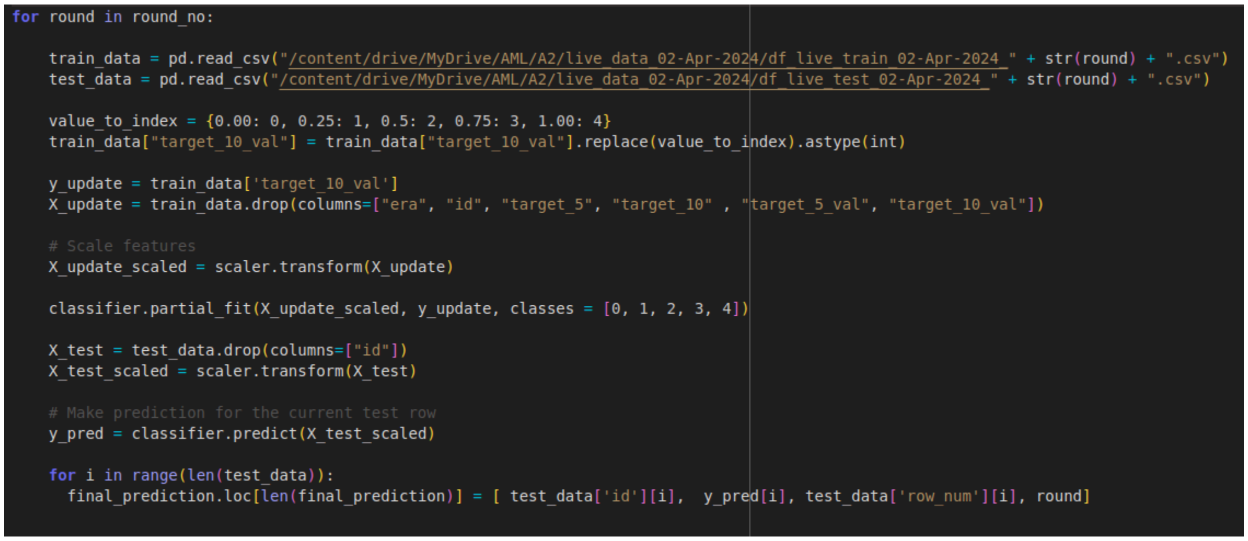
AML ASSIGNMENT - 2
MILESTONE - 2
GROUP - 1

Note:-

We have used the same models that we used in Milestone 1.

1) **First, we trained the model on the given dataset:- df_train_suffled.csv and df_val.csv**
2) **Then, we used continual learning, updated the weights using the train set, and predicted the test sets of the corresponding round.**
3) **Repeated step 2 for all the 5 days.**
4) **Predictions file naming convention:- prediction_dd-mm-yyyy.csv**
5) **Prediction files are saved on GitHub under path:- A2/Milestone2**
6) **A snippet of the Continual learning loop code is pasted below.**



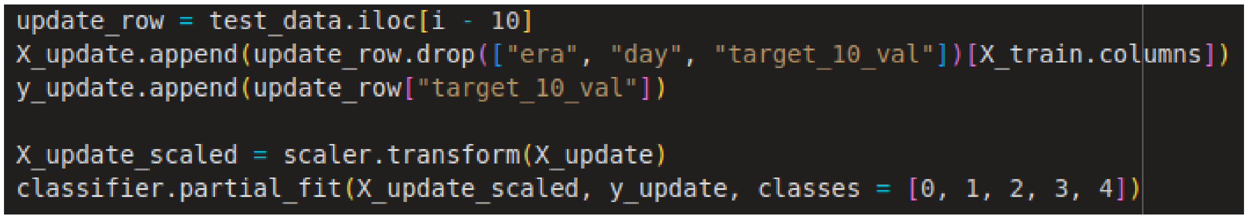
```python
for round in round_no:

    train_data = pd.read_csv("/content/drive/MyDrive/AML/A2/live_data_02-Apr-2024/df_live_train_02-Apr-2024_" + str(round) + ".csv")
    test_data = pd.read_csv("/content/drive/MyDrive/AML/A2/live_data_02-Apr-2024/df_live_test_02-Apr-2024_" + str(round) + ".csv")

    value_to_index = {0.00: 0, 0.25: 1, 0.5: 2, 0.75: 3, 1.00: 4}
    train_data["target_10_val"] = train_data["target_10_val"].replace(value_to_index).astype(int)

    y_update = train_data['target_10_val']
    X_update = train_data.drop(columns=["era", "id", "target_5", "target_10" , "target_5_val", "target_10_val"])

    # Scale features
    X_update_scaled = scaler.transform(X_update)

    classifier.partial_fit(X_update_scaled, y_update, classes = [0, 1, 2, 3, 4])

    X_test = test_data.drop(columns=["id"])
    X_test_scaled = scaler.transform(X_test)

    # Make prediction for the current test row
    y_pred = classifier.predict(X_test_scaled)

    for i in range(len(test_data)):
      final_prediction.loc[len(final_prediction)] = [ test_data['id'][i],  y_pred[i], test_data['row_num'][i], round]
```

Online Learning:-

Online learning, also known as incremental learning or sequential learning, is a machine learning paradigm where the model is trained continuously on streaming data or mini-batches of data. In online learning, the model **updates its parameters incrementally** with each new data point or mini-batch without revisiting past data. This approach is well-suited for scenarios where data arrives sequentially, and the underlying data distribution may change over time **(concept drift)**.

