Table of Contents

PROJECT OVERVIEW **1**

PROBLEM STATEMENT 1 ..... **1**

Question 1
Code Snippet 1
Output 1

PROBLEM STATEMENT 2 ..... **2**

Question 2
Code Snippet 2
Output 2

PROBLEM STATEMENT 3 ..... **3**

Question 3
Design Process Procedure 3
Considered MLPClassifier tradeoffs 4

PROBLEM STATEMENT 4 ..... **4**

Question 4
Code Snippet 4
Output 4

PROBLEM STATEMENT 5, 6 & 7 ..... **5**

Question 5
Solution 5
Observations 6

PROBLEM STATEMENT 8 ..... **6**

Question 6
Solution 6
Observations 8

PROJECT 3 CONCLUSION **9**

APPENDIX **10**

Code 1 10
Code 2 11

List of Tables

Table 1: Code 1- Split of Data into Training, Validation and Testing with values. 2
Table 2: Code 1 - Cross-Validation Scores for each iteration and the average. 3
Table 3: Code 1 - Input vs Output values comparison along with red highlight (worst performing scenario) and green highlight (Best performing scenario). 5
Table 4: Code 2 - Split of Data into Training, Validation and Testing with values. 7
Table 5: Code2 - Cross-Validation Scores for each iteration and the average. 7
Table 6: Code 2 - Input vs Output values comparison along with red highlight (worst performing scenario) and green highlight (Best performing scenario). 8

Project Overview

The project 3 aims to expand learning through practice upon the concepts of Neural Network multilayer perceptron learning algorithms with focus upon MLPClassifier and the MLPRegressor.

Problem Statement 1

Question

The expectation is to write a python code that splits the Wisconsin Breast Cancer data set available in the *Sklearn* dataset library into Training/Validation and Testing subsets through a split ratio of 80% and 20% respectively. The key objective is to mention and describe how the split is conducted along with providing the random value used for the split for the purpose of recreation.

Code Snippet

The following is the code snippet for the question given:

```
# Splitting the datasets into Training/Validation and Testing
ftr_train_val, ftr_test, trgt_train_val, trgt_test = train_test_split(ftr, trgt,
test_size=0.2, random_state=19)

# Further splitting the Training/Validation set into Training and Validation sets
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train_val,
trgt_train_val, test_size=0.5, random_state=19)

# Printing the number of samples in the original dataset and split subsets
print("The Total number of samples in the Wisconsin Breast Cancer Dataset is: \n")
print(len(ftr))

print("The number of samples in the Training Dataset is: \n")
print(len(ftr_train))

print("The number of samples in the Validation Dataset is: \n")
print(len(ftr_val))

print("The number of samples in the Testing Dataset is: \n")
print(len(ftr_test))
```

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 1 CAN BE FOUND IN THE APPENDIX WITHIN THE CODE 1 OF THIS DOCUMENT.

Output

```
The Total number of samples in the Wisconsin Breast Cancer Dataset is:

569
The number of samples in the Training Dataset is:

227
The number of samples in the Validation Dataset is:

228
The number of samples in the Testing Dataset is:

114
```

The above output clearly identifies the total number of samples in the Wisconsin Breast Cancer Dataset which is 569 and the number of samples divided for Training and Validation is 455 i.e. 80% (79.96%). Additionally, the Training and Validation is further split in half with 227 samples shared with Training and 228 samples shared with Validation. The number of samples divided for Testing is 114 i.e. 20% (20.03%). The random state value provided for the split is given as 19.

Category	Percentage split	Number of rows (samples)
Training and Validation	80%	455 (227+228)
Training	50% of 80%	227
Validation	50% of 80%	228
Testing	20%	114

Table 1: Code 1- Split of Data into Training, Validation and Testing with values.

Problem Statement 2

Question

The requirement here is to either create an additional split for the validation sub-dataset or to use cross validation technique to tune the overall program's MLP Hyperparameters.

Code Snippet

The following is the code snippet for the question given:

```
# Performing Cross Validation on the training and validation dataset
cv_scores = cross_val_score(MLPClassifier(), ftr_train_val, trgt_train_val, cv=10)

# Printing the performance of the Cross Validation
print("Cross Validation Scores: \n")
print(cv_scores)

print("Average Score of Cross Validation: \n")
print(cv_scores.mean())
```

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 2 CAN BE FOUND IN THE APPENDIX WITHIN THE CODE 1 OF THIS DOCUMENT.

