Assignment 3

Author: Aditya Mulik
NUID: 002127694
Email ID: mulik.a@northeastern.edu

Notes:

1. I have used Google Colab for running the jupyter files
2. The datasets are loaded in Google Colab using it's Gdrive API
3. I have linked the dataset or uploaded it in the zip file as per respective question and mentioned in its respective notes

---

Question 1: Expand the basic code for building a DNN on the Pima Indian Diabetic Dataset to include:

(a) **pre-process the data by scaling/standardising the 8 columns**

Ans: Below are the currently supported pre-processing methods provided by sklearn

1. MinMaxScaler
2. StandardScaler
3. Normalizer
4. Binarizer

For testing purposes, I used the MinMaxScaler as per the below screenshot. I have added examples of other implementations as well and commented on them.

```
[7] #scale Features from 0 to 1
    transforms = MinMaxScaler(feature_range=(0,1))
    scaler_X =  transforms.fit_transform(X)
    print(scaler_X)

[[0.35294118 0.74371859 0.59016393 ... 0.50074516 0.23441503 0.48333333]
 [0.05882353 0.42713568 0.54098361 ... 0.39642325 0.11656704 0.16666667]
 [0.47058824 0.91959799 0.52459016 ... 0.34724292 0.25362938 0.18333333]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.390462   0.07130658 0.15      ]
 [0.05882353 0.63316583 0.49180328 ... 0.4485842  0.11571307 0.43333333]
 [0.05882353 0.46733668 0.57377049 ... 0.45305514 0.10119556 0.03333333]]
```

(b) **Split the entire dataset into three parts instead of two as we currently do. One is train, two is validation, and then a test set. Build DNN model with train data, tune hyper-parameters with validation data, and finally evaluate performance on the test data**

Ans: I split the data in the train-validation-test dataset in a range of **70-15-15 ratio** by reusing the sklearn's train_test_split twice.

Later I plugged the validation set to the hyperparameters by replacing the test test and evaluated the model on the test set.

Split data into train, validation and test

```
[8] df.shape

(768, 9)

[13] X_train, X_test, Y_train, Y_test = train_test_split(scaler_X, Y, test_size=0.15, random_state=1)
     X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.15, random_state=1)

[15] print(X_train.shape,Y_train.shape,X_val.shape,Y_val.shape,X_test.shape,Y_test.shape)

(554, 8) (554, 1) (98, 8) (98, 1) (116, 8) (116, 1)
```

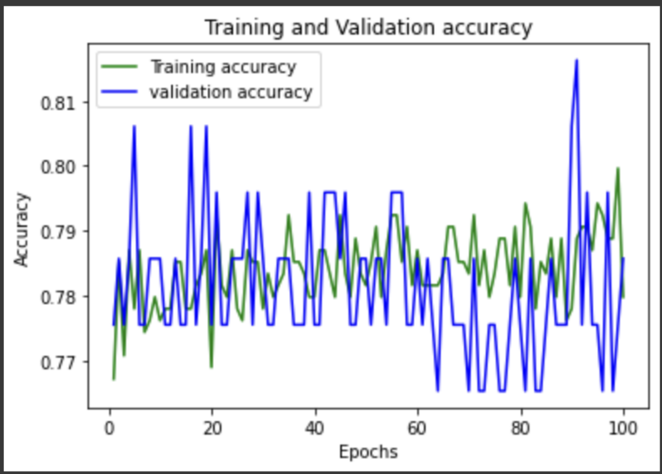
```
[20] # Fit the DNN with your train data

     model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=100, batch_size=5)
```

(c) **Make Epoch versus train set accuracy, and validation set accuracy**

Ans: I've plotted the graph of train/ val accuracy against the 100 epochs used.

