**Capstone Project Neural Networks**

**Introduction**

This report presents a capstone project that utilizes neural networks to predict student satisfaction scores based on various factors and demographic information. The project involves reading data from Google Big Query tables, cleaning and preprocessing the data, and building a neural network using the Keras library. The report outlines the steps taken to create the neural network and evaluates the accuracy of the model.

**Data Collection and Preprocessing**

The project involves reading data from various tables stored in a Google Big Query dataset using the pandas\_gbq library. The data is cleaned and preprocessed to create two data frames representing the accommodation feedback and academic feedback. The satisfaction score for each student is calculated based on different factors using a weighted average. The categorical variables in the Demographics table are encoded using the LabelEncoder function from scikit-learn. Four different data frames are created, each containing specific demographic information. Finally, the different data frames are combined to create a single data frame containing all the relevant information for each student.

**Model Building and Evaluation of feedforward neural network (FFNN)**

The code defines and trains a feedforward neural network (**FFNN**) using the Keras library with the TensorFlow backend.

First, the necessary libraries are imported: train\_test\_split, StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, mean\_absolute\_error, mean\_squared\_error, r2\_score, Sequential, Dense, Dropout, Activation, Adam, SGD, RMSprop, Nadam, and backend from Keras.

The swish function is also defined, which applies the swish activation function to a tensor using the sigmoid function with a specified beta value.

The data is then standardized using **StandardScaler** before being split into training and testing sets using the **train\_test\_split** function.

The neural network model is defined using the Sequential model, which is a linear stack of layers. The model has an input layer with 300 neurons**,** followed by five hidden layers with **200, 200, 100, 100, and 50 neurons**, respectively. Each hidden layer uses the swish activation function and includes a dropout layer with a rate of **0.5** to prevent overfitting. The output layer has a single neuron and uses a linear activation function.

The model is compiled using the **Adam optimizer** with a learning rate of **0.003** and the mean squared error loss function.

The model is trained using the fit function with the training data, for 250 epochs and with a batch size of 32.

After training, the model is evaluated on the testing data using mean squared error (**mse**), mean absolute error (**mae**), root mean squared error (**rmse**), and r-squared (**r2**) metrics.

Finally, the predicted and actual values are plotted using matplotlib and saved in a DataFrame, which includes an ID column.

The **Acu\_vs\_pre** DataFrame is then rounded to two decimal places and its summary information is displayed using the **info()** function.

**Creating schema for Act\_vs\_pre tables and pushing it to google BigQuery**

The above code creates a schema for a new table called **Act\_vs\_pre** in the BigQuery dataset **Capstone\_Project** under the project **surveyproject-378222.**

First, it drops the table if it already exists. Then it defines the schema of the new table with three columns - **ID** (an integer column), **Actual** (a float column), and **Predicted** (a float column).

After that, the code loads the data from the Pandas DataFrame **Acu\_vs\_pre** into the **Act\_vs\_pre** table in BigQuery using the **load\_table\_from\_dataframe()** method of the **bigquery** client. The **WRITE\_TRUNCATE** write disposition is used to overwrite any existing data in the table.

Finally, the code reads the data from the **Act\_vs\_pre** table in BigQuery using the **read\_gbq()** method of the **pandas\_gbq** module and stores it in a Pandas DataFrame called **Act\_vs\_pre**.

**Model Building and Evaluation of CNN**

The above code is implementing a Convolutional Neural Network (CNN) model for regression analysis. The code is divided into several parts:

**Data Preprocessing:**

* The input data X is reshaped to include an additional dimension to represent the channel (axis=2) using **np.expand\_dims().**
* The data is split into training and testing sets using **train\_test\_split()** from **sklearn.model\_selection.**

**Model Creation:**

* The model is created using **Sequential()** from **tensorflow.keras.models.**
* The first layer is a 1D convolutional layer with 50 filters, padding='same', kernel\_size=3 and activation function **swish().**
* The second layer is another 1D convolutional layer with 30 filters, padding='same', kernel\_size=3 and activation function **swish().**
* The third layer is a flatten layer to convert the 2D output of the previous layer to a 1D array.
* The fourth layer is a dense layer with 20 units and activation function **swish().**
* The final layer is a dense layer with 1 unit and activation function **'linear'**.

**Model Compilation:**

* The model is compiled using **compile()** with loss function **'mean\_squared\_error'** and optimizer **Adam(learning\_rate=0.001**).

**Model Training:**

* The model is trained using **fit()** with training data **(X\_train and y\_train)** for 200 epochs, batch size of 128, and validation data **(X\_test and y\_test).**

**Model Evaluation:**

* The trained model is evaluated on the test set using **predict()** to obtain predicted values **(y\_pred).**
* Three evaluation metrics are calculated: **Mean Absolute Error (MAE), Root Mean Squared Error (RMSE),** and **R-squared (R2)** using **mean\_absolute\_error(), mean\_squared\_error(),** and **r2\_score()** respectively.

**Visualizing Results:**

* The actual and predicted values are plotted using **matplotlib.pyplot** to visualize the performance of the model on the test set.