Output

```
Cross Validation Scores:

[0.97826087 0.91304348 0.93478261 0.82608696 0.93478261 0.91111111
 0.91111111 0.95555556 0.86666667 0.93333333]
Average Score of Cross Validation:

0.9164734299516908
```

The output at first displays the cross-validation scores with a cross validation count of number of folds is given as 10. This means that the validation subset is divided into 10 equal folds and each fold is compared with the remaining 9 other to evaluate the model's performance based on 10 iterations (in this case). The average value of these 10-fold iterations is also displayed and the value obtained here is 0.9164. This shows that the model is already leaning on a good learning curve.

Iteration Count	Cross Validation Scores
1	0.97826087
2	0.91304348
3	0.93478261
4	0.82608696
5	0.93478261
6	0.91111111
7	0.91111111
8	0.95555555
9	0.86666667
10	0.93333333
Average	**0.9164734299516908**

Table 2: Code 1 - Cross-Validation Scores for each iteration and the average.

Problem Statement 3

Question

The expectation with this question is to provide a procedure to document the design process and to identify and mention the tradeoffs considered in using a MLPClassifier.

Design Process Procedure

The following are the general steps involved (along with short descriptions in relation to the Project 3 fulfillment) in the establishment of design as a precursor to the MLPClassifier:

1. Problem Definition:

In this case, the problems are clearly defined in what is to be achieved along with the necessary inputs (Ex. Choice of Dataset), procedural steps and the necessary outputs (Ex. MLP and Cross Validation scores).

2. Data Collection and Preprocessing:

This step involved prepping the data (cleaning, normalization etc.) before operations can be done using it. However, in this case, since we are already using a prepped dataset such as the Wisconsin Breast Cancer Dataset from the Sklearn dataset library, it is directly used for operations.

3. Model Architecture:

The model architecture focusses on the finer details of how the data is to be analyzed based on a certain set of hyperparameters such as number of layers with the number of neurons in each layer along with the number of epochs that is to be conducted et al.

4. Training and Hyperparameter tuning:

This stage of the process trains the model from the split training sub-dataset and tuning the hyperparameters such as the number of neurons, learning rate etc. The usage of cross validation technique is also considered for hyperparameter to achieve the best score for the model.

5. Model Evaluation:

This step focusses on the model's performance upon the unseen data i.e. testing sub-dataset. This evaluation is done by using metrics such as scores and the expected efficiency is obtained based on the problem statement's requirement.

6. Documentation

This stage is a vital step of the process since it involves documenting the entire process procedure, the decisions made and the tuning of the hyperparameters to achieve desired results and to satisfy the problem statement. This step also provides an insight upon the entire activity conducted from the start till the end.

Considered MLPClassifier tradeoffs

1. Increasing the complexity of a model does improve the model's efficiency and its ability to solve complex problems, however, tuning the complexity of a model greater than a limit causes overfitting in the model.
2. Increasing the number of layers or the number of neurons improves the performance of the model but also increases the training time for any dataset.
3. MLP Classifiers are black box systems due to their complexity and thus leaves little room for interpretation and decision making in complex scenarios. However, the replacement for MLP Classifiers can either be linear classifiers or decision trees which can interpret much easily. But the performance of an MLP Classifier is unmatched.
4. Data Availability affects the model complexity as well i.e. with abundant labelled examples from the dataset, complex models like MLP Classifiers can be used else there is a risk of overfitting when the dataset isn't large and unlabeled.

Problem Statement 4

Question

The requirement here is to use an MLP Classifier to train, validate and test an MLP Model. Any number of features from the dataset can be used. The random state value is to be mentioned for the purpose of model recreation.

