Name: Shrey Pendurkar
Class: D15C
Batch: B
Roll No: 38

MLDL - Experiment 1

Aim

Implement Linear and Logistic Regression on real-world datasets.

Dataset Source

- Dataset: Weight-Height Dataset
- Kaggle: https://www.kaggle.com/datasets/mustafaali96/weight-height

---

Dataset Description

The Weight-Height dataset is a widely used bivariate dataset designed for introductory regression tasks in machine learning. It contains 10,000 records describing individuals with three attributes: Gender, Height (in inches), and Weight (in pounds). The relationship between height and weight in this dataset is almost perfectly linear with very little noise, making it ideal for demonstrating and validating Simple Linear Regression models. The data is provided as a CSV (Comma Separated Values) file named `weight-height.csv`, which includes one categorical column (Gender) and two numerical columns (Height and Weight) suitable for modeling and visualization.

Theory

Regression analysis studies how a target (dependent) variable changes in response to one or more predictor (independent) variables. When the target is continuous (such as weight, salary, or temperature), regression models are appropriate. Simple Linear Regression (SLR) focuses on a single predictor and assumes a linear relationship between the input and the output.

The general form is:

$$y = mx + c$$

In machine learning notation:

$$y = \beta_0 + \beta_1 x$$

Where:

- y: Output (Weight).
- x: Input (Height).
- β_0: Intercept.
- β_1: Slope (coefficient).

The goal of Linear Regression is to find the best-fit line that minimizes the error between predicted and actual values. Using Ordinary Least Squares (OLS), the optimal parameters are:

$$\boxed{\begin{gathered}\beta_1 = \frac{n\sum xy - (\sum x)(\sum y)}{n\sum x^2 - (\sum x)^2} \\ \beta_0 = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n\sum x^2 - (\sum x)^2}\end{gathered}}$$

OLS computes these in a single closed-form step by minimizing the sum of squared residuals.

Limitations

Key limitations of Simple Linear Regression (same structure as in your reference):

1. Assumption of Linearity
   It enforces a straight-line relationship and assumes a constant change in weight for each unit increase in height. If the true relationship is curved or non-linear, the model underfits.
2. Sensitivity to Outliers
   Because OLS squares errors, extreme outliers can heavily influence the

regression line, altering slope and intercept to fit a few anomalous points at the expense of overall accuracy.

3. Heteroscedasticity
   SLR assumes constant variance of errors (homoscedasticity). If variance grows with height (wider spread at taller heights), this assumption is violated and statistical inferences become unreliable.
4. Extrapolation Risk
   The linear relationship is valid only within the observed range of heights. Predictions far outside the data range can be unrealistic (e.g., negative weights).
5. Independence of Observations
   The method assumes each sample is independent. In time-series or grouped settings, autocorrelation violates this, causing underestimated standard errors.

Workflow – Linear Regression

1. Data Collection
   - Load `weight-height.csv` using Pandas into a DataFrame containing `Gender`, `Height`, and `Weight`.
2. Data Preprocessing
   - Select Height as the input feature `X` and Weight as the target `y`.
   - Reshape `Height` to a 2D array as required by scikit-learn.
   - No missing-value handling is needed since the dataset is clean.
3. Train–Test Split
   - Split into 80% training and 20% testing using `train_test_split`.
   - The training set is used to learn the height–weight relationship; the test set evaluates generalization.
4. Model Training
   - Fit a `LinearRegression` model on the training data so it learns the slope and intercept that minimize squared error.
5. Prediction
   - Use the trained model to predict weight values for the test heights.
6. Model Evaluation
   - Compute R^2 (coefficient of determination) to measure how well predictions match actual weights.
   - Print the learned coefficient (slope) and intercept, which form the final regression equation.
7. Data Visualization

    - Plot a scatter diagram of actual test data points (Height vs Weight) and overlay the fitted regression line.
    - This visually shows how closely the line follows the data.
8. Conclusion
    - Conclude that Simple Linear Regression successfully models the near-linear relationship between height and weight and that the high R^2 and well-aligned line indicate a good fit.

Performance Analysis

The performance of the implemented Simple Linear Regression model on the Weight–Height dataset is evaluated using the R^2 score together with the learned slope and intercept of the regression line.

