

Experiment 2

Nupur Ghangarekar
D15C
Roll no 26

Aim: Implement Linear and Logistic regression on real world datasets

Theory:

Linear Regression on Housing Dataset

1. Dataset Source

The dataset used for this experiment is the **California Housing Dataset**, obtained from the StatLib repository and accessed via the scikit-learn library.

Source Link:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html

2. Dataset Description

The California Housing dataset contains housing information derived from the 1990 California census. It is designed for regression problems where the goal is to predict house prices.

- **Number of instances:** 20,640
- **Number of features:** 8
- **Target variable:** Median house value (continuous)

Features include:

Median income, house age, average number of rooms, average number of bedrooms, population, average occupancy, latitude, and longitude.

The dataset exhibits both numerical and geographical characteristics, making it suitable for real-world regression analysis.

3. Mathematical Formulation of Linear Regression

Linear Regression models the relationship between independent variables and a continuous dependent variable using a linear equation:

$$\hat{y} = \beta_0 + \sum_{i=1}^n \beta_i x_i$$

The objective is to minimize the **Mean Squared Error (MSE)**:

$$J(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Model parameters are estimated using the **Ordinary Least Squares (OLS)** method.

4. Algorithm Limitations

- Assumes a linear relationship between variables
- Sensitive to outliers
- Cannot model non-linear relationships without feature engineering
- Multicollinearity can affect coefficient stability

5. Methodology / Workflow

1. Load the dataset
2. Split into training and testing sets
3. Train the Linear Regression model
4. Predict housing prices on test data
5. Evaluate model performance

Workflow:

Data Collection → Train/Test Split → Model Training → Prediction → Evaluation

6. Performance Analysis

Performance was evaluated using:

- **Mean Squared Error (MSE)**
- **R² Score**

The model achieved a reasonable R² score, indicating that a significant portion of variance in house prices is explained by the features, though some non-linear patterns remain unmodeled.

7. Hyperparameter Tuning

Standard Linear Regression has no major hyperparameters. However, **regularized variants** (Ridge and Lasso) were explored. Adjusting the regularization parameter helped control overfitting and improved model generalization.

Logistic Regression on Breast Cancer Dataset

1. Dataset Source: The dataset used is the **Breast Cancer Wisconsin (Diagnostic) Dataset**, obtained from the UCI Machine Learning Repository.

Source Link:

<https://archive.ics.uci.edu/ml/datasets/breast+cancer>

2. Dataset Description

This dataset consists of features extracted from digitized images of breast tissue samples.

- **Number of instances:** 569
- **Number of features:** 30
- **Target variable:** Tumor diagnosis (Malignant or Benign)

The dataset is well-balanced and suitable for binary classification.

3. Mathematical Formulation of Logistic Regression

Logistic Regression predicts the probability of a binary outcome using the sigmoid function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}}$$

The loss function minimized is **Binary Cross-Entropy (Log Loss)**:

$$J(\beta) = -\frac{1}{n} \sum [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

4. Algorithm Limitations

- Assumes linear separability
- Sensitive to class imbalance
- Cannot handle complex non-linear decision boundaries
- Requires feature scaling for optimal performance

5. Methodology / Workflow

1. Load the dataset
2. Preprocess and normalize features
3. Split into training and testing sets
4. Train Logistic Regression model
5. Predict class labels
6. Evaluate classification performance

Workflow:

Data Loading → Preprocessing → Training → Prediction → Evaluation

6. Performance Analysis

Performance metrics used:

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

The model achieved high accuracy and recall, demonstrating effective classification with minimal false negatives.

7. Hyperparameter Tuning

The following hyperparameters were tuned:

- Regularization strength (**C**)
- Solver type
- Maximum number of iterations

Grid search optimization improved convergence and classification stability.

