are added in neural network architecture. Dropout can help prevent overfitting by randomly dropping a certain proportion of neurons during training, forcing the network to learn more robust features. Fixing overfitting helps in building models that make accurate predictions on diverse datasets.

Now the summary of the model architecture after adding Dropout layers in neural network architecture. And the plot after adding Dropout layers in neural network architecture.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 8)                 72

dropout_8 (Dropout)          (None, 8)                 0

dense_13 (Dense)             (None, 8)                 72

dropout_9 (Dropout)          (None, 8)                 0

dense_14 (Dense)             (None, 1)                 9

=================================================================
Total params: 153 (612.00 Byte)
Trainable params: 153 (612.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

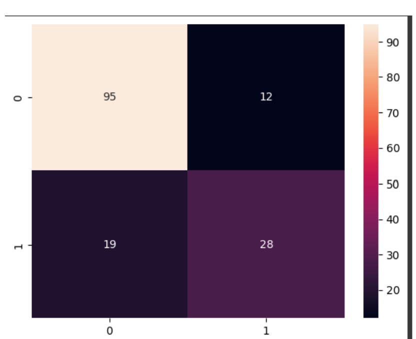


ANN neural network relevant metrics for the case and predict with new data

ANN neural network:

By using the trained model, the outcome for the text data (X_test) is predicted. Then the predicted probability is compared against a threshold of 0.5.

A confusion matrix was used to estimate the result which is visualized using the heatmap. Also, accuracy, precision, and recall are calculated.

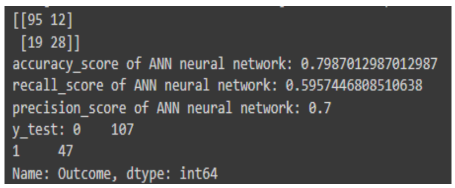


Here,
TN(True Negative): In the confusion matrix, there are 95 instances where the actual class was negative and the model correctly predicted them as negative.

TP(True Positive): In the confusion matrix, there are 28 instances where the actual class was positive and the model correctly predicted them as positive.

FP(False Positive): In the confusion matrix, there are 12 instances where the actual class was negative and the model incorrectly predicted them as positive.

FN(False Negative): In the confusion matrix, there are 19 instances where the actual class was positive and the model incorrectly predicted them as negative.



```
[[95 12]
 [19 28]]
accuracy_score of ANN neural network: 0.7987012987012987
recall_score of ANN neural network: 0.5957446808510638
precision_score of ANN neural network: 0.7
y_test: 0    107
1    47
Name: Outcome, dtype: int64
```

Accuracy score for ANN neural network:
The accuracy score is 0.7987012987012987 means that the model's correctness is 79.87%.

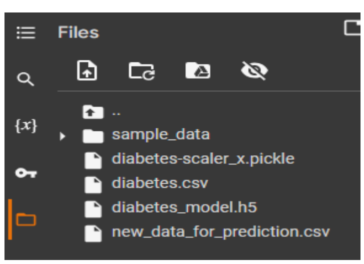
Precision score for ANN neural network:
The precision score is 0.7 means that out of all the positive predictions made by this model, approximately 70% of them were correct.

Recall score for ANN neural network:
The recall score is 0.5957446808510638 means that this model correctly identified approximately 59.57% of the actual positive cases in the dataset.

y_test: Actual Distribution:
displays the count of actual positive (1) and negative (0) instances in the test dataset.

Outcome 0: There are 107 instances where the target variable Outcome is labeled as 0
Outcome 1: There are 47 instances where the target variable Outcome is labeled as 1

Then saves the trained neural network model ('diabetes_model.h5') and the StandardScaler instance ('diabetes-scaler_x.pickle') using the model.save() method and Python's pickle.dump() function, respectively, for future use in predictions without retraining or re-scaling.



Predict with new data:

Now load a pre-trained neural network model ('diabetes_model.h5') using load_model() function , restoring its architecture and weights. Simultaneously, it loads a previously saved 'scaler_x' instance ('diabetes-scaler_x.pickle') using Python's pickle.load() for consistent data preprocessing during predictions, facilitating model deployment without retraining or re-scaling.

Now reads new data from 'new_data_for_prediction.csv', scales it using the pre-loaded scaler (sc), and predicts outcomes (ynew) using the trained neural network model (model). The original data (Xnew_org) and the scaled data (Xnew) are stored separately for future reference or analysis.

