TAYLOR'S UNIVERSITY

Wisdom • Integrity • Excellence

ITS66904 Big Data Technologies

ASSIGNMENT

Cover Sheet

HAND OUT DATE: **8th February 2022**

HAND IN DATE: **20th February 2022**

WEIGHTAGE: **15%**

INTAKE: **Semester January 2022**

Instructions to student:

- This is an individual assignment.
- Complete this cover sheet and attach it to your assignment – this should be your first page!

Student declaration:	
I declare that: ▪ ***I understand what is meant by plagiarism*** ▪ ***The implication of plagiarism has been explained to me by our lecturer*** ***This assignment is my own work.***	
Names	**Student ID**
MD MESBAHUR RAHMAN	0344684

Asgn_0344684_ITS66904

MD Mesbahur Rahman (0344684)

February 19, 2022

Contents

1 Introduction

This assignment will be showing the usage of Pandas, Numpy, and KERAS to create a prediction model. We will be acquiring a dataset, making sure that the entries in said dataset is valid. After this stage, we will be formatting the data such that it can be read by KERAS. KERAS will be used to develop the prediction model. This prediction model will take the formatted input from the dataset and use a neural network to compare the input variables to the output variable, and hence make predictions.

2 The Dataset

The dataset we will be using describes different details of a student, and whether or not they have received a distinction award. The dataset contains 50,000 entries and has 8 attributes each. These attribues are as follows:

id : The unique identifier of the record and the student.

name : The name of the student

age : The age of the student

gender: The biological gender of the student; with a value of 0 indicating Female, and a value of 1 indicating Male.

section : The section of the student. Students in lower numbered sections are supposedly better in academics.

subject_count: The number of subjects taken by the student.

avg_study_hrs: The number of hours a student spends studying per day.

distinction: Indicates if a student has recieved a distinction. A value of 1 indicated they've received the award.

The dataset is loaded into our application using the following commands in python

[21]:
```
#importing pandas
import pandas as pd

#df = DataFrame
df = pd.read_csv("records.csv",low_memory=False) #low_memory = False because we␣
 ↪are using a large dataset.

df.head(20)
```

[21]:

	id	name	age	gender	section	subject_count	\
0	0	Jason Dust	20.0	0	6.0	4.0	
1	1	Jason De Ville	18.0	0	2.0	3.0	

2	2	Sofia Discord	24.0	1	3.0	5.0
3	3	Noah Rundown	17.0	0	2.0	2.0
4	4	James Static	23.0	0	9.0	3.0
5	5	Amelia Wendigo	24.0	1	3.0	6.0
6	6	Leah Shivers	19.0	1	9.0	3.0
7	7	Madison Venom	20.0	1	8.0	5.0
8	8	Jameson Checkmate	19.0	0	3.0	5.0
9	9	Camila Vader	24.0	1	8.0	2.0
10	10	Nicholas Mimic	16.0	0	NaN	-2.0
11	11	Nova Diamondback	24.0	1	5.0	4.0
12	12	Charlotte Surge	23.0	1	8.0	4.0
13	13	Easton	22.0	0	2.0	2.0
14	14	Parker Buttons	21.0	0	2.0	5.0
15	15	Olivia Shivers	22.0	1	4.0	6.0
16	16	Wyatt Zorg	19.0	0	4.0	6.0
17	17	Connor Diamondback	22.0	0	9.0	6.0
18	18	Leo Zorg	23.0	0	3.0	2.0
19	19	Jonathan Mutilator	22.0	0	3.0	5.0

	avg_study_hrs	distinction
0	3.0	1
1	2.0	0
2	7.0	1
3	2.0	0
4	4.0	1
5	6.0	1
6	5.0	1
7	1.0	0
8	1.0	0
9	NaN	1
10	5.0	1
11	7.0	1
12	1.0	1
13	3.0	1
14	3.0	0
15	1.0	0
16	3.0	0
17	3.0	1
18	3.0	1
19	4.0	0

It can be seen that the data has been successfully loaded. We can now move onto exploring our data.

