Criterion C: Development

Word Count: 1060

This application was written using Python (programming language). Below, I have listed all the modules including the main integrated file utilized for the backend implementation.

List of the Techniques

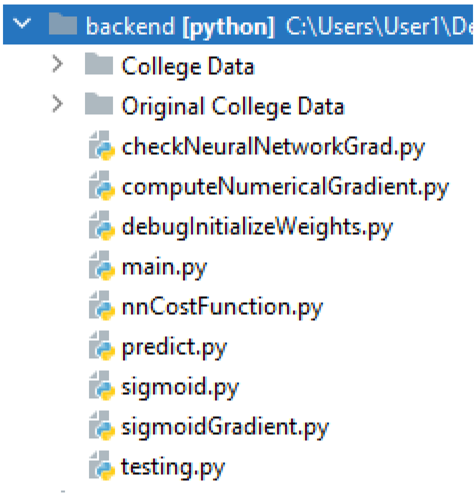
Number	Technique
1	Different methods were created to be used in the application
2	Use of certain Python Libraries
3	2D Arrays, conditional, and loop statements
4	Frontend and Backend folders are packaged in one parent folder
5	Loading Data from csv files
6	Abstract Data Structures (Trees)

List of the Python Libraries and modules

Number	Library
1	Numpy – a numerical library used for working with arrays and performing mathematical and statistical calculations on them.
2	Sklearn – a machine learning library
3	Joblib – a library used to provide lightweight pipelining
4	Os – a python module that provides functions to interact with the operation system of the device
5	Scipy – a numerical library which is an extension of the Numpy Library and is used for scientific calculations.
6	Decimal – a python module which provides support for fast correctly-rounded decimal floating point arithmetic.
7	Sys – a python module that provides functions to manipulate different parts of the Python Runtime Environment

Backend

Below is the list of all the Python Scripts of the modules and folders containing the college data that have been used to implement the backend (calculating the optimized weights ‘Theta1’ and ‘Theta2’ values for the colleges) of the application:



The Python Script ‘**main.py**’ is the final integrated program that utilizes the above listed modules and the college data folder as part of the backend implementation to calculate and store the optimized weights. Also, the folder ‘Original College Data’ consists of the preformatted data files (.csv) that was obtained from the US Govt[1] website, for later utilization by front-end GUI. This is part of back-end implementation as tampering the values will result in calculation errors. Hence, access restricted to admin only.

[1] *Common Data Set*, 18 Nov. 2000, commondataset.org/.

Sigmoid

This is the module which calculates and returns the sigmoid value for the variable passed into its parameter. The code for the module is depicted below:

```python

import numpy as np


def sigmoid(z):
    g = np.zeros(z.shape)
    g = 1 / (1 + np.exp(-z))

    return g

```

SigmoidGradient

This is the module which calculates and returns the gradient of the sigmoid function for the variable passed into its parameter. The code for the module is depicted below:

```python

import sigmoid as s


def sigmoidGradient(z):
    g = s.sigmoid(z)
    g = g * (1 - g)

    return g