Then convert scaled data back to its original unscaled format, enabling interpretation or analysis in the data's initial scale.

finally, prediction is done for new data by interpreting them with a 0.5 threshold: values ≥ 0.5 are labeled as 1 (diabetes present), while values < 0.5 are labeled as 0 (no diabetes).

```
Pregnancies                   1.0
Glucose                     188.0
BloodPressure                65.0
SkinThickness                25.0
Insulin                      87.0
BMI                          38.1
DiabetesPedigreeFunction      0.8
Age                          36.0
Name: 0, dtype: float64
Predicted Status (threshold 0.5): 1

Pregnancies                   2.0
Glucose                     140.0
BloodPressure                80.0
SkinThickness                28.0
Insulin                     110.0
BMI                          22.0
DiabetesPedigreeFunction      0.6
Age                          28.0
Name: 1, dtype: float64
Predicted Status (threshold 0.5): 0

Pregnancies                   5.0
Glucose                     100.0
BloodPressure                75.0
SkinThickness                35.0
Insulin                      90.0
BMI                          28.0
DiabetesPedigreeFunction      0.3
Age                          40.0
Name: 2, dtype: float64
Predicted Status (threshold 0.5): 0
```

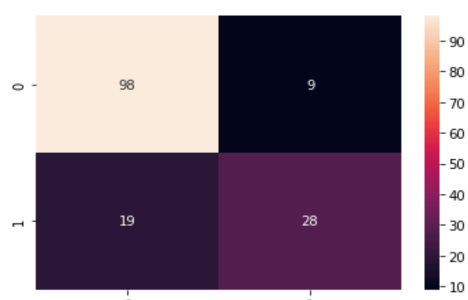
Logistic regression relevant metrics for the cases and predict with new data

Logistic regression:

By using Logistic Regression, the model was trained. And finally, Predicting outputs with X_test as inputs.

Confusion matrix for logistic regression:

The result was Estimated by a confusion matrix.



Here,
TN(True Negative): In the confusion matrix, there are 98 instances where the actual class was negative and the model correctly predicted them as negative.

TP(True Positive): In the confusion matrix, there are 28 instances where the actual class was positive and the model correctly predicted them as positive.

FP(False Positive): In the confusion matrix, there are 9 instances where the actual class was negative and the model incorrectly predicted them as positive.

FN(False Negative): In the confusion matrix, there are 19 instances where the actual class was positive and the model incorrectly predicted them as negative.

```
confusion matrix of logistic regression:  [[98  9]
 [19 28]]
accuracy score of logistic regression: 0.82
precision score of logistic regression: 0.76
recall score of logistic regression: 0.60
```

Accuracy score for logistic regression model:
The accuracy score is 0.82 means that the model's correctness is 82%.

Precision score for logistic regression model:
The precision score is 0.76 means that out of all the positive predictions made by this model, approximately 76% of them were correct.

Recall score for logistic regression model:
The recall score is 0.60 means that this model correctly identified approximately 60% of the actual positive cases in the dataset.

Predict with new data:

new data is read from 'new_data_for_prediction.csv' file and predicted by the trained model. The prediction output was good enough for getting the information about whether the patient has diabetes or not.

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            1      188             65             25       87  38.1
1            2      140             80             28      110  22.0
2            5      100             75             35       90  28.0

   DiabetesPedigreeFunction  Age  Outcome
0                       0.8   36        1
1                       0.6   28        1
2                       0.3   40        1
```

Conclusions of the results

The logistic regression model has a higher accuracy rate than the ANN neural network model. Higher accuracy generally is a positive indicator.

Precision measures the proportion of true positive predictions among all positive predictions made by the model. A higher precision means fewer false positives. In this case, the logistic regression model also outperforms the ANN neural network model in terms of precision.

Recall measures the proportion of true positive predictions among all actual positives. It indicates how well the model captures positive instances. Also, the logistic regression model has a higher recall compared to the ANN neural network.

Based on these metrics, logistic regression appears to perform better than the ANN neural network for this dataset.

Get a better model:

To enhance model performance, need to focus on data preprocessing, hyperparameter tuning, regularization techniques to reduce overfitting, etc. By increasing the complexity or depth of the network architecture and optimizing hyperparameters like learning rate, batch size, and the number of neurons in hidden layers possible to improve the performance of the Artificial Neural Network (ANN) model.