3 Exploring the Data

We will first see an overview of the different datatypes of the data. We can do this using a pandas dataframe method.

[2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             50000 non-null  int64
 1   name           49990 non-null  object
 2   age            48987 non-null  float64
 3   gender         50000 non-null  int64
 4   section        48959 non-null  float64
 5   subject_count  49496 non-null  float64
 6   avg_study_hrs  48992 non-null  float64
 7   distinction    50000 non-null  int64
dtypes: float64(4), int64(3), object(1)
memory usage: 3.1+ MB
```

We can see that the datatypes of each column is correct. We can see that there are a few null items in some of the columns. This is missing data. We can count the missing data using another method.

[3]:
```
def showMissing(df):
    missing = df.isna().sum()
    missing = pd.DataFrame(missing, columns =["Null Values"])
    print(missing)
showMissing(df)
```

```
               Null Values
id                       0
name                    10
age                   1013
gender                   0
section               1041
subject_count          504
avg_study_hrs         1008
distinction              0
```

This shows us the number of null values in each column.

3.1 Invalid values

Additionally, we must look at a statistical overview of the data to see if there are any invalid values amongst the records. We accomplish this using the "describe" method for pandas dataframes.

[4]:
```
df.describe()
```

[4]:

	id	age	gender	section	subject_count	\
count	50000.000000	48987.000000	50000.000000	48959.000000	49496.000000	
mean	24999.500000	19.995999	0.501440	5.023898	3.589381	
std	14433.901067	2.585433	0.500003	2.576157	2.258649	
min	0.000000	16.000000	0.000000	1.000000	-6.000000	
25%	12499.750000	18.000000	0.000000	3.000000	3.000000	
50%	24999.500000	20.000000	1.000000	5.000000	4.000000	
75%	37499.250000	22.000000	1.000000	7.000000	5.000000	
max	49999.000000	24.000000	1.000000	9.000000	6.000000	

	avg_study_hrs	distinction
count	48992.000000	50000.000000
mean	4.050498	0.646580
std	2.313286	0.478036
min	0.000000	0.000000
25%	2.000000	0.000000
50%	4.000000	1.000000
75%	6.000000	1.000000
max	8.000000	1.000000

We can see that the minimum value for the "subject_count" is "-6". This is not possible because a student is not able to take a negative number of subjects. Hence, we must mark all negative values for this columns as "NaN" or missing. We can do this using Numpy.

[5]:
```
#importing numpy
import numpy as np

#Ignores false positive warnings for improper editing of data.
import warnings
warnings.simplefilter(action='ignore')

df["subject_count"][df["subject_count"]<=0] = np.nan
df.describe()
```

[5]:

	id	age	gender	section	subject_count	\
count	50000.000000	48987.000000	50000.000000	48959.000000	46980.000000	
mean	24999.500000	19.995999	0.501440	5.023898	3.996509	
std	14433.901067	2.585433	0.500003	2.576157	1.416736	
min	0.000000	16.000000	0.000000	1.000000	2.000000	
25%	12499.750000	18.000000	0.000000	3.000000	3.000000	
50%	24999.500000	20.000000	1.000000	5.000000	4.000000	
75%	37499.250000	22.000000	1.000000	7.000000	5.000000	
max	49999.000000	24.000000	1.000000	9.000000	6.000000	

	avg_study_hrs	distinction
count	48992.000000	50000.000000

```
mean        4.050498      0.646580
std         2.313286      0.478036
min         0.000000      0.000000
25%         2.000000      0.000000
50%         4.000000      1.000000
75%         6.000000      1.000000
max         8.000000      1.000000
```

[6]:
```
showMissing(df)
```