```

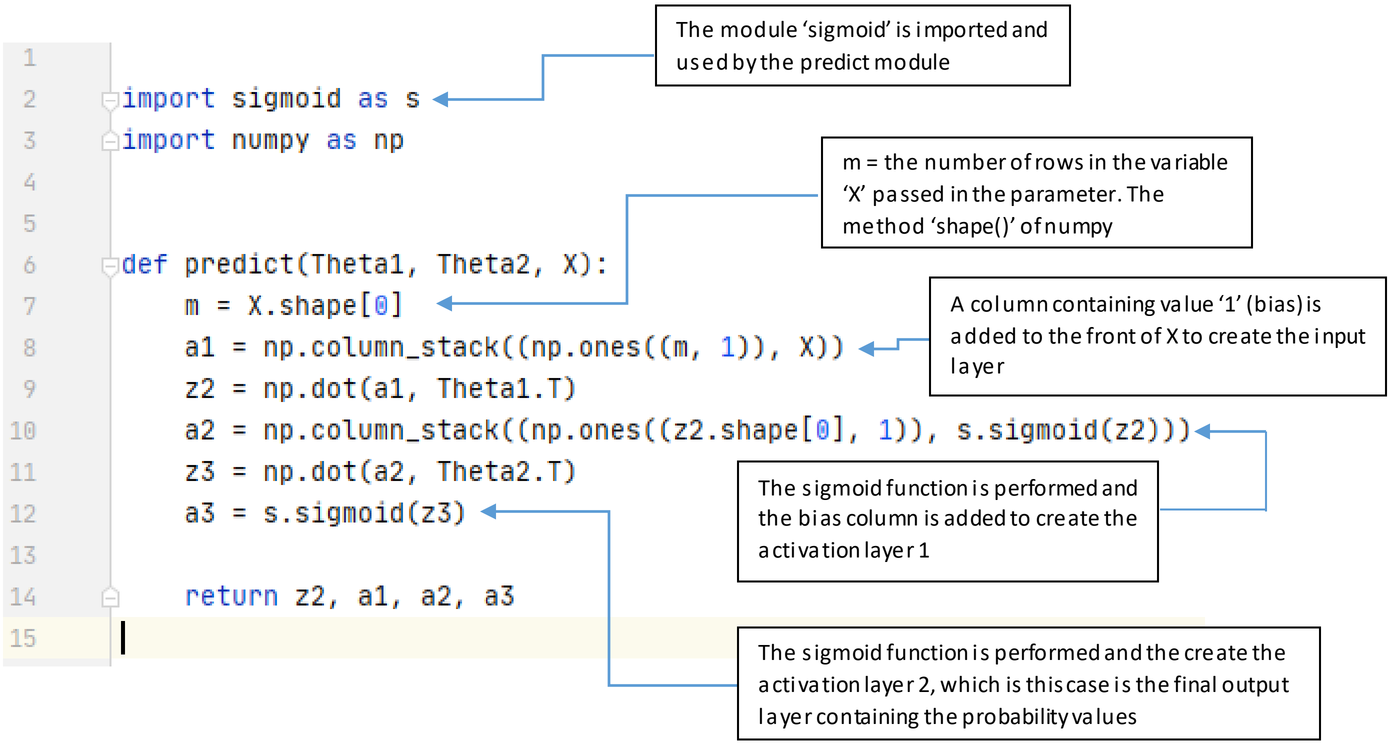
The module 'sigmoid' is imported and used by the 'sigmoidGradient' module

Predict

This is the module which performs the Neural Network's prediction function and returns the values of the various layers of the Neural Network. It also:

- modifies the input data to create the input layer
- Performs the activation functions on the input layer
- Creates the activation layer (hidden layer), and
- Calculates the final layer which consists of the final probability.

The code for the module is depicted below:



z2 and z3 are the variables containing the values after performing the dot product of a1 and a2, with the transpose of Theta1 and Theta2 respectively (the library Numpy is used and its method 'dot ()' is utilized to perform the dot product).

NnCostFunction

This is the module which calculates and returns the cost and the gradient values of the Neural Network being implemented.