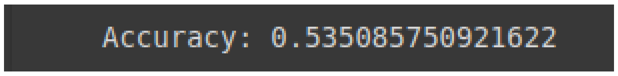
a) **Online learning with liner model (SGDClassifier)**
   First, train the model completely using cf_train.csv.
   We initialize an SGDClassifier with parameters such as loss function, regularization strength (alpha), maximum number of iterations, and random seed. We iteratively update the model parameters using partial_fit, which allows the model to **learn incrementally**. The class parameter specifies the possible class labels in the dataset. Finally, we make predictions using the trained model on test data row-by-row and **update the model**

parameters using the current_row - 10 data point. The prediction method returns the predicted class labels for the input data.



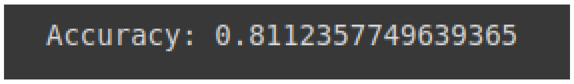
```
update_row = test_data.iloc[i - 10]
X_update.append(update_row.drop(["era", "day", "target_10_val"])[X_train.columns])
y_update.append(update_row["target_10_val"])

X_update_scaled = scaler.transform(X_update)
classifier.partial_fit(X_update_scaled, y_update, classes = [0, 1, 2, 3, 4])
```

Accuracy achieved on noisy data:-



```
Accuracy: 0.535085750921622
```

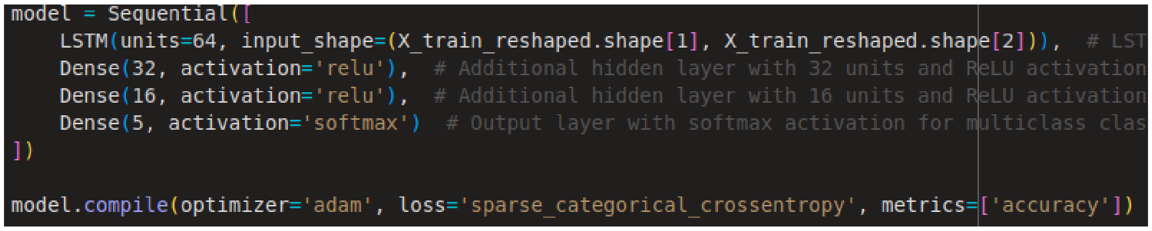
Accuracy achieved without noise data:-



```
Accuracy: 0.8112357749639365
```

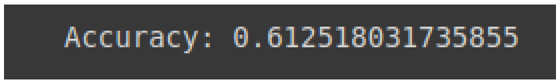
b) Online Learning with Neural Nets and LSTM (RNNs)

Similar to the method described above. We first trained the model on training data and the predicted row-by-row and simultaneously updated the model parameters using the current_row - 10 data point.



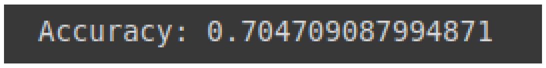
```
model = Sequential([
    LSTM(units=64, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])),  # LST
    Dense(32, activation='relu'),  # Additional hidden layer with 32 units and ReLU activation
    Dense(16, activation='relu'),  # Additional hidden layer with 16 units and ReLU activation
    Dense(5, activation='softmax')  # Output layer with softmax activation for multiclass clas
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Accuracy achieved on noisy data:-



```
Accuracy: 0.612518031735855
```

Accuracy achieved without noisy data:-



```
Accuracy: 0.704709087994871
```

Continual ensemble learning (to handle noisy datasets):-

Continual ensemble learning is a machine learning approach where multiple models, often diverse, are sequentially trained on incoming data streams or tasks, and their predictions are combined to adapt to evolving concepts or distributions over time. Each model in the ensemble learns from new data while maintaining the knowledge learned from previous tasks or data instances. By continuously updating and integrating new models into the ensemble, continual ensemble learning aims to mitigate concept drift, adapt to changing data characteristics, and improve overall predictive performance over time, making it well-suited for dynamic and evolving environments.

```
ensemble_models = []
for _ in range(5):  # Train 5 different models for ensemble
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    ensemble_models.append(model)
```

Firstly, we trained the model on the cf_train.csv dataset. Similar to the method described above. Then, predicted row-by-row and simultaneously updated the model parameters using the current_row - 10 data point **(incremental weight updates).**

Accuracy achieved on noisy dataset:-

```
Accuracy: 0.38174174174174175
```

Accuracy achieved without noisy data:-

```
Accuracy: 0.6843343343343343
```