Code Snippet

```
# Creating an MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=1000, random_state=53)

# Training the MLP classifier on the training dataset
mlp.fit(ftr_train, trgt_train)

# Evaluating the MLP classifier on the training dataset
training_score = mlp.score(ftr_train, trgt_train)

# Evaluating the MLP classifier on the validation dataset
validation_score = mlp.score(ftr_val, trgt_val)

# Evaluating the MLP classifier on the testing dataset
test_score = mlp.score(ftr_test, trgt_test)

# Printing the scores for the MLP Classifier
print("Training Score: \n")
print(training_score)

print("Validation Score: \n")
print(validation_score)

print("Test Score: \n")
print(test_score)
```

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 4 CAN BE FOUND IN THE APPENDIX WITHIN THE CODE 1 OF THIS DOCUMENT.

Output

```
Training Score:

0.9383259911894273
Validation Score:

0.9078947368421053
Test Score:

0.9385964912280702
```

It can be observed that the output prints the score of all the all the three subsets separately.

Problem Statement 5, 6 & 7

Question

The requirement for problem statement 5 is to provide a list of hyperparameters that were used as default values as input along with the new values that were modified from the original values. The requirement for problem statement 6 is to provide a score values of each dataset (Training, Validation and Testing). The requirement for problem statement 7 is to note the observations along with pointers for improvement.

Solution

Slno	Input Values		Output Values			
	Type	Value	Average Cross Validation Score	Training Score	Validation Score	Testing Score
1	Train/Val and Test Split	80:20	0.91647343	0.938325991	0.907894737	0.938596491
	Train and Val Split	50:50				
	cviterations	10				
	Number of layers and number of neurons per layer	100;100				
	Number of Epochs	1000				
2	Train/Val and Test Split	80:20	0.905469385	0.936813187	0.857142857	0.947368421
	Train and Val Split	80:20				
	cviterations	3				
	Number of layers and number of neurons per layer	1000;150				
	Number of Epochs	3000				
3	Train/Val and Test Split	80:20	0.9209319	0.78021978	0.663003663	0.754385965
	Train and Val Split	40:60				
	cviterations	15				
	Number of layers and number of neurons per layer	150;1000				
	Number of Epochs	5000				
4	Train/Val and Test Split	80:20	0.936264822	0.934065934	0.923076923	0.929824561
	Train and Val Split	80:20				
	cviterations	20				
	Number of layers and number of neurons per layer	1500;1500				
	Number of Epochs	10000				
5	Train/Val and Test Split	80:20	0.914273504	0.831295844	0.782608696	0.885964912
	Train and Val Split	90:10				
	cviterations	18				
	Number of layers and number of neurons per layer	900;500				
	Number of Epochs	1000				

Table 3: Code 1 - Input vs Output values comparison along with red highlight (worst performing scenario) and green highlight (Best performing scenario).

Observations

The model shows the consistency in the scores among average cross validation, training, validation, and testing scores with the varying input hyperparameters accordingly. The best-case scenario can be observed in Table 3 where the output testing score is 0.947. The worst-case scenario can be observed in Table 3 where the output testing score is 0.754.

The feedback on pointers for improvement would be to increase the number of iterations in the cross validation as that would impact the average cross validation and the validation scores, which in turn would make the model robust. Along with this, an increase to the ratio of number of neurons per layer in the MLP Classifier would significantly make the mode further complex but also enhance the chances for a better case scenario.

It was also observed that the increase to the number of epochs would not directly impact towards a best-case scenario, rather it would streamline the results and push the output towards a further normalized data as seen in case 4 with 10000 epochs from Table 3.

Problem Statement 8

Question

The requirement for problem statement 8 is to repeat the steps performed for Code 1 with change in use of MLP Regressor in place of a MLP Classifier along with adding a shuffle to the dataset before the operations are done on it. Furthermore, the same inputs, outputs and observations are to be recorded.

Solution

The following is the output obtained for using a MLP Regressor upon the Diabetes dataset present in SK Learn dataset following the same procedure as Code 1:

```
The Total number of samples in the Wisconsin Breast Cancer Dataset is:

442
The number of samples in the Training Dataset is:

317
The number of samples in the Validation Dataset is:

36
The number of samples in the Testing Dataset is:

89

Cross Validation Scores:

[-3.23383008 -3.1745197  -2.36219969 -3.16064332]
Average Score of Cross Validation:

-2.982798197130794

Training Score:

0.5575521641082267
Validation Score:

0.5520001509791401
Test Score:

0.23943396231077252
```

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 8 CAN BE FOUND IN THE APPENDIX WITHIN THE CODE 2 OF THIS DOCUMENT.