The code for the module is depicted below:

```
1
2   import numpy as np
3   import predict as pr
4   import sigmoidGradient as sg
5   def nnCostFunction(nn_params, input_layer_size, hidden_layer_size, num_labels, X, y, lambda_reg):
6       Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
7                           (hidden_layer_size, input_layer_size + 1), order='F')
8       Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
9                           (num_labels, hidden_layer_size + 1), order='F')
10      m = X.shape[0]
11      z2, a1, a2, a3 = pr.predict(Theta1, Theta2, X)
12      cost = np.sum((-y) * np.log(a3) - ((1 - y) * np.log(1 - a3)))
13      J = (1.0 / m) * cost
14      sumOfTheta1 = np.sum(Theta1[:, 1:] ** 2)
15      sumOfTheta2 = np.sum(Theta2[:, 1:] ** 2)
16      J = J + ((lambda_reg / (2.0 * m)) * (sumOfTheta1 + sumOfTheta2))
17      delta3 = a3 - y
18      delta2 = (np.dot(delta3, Theta2)) * np.column_stack((np.ones((z2.shape[0], 1)), sg.sigmoidGradient(z2)))
19      delta2 = delta2[:, 1:]
20      Theta2_grad = (np.dot(delta3.T, a2)) * (1 / m)
21      Theta1_grad = (np.dot(delta2.T, a1)) * (1 / m)
22      Theta2_grad = Theta2_grad + (
23              (float(lambda_reg) / m) * np.column_stack((np.zeros((Theta2.shape[0], 1)), Theta2[:, 1:])))
24      Theta1_grad = Theta1_grad + (
25              (float(lambda_reg) / m) * np.column_stack((np.zeros((Theta1.shape[0], 1)), Theta1[:, 1:])))
26      grad = np.concatenate(
27          (Theta1_grad.reshape(Theta1_grad.size, order='F'), Theta2_grad.reshape(Theta2_grad.size, order='F')))
28
29      return J, grad
```

The modules 'sigmoid' and 'predict' are imported and used

'nn_params' is unrolled back into the respective weights 'Theta1' and 'Theta2'

The Unregularized Cost Function value is calculated

The Regularized Cost Function value is calculated

Unregularized back-propagation takes place and unregularized gradients of the weights are calculated and stored in 'Theta1_grad', 'Theta2_grad' respectively

Regularization is added to 'Theta1_grad', 'Theta2_grad' and rolled into one long vector 'grad

The regularized Cost Function value 'J' and regularized gradients vector 'grad' are returned (**line 29**).

ComputeNumericalGradient

This is the module which calculates and returns the numerical gradient values for a Neural Network based on the central difference formula:

$$\frac{d}{d\theta}J(\theta) = \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

where J(Ө) is a function of Ө (in this case the function being the nnCostFunction and 'Ө' being the rolled vector 'nn_params') being minimized and ϵ is epsilon which is set to a very small constant, usually having the value 10^{-4} (1e-4).

Thus, the code for the module is depicted below:

```

import numpy as np


def computeNumericalGradient(J, theta):
    numgrad = np.zeros(theta.shape[0])
    perturb = np.zeros(theta.shape[0])
    e = 1e-4
    for p in range(theta.size):

        # Set perturbation vector
        perturb[p] = e
        loss1 = J(theta - perturb)
        loss2 = J(theta + perturb)

        # Compute Numerical Gradient
        numgrad[p] = (loss2[0] - loss1[0]) / (2*e)
        perturb[p] = 0

    return numgrad

```

Debugging

This is the module which is used to generate random test data based on the number of rows and columns.

The code for the module is depicted below:

```python

import numpy as np


def debugging(rows, cols):
    data = np.zeros((rows, cols))
    data = np.reshape(np.sin(range(data.size)), data.shape) / 10

    return data

```

Initializing array 'data' having the number rows and columns passed in as parameters of the module, and having all values in it initially as 0

Modifying 'data' array values using sine function ensures that 'data' always consists of the same values and will be useful for debugging

CheckNeuralNetworkGrad

This is the module which is used to create a small neural network in order to check if backpropagation has been implemented correctly. It will compare the difference between the numerical gradient and analytical gradient produced by backpropagation implementation in the module 'nnCostFunction', for a test data, and weights producing using the module 'debugging'.