```
                Null Values
id                        0
name                     10
age                    1013
gender                    0
section                1041
subject_count          3020
avg_study_hrs          1008
distinction               0
```

The number of missing entries for the subject count has increased, and hence we can see that we have successfuly marked the invalid values as missing.

4 Cleaning the Dataset

Now that we have seen and marked the missing values in our dataset, we have to "clean" our data. We will do this by using the appropriate method to adjust each value such that they are valid.

4.1 Numerical Values

For our numerical values, we are able to replace each missing value with the mean value of the corresponding attribute, and round it to the nearest integer.

[7]:
```
df["age"] = df["age"].replace(np.nan, round(df["age"].mean()))
df["section"] = df["section"].replace(np.nan, round(df["section"].mean()))
df["subject_count"] = df["subject_count"].replace(np.nan,␣
 ↪round(df["subject_count"].mean()))
df["avg_study_hrs"] = df["avg_study_hrs"].replace(np.nan,␣
 ↪round(df["avg_study_hrs"].mean()))
```

We can now check again for missing values.

[8]:
```
showMissing(df)
```

```
                Null Values
id                        0
name                     10
age                       0
```

```
gender                0
section               0
subject_count         0
avg_study_hrs         0
distinction           0
```

We have successfully replaced the missing values in the numerical columns using the mean value.

4.2 Object values

We can see that the "name" column still has a number of missing values. There is no method we can use to replace the missing value with a valid value and hence we will drop the records entirely.

[9]:
```
#storing the clean dataset in a different variable
clean_df = df.dropna()
```

[10]:
```
showMissing(clean_df)
```

```
               Null Values
id                       0
name                     0
age                      0
gender                   0
section                  0
subject_count            0
avg_study_hrs            0
distinction              0
```

We can see that our dataset is now void of missing or invalid values. Hence, we can say that our data cleaning is complete. We will be saving the cleaned records in a seperate "csv" file.

[11]:
```
clean_df.to_csv('clean_records.csv', index=False)
```

5 Model Development

The clean records provide us with an appropriate dataset that can be used by the prediction model.

First, we will load the dataset using Numpy.

The first 2 columns in our dataset, "id" and "name" are unique identifiers; hence, they cannot be used for prediction. We will be ignoring these attributes.

[12]:
```
#usecols dodges the id and the name
records = np.loadtxt("clean_records.csv", delimiter =",",skiprows = 1,usecols␣
↪=(2,3,4,5,6,7))
```

We then will split our dataset into 2 sections. The first section is our feature matrix; This contains the features that will be used to make the prediction. The second section is our output variable; This is the variable that our model will be predicting.

[13]:
```
#the variable x contains the feature matrix
x = records[:,0:5]
# the variable y contains the output variable
y = records[:,5]
```

We will now be building the prediction model using a neural network.

To build our neural network, we will be using a sequential model. This is because there are multiple inputs and one output in our dataset.

We will use a number of dense layers , with a number of neurons in each layer, along with an output neuron. The number of dense layers and the number of neurons in each layer is configured by trial and error. In general, the more neurons and the more layers, we will get more accurate returns. However, this will increase our compute times drastically. Hence, we must find a balance between model accuracy and computation time.

After testing, we've seen that an optimal configuration of the neurons is:

```
20 Neurons in the first Dense Layer
10 Neurons in the second Dense Layer
1 Neuron in the last Dense layer
```

The activation type of each layer describes a mathematical function. We use "relu" for the first 2 layers and "sigmoid" for the last layer.

[14]:
```
#import relevant classes and methods from keras
from keras.models import Sequential
from keras.layers import Dense

#this is the neural network
predictor  = Sequential()

#first layer
predictor.add(Dense(20, input_dim=5, activation='relu'))
#second layer
predictor.add(Dense(10, activation='relu'))
#third layer
predictor.add(Dense(1, activation='sigmoid'))
```

Now that we've configured our neural network, we can proceed to compiling the model using our feature matrix and target values. There are a few parameters we will need to configure our model to compile with so that it functions correctly.

The "loss" parameter will be filled with a method appropriate to our output class. Our output can only be classified in either 0 or 1, and hence we will be using binary crossentropy.

The "optimizer" parameter determines how the weightings of the neurons are calculated. We will be using the Adam optimizer.

The parameter, "metrics" is used so we are able to see the current self-determined accuracy of the model.

[15]:
```
predictor.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

Next we must fit the model to the dataset.