The output shows that the *Diabetes* dataset contains a total of 442 samples of rows of data among which 353 i.e. 80% (79.86%) of the original data is further divided into Training and Validation at 50% each i.e. 176 and 177 samples respectively. The testing sub-dataset contains a sample of 89 rows of data which is 20% (20.13%) of the original data.

Category	**Percentage split**	**Number of rows (samples)**
Training and Validation	80%	353 (176+177)
Training	50% of 80%	176
Validation	50% of 80%	177
Testing	20%	89

Table 4: Code 2 - Split of Data into Training, Validation and Testing with values.

The output also displays the cross-validation scores with a cross validation count of number of folds is given as 4. This means that the validation subset is divided into 4 equal folds and each fold is compared with the remaining 3 other to evaluate the model's performance based on 4 iterations (in this case).

The average value of these 4-fold iterations is also displayed and the value obtained here is -2.9827. This shows that the model is either failing due to the mismatch in the decision chosen for the original subset used for the pertaining methodology of approach of operations.

Iteration Count	**Cross Validation Scores**
1	-3.23383008
2	-3.1745197
3	-2.36219969
4	-3.16064332
Average	**-2.982798197130794**

Table 5: Code2 - Cross-Validation Scores for each iteration and the average.

The Table 5 shows the cross-validation scores of each iteration performed.

Slno	Input Values		Output Values			
	Type	Value	Average Cross Validation Score	Training Score	Validation Score	Testing Score
1	Train/Val and Test Split	80:20	-3.09345511	0.51648108	0.528020014	0.26298702
	Train and Val Split	50:50				
	cv iterations	4				
	Number of layers and number of neurons per layer	50;50				
	Number of Epochs	1000				
2	Train/Val and Test Split	80:20	-3.105126489	0.572557996	0.528463194	0.225699832
	Train and Val Split	80:20				
	cv iterations	6				
	Number of layers and number of neurons per layer	20;40				
	Number of Epochs	3000				
3	Train/Val and Test Split	80:20	-3.569697024	-1.148202381	-1.107390694	-1.162233138
	Train and Val Split	40:60				
	cv iterations	2				
	Number of layers and number of neurons per layer	5;80				
	Number of Epochs	500				
4	Train/Val and Test Split	80:20	-3.08154504	0.5504912	0.528807982	0.226522291
	Train and Val Split	80:20				
	cv iterations	9				
	Number of layers and number of neurons per layer	1000;10				
	Number of Epochs	500				
5	Train/Val and Test Split	80:20	-3.15615394	0.648964589	0.481508944	0.190441928
	Train and Val Split	90:10				
	cv iterations	10				
	Number of layers and number of neurons per layer	1000;500				
	Number of Epochs	1500				

Table 6: Code 2 - Input vs Output values comparison along with red highlight (worst performing scenario) and green highlight (Best performing scenario).

Observations

The model shows the consistency in the poor scores among average cross validation. The scores of training, validation, and testing scores with the varying input hyperparameters accordingly with the Testing scores not aligning with the model's learning and validation in all the scenarios. The best-case scenario can be observed in Table 6 where the output testing score is 0.262. The worst-case scenario can be observed in Table 6 where the output testing score is -1.162.

The feedback on pointers for improvement would be to change the model from the source as clearly the model shows that MLP Regressor isn't the best fit for the dataset chosen. Added to this, the number of iterations performed for cross validation also has a limit due to the

dataset being limited in the number of features present in it and the lesser number of samples also adds towards the model to underperform.

From upon observation into Table 6 alone, it can be noticed that the model is exhibiting underfitting attributes and that the results from this model can only be relied upon to a maximum of just over 20%. This maximum failure exhibited by the model deems it unfit for practical application and use.

Project 3 Conclusion

Project 3 helped in cementing the bridge towards neural network and machine learning. The concepts and the routine approach towards the problem statements which made it even simpler to understand and manipulate to learn the concept better.
Project 3 focusses mainly on the neural networks using MLP (Multi-Layer Perceptron) approach with both Classifier and Regressor. Using these on the already familiar datasets i.e. Wisconsin Breast Cancer dataset and the Diabetes dataset made the task easier from the pre-existing and learnt lessons from the previous projects.

