Marwadi University Marwadi Chandarana Group NAAC A+	Marwadi University Faculty of Engineering and Technology Department of Information and Communication Technology	
Subject: ML (01CT1519)	AIM: Implement Regression as Classification problem.	
Lab Experiment No: 01	Date: 15/08/2025	Enrollment No: 92301733057

Implement Regression as Classification problem. Explain the process using a dummy example by taking 3 features, 3 classes and N observations.

What changes should be done in the code of Multi-Variable Linear Regression?

Theory : Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. In simpler terms, It finds the **best fit** straight line that represents the relationship between variables. Linear regression is typically used when the dependent variable is continuous (e.g., predicting house prices, temperature, etc.). When used for classification, linear regression is adapted to predict a binary (0 or 1) outcome.

Code:

```
import numpy as np

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from collections import Counter



# Generate dummy data

np.random.seed(42)
N = 200
data = pd.DataFrame({
'Hours_Studied': np.random.uniform(1, 15, N),
'Assignments_Completed': np.random.randint(0, 10, N),
'Previous_Exam_Score': np.random.uniform(40, 100, N)
})



data['Final_Exam_Score'] = np.clip((

data['Hours_Studied'] * 3 +
data['Assignments_Completed'] * 2 +
data['Previous_Exam_Score'] * 0.4 +
np.random.normal(0, 5, N)
), 0, 100)


#It defines four distinct score ranges (bins) and corresponding class labels (0 for 'F', 1 for 'C', etc.).


bins = [-1, 59, 74, 89, 101]

labels = [0, 1, 2, 3] # 0=F, 1=C, 2=B, 3=A
data['Grade_Class'] = pd.cut(data['Final_Exam_Score'], bins=bins,
labels=labels).astype(int)



X = data[['Hours_Studied', 'Assignments_Completed', 'Previous_Exam_Score']]

y = data['Grade_Class']



X_train, X_test, y_train, y_test = train_test_split(

X, y, test_size=0.25, random_state=42, stratify=y
)


# Convert to NumPy arrays for our calculations
```

```
X_train_np = X_train.values
y_train_np = y_train.values
X_test_np = X_test.values
```

#For a single new student, it calculates its distance to all students in the training data, finds the k closest ones (the "neighbors"), and returns the most common grade among them.

```
def calculate_distance(point1, point2):

"""Calculates the straight-line (Euclidean) distance between two points."""
return np.sqrt(np.sum((point1 - point2)**2))

def predict_by_neighbors(X_train, y_train, x_new, k=5):
"""Makes a prediction for a single new data point."""
# 1. Create a list to store distances from the new point to all training points
distances = []
for train_point, train_label in zip(X_train, y_train):
dist = calculate_distance(x_new, train_point)
distances.append((dist, train_label))
# 2. Sort the list by distance (from smallest to largest)
distances.sort(key=lambda item: item[0])
# 3. Get the labels of the top 'k' closest neighbors
neighbors = distances[:k]
neighbor_labels = [label for dist, label in neighbors]
# 4. Find the most common label (the majority vote)
most_common = Counter(neighbor_labels).most_common(1)
return most_common[0][0]
```

```
# Create a list to store all our predictions for the test set

all_predictions = []
```

#It iterates through every student in the test set (X_test_np) and uses the predict_by_neighbors function to generate a grade prediction for each one.

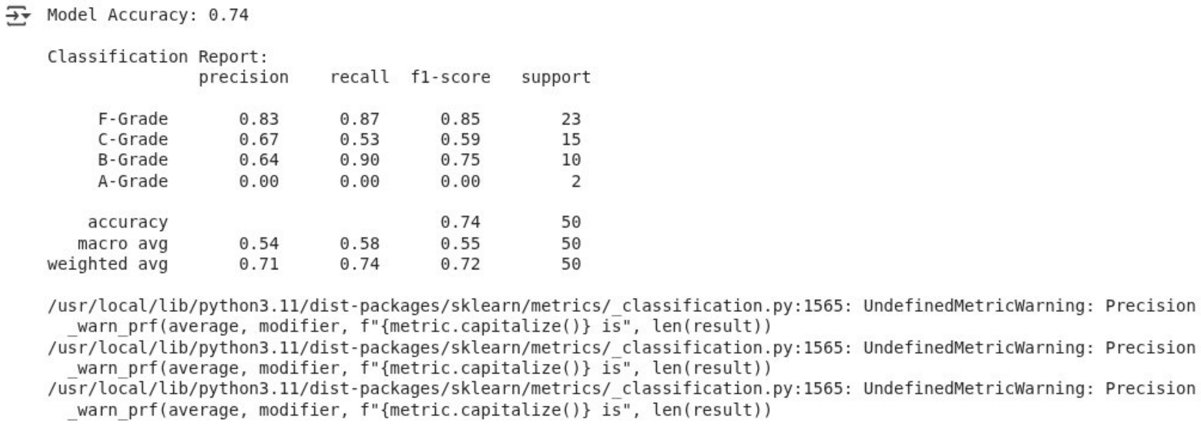
It then compares these all_predictions to the true grades (y_test) and calculates standard classification metrics “Accuracy and a Classification Report” to measure how well the model performed.

```
# Loop through every student in our test data and make a prediction

for test_student_features in X_test_np:
prediction = predict_by_neighbors(X_train_np, y_train_np, test_student_features, 
k=5)
all_predictions.append(prediction)
# --- EVALUATE THE MODEL ---

class_names = ['F-Grade', 'C-Grade', 'B-Grade', 'A-Grade']
print("Model Accuracy:", accuracy_score(y_test, all_predictions))
print("\nClassification Report:\n", classification_report(y_test, all_predictions, 
target_names=class_names))
```

Output:



```
Model Accuracy: 0.74

Classification Report:
               precision    recall  f1-score   support

     F-Grade       0.83      0.87      0.85        23
     C-Grade       0.67      0.53      0.59        15
     B-Grade       0.64      0.90      0.75        10
     A-Grade       0.00      0.00      0.00         2

    accuracy                           0.74        50
   macro avg       0.54      0.58      0.55        50
weighted avg       0.71      0.74      0.72        50

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

--

What changes should be done in the code of Multi-Variable Linear Regression?

To convert a standard Multi-Variable Linear Regression script to solve this problem as a classification task, these three changes can be done:

1. **Transform the Target Variable (y):** The most important change is to stop using the continuous target variable (e.g., Final_Exam_Score). First **discretize** it into a finite number of classes (e.g., Grade_Class with values 0, 1, 2, 3), as done in the script with pd.cut. The model's goal is no longer to predict the exact score but to predict which category the score falls into.
2. **Change the Model Algorithm:** Replace the LinearRegression model with a **classification algorithm**. A linear regression model is designed to output a continuous number by fitting a line, not a discrete class label. Other common choices would be LogisticRegression, DecisionTreeClassifier, or RandomForestClassifier from scikit-learn.
3. **Use Classification Metrics for Evaluation:** Regression metrics like Mean Squared Error (MSE) or R-squared (R2) are no longer applicable. Switch to **classification metrics** to evaluate the model.

Conclusion

The model was able to predict the correct grade a good percentage of the time.

If the target is categorical, we can use a proper classification algorithm such as Logistic Regression, a Decision Tree, or KNN to build the model.

The technique was the **discretization** of the continuous target variable (Final_Exam_Score) into four distinct classes ('F', 'C', 'B', 'A'). This method gives us a straightforward way to get that answer.