The code for the module is depicted below:

```python
import numpy as np
import debugging as deb
import nnCostFunction as nncf
import computeNumericalGradient as cng
from decimal import Decimal
import sys


def checkNeuralNetworkGrad(lambda_reg):
    input_layer_size = 5
    hidden_layer_size = 10
    num_labels = 1
    m = 6
    Theta1 = deb.debugging(hidden_layer_size, input_layer_size + 1)
    Theta2 = deb.debugging(num_labels, hidden_layer_size + 1)
    X = deb.debugging(m, input_layer_size)
    y = np.random.randint(0, num_labels + 1, (m, num_labels))
    nn_params = np.concatenate((Theta1.reshape(Theta1.size, order='F'), Theta2.reshape(Theta2.size, order='F')))

```

The modules 'debugging', 'nnCostFunction', and 'computeNumericalGradient' are imported and used

Test weights 'Theta1' and 'Theta2', input data and the output data 'X' and 'y' are generated using the debugging method of the module 'debugging'

Test weights 'Theta1' and 'Theta2' are rolled into one long single-dimensional array 'nn_params'

Short hand called ‘CostFunc’ of the nnCostFunction method present in the module ‘nnCostFunction’ is created

```python
20 def costFunc(p):
21     return nncf.nnCostFunction(p, input_layer_size, hidden_layer_size,
22                                num_labels, X, y, lambda_reg)
23 _, grad = costFunc(nn_params)
24 numgrad = cng.computeNumericalGradient(costFunc, nn_params)
25 diff = Decimal(np.linalg.norm(numgrad - grad)) / Decimal(np.linalg.norm(numgrad + grad))
26 if diff > 1e-9:
27     print("Error, relative difference is greater than 1e-9, check difference")
28     sys.exit()
29 
```

The analytical and numerical gradient are calculated using which their relative difference is computed and stored in variable ‘diff’

The short hand ‘CostFunc’ is created because the Cost Function of the Neural Network being implemented is the function that has to be minimized in the ‘computeNumericalGradient’ method of the module ‘computeNumericalGradient’ in order to calculate the numerical gradients for test data and rolled weights array ‘nn_params’ generated using the debugging method of the module ‘debugging’, in the method checkNeuralNetworkGrad of the module ‘checkNeuralNetworkGrad’.

If the implementation of the backpropagation is correct, the value of ‘diff’ should always be less than **1e-9** (10^{-9}).

Thus to ensure the correctness of backend, an ‘if’ **condition** is created from **lines 26-28** which would terminate the program if **‘diff’ > 1e-9**, displaying the error that the value of diff is greater than 1e-9.

<u>Main</u>

```python

import numpy as np
import nnCostFunction as nncf
import checkNeuralNetworkGrad as cnng
import predict as pr
from sklearn import metrics
from sklearn.model_selection import train_test_split
from joblib import dump
import os
from scipy.optimize import minimize
from sklearn import preprocessing


my_path = os.path.abspath(os.path.dirname(__file__))
hidden_layer_size = 10
num_labels = 1
```

All the python libraries and modules developed to be utilized are imported

Relative referencing is utilized to obtain the path wherein this script resides on a system. Relative referencing is utilized as the location of the folder containing the script might differ when the application is setup for the client

hidden_layer_size is the number of nodes in the 2nd layer of the Neural Network and 'num_labels' is the number of nodes in the last layer (output layer) of the Neural Network to be implemented

```python
all_statusfiles = []
colleges = []
all_featurefiles = []
for r, d, f in os.walk(my_path + '/College Data/'):
    colleges.append(d)
    for item in f:
        if 'Admissionstatus.csv' in item:
            all_statusfiles.append(os.path.join(r, item))
        elif 'Featuresdata.csv' in item:
            all_featurefiles.append(os.path.join(r, item))

collegelist = np.array(colleges[0])
num_of_colleges = len(collegelist)
```

In order to obtain the list of the colleges, the path wherein the files '**Admissionstatus.csv**' is stored for each test type (ACT, SAT) in every college, and the path wherein the files '**Featuresdata.csv**' is stored for each test type (ACT, SAT) in every college, a **depth-first preorder traversal** is done and the list of colleges and the paths are appended to the lists **colleges**, **all_statusfiles**, and **all_featurefiles** accordingly

The list **colleges** consists of numerous nested lists of the subdirectories because each college also consists of the subdirectories 'ACT' and 'SAT'. However, since only the list of the name of colleges is required, the data in the 1st index of 'colleges' list is extracted and put into an array **collegelist**. It has been converted into an array as few operations have to be performed on it as an array

The number of colleges is calculated and stored in the variable by finding the number of elements in the array '**collegelist**'

Since the weights need to be optimized for files containing data for ACT and SAT test, each college has to be repeated twice. For example, an array having elements [1, 2] will be transformed to [1, 1, 2, 2].