Appendix

Code 1

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neural_network import MLPClassifier

# Loading and initializing the dataset
data = load_breast_cancer()
ftr = data.data
trgt = data.target

# Splitting the datasets into Training/Validation and Testing
ftr_train_val, ftr_test, trgt_train_val, trgt_test = train_test_split(ftr, trgt,
test_size=0.2, random_state=19)

# Further splitting the Training/Validation set into Training and Validation sets
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train_val, trgt_train_val,
test_size=0.5, random_state=19)

# Printing the number of samples in the original dataset and split subsets
print("The Total number of samples in the Wisconsin Breast Cancer Dataset is: \n")
print(len(ftr))

print("The number of samples in the Training Dataset is: \n")
print(len(ftr_train))

print("The number of samples in the Validation Dataset is: \n")
print(len(ftr_val))

print("The number of samples in the Testing Dataset is: \n")
print(len(ftr_test))

# Performing Cross Validation on the training and validation dataset
cv_scores = cross_val_score(MLPClassifier(), ftr_train_val, trgt_train_val, cv=10)

# Printing the performance of the Cross Validation
print("Cross Validation Scores: \n")
print(cv_scores)

print("Average Score of Cross Validation: \n")
print(cv_scores.mean())

# Creating an MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=1000, random_state=53)

# Training the MLP classifier on the training dataset
mlp.fit(ftr_train, trgt_train)

# Evaluating the MLP classifier on the training dataset
training_score = mlp.score(ftr_train, trgt_train)

# Evaluating the MLP classifier on the validation dataset
validation_score = mlp.score(ftr_val, trgt_val)

# Evaluating the MLP classifier on the testing dataset
test_score = mlp.score(ftr_test, trgt_test)

# Printing the scores for the MLP Classifier
print("Training Score: \n")
print(training_score)

print("Validation Score: \n")
print(validation_score)

print("Test Score: \n")
print(test_score)
```

Code 2

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.utils import shuffle
from sklearn.neural_network import MLPRegressor

#Loading the Dataset
data = load_diabetes()

#Shuffling and inilializing the sub-datasets
ftr, trgt = shuffle(data.data, data.target, random_state = 33)

# Splitting the datasets into Training/Validation and Testing
ftr_train_val, ftr_test, trgt_train_val, trgt_test = train_test_split(ftr, trgt, test_size =
0.2, random_state = 82)

# Further splitting the Training/Validation set into Training and Validation sets
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train_val, trgt_train_val,
test_size = 0.1, random_state = 80)

#Printing the number of samples in original dataset and split subsets
print("The Total number of samples in the Diabetes Dataset is: \n")
print(len(ftr))

print("The number of samples in the Training and Validation Dataset is: \n")
print(len(ftr_train_val))

print("The number of samples in the Testing Dataset is: \n")
print(len(ftr_test))

# Performing Cross Validation on the training and validation dataset
cv_scores = cross_val_score(MLPRegressor(), ftr_train_val, trgt_train_val, cv = 4)

# Printing the performance of the Cross Validation
print("Cross Validation Scores: \n")
print(cv_scores)

print("Average Score of Cross Validation: \n")
print(cv_scores.mean())

#Creating a MultiLayer Perceptron Regressor
mlp = MLPRegressor(hidden_layer_sizes =(50, 50), max_iter = 1000, random_state = 53)

# Training the MLP Regressor on the training dataset
mlp.fit(ftr_train, trgt_train)

# Evaluating the MLP Regressor on the training dataset
training_score = mlp.score(ftr_train, trgt_train)

# Evaluating the MLP Regressor on the validation dataset
validation_score = mlp.score(ftr_val, trgt_val)

# Evaluating the MLP Regressor on the testing dataset
test_score = mlp.score(ftr_test, trgt_test)

# Printing the scores for the MLP Regressor
print("Training Score: \n")
print(training_score)

print("Validation Score: \n")
print(validation_score)

print("Test Score: \n")
print(test_score)
```