Neural Networking Model Training Assignment

BSCS 8th SEMESTER

STUDENT DETAILS

Student Name: Syed Ahtsham

Student ID Number: 04071813015

ASSIGNMENT DETAILS

Subject Title: Introduction to Machine Learning

Submission Date: June 24, 2022

Submit To: Dr. Onaiza Maqbool

Chapter 1

Problem Statement and Dataset Description

Chapter 1:

Problem statement:

System: Bottled Water Classifier as Potable or not on the basis of its Features

Most of the times, we purchase the bottled water from the markets without noticing whether it’s potable or not. Potable water means the water that’s drinkable. There have come a number of companies in the Pakistan selling bottled water across the country. My project’s aim is to classify the water from different companies that comes in bottles, on the basis of features which are taken after testing the bottled water from different companies in the labs. And applying machine learning to classify the water as potable or not.

Dataset Description:

I have gathered the dataset from the quarterly reports published by the **Pakistan Council of Research in Water Resources (PCRWR)** on their website.

Website Address: https://pcrwr.gov.pk/bottled-water/

A total of 2496 data samples of bottled water exist in the dataset which is collected by pcrwr from different water supplying companies.

Out of which around **2000** were Potable water samples (**positive**) and around **350** were impotable samples (**negative**).

I reduced the dataset to tackle the class imbalance issue, and now the dataset has equal number of positive and negative classes for better training.

The dataset contains **14 features and 1 class** column having two possible classes/labels; **Potable or Im-Potable (0/1).**

The Features are namely as:

pH, EC, TDS, Ca, Mg, Hard, HCO3, Cl, Na, F, K, NO3, SO4, As

These are the different features that exists in water like its pH, Calcium ratio, Magnesium ratio and so on.

Class/Label: **Potability**

Chapter 2

Methodology

This chapter includes all the methods applied to each module of the system. In each module, I used different methodologies relevant to them to make them working.

To implement all modules we used **Python Programming Language using different libraries and modules like pandas, pyplotlib, numpy, sk learn, keras, tensor flow etc.**

Methodologies for different modules are given below:

Module 1:

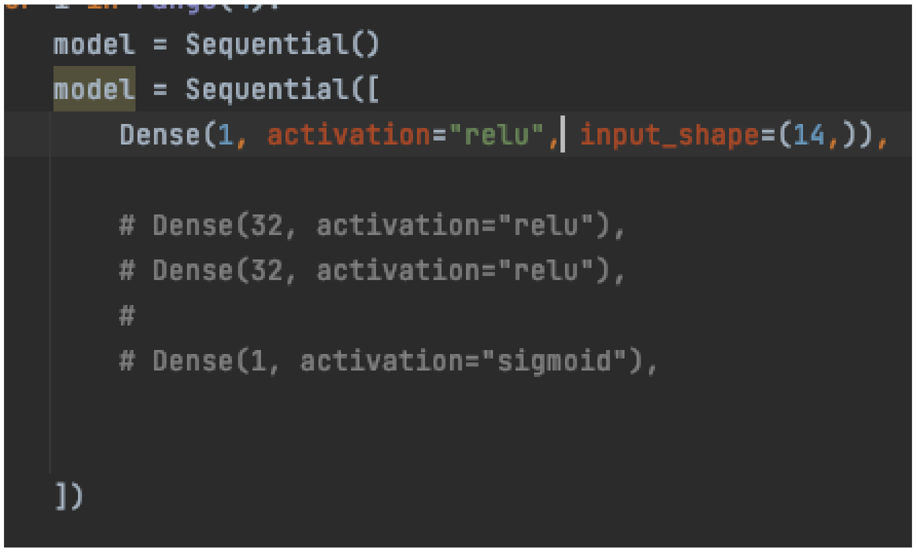
1) For extracting and preprocessing the dataset I used the Word, Excel and Python's preprocessing techniques to prepare my dataset.
2) For creating, reading and writing to the files, I used Python's file module with open function. And I used the Python's pandas library to read the csv dataset files into my dataframes.