Epoch vs Train/ Validation

```
loss_train = history.history['accuracy']
loss_val = history.history['val_accuracy']
epochs = range(1,101)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



(d) **Report results using nice ROC curves, report AUC values. Feel free to use code from our course, or from elsewhere**

Ans: A Roc_Auc score of 84.24% received on model evaluation and the code snippet with the plot diagram is added below.

Model Evaluation

ROC vs AUC implementation using sklearn's roc_auc_score metrics

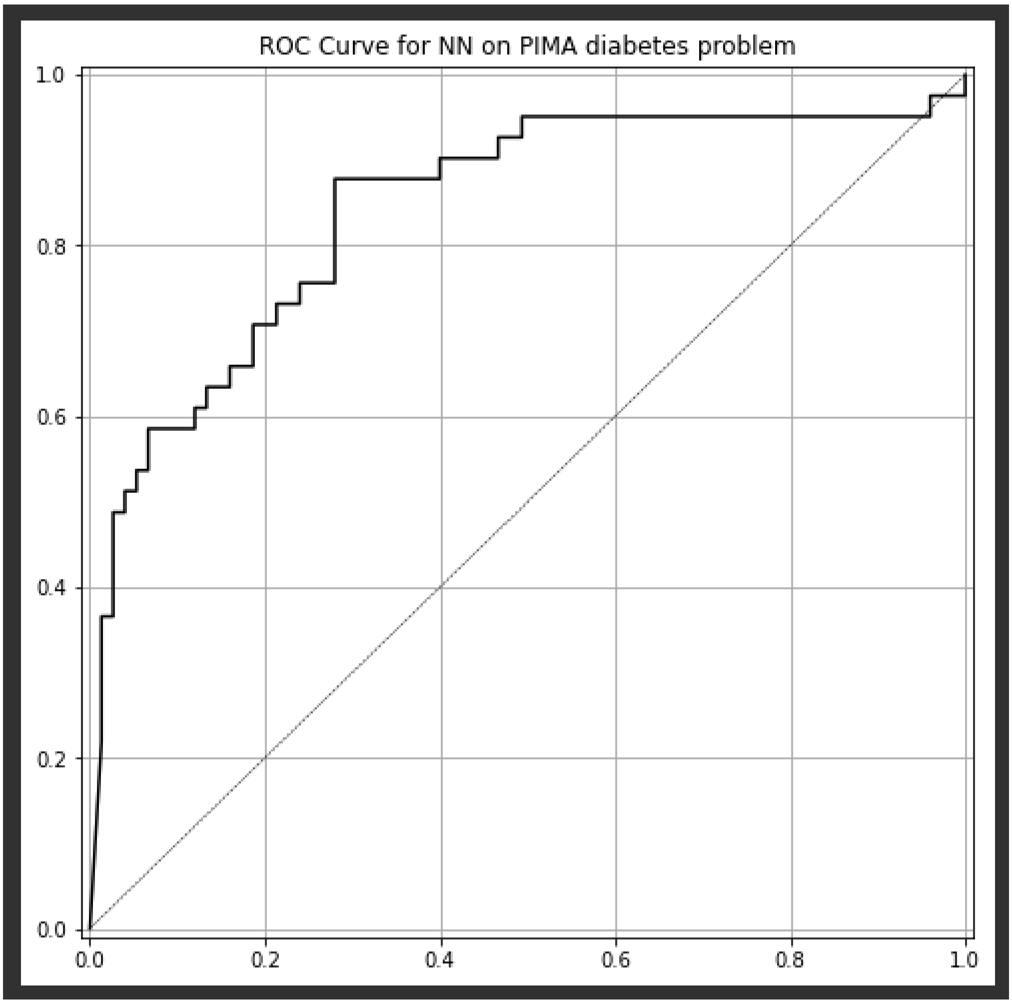
```
[45] y_pred_prob_nn_1 = model.predict(X_test)
```

```
[46] y_pred_prob_nn_1[:10]
```

```
array([[0.5604001 ],
       [0.36893898],
       [0.11501053],
       [0.05097717],
       [0.14265424],
       [0.17524552],
       [0.4865586 ],
       [0.07613519],
       [0.10438582],
       [0.07139322]], dtype=float32)
```

```
[47] print(roc_auc_score(Y_test,y_pred_prob_nn_1))
```

```
0.842439024390244
```



(e) **How would you increase dataset size? Try out at least two approaches and re-evaluate the model performance on this new and augmented dataset.**

Ans: I used the **SMOTE technique** from the imblearn library to augment the dataset for the outcome column of the pima dataset. Another method is to add **gaussian noise** to the dataset by adding dummy or random values to increase the quantity of the dataset which can be used for training the model and increasing its efficiency.

Another method to augment data is by using **oversampling** available using the imblearn library.

Data Augmentation

```
df3 = df.copy()
target = 8

print('Original class distribution:')
print(df3[target].value_counts())

xf = df3.columns
X_t = df3.drop([target],axis=1)
Y_t = df3[target]

smote = SMOTE()
X_t, Y_t = smote.fit_resample(X, Y)

df3 = pd.DataFrame(X_t, columns=xf)
df3[target] = Y_t

print('\nClass distribution after applying SMOTE Technique:',)
print(Y_t.value_counts())
```

```
Original class distribution:
0    500
1    268
Name: 8, dtype: int64

Class distribution after applying SMOTE Technique:
8
0    500
1    500
dtype: int64
```

```
# Example to add noice to dataset
# x is my training data
# mu is the mean
# std is the standard deviation
mu=0.0
std = 0.1
def gaussian_noise(x,mu,std):
    noise = np.random.normal(mu, std, size = x.shape)
    x_noisy = x + noise
    return x_noisy
```

Note:
All the implementation has been done in the notebook and attached as per **Question_1.ipynb** file

Dataset: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Question 2: Please describe at least two ways of ensembling together DNNs and RFs. Take any dataset from Kaggle and

(1) train an RF model (2) train a DNN, and (3) a hybrid DNN and RF model. Provide detailed model and result comparison

a. **Two ways of ensembling DNNs & RFs**

Ans: The most common ways of ensembling two models which results as a heterogeneous Ensemble Models, below are some methods which can be used to aggregate the two models:

1. Bagging
2. Boosting
3. Stacking

b. **Model and result comparison with below criteria.**
   i. **Train an RF model**
   ii. **Train an DNN model**
   iii. **Hybrid DNN and RF model**

Ans:

My method of creating an ensemble was with stacking, selecting the best features from the RandomForestClassifier and Deep Neural Network's Sequential model and using it as a base learner and predicting the results on another LogisticRegression Model which results in higher accuracy.

Below is the comparison for the criteria.

No	Machine Learning Model	Accuracy	Notebook
1	Random Forest Classifier	72.75%	**Q2_RF.ipynb**
2	Deep Neural Network	80.42%	**Q2_DNN.ipynb**
3	Hybrid DNN + RF	81.21%	**Q2_Hybrid.ipynb**

Dataset: https://www.kaggle.com/datasets/osmi/mental-health-in-tech-survey