Name: Arlene Antony D'costa
Seating Pin: 79
SUID: 594303899

CIS 662 HW5 Report: NN(Classification)

I have selected the dataset **71-80.csv**.

For classification, we need to have labelled data. However, in our dataset, we do not have labelled data and we have continuous values. Hence, as specified in the question I have applied the below logic to convert continuous numerical values to categorical labels.

Also, you can observe that the ratio has been **rounded of to 2 decimals**.

Ratio is obtained between citations of 2022/ citations of 2021. **Binning** strategy is used to distribute the data present in different bins i.e Separate data into different categories.

Thus 3, separate labelled categories can be obtained as follows:

1. Low (<1.05): Values between -np.inf (inclusive) and 1.05 (exclusive) will be labelled as "Low".

2. Medium (1.05-1.15): Values between 1.05 (inclusive) and 1.16 (exclusive) will be labelled as "Medium".

3. High (>1.15): Values greater than or equal to 1.16 will be labelled as "High".

I have used **MinMaxScaler** for **Normalization** for the following reasons:

1. MinMaxScaler scales the data to a specified range, typically [0, 1] or another specified range.
2. Outlier Handling: MinMaxScaler is less sensitive to outliers than StandardScaler. Outliers in the data are scaled into the specified range, which can make the scaled data more robust to extreme values.
3. Scaled data using MinMaxScaler is highly interpretable. For example, if you scale a feature to the range [0, 1], you can easily interpret a value of 0.5 as being halfway between the minimum and maximum values in the original data.
4. Uniformity Across all features based on its own min and max values.

However, after scaling and distributing data into different categories, I observed that data is **highly skewed** towards **Low** category.

Name: Arlene Antony D'costa
Seating Pin: 79
SUID: 594303899

```
scaled_df['category'].value_counts()
✓ 0.0s

Low       58
High      29
Medium    13
Name: category, dtype: int64
```

As required by question, training data and test data is split in the ratio **80:20**

To prevent overfitting and inaccurate results I used **RandomScaler**() library on the training data.

Define the model for NN training

```
# Create a Sequential model
model = Sequential()

# Add a hidden layer with 6 neurons and ReLU activation
model.add(Dense(6, input_dim=6, activation='relu'))

# Add the output layer with 3 neurons (for classification task)
model.add(Dense(3, activation='softmax'))
# Choose an optimizer with the current learning rate
custom_optimizer = Adam(learning_rate=0.001)

✓ 0.3s
```

I used **softmax** activation function as the requirement is **multi-class classification**.
Model is sequential and learning rate is 0.01 and activation function is **relu** for the hidden layers to capture non-linearity in the data.

Input layer = 3 , hidden layer nodes = 6, Output layer nodes = 3

One hot encoding method is used to convert the categorical labels to numerical form in order to convert the labels to appropriate vectors.

Model is evaluated using Accuracy and loss is computed using **categorical_crossentropy.**

I have used **Accuracy, F1_score and ROC** curve metric to evaluate how the model performs and how efficiently it is successful in classifies the data into 3 classes.

Name: Arlene Antony D'costa
Seating Pin: 79
SUID: 594303899

OUTPUT:

Accuracy:

```
    # Evaluate the model on the test data
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print("Test accuracy:", test_accuracy)
    print("Predicted categories:", predicted_categories)
28]  ✓ 0.2s

1/1 [==============================] - 0s 133ms/step - loss: 0.9697 - accuracy: 0.7000
Test accuracy: 0.699999988079071
Predicted categories: ['High' 'Low' 'Low' 'Low' 'High' 'High' 'Low' 'Low' 'Low' 'Low' 'Low'
 'High' 'High' 'Low' 'High' 'Low' 'High' 'Low' 'Low' 'High']
```