Module 2:

1) After successfully creating and importing the dataset as dataframe, I performed the MinMax Scaling to the dataset. Which made the datapoints come in a common range.
2) After this I performed train_test split on my dataset with 70:30 ratio. And got Training set and a testing set for features as well as for class label.
3) Now, I've my training and testing data. I divided the Testing data again into Testing and Validation datasets.
4) The training dataset will be used to check the model's efficiency while the algorithm is running.

Module 3:

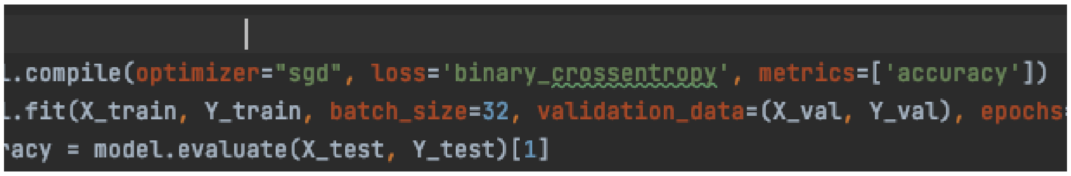
1) Now, I have created the Sequential model architecture for my Artificial Neural Network using following lines of code.



```
model = Sequential()
model = Sequential([
    Dense(1, activation="relu", input_shape=(14,)),

    # Dense(32, activation="relu"),
    # Dense(32, activation="relu"),
    #
    # Dense(1, activation="sigmoid"),


])
```

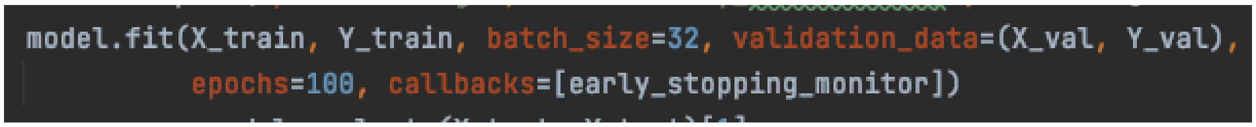
2) I inserted the number of neurons as first parameter to the dense layer inside the sequential model, and the second parameter is for the activation function that I intend to use, and the third parameter tells that I've 14 features in my input layer.

3) Then I compiled my sequential model using compile function, and giving the optimizer as stochastic gradient descent (sgd), binary_crossentropy as my model's loss function and the metrics as accuracy.



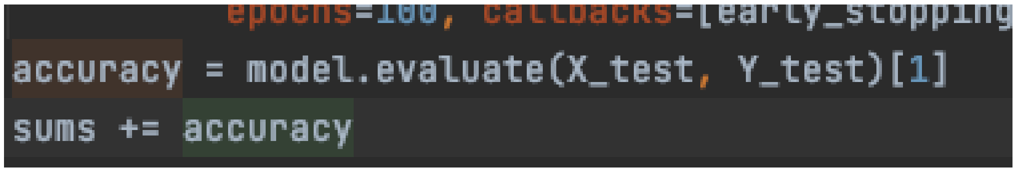
```
.compile(optimizer="sgd", loss='binary_crossentropy', metrics=['accuracy'])
.fit(X_train, Y_train, batch_size=32, validation_data=(X_val, Y_val), epochs
acy = model.evaluate(X_test, Y_test)[1]
```

4) After this I called the model.fit function which will fit the model in other words train the model on training dataset. And the batch size is given as 32 since I used the sgd in which batch of 32 will update its weights and then the next batch from the data points will update and so on. Validation data is used for checking the improvement in the model while it's being trained. And the parameter epochs is set to 100 which means that the model will be trained for 100 cycles. And after all the callbacks say that the model should stop running for more cycles if there is no further improvement in the model, if the model didn't improve for early_stopping_monitor times, then it'll stop.



