MLOps Project: Predictive Modeling for Student Final Scores based on Initial Assessments

ASSIGNMENT # 02

Ahmad Makki

MSDSF23M022

Problem Statement

Develop a predictive solution to anticipate students' final scores in a course based on their performance in a sequence of initial assessment activities. Given historical data featuring consistent assessment order and grading scheme, the goal is to build models that accurately forecast final scores for current iterations of the course, starting prediction after the 5th activity and extending to the final assessment.

This project addresses the need for early identification of students' academic progress and facilitates targeted interventions to enhance learning outcomes.

Dataset Features:

Sr#	Roll No.	Name	Q1	Q2	A1	Q3	Q4	Midterm	Q5	A2	Q6	Q7	Q8	Final	Total
Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature	Feature

ICT Dataset

Data Pre Processing:

In this part, pre-processes data for a machine learning task:

1. It loads historical and test data from Excel files, filling missing values with zeros.
2. It defines features and target variables, then calculates weighted features based on specific weights and marks.
3. Finally, it computes the total score for both training and testing data using the weighted featues.

es.

In [1]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV
from openpyxl import Workbook
import warnings
warnings.filterwarnings("ignore")

# Load historical data
data = pd.read_excel('ICT_Result_Data_Morning.xlsx')
test_data = pd.read_excel('ICT_Result_Data_Evening.xlsx')

data.fillna(0, inplace=True)
test_data.fillna(0, inplace=True)

# Define features and target
features = ['Q1', 'Q2', 'A1', 'Q3', 'Q4', 'Midterm', 'Q5', 'A2', 'Q6', 'Q7', 'Q8', 'Final', 'Total']
weights = [2.625, 2.625, 2, 2.625, 2.625, 35, 2.625, 2, 2.625, 2.625, 2.625, 40, 100]
marks = [30, 49, 100, 30, 15, 35, 45, 100, 32, 24, 40, 40, 100]

# Calculate weighted features for Training
weighted_features = data[features[:-1]].multiply(weights[:-1]).divide(marks[:-1]).round()
data['Total'] = weighted_features.sum(axis=1)
target = data['Total']

# Calculate weighted features for Testing data
test_weighted_features = test_data[features[:-1]].multiply(weights[:-1]).divide(marks[:-1]).round()
test_data['Total'] = test_weighted_features.sum(axis=1)
test_target = test_data['Total']
```

Data Modeling and Evaluation:

- Defines a function `calculate_mse(predictions, actual)` to compute Mean Squared Error.
- Initializes a Random Forest Regressor model with a specified random state.
- Sets up lists to store MSE for each step of feature selection.
- Iterates through each feature to predict the total score.
  - Selects features up to the current iteration for training and testing.
  - Splits the data into training and validation sets.
  - Trains the model on the training data.
  - Computes predictions for training, validation, and testing data.
  - Calculates MSE for each set and stores the values.
  - Prints MSE for training, validation, and testing data after each feature addition.

In [2]:
```python
# # Models to train with their respective hyperparameters to tune
# models = {
#     "Ridge": (Ridge(), {'alpha': [0.0001, 0.001, 0.01, 0.1]}),
#     "Lasso": (Lasso(), {'alpha': [0.0001, 0.001, 0.01, 0.1]}),
#     "MLPRegressor": (MLPRegressor(), {'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)], 'activation': ['relu', 'tanh','logistic']}),
#     "Linear Regression": (LinearRegression(), {}),
#     "Random Forest": (RandomForestRegressor(), {'n_estimators': [10, 50, 100, 200], 'max_depth': [None, 10, 20]})
# }

models = {
    "Ridge": (Ridge(), {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}),
    "Lasso": (Lasso(), {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}),
    "MLPRegressor": (MLPRegressor(), {'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
                                  'activation': ['relu', 'tanh', 'logistic'],
                                  'solver': ['adam', 'lbfgs'],
                                  'alpha': [0.0001, 0.001, 0.01],
                                  'learning_rate': ['constant', 'adaptive']}),
    "Linear Regression": (LinearRegression(), {}),
    "Random Forest": (RandomForestRegressor(), {'n_estimators': [10, 50, 100, 200],
                                            'max_depth': [None, 10, 20, 30],
                                            'min_samples_split': [2, 5, 10],
                                            'min_samples_leaf': [1, 2, 4]})
}

results = []
mses = []
model_names = []
min_mse = float('inf')
topka_model = None
topki_pred = None
all_abs_errors = []
errors = []
result = []

result_df = pd.DataFrame(test_data['Roll No.'], columns=['Roll No.'])
result_df['Actual Total'] = test_data['Total']
errors_df = pd.DataFrame(test_data['Roll No.'], columns=['Roll No.'])