Since the weights need to be calculated and optimized twice i.e. for data with GPA Score Results and data with IB Predicted Score Results.

The number of nodes in the 1st layer i.e. input layer is calculated based on the no: of features/columns in **x**

Elements of **all_featurefiles** containing the features data are accessed in a loop and are loaded as a 2-D array

Elements of **all_statusfiles** containing the admission status data are accessed in a loop and initially loaded as a 1D array. However, a new axis is added and transposed to convert it to a 2D array and in the required format

m is the number of rows (records) in the array '**X**'

```
30 collegelist = np.repeat(collegelist, 2)
31 for j in range(2):
32     for i in range(len(all_featurefiles)):
33         x = np.loadtxt(all_featurefiles[i], delimiter=",")
34         if j == 1:
35             x = np.delete(x, 1, 1)
36         elif j == 0:
37             x = np.delete(x, 2, 1)
38         input_layer_size = x.shape[1]
39         scaler = preprocessing.StandardScaler().fit(x)
40         x = scaler.transform(x)
41         abc = np.loadtxt(all_statusfiles[i])[np.newaxis]
42         Y = abc.T
43         x_train, x_test, y_train, y_test = train_test_split(x, Y, test_size=0.3)
44         X = x_train
45         y = y_train
46         m = X.shape[0]
47         Theta1 = np.random.rand(hidden_layer_size, input_layer_size + 1)
48         Theta2 = np.random.rand(num_labels, hidden_layer_size + 1)
49         nn_params = np.concatenate((Theta1.reshape(Theta1.size, order='F'), Theta2.reshape(Theta2.size, order='F')))
```

In **lines 34-37**, based on the if condition stated, for 1st outer loop, the column containing the IB Predicted Score is deleted and therefore all the processes in the inner loop will be carried out for data with GPA Score. For 2nd outer loop, the column containing the GPA Score is deleted and therefore all the processes in the inner loop will be carried out for data with IB Predicted Score.

In **lines 39-40**, the feature dataset loaded as a 2-D array in variable '**x**', is further processed using the StandardScaler method () of sklearn, which standardizes all the features by removing the mean and scaling it to unit variance.

In **lines 43-45**, the feature data and the corresponding admission status data loaded as 2D arrays in '**x**' and '**Y**' are further split into training data and testing data based on a 70/30 % split and therefore the training features data is stored as '**X**' and testing features data as '**x_test**'. The corresponding admission status data is also split and therefore the training admission status data is stored as '**y**' and testing admission status data as '**y_test**'. '**X**' and '**y**' will be utilized for training the neural network while ‘**x_test**’ and ‘**y_test**’ will be utilized to check how the performance of the created model is for its predictions.

In **lines 47-48**, the weights ‘**Theta1**’ and ‘**Theta2**’ are initialized as 2D arrays with their sizes based on the value of ‘**input_layer_size**’, ‘**hidden_layer_size**’, and ‘**num_labels**’, and contain random values which will accordingly be optimized. Furthermore, in **line 49**, the weights are then rolled into a 1D array and stored as ‘**nn_params**’.

```
50  lambda_reg = 0
51  # Checking Backpropagation
52  cnng.checkNeuralNetworkGrad(lambda_reg)
53  # Checking Backpropagation with Regularization
54  lambda_reg = 0.01
55  cnng.checkNeuralNetworkGrad(lambda_reg)
56
57  maxiter = 100000
58  myargs = (input_layer_size, hidden_layer_size, num_labels, X, y, lambda_reg)
59  results = minimize(nncf.nnCostFunction, x0=nn_params, args=myargs,
60                     options={'disp': True, 'maxiter': maxiter, 'ftol': 0}, method="L-BFGS-B", jac=True)
61
62  nn_params = results.x
```

lambda_reg is the regularization term used for fine tuning the weights. The checkNeuralNetworkGrad method is utilized to check the implementation of the backpropagation with and without regularization and is used from the imported module ‘**checkNeuralNetworkGrad**’