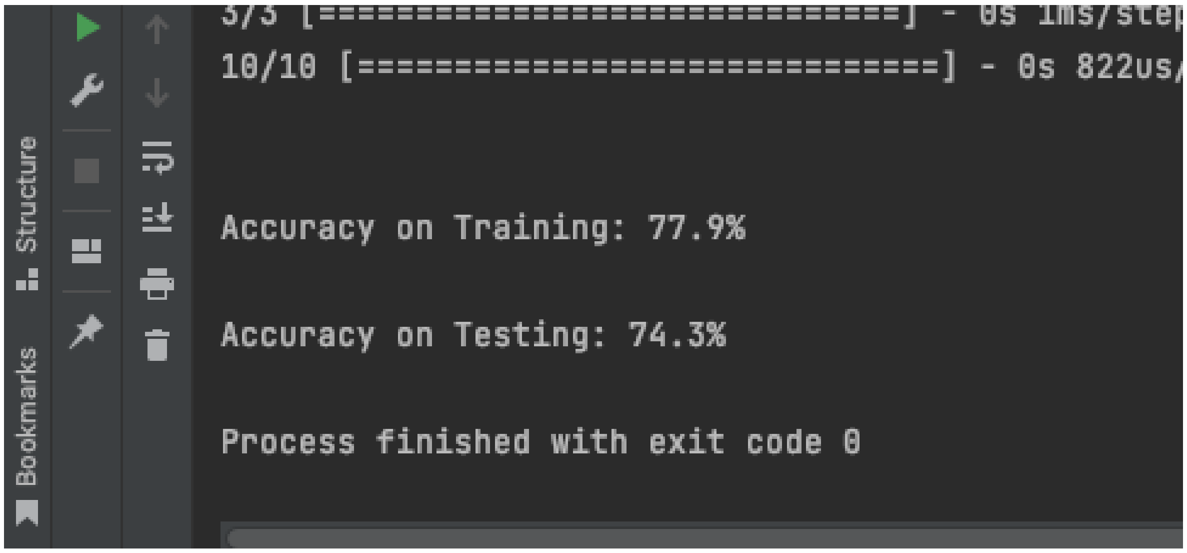
```
model.fit(X_train, Y_train, batch_size=32, validation_data=(X_val, Y_val),
          epochs=100, callbacks=[early_stopping_monitor])
```

5) Once the model has been trained, I evaluated it on my test set. And stored the accuracy from the 1 index into the accuracy variable. I performed this whole process in a loop to take the mean of the accuracies of the evaluation.



```
accuracy = model.evaluate(X_test, Y_test)[1]
sums += accuracy
```

6. My model's accuracy on the Training and Testing data when I used just the input and output layer with no hidden layer in between, and the activation function is used **"Relu"** is given below:

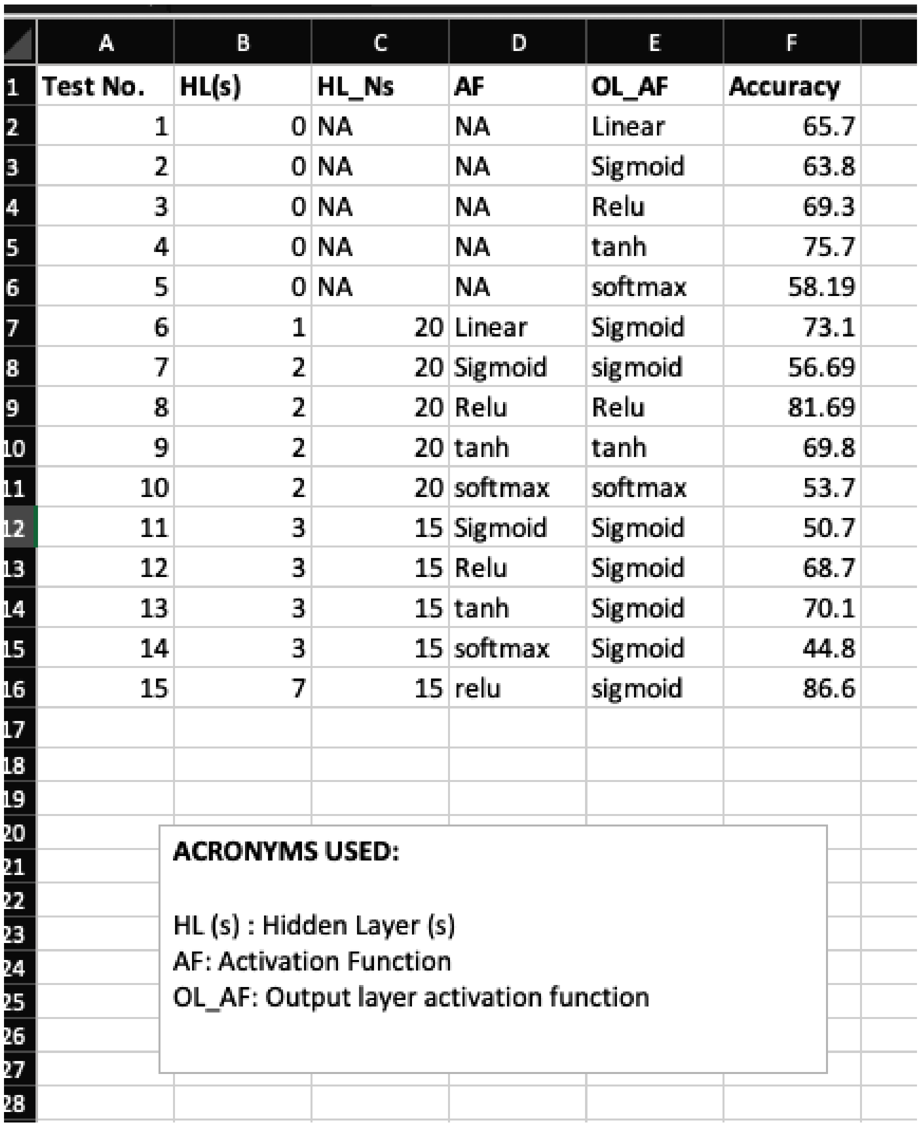


```
3/3 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 822us/


Accuracy on Training: 77.9%

Accuracy on Testing: 74.3%

Process finished with exit code 0
```

Variations in Number of Neurons,

Activation Function, and Hidden Layers

I run the classifier for different configurations of its hidden layers, number of neurons in the hidden layer and the activation functions, I got the following results:



Test No.	HL(s)	HL_Ns	AF	OL_AF	Accuracy
1	0	NA	NA	Linear	65.7
2	0	NA	NA	Sigmoid	63.8
3	0	NA	NA	Relu	69.3
4	0	NA	NA	tanh	75.7
5	0	NA	NA	softmax	58.19
6	1	20	Linear	Sigmoid	73.1
7	2	20	Sigmoid	sigmoid	56.69
8	2	20	Relu	Relu	81.69
9	2	20	tanh	tanh	69.8
10	2	20	softmax	softmax	53.7
11	3	15	Sigmoid	Sigmoid	50.7
12	3	15	Relu	Sigmoid	68.7
13	3	15	tanh	Sigmoid	70.1
14	3	15	softmax	Sigmoid	44.8
15	7	15	relu	sigmoid	86.6

ACRONYMS USED:

HL (s) : Hidden Layer (s)
AF: Activation Function
OL_AF: Output layer activation function

This shows that when we increased the number of hidden layers in the model, the model had more accuracy. And also, the type of activation function that I used. When I used Relu as my hidden layers' activation function I got higher accuracy as compared to the other activation function's accuracy.

Conclusion:

The usage of hidden layers and number of neurons in the hidden layers, as well as the activation function depends on the kind of problem we're solving, the features it have and the pattern of the data. If it's linearly separable we may use the linear activation function. And if we have more features then we need a more stronger activation function. And hidden layers. As well as large number of the neurons.