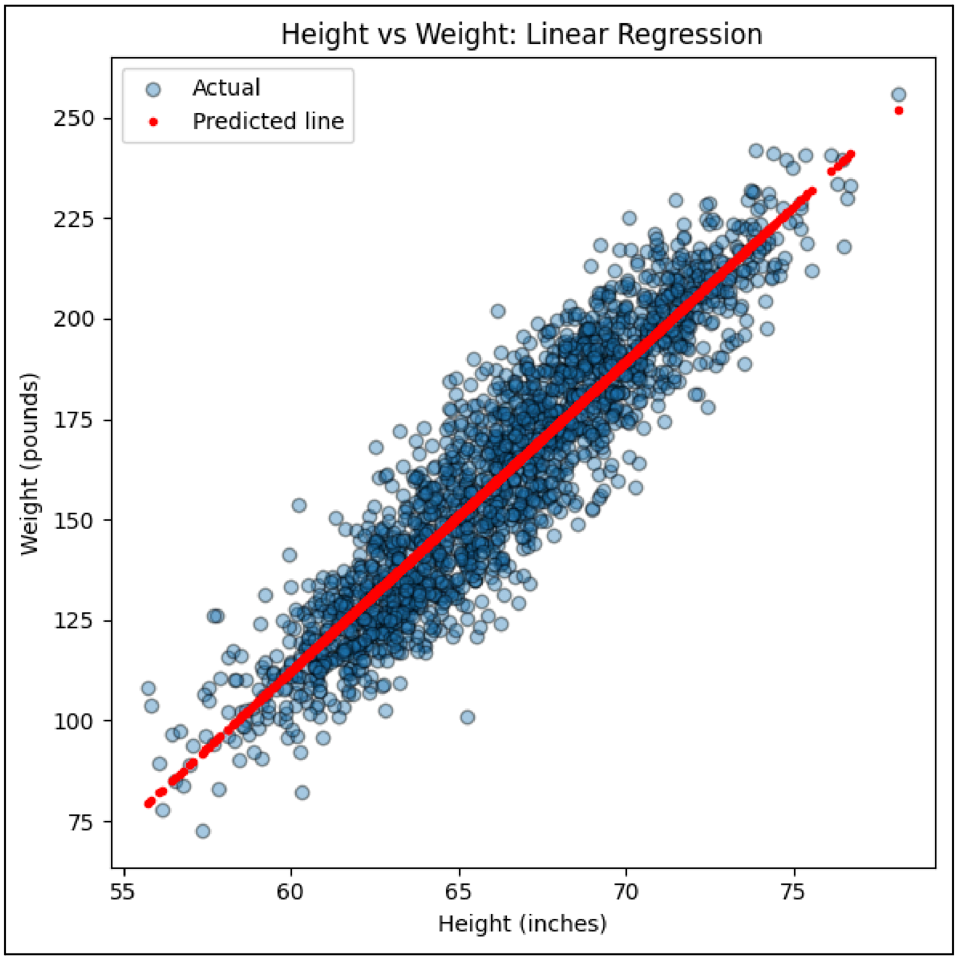
1. R^2 Score (Coefficient of Determination)
   The obtained $R2=0.8577$ indicates that approximately 85.77% of the variation in weight can be explained by a person's height. This shows that height is a very strong predictor of weight in this dataset and that the linear model captures most of the underlying relationship. The remaining 14.23% of the variation is likely due to factors not included in the model, such as body composition, age, gender differences, and lifestyle, as well as natural biological variability. Overall, the model demonstrates a good fit and strong predictive performance for this simple setting.
2. Interpretation of the Slope (Coefficient)
   The learned slope is 7.7022, meaning that for every 1-inch increase in height, the model predicts an average increase of about 7.7 pounds in weight. This confirms a clear positive linear relationship: taller individuals are expected to weigh more, and the magnitude of the slope reflects a relatively steep dependence of weight on height.
3. Interpretation of the Intercept
   The intercept is −349.7878, which is the predicted weight when height is mathematically set to zero inches. Although this value has no physical meaning (no person has 0 inches height), it is necessary for defining the regression line and positioning it correctly relative to the observed data points. The intercept should therefore be interpreted as a model artifact rather than a realistic weight prediction.

Hyperparameter Tuning

Linear Regression here has no tunable hyperparameters; OLS directly computes the unique best-fit line, so hyperparameter tuning is not applicable.