# Train and evaluate models
for model_name, (model, param_grid) in models.items():
    for i, feature in enumerate(features[5:-1]):  # Exclude 'Total'
        # Train-test split
        X_train, X_val, y_train, y_val = train_test_split(weighted_features[[feature]], target, test_size=0.2, random_state=42)

        # Grid Search for hyperparameter tuning
        grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
        grid_search.fit(X_train, y_train)

        # Get best model and predict
        best_model = grid_search.best_estimator_
        train_pred = best_model.predict(X_train)
        val_pred = best_model.predict(X_val)
        test_pred = best_model.predict(test_weighted_features[[feature]])

        # Calculate MSE
        train_mse = mean_squared_error(y_train, train_pred)
        val_mse = mean_squared_error(y_val, val_pred)
        test_mse = mean_squared_error(test_target, test_pred)

        # Store results
        results.append({'Model': model_name, 'Feature': feature, 'Train MSE': train_mse, 'Validation MSE': val_mse, 'Test MSE': test_mse})

        mses.append(test_mse)
        model_names.append(model_name)

        if test_mse < min_mse:
            min_mse = test_mse
            topka_model = model_name
            topki_pred = test_pred

        predicted_column_name = f'Predicted Marks {feature}'
        result_df[predicted_column_name] = test_pred
        abs_errors = abs(test_target - test_pred)
        all_abs_errors.append(abs_errors)
```

Score Prediction and Error Analysis:

This segment loops through each student to compute prediction errors:

- It iterates over unique student IDs to isolate data for each student.
- Extracts features and targets for training and testing.
- Trains the model and predicts total scores for the test data.
- Calculates absolute errors and stores statistics including average, minimum, and maximum errors.
- Converts the error statistics into a DataFrame and saves the results to an Excel sheet for further analysis.

In [3]:
```python
# # Create the errors dataframe for CSV output
# errors_df['Avg Absolute Error'] = pd.concat(all_abs_errors, axis=1).mean(axis=1)
# errors_df['Min Absolute Error'] = pd.concat(all_abs_errors, axis=1).min(axis=1)
# errors_df['Max Absolute Error'] = pd.concat(all_abs_errors, axis=1).max(axis=1)

# errors.append({'Avg Absolute Error': errors_df['Avg Absolute Error'], 'Min Absolute Error': errors_df['Min Absolute Error'], 'Max Absolute Error': errors_df['Max Absolute Error']})

# Create the errors dataframe for CSV output
errors_df['Avg Absolute Error'] = pd.concat(all_abs_errors, axis=1).mean(axis=1)
errors_df['Min Absolute Error'] = pd.concat(all_abs_errors, axis=1).min(axis=1)
errors_df['Max Absolute Error'] = pd.concat(all_abs_errors, axis=1).max(axis=1)

errors_df = errors_df[['Avg Absolute Error', 'Min Absolute Error', 'Max Absolute Error']]

# Transpose errors_df before appending
errors.append({'Avg Absolute Error': errors_df['Avg Absolute Error'],
               'Min Absolute Error': errors_df['Min Absolute Error'],
               'Max Absolute Error': errors_df['Max Absolute Error']})

# Convert results to DataFrame
results_df = pd.DataFrame(results)
errors_df = pd.DataFrame(errors)

# Save results to Excel
results_df.to_excel('ICT_Model_Results.xlsx', index=False)
errors_df.to_excel('ICT_Abs_Error.xlsx', index=False)
result_df.to_excel('ICT_Marks_Pred.xlsx', index=False)
```

Summary:

This code workflow encompasses comprehensive steps in preparing, modeling, and evaluating data for predicting student scores. Initially, it conducts data preprocessing by loading historical and test datasets, filling missing values, and computing weighted features for both training and testing data. Following this, it employs a Random Forest Regressor model to predict total scores, iterating through feature selections and evaluating model performance using Mean Squared Error (MSE). Moreover, it extends the analysis to individual students, predicting their scores and analyzing prediction errors, which are then aggregated and saved for further examination. This structured approach ensures thorough data processing, effective model training, and insightful error analysis, facilitating informed decision-making in educational assessment and intervention strategies.

CC Dataset

Data Preprocessing:

This code segment illustrates the data preprocessing steps for a machine learning task:

- It imports necessary libraries such as pandas for data manipulation and sklearn for machine learning algorithms.
- Historical and test datasets are loaded from Excel files, with missing values replaced by zeros.
- Features and target variables are defined based on specific weights and marks, reflecting their importance in the assessment.
- Weighted features are calculated for both training and testing data to account for varying importance levels.
- These weighted features are used to compute the total score, serving as the target variable for predictive modeling.

In [6]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from scipy.stats import randint


# Load historical data
data = pd.read_excel('CC_Data_Morning.xlsx')
test_data=pd.read_excel('CC_Data_Evening.xlsx')


data.fillna(0, inplace=True)
test_data.fillna(0, inplace=True)


# Define features and target
features = ['A1', 'Q1', 'A2', 'Q2', 'A3', 'A4', 'Q3', 'Mid', 'AWS Labs', 'Q4', 'A5', 'Q5', 'A6', 'Final']
weights = [1, 1.5, 1, 1.5, 1, 4, 1.5, 35, 3, 1.25, 4, 1.25, 4, 40]
marks = [10, 21, 10, 30, 100, 10, 41, 35, 10, 40, 100, 20, 100, 40]

# Calculate weighted features for Training
weighted_features = data[features].multiply(weights).divide(marks).round()
data['Total'] = weighted_features.sum(axis=1)
target = data['Total']

# Calculate weighted features for Testing data
test_weighted_features = test_data[features].multiply(weights).divide(marks).round()
test_data['Total'] = test_weighted_features.sum(axis=1)
test_target = test_data['Total']
```

Model Evaluation:

- Defines a function to calculate Mean Squared Error (MSE) for model evaluation.
- Initializes a Random Forest Regressor model with a specified random state.
- Iterates through features to predict the total score, splitting the data, training the model, and computing MSE for training, validation, and testing datasets.