The neural network is trained, wherein with each iteration the cost function value decreases. The iterations stop either when the cost function value converges or when the total no: of iterations executed = **maxiter**

The optimized rolled weights are calculated while training the neural network and thus **nn_params** is updated

```
63  Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
64                      (hidden_layer_size, input_layer_size + 1), order='F')
65  Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
66                      (num_labels, hidden_layer_size + 1), order='F')
67  y_test_prob = pr.predict(Theta1, Theta2, x_test)[3]
68  precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_test_prob)
69  auc = metrics.auc(recall, precision)
70  aucarray = np.array([auc])
```

Obtaining the optimized weights **Theta1** and ‘**Theta2** back from **nn_params**

The probability of admission is calculated for **x_test** multi-dimensional array using the predict method in the imported module ‘**predict**’, based on which the Area under the Precision-Recall curve (AUC) is calculated and stored in **aucarray** (auc is converted to array because only arrays can be saved in a.csv file)

The AUC score is an indicator which checks the accuracy of prediction based on the optimized weights calculated. The value for this ranges from 0 to 1. A model having an AUC value greater than or equal to 0.7 indicates a high precision in making its predictions.

```python
if j == 1:
    if i % 2 == 0:
        IBPath = my_path + "/Optimized Weights for IB/" + collegelist[i] + "/" + "ACT"
    elif i % 2 != 0:
        IBPath = my_path + "/Optimized Weights for IB/" + collegelist[i] + "/" + "SAT"
    if not os.path.exists(IBPath):
        os.makedirs(IBPath)
    np.savetxt(os.path.join(IBPath, 'Theta1.csv'), Theta1, delimiter=',')
    np.savetxt(os.path.join(IBPath, 'Theta2.csv'), Theta2, delimiter=',')
    np.savetxt(os.path.join(IBPath, 'Auc.csv'), aucarray)
    dump(scaler, os.path.join(IBPath, 'scaler_file.joblib'))
elif j == 0:
    if i % 2 == 0:
        GPAPath = my_path + "/Optimized Weights for GPA/" + collegelist[i] + "/" + "ACT"
    elif i % 2 != 0:
        GPAPath = my_path + "/Optimized Weights for GPA/" + collegelist[i] + "/" + "SAT"
    if not os.path.exists(GPAPath):
        os.makedirs(GPAPath)
    np.savetxt(os.path.join(GPAPath, 'Theta1.csv'), Theta1, delimiter=',')
    np.savetxt(os.path.join(GPAPath, 'Theta2.csv'), Theta2, delimiter=',')
    np.savetxt(os.path.join(GPAPath, 'Auc.csv'), aucarray, )
    dump(scaler, os.path.join(GPAPath, 'scaler_file.joblib'))
```

In the following line of code above, based on the values of the inner loop and outer loop (nested if condition), the exam score type (IB Predicted Score/GPA Score), test type score (ACT/SAT), the path of where the optimized weights **Theta1** and **Theta2**, the AUC score (**aucarray**), and scaler are to be stored in appropriate file formats (.csv or .joblib) is determined and stored there so that it can be used for future prediction by the GUI Program. If the path does not exist, it is created.

The annotated code of the complete application has been attached in the Appendix.

Bibliography and References

1. Bonnin, Rodolfo, and Claudio Delrieux. Machine Learning for Developers: Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning. Packt, 2017.

2. *Common Data Set*, 18 Nov. 2000, commondataset.org/.

3. "Machine Learning." *Coursera*, www.coursera.org/learn/machine-learning.

4. NG, Andrew. *Basics of Neural Network Programming*. cs230.stanford.edu/files/C1M2.pdf.