Code and Output - Linear Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import r2_score

# 1. Data Collection
df =
pd.read_csv('weight-height.csv')

print("First 5 rows:")
print(df.head())
print("\nShape:", df.shape)

# 2. Data Preprocessing
X = df[['Height']].values   # 2D
array
y = df['Weight'].values

# 3. Train–Test Split (80/20)
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42
)

# 4. Model Training
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# 5. Prediction
y_pred = lin_reg.predict(X_test)

# 6. Evaluation
r2 = r2_score(y_test, y_pred)
slope = lin_reg.coef_[0]
intercept = lin_reg.intercept_

print("\n--- Linear Regression
Results ---")
print(f"R²:        {r2:.4f}")
print(f"Slope:     {slope:.4f}")
print(f"Intercept: {intercept:.4f}")

# 7. Visualization
plt.figure(figsize=(6, 6))
plt.scatter(X_test, y_test,
alpha=0.4, label='Actual',
edgecolor='k')
plt.plot(X_test, y_pred, 'r.',
label='Predicted line')
plt.xlabel('Height (inches)')
plt.ylabel('Weight (pounds)')
plt.title('Height vs Weight: Linear
Regression')
plt.legend()
plt.tight_layout()
plt.show()
```



Logistic Regression

Dataset Source

- Dataset: Advertising Dataset
- Kaggle: https://www.kaggle.com/datasets/gabrielsantello/advertisement-click-on-ad

---

Dataset Description

The Advertising dataset is a multivariate dataset commonly used for binary classification tasks such as Logistic Regression. It contains 1,000 observations describing web users' demographic characteristics and online browsing behavior, with the objective of predicting whether a user will click on an online advertisement.

The dataset includes a mix of numerical and categorical variables. Important numerical features are:

- Daily Time Spent on Site (minutes)
- Age (years)
- Area Income (average income of the user's geographical area)
- Daily Internet Usage (minutes)
- Male (binary indicator: 1 for male, 0 for female)

Theory

Logistic Regression is a supervised learning algorithm used for classification, especially binary classification (e.g., Click / No Click). Unlike Linear Regression, which predicts continuous values, Logistic Regression predicts a probability between 0 and 1 that an input belongs to the positive class.

The model first computes a linear combination:

$$\boxed{z = b_0 + b_1 x}$$

Then applies the Sigmoid (logistic) function:

$$\boxed{y = \frac{1}{1 + e^{-z}}}$$

Where:

- y: Estimated probability that the user clicked the ad.
- z: Linear score from input features.

Properties:

- Large positive
- $z \to y \approx 1.$

- Large negative $z \rightarrow y \approx 0$.
- $z = 0 \rightarrow y = 0.5$, the threshold of maximum uncertainty.

The S-shaped sigmoid curve maps any real input to $(0,1)$, making it ideal for probabilities.

Limitations

Key limitations (same structure as your PDF):

1. Linear Decision Boundary
   Logistic Regression assumes that classes can be separated by a linear hyperplane. It struggles with complex non-linear class boundaries (e.g., circular or spiral separation).
2. Sensitivity to Multicollinearity
   Highly correlated features (e.g., salary in two currencies) lead to unstable coefficients, making the importance of individual predictors hard to interpret, even if predictive accuracy remains high.
3. Impact of Outliers
   Extreme points can strongly influence the decision boundary as the model tries to reduce their loss, potentially reducing accuracy on normal data.
4. Need for Adequate Sample Size
   Parameters are estimated by Maximum Likelihood Estimation. With very small datasets, coefficient estimates can be biased or unstable; a common rule is at least ~10 observations per predictor.
5. Class Imbalance
   When one class dominates (e.g., 99% no-click), the model may simply predict the majority class and still achieve high accuracy, masking poor performance on the minority class.

Workflow – Logistic Regression

1. Data Collection
   - Load `advertising.csv` into a Pandas DataFrame and inspect shape and sample rows.
2. Data Preprocessing
   - Drop text and non-numeric columns: `Ad Topic Line`, `City`, `Country`, `Timestamp`.

- Keep numeric predictors: `Daily Time Spent on Site`, `Age`, `Area Income`, `Daily Internet Usage`, `Male`.
- Set `Clicked on Ad` as the target variable `y`.

3. Splitting the Dataset