In [7]:
```python
models = {
    "Ridge": (Ridge(), {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}),
    "Lasso": (Lasso(), {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}),
    "MLPRegressor": (MLPRegressor(), {'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
                                  'activation': ['relu', 'tanh', 'logistic'],
                                  'solver': ['adam', 'lbfgs'],
                                  'alpha': [0.0001, 0.001, 0.01],
                                  'learning_rate': ['constant', 'adaptive']}),
    "Linear Regression": (LinearRegression(), {}),
    "Random Forest": (RandomForestRegressor(), {'n_estimators': [10, 50, 100, 200],
                                            'max_depth': [None, 10, 20, 30],
                                            'min_samples_split': [2, 5, 10],
                                            'min_samples_leaf': [1, 2, 4]})
}

results = []
mses = []
model_names = []
min_mse = float('inf')
topka_model = None
topki_pred = None
all_abs_errors = []
errors = []
result = []

result_df = pd.DataFrame(test_data['Roll No.'], columns=['Roll No.'])
result_df['Actual Total'] = test_data['Total']
errors_df = pd.DataFrame(test_data['Roll No.'], columns=['Roll No.'])


# Train and evaluate models
for model_name, (model, param_grid) in models.items():
    for i, feature in enumerate(features[5:-1]):  # Exclude 'Total'
        # Train-test split
        X_train, X_val, y_train, y_val = train_test_split(weighted_features[[feature]], target, test_size=0.2, random_state=42)

        # Grid Search for hyperparameter tuning
        grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
        grid_search.fit(X_train, y_train)

        # Get best model and predict
        best_model = grid_search.best_estimator_
        train_pred = best_model.predict(X_train)
        val_pred = best_model.predict(X_val)
        test_pred = best_model.predict(test_weighted_features[[feature]])

        # Calculate MSE
        train_mse = mean_squared_error(y_train, train_pred)
        val_mse = mean_squared_error(y_val, val_pred)
        test_mse = mean_squared_error(test_target, test_pred)

        # Store results
        results.append({'Model': model_name, 'Feature': feature, 'Train MSE': train_mse, 'Validation MSE': val_mse, 'Test MSE': test_mse})

        mses.append(test_mse)
        model_names.append(model_name)

        if test_mse < min_mse:
            min_mse = test_mse
            topka_model = model_name
            topki_pred = test_pred

        predicted_column_name = f'Predicted Marks {feature}'
        result_df[predicted_column_name] = test_pred
        abs_errors = abs(test_target - test_pred)
        all_abs_errors.append(abs_errors)
```

Student Error Analysis:

- Initializes a dictionary to store error statistics for each student including average, minimum, and maximum absolute errors.
- Iterates through each student to calculate prediction errors using the trained model.
- Computes absolute errors, aggregates statistics, and stores them in the error dictionary.
- Converts the error dictionary into a DataFrame and saves the results to an Excel sheet for further analysis.

In [8]:
```python
# Create the errors dataframe for CSV output
errors_df['Avg Absolute Error'] = pd.concat(all_abs_errors, axis=1).mean(axis=1)
errors_df['Min Absolute Error'] = pd.concat(all_abs_errors, axis=1).min(axis=1)
errors_df['Max Absolute Error'] = pd.concat(all_abs_errors, axis=1).max(axis=1)

errors_df = errors_df[['Avg Absolute Error', 'Min Absolute Error', 'Max Absolute Error']]

# Transpose errors_df before appending
errors.append({'Avg Absolute Error': errors_df['Avg Absolute Error'],
               'Min Absolute Error': errors_df['Min Absolute Error'],
               'Max Absolute Error': errors_df['Max Absolute Error']})

# Convert results to DataFrame
results_df = pd.DataFrame(results)
errors_df = pd.DataFrame(errors)

# Save results to Excel
results_df.to_excel('CC_Model_Results.xlsx', index=False)
errors_df.to_excel('CC_Abs_Error.xlsx', index=False)
result_df.to_excel('CC_Marks_Pred.xlsx', index=False)
```

Summary

The code encapsulates a thorough data processing and predictive modeling pipeline. It begins by setting up the necessary data structures and importing libraries for analysis, including pandas for data manipulation and scikit-learn for machine learning tasks. Historical and test datasets are loaded and preprocessed, with missing values replaced by zeros and features weighted based on their importance.