Linear and Logistic Regression

Name: RUQAIA AL-TUWEJRI

Introduction:

Regression techniques are fundamental tools in data science and machine learning. They are used to understand the relationship between variables and to make predictions.
Two of the most widely used regression models are:
- Linear Regression: for predicting continuous numerical values.
- Logistic Regression: for binary classification problems.

Both models belong to supervised learning and are trained using labeled data to make predictions on unseen data.

1. Linear Regression:

Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables using a straight line.

Linear Regression Equation:

The general form of the linear regression equation is:

$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$

Where:

$\hat{y}$: Predicted value (dependent variable)

β_0: Intercept term (the value of $\hat{y}$ when all $x_i = 0$)

$\beta_1, \beta_2, \ldots, \beta_n$: Coefficients for each independent variable $x_1, x_2, \ldots, x_n$

$x_1, x_2, \ldots, x_n$: Independent variables (features)

Practical Example:

Suppose we want to predict the price of a house based on the number of bedrooms it has. Let's define:

$\hat{y}$: Predicted house price

x: Number of bedrooms

$\beta_0 = 50{,}000$: Base price of the house

β_1 = 30,000: Additional price per bedroom

The linear regression equation becomes: $\hat{y}$ = 50,000 + 30,000 × x

Example Calculation:

If a house has 3 bedrooms (x = 3):

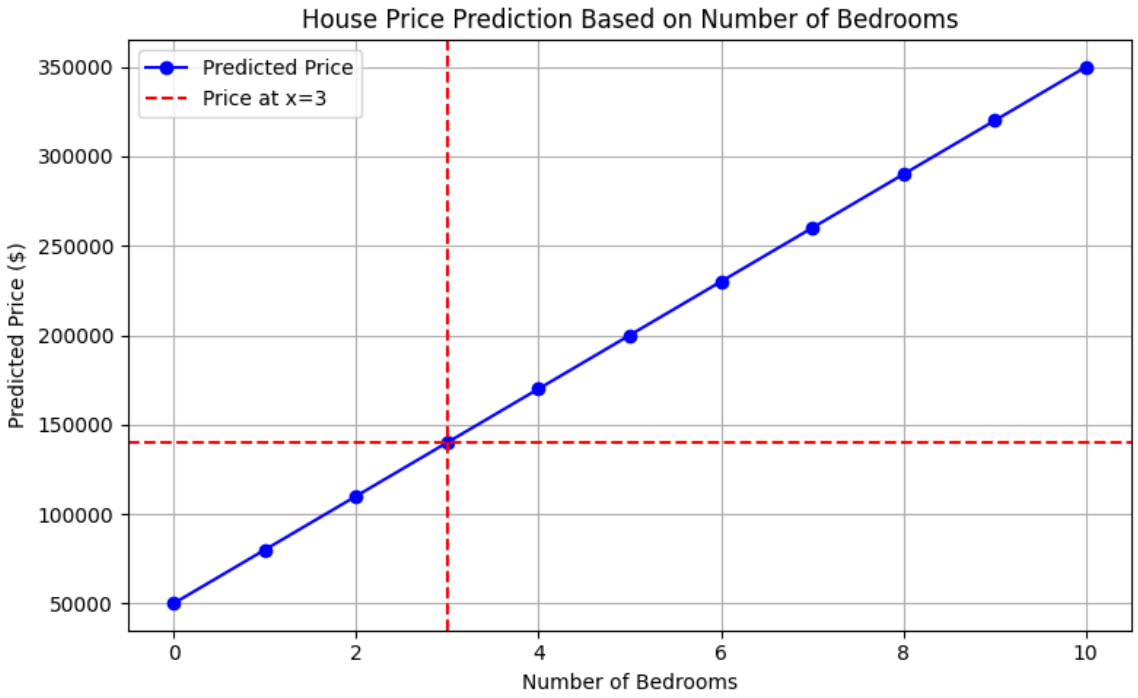
$\hat{y}$ = 50,000 + 30,000 × 3 = 140,000

So, the predicted price of the house would be $140,000.

Python Implementation + Visualization:

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

# Parameters of the linear regression model
beta_0 = 50000  # Base price
beta_1 = 30000  # Price per bedroom

# Define the linear regression function
def predict_price(bedrooms):
    return beta_0 + beta_1 * bedrooms

# Example: Predict price for a house with 3 bedrooms
x_example = 3
predicted_price = predict_price(x_example)
print(f"Predicted price for a house with {x_example} bedrooms:
${predicted_price:,.0f}")

# Generate data for visualization
x_values = np.arange(0, 11)  # Bedrooms from 0 to 10
y_values = predict_price(x_values)

# Plotting
plt.figure(figsize=(8, 5))
plt.plot(x_values, y_values, marker='o', linestyle='-',
color='blue', label='Predicted Price')
plt.axhline(y=predicted_price, color='red', linestyle='--',
label=f'Price at x={x_example}')
plt.axvline(x=x_example, color='red', linestyle='--')
plt.title('House Price Prediction Based on Number of Bedrooms')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Predicted Price ($)')
plt.grid(True)
```

```
plt.legend()
plt.tight_layout()
plt.show()
```



What is a Loss Function?

A Loss Function is a metric used to measure how well a model performs in making predictions. it measures the difference between the actual values and the predicted values.

The smaller the loss, the better the model's performance.

What is MSE (Mean Squared Error)?

MSE is one of the most common loss functions used in linear regression, and it's calculated as follows:

$$MSE = \Sigma(y_i - p_i)^2/n$$

Updated Python Code with MSE and Visualization:

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

# Parameters of the linear regression model
beta_0 = 50000  # Base price
beta_1 = 30000  # Price per bedroom

# Define the prediction function
def predict_price(bedrooms):
    return beta_0 + beta_1 * bedrooms

# Bedrooms from 1 to 10
x = np.arange(1, 11)

# Predicted prices using the model
y_pred = predict_price(x)

# Simulate actual prices with some noise (to mimic real-world data)
np.random.seed(42)  # for reproducibility
noise = np.random.normal(0, 10000, size=len(x))  # noise with std
deviation of 10,000
y_actual = y_pred + noise  # actual prices = predicted + noise

# Compute Mean Squared Error (MSE)
n = len(x)
mse = np.mean((y_pred - y_actual) ** 2)
print(f"Mean Squared Error (MSE): {mse:,.2f}")

# Plotting predictions vs actual values
plt.figure(figsize=(10, 6))
plt.plot(x, y_pred, label='Predicted Prices', color='blue',
marker='o')
plt.scatter(x, y_actual, label='Actual Prices (with noise)',
color='orange', s=80)
plt.title('Predicted vs Actual House Prices')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Price ($)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

the Plot Show?

- **Blue line**: Predictions