   - Split into 70% training and 30% testing sets using `train_test_split` with `random_state=101` to keep results reproducible.
4. Hyperparameter Tuning & Model Training
   - Define a parameter grid over `C` (0.001 to 1000, log-spaced) and `penalty` (`l1`, `l2`) using the `liblinear` solver.
   - Use GridSearchCV with 5-fold cross-validation to train and validate 70 model configurations and select the best estimator.
5. Prediction
   - Use the best model to predict class labels on `X_test`. The model outputs probabilities and then assigns class 0/1 based on a threshold (default 0.5).
6. Model Evaluation
   - Compute Accuracy on the test set.
   - Generate a Classification Report (precision, recall, F1-score) for both classes.
   - Compute and visualize a Confusion Matrix (TP, TN, FP, FN).
7. Data Visualization
   - Plot the confusion matrix as a Seaborn heatmap to visually inspect where the model makes mistakes (missed clicks vs false alarms).
8. Conclusion
   - Summarize that an optimally tuned Logistic Regression model can reliably predict whether a user will click an ad based on browsing and demographic features.

Performance Analysis

The performance of the Logistic Regression model on the Advertising dataset is evaluated using the Accuracy Score, Classification Report, and Confusion Matrix.

1. Accuracy Score
   The model achieves an accuracy of 97.67% on the test set (293 correct predictions out of 300 samples). This indicates that the selected features—such as Daily Time Spent on Site, Age, Area Income, Daily Internet Usage, and the Male indicator—are highly informative for predicting whether a user will click on an

advertisement. The small error rate (about 2.33%) shows that the classifier generalizes very well to unseen data.

2. Confusion Matrix Analysis
   The confusion matrix is:
3. [155 2]
4. [5 138]
   - True Negatives (155): The model correctly predicted “Did not Click” (0) for 155 users who actually did not click the ad.
   - False Positives (2): Only 2 users were incorrectly predicted as “Click” (1) when they actually did not click, representing a very low false alarm rate.
   - False Negatives (5): The model missed 5 users who actually clicked (1) but were predicted as “No Click” (0), indicating a small number of missed opportunities.
   - True Positives (138): The model correctly predicted “Click” for 138 actual clickers.
5. The dominance of the diagonal entries (155 and 138) and the very small off-diagonal entries (2 and 5) demonstrate that the model distinguishes the two classes with high reliability.
6. Classification Report
   For the negative class (0 – Did not Click):
   - Precision: 0.97 → When the model predicts “No Click”, it is correct 97% of the time.
   - Recall: 0.99 → It correctly identifies 99% of all actual non-clickers.
   - F1-score: 0.98 → Strong balance between precision and recall for the majority class.
7. For the positive class (1 – Clicked on Ad):
   - Precision: 0.99 → When the model predicts “Click”, it is correct 99% of the time, meaning very few wasted marketing efforts on non-interested users.
   - Recall: 0.97 → It successfully captures 97% of all users who actually clicked, missing only a small fraction of potential customers.
   - F1-score: 0.98 → Excellent trade-off between catching true clickers and avoiding false alarms.
8. The macro and weighted averages (precision, recall, F1 ≈ 0.98) confirm that performance is consistently high for both classes, with no major imbalance in how well each class is handled.

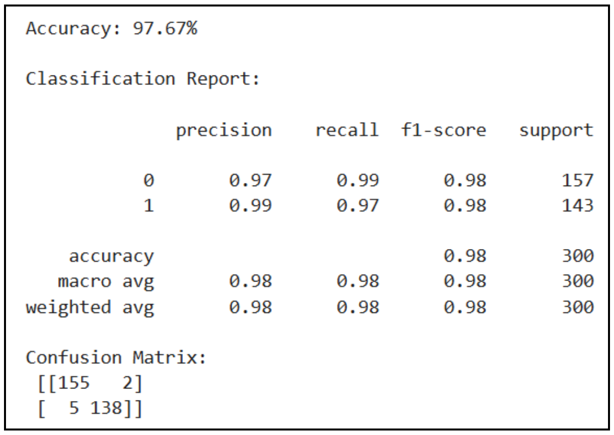
Hyperparameter Tuning

Hyperparameter tuning was performed to ensure that the Logistic Regression model is optimally regularized and not just relying on default settings.

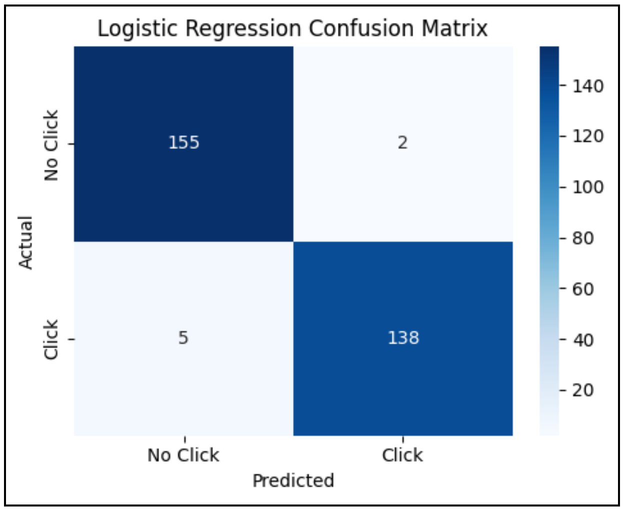
1. Search Strategy
   A grid search with cross-validation was used to explore combinations of:
   - C (inverse regularization strength):
   - $[0.001, 0.01, 0.1, 1, 10, 100, 1000]$
   - $[0.001, 0.01, 0.1, 1, 10, 100, 1000]$
   - penalty: `l1` and `l2`
   - solver: `liblinear` (supports both L1 and L2 penalties and works well for small datasets)
2. For each combination, 5-fold cross-validation was performed on the training set, resulting in 70 model configurations being evaluated. The combination that yielded the highest cross-validated accuracy was selected as the best model.
