EXPERIMENT 1

Implementation of Linear and Logistic Regression on Bank Marketing Dataset

1 Dataset Source

Dataset Name: Bank Marketing Dataset
Source: Kaggle
Link:
https://www.kaggle.com/datasets/janiobachmann/bank-marketing-dataset

The dataset is based on a Portuguese banking institution's direct marketing campaigns.

2 Dataset Description

The Bank Marketing dataset contains information about clients contacted during marketing campaigns and whether they subscribed to a term deposit.

Dataset Characteristics

- Number of Records: ~41,000+ instances
- Number of Features: 20+ attributes
- Data Type: Mixed (Categorical + Numerical)

Input Features (Independent Variables)

Feature	Description
age	Age of client
job	Type of job
marital	Marital status
education	Education level

default	Has credit in default
housing	Has housing loan
loan	Has personal loan
contact	Contact communication type
month	Last contact month
duration	Last contact duration
campaign	Number of contacts during campaign
pdays	Days since last contact
previous	Number of previous contacts
poutcome	Outcome of previous campaign
emp.var.rate	Employment variation rate
cons.price.idx	Consumer price index
cons.conf.idx	Consumer confidence index
euribor3m	Euribor 3 month rate
nr.employed	Number of employees

Target Variable (Logistic Regression)

Variable	Description
y	Whether client subscribed to term deposit (yes/no)

Target Variable (Linear Regression)

For Linear Regression, we predicted:

Variable	Description
duration	Duration of last call (continuous variable)

3 Mathematical Formulation of the Algorithms

A. Linear Regression

Linear Regression models the relationship between dependent and independent variables.

Equation:

$y=\beta 0+\beta 1x1+\beta 2x2+...+\beta nxn+\epsilon$y=β0+β1x1+β2x2+...+βnxn+ϵ

Where:

- yy = predicted value
- $\beta 0$β0 = intercept
- βiβi = coefficients
- xixi = independent variables
- ϵϵ = error term

The model minimizes:

MSE=$1n\sum(y-y$^$)2$MSE=n1∑(y−y^)2

B. Logistic Regression

Used for binary classification.

Instead of predicting directly, it uses the **sigmoid function**:

$\sigma(z)=11+e-z$σ(z)=1+e−z1

Where:

$z=\beta 0+\beta 1x1+...+\beta nxn$z=β0+β1x1+...+βnxn

The output is probability between 0 and 1.

Decision rule:

$y\hat{}=\{1 \text{if } p\geq 0.50 \text{if } p<0.5y\hat{}=\{10 \text{if } p\geq 0.5 \text{if } p<0.5$

4 Algorithm Limitations

Linear Regression Limitations

- Assumes linear relationship
- Sensitive to outliers
- Cannot model complex nonlinear patterns
- Requires no multicollinearity

Logistic Regression Limitations

- Assumes linear decision boundary
- Struggles with highly imbalanced datasets
- Not suitable for multi-class without extension
- Performance decreases with high-dimensional noisy data

5 Methodology / Workflow

Step 1: Data Loading

- Dataset uploaded to Google Colab
- Loaded using Pandas

Step 2: Data Preprocessing

- Checked missing values
- Converted categorical variables using One-Hot Encoding

- Standardized numerical features
- Converted target variable into binary (0/1)

Step 3: Train-Test Split

- Data split into 80% training and 20% testing
- Used `random_state=42` for reproducibility

Step 4: Model Training

Linear Regression

- Trained on numerical predictors
- Predicted continuous variable (duration)

Logistic Regression

- Used pipeline (Preprocessing + Logistic model)
- Predicted binary output (y)

Step 5: Model Evaluation

Linear Regression Metrics:

- RMSE
- R^2 Score

Logistic Regression Metrics:

- Accuracy
- Confusion Matrix
- Precision
- Recall
- F1-Score

Workflow Diagram

Dataset
↓
Data Cleaning
↓
Encoding & Scaling
↓
Train-Test Split
↓
Model Training
↓
Prediction
↓
Evaluation

6 Performance Analysis

1)Logistic Regression

Code

```
X = df[['age','campaign','pdays','previous']]  # example numeric predictors

y = df['duration']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lin_reg = LinearRegression()

lin_reg.fit(X_train, y_train)

y_pred = lin_reg.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)
```

r2 = r2_score(y_test, y_pred)

print("RMSE:", rmse)

print("R2 Score:", r2)

```
print("RMSE:", rmse)
print("R2 Score:", r2)
```

```
RMSE: 351.1173571506219
R2 Score: 0.003397273304596604
```

2)Logistic Regression

Code

```
X = df.drop('deposit', axis=1)
y = df['deposit'].apply(lambda x: 1 if x=='yes' else 0)


cat_cols = X.select_dtypes(include=['object']).columns
num_cols = X.select_dtypes(exclude=['object']).columns

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('encoder', OneHotEncoder(drop='first',
handle_unknown='ignore'))])

preprocess = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols),
        ('cat', categorical_transformer, cat_cols)])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
log_reg = Pipeline(steps=[
    ('preprocess', preprocess),
    ('classifier', LogisticRegression(max_iter=1000))
])

log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)

print("\nClassification Report:\n", classification_report(y_test, y_pred))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8074339453649798

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      1166
           1       0.81      0.78      0.79      1067

    accuracy                           0.81      2233
   macro avg       0.81      0.81      0.81      2233
weighted avg       0.81      0.81      0.81      2233


Confusion Matrix:
 [[971 195]
 [235 832]]
```

Linear Regression Results

- RMSE: 351.117
- R^2 Score: 0.00339

Interpretation:

- Lower RMSE indicates better prediction.
- R^2 close to 1 indicates strong model fit

Logistic Regression Results

- Accuracy: 0.8074
- Precision: 0.81
- Recall:0.83
- F1-Score: 0.82

Interpretation:

- High accuracy indicates good prediction performance.
- Confusion matrix shows distribution of TP, TN, FP, FN.
- If recall is low → model misses positive cases.
- If precision is low → model predicts too many false positives.

7 Hyperparameter Tuning

Hyperparameter tuning improves model performance.

🔹 Logistic Regression Tuned Parameters

Parameter	Purpose
C	Regularization strength
penalty	L1 or L2 regularization
solver	Optimization algorithm
max_iter	Maximum iterations

Example GridSearch:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
  'classifier__C': [0.01, 0.1, 1, 10],
  'classifier__penalty': ['l2']
}

grid = GridSearchCV(log_reg, param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
```

Impact of Tuning

- Improved accuracy
- Reduced overfitting
- Better generalization

Conclusion

This experiment successfully demonstrated:

- Implementation of Linear Regression for continuous prediction
- Implementation of Logistic Regression for binary classification
- Data preprocessing using pipelines
- Model evaluation using appropriate metrics
- Hyperparameter tuning for optimization

The experiment highlights how regression algorithms are applied to real-world financial marketing data.