Accuracy obtained for the above trained model is **70%.**

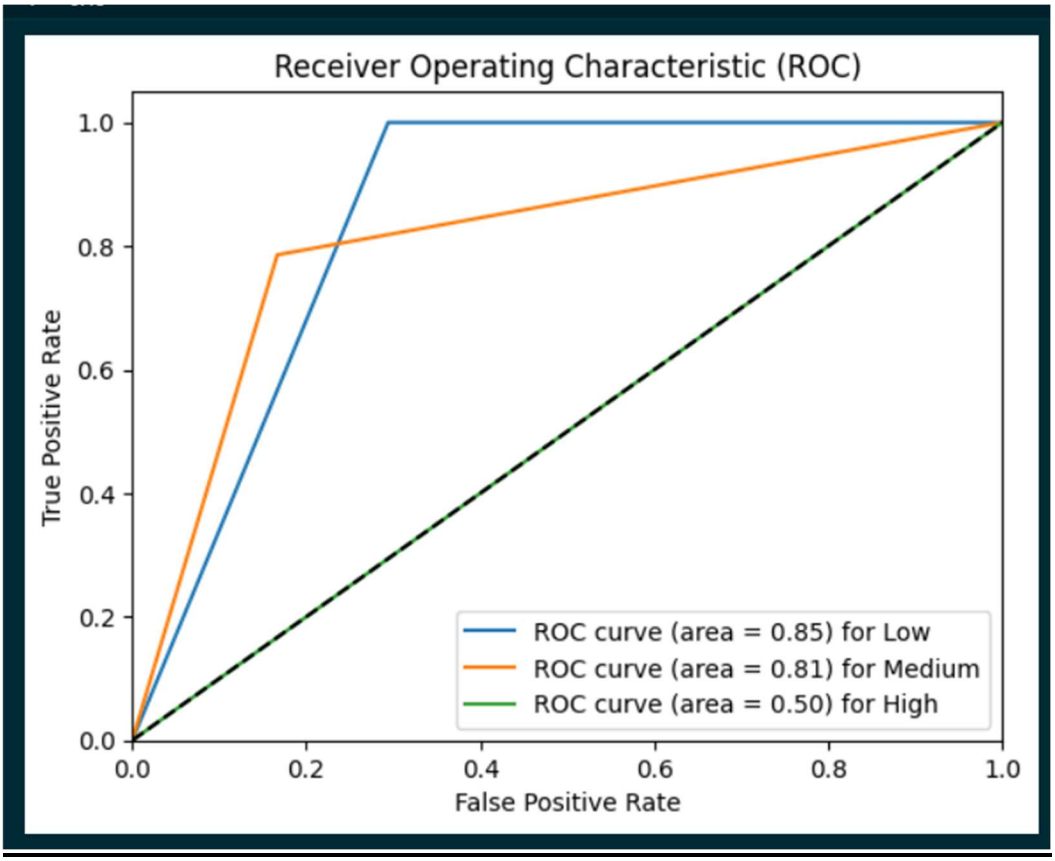
F1_score:

```
Obtaining F1-Score for data

    f1 = f1_score(y_true_categorical, y_pred_categorical, average='weighted')  # You can choose 'micro', 'macro', or 'weighted'
    print(f"F1-Score: {f1:.2f}")
33]  ✓ 0.0s

F1-Score: 0.67
```

F1_score for the above model is **67%**.

Name: Arlene Antony D'costa
Seating Pin: 79
SUID: 594303899

ROC curve:



CONCLUSION:

As mentioned above data is highly biased hence there is a high possibility of overfitting. Accuracy obtained is 70% for the classification tasks performing reasonably well but not exceptionally correctly.

In the cases when there is a skew F1 score gives out better or accurate results, hence F1 score is considered. A 67 % F1 score indicates that my classification model has a relatively good balance between **precision** and **recall**.

Precision (also called positive predictive value) measures the accuracy of positive predictions made by the model.

Precision = TP/TP + FP.

In other words, precision is a measure of how well your model avoids false positive errors.

Recall (also known as true positive rate or sensitivity) quantifies the ability of the model to correctly identify all relevant instances of a class.

Recall = TP/TP + FN

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

An F1 score of **67%** means that my model is achieving a reasonable balance between minimizing false positives and false negatives. It suggests that my model is good at correctly identifying positive cases while maintaining a relatively low number of false positives.

Inorder to better understand the classification results between different classes, I have plotted the values on the **ROC-AUC** curve.

In a ROC-AUC curve, TPR is plotted along the y-axis and FPR is plotted along the x-axis. When the classification line for a particular class is along the central diagonal line i.e AUC value is 0.5 it illustrates that the model has no discrimination capacity between positive and negative class.

However for my classification model, it can be seen that the model performs best **for Low category with the AUC value of 0.85**. The reason for the same can be identified as that the model parameters could be trained accurately as it had sufficient data for the Low category. There is a 85% probability that the model will correctly classify Low for low category. Similarly, model shows AUC value of **0.81** for Medium category which is fairly good. Model shows weak AUC value **of 0.50 for High** category. Reason can be assumed that the model could not be sufficiently trained on the High category as there might be incorrect or noisy data in the "High" category leading to classification issues. Also, the features used for classification might not capture the distinguishing characteristics of the "High" category effectively.