3. Best Hyperparameters
   The grid search identified the following optimal configuration:
   - C = 100
   - penalty = 'l1'
   - solver = 'liblinear'
4. A relatively large C = 100 means the model uses weaker regularization, allowing coefficients more freedom to fit the data, while L1 penalty encourages sparsity by driving some less important coefficients exactly to zero. This combination suggests that the dataset contains a few strong predictors and some weaker ones that can be safely down-weighted or removed.
5. Impact of Tuning
   With these tuned hyperparameters, the model attains an accuracy of 97.67% and excellent precision/recall for both classes, as reflected in the confusion matrix and classification report. This confirms that the chosen regularization settings provide a good balance between bias and variance, avoiding both underfitting and overfitting. The tuning process therefore gives statistical confidence that the final Logistic Regression model is the best-performing and most reliable configuration for this dataset.

Code and Output - Logistic Regression

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import
train_test_split, GridSearchCV

from sklearn.linear_model import
LogisticRegression

from sklearn.metrics import
accuracy_score,
classification_report,
confusion_matrix


# 1. Data Collection

df = pd.read_csv('advertising.csv')


print("First 5 rows:")

print(df.head())

print("\nShape:", df.shape)


# 2. Data Preprocessing

drop_cols = ['Ad Topic Line',
'City', 'Country', 'Timestamp']

df = df.drop(columns=drop_cols)

X = df.drop(columns=['Clicked on
Ad'])

y = df['Clicked on Ad']


# 3. Train–Test Split (70/30)

X_train, X_test, y_train, y_test =
train_test_split(

    X, y, test_size=0.3,
random_state=101

)


# 4. Hyperparameter Tuning &
Training

param_grid = {

    'C': [0.001, 0.01, 0.1, 1, 10,
100, 1000],

    'penalty': ['l1', 'l2'],

    'solver': ['liblinear']

}


log_reg =
LogisticRegression(max_iter=1000)


grid = GridSearchCV(

    estimator=log_reg,

    param_grid=param_grid,
    cv=5,

    scoring='accuracy',

    n_jobs=-1

)


grid.fit(X_train, y_train)


best_model = grid.best_estimator_

print("\nBest Parameters:",
grid.best_params_)


# 5. Prediction

y_pred = best_model.predict(X_test)


# 6. Evaluation

acc = accuracy_score(y_test, y_pred)

print(f"\nAccuracy:
{acc*100:.2f}%\n")


print("Classification Report:\n")

print(classification_report(y_test,
y_pred))


cm = confusion_matrix(y_test,
y_pred)

print("Confusion Matrix:\n", cm)


# 7. Visualization: Confusion Matrix
Heatmap

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',

            xticklabels=['No Click',
'Click'],

            yticklabels=['No Click',
'Click'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Logistic Regression
Confusion Matrix')

plt.tight_layout()

plt.show()
```



```
Accuracy: 97.67%

Classification Report:

              precision    recall  f1-score   support

           0       0.97      0.99      0.98       157
           1       0.99      0.97      0.98       143

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300

Confusion Matrix:
 [[155   2]
 [  5 138]]
```



Conclusion

The overall practical successfully demonstrates how Linear Regression and Logistic Regression can be applied to real-world datasets for regression and classification tasks, respectively. Using the Weight–Height dataset, Simple Linear Regression accurately models the strong linear relationship between height and weight, achieving a high R^2 score and a clear, interpretable equation that confirms height as a powerful predictor of weight. On the other hand, applying tuned Logistic Regression to the Advertising dataset shows that user demographic and browsing features can very effectively predict whether a user will click on an ad, as evidenced by the high accuracy, excellent precision, recall, and near-perfect confusion matrix. Together, these experiments highlight the versatility of regression techniques: Linear Regression excels at predicting continuous outcomes with interpretable trends, while Logistic Regression, especially with proper hyperparameter tuning, provides robust and reliable binary classification in practical business scenarios.