The "epochs" parameter defines the cutoff point of the testing. Theoretically, the epoch should be as high as possible to gain as high an accuracy as possible, however, there are diminishing returns and higher epochs can take significantly higher computation time. Through trial an error, we've seen that an epochs value of 3000 gives us a model with relaible results. A lower value gives us a less than desirable accuracy. A higher value takes too long and gives us significantly diminished increase in accuracy.

The "batch_size" parameter defines the size of the input. We've set this to be the size of the dataset.

[16]:
```
predictor.fit(x,y, epochs = 3000,batch_size = (clean_df.shape[0]))
```

```
Epoch 2998/3000
1/1 [==============================] - 0s 68ms/step - loss: 0.1778 - accuracy:
0.9520
Epoch 2999/3000
1/1 [==============================] - 0s 68ms/step - loss: 0.1778 - accuracy:
0.9522
Epoch 3000/3000
1/1 [==============================] - 0s 70ms/step - loss: 0.1778 - accuracy:
0.9522
```

[16]:
```
<keras.callbacks.History at 0x22739be00d0>
```

Next we will evaluate the model.

[17]:
```
_,accuracy = predictor.evaluate(x, y)
print("Accuracy: ", accuracy)
```

```
1563/1563 [==============================] - 1s 445us/step - loss: 0.1778 -
accuracy: 0.9520
Accuracy:  0.9520103931427002
```

It can be seen that we get an accuracy of above 95% This tells us our model is reliable.

6 Predictions using the completed model

The model is complete and we will be using it on a test data set, and comparing it's predictions with the actual value.

The test data contains 300 records.

Once again, we will load the test data using Pandas and Numpy.

[18]:
```
test_data = pd.read_csv("test_data.csv")
```

```
#drop the unique identifiers
test_data = test_data.drop(["id","name"], axis = 1)

#seperate the target variable from the feature matrix
#this is the target variable
test_target = test_data["distinction"]
#reshaping
final_output = np.array(test_target.values)
final_output = np.reshape(final_output, (-1, 1))

#this is the feature matrix
test_features = test_data.drop("distinction",axis = 1)

#save the feature set to a csv to load into numpy.loadtext later
test_features.to_csv("test_features.csv", index = False)

feature_matrix = np.loadtxt("test_features.csv", delimiter =",",skiprows = 1)

final_feature_matrix = feature_matrix[:,0:len(feature_matrix)]
```

Now, we make the predictions using the model.

[19]:
```
#this array stores the predictions made from the model
#the predictions are made using the input from the test dataset
predictions = predictor.predict_on_batch(final_feature_matrix)
```

6.1 Confusion Matrix

We now have our predicted values. Our test data contains 300 items and hence we need a way to visualize our results. We will be making use of a confusion matrix.

This allows us to compare the actual values with the predicted values.

[20]:
```
#The predictions are in the form of continuous values.
#This converts the continuous values to 1 or 0.
for r in range(len(predictions)):
    predictions[r][0] = round(predictions[r][0])

#importing sklearn for the confusion matrix
from sklearn.metrics import confusion_matrix

cf_matrix = confusion_matrix(final_output, predictions)

matrix = pd.DataFrame(cf_matrix, index = ["Actual False"," Actual True"],␣
 ↪columns =["Predicted False","Predicted True"])
matrix
```

[20]:

	Predicted False	Predicted True
Actual False	83	9
Actual True	4	204

The confusion matrix shows us that the predictions are able to mostly predict the correct value. We are able to confirm that the model works as intended.

7 Conclusion

We've chosen an appropriate dataset and have taken the steps to remove unwanted data. We were able to successfully create a neural network and use our dataset to fit the model to the data's specifications. We've tested the model against a seperate test dataset. It can be seen that the model successfully predicts items outside its own training set. We can say that we have created an accurate prediction model.