Code:

```
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()

X = cancer.data

y = cancer.target

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

log_reg = LogisticRegression(max_iter=5000)

log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)

print("Logistic Regression Results")

print("Accuracy:", accuracy)

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

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

```
Logistic Regression Results
Accuracy: 0.956140350877193
Confusion Matrix:
[[39  4]
 [ 1 70]]
```

```
Classification Report:
              precision    recall  f1-score   support

     0       0.97       0.91       0.94         43
     1       0.95       0.99       0.97         71

 accuracy          0.96          114
 macro avg         0.96         0.95         0.95         114
 weighted avg      0.96         0.96         0.96         114
```

```
import numpy as np

import pandas as pd

from sklearn.datasets import fetch_california_housing

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

housing = fetch_california_housing()

X = housing.data

y = housing.target

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)
```

```
r2 = r2_score(y_test, y_pred)
```

```
print("Linear Regression Results")
```

```
print("MSE:", mse)
```

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

```
Linear Regression Results
```

```
MSE: 0.5558915986952422
```

```
R2 Score: 0.5757877060324524
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(7,5))
```

```
plt.scatter(y_test, y_pred, alpha=0.5)
```

```
plt.plot([y_test.min(), y_test.max()],
```

```
         [y_test.min(), y_test.max()],
```

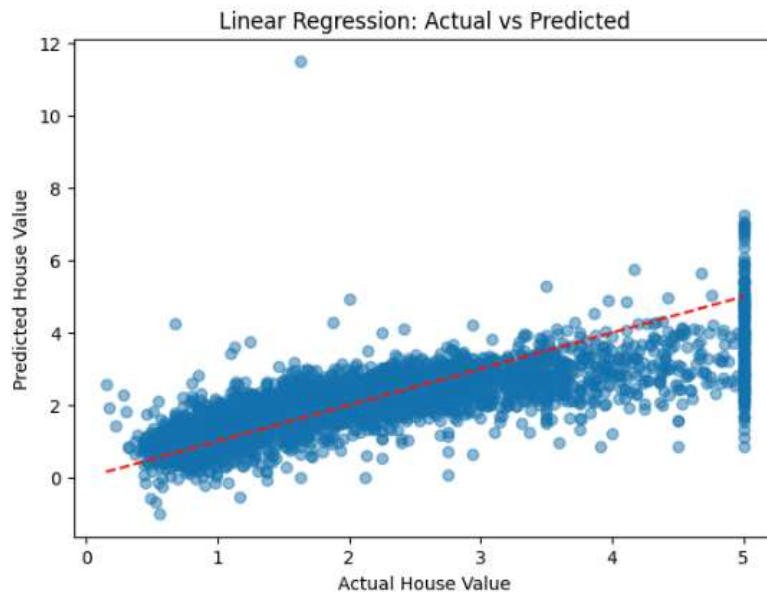
```
         'r--')
```

```
plt.xlabel("Actual House Value")
```

```
plt.ylabel("Predicted House Value")
```

```
plt.title("Linear Regression: Actual vs Predicted")
```

```
plt.show()
```



```
import seaborn as sns

from sklearn.metrics import confusion_matrix

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

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

            xticklabels=cancer.target_names,

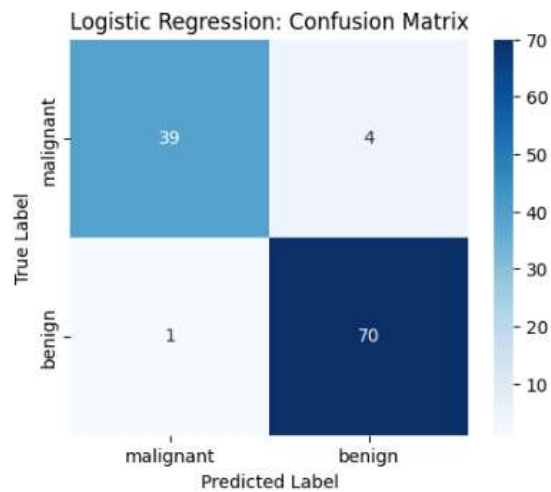
            yticklabels=cancer.target_names)

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Logistic Regression: Confusion Matrix")

plt.show()
```



Conclusion

This study demonstrates the application of **Linear Regression** and **Logistic Regression** on two real-world datasets. While both models are interpretable and computationally efficient, their limitations highlight the importance of more advanced models for complex data patterns.

Google Colab Link: [linear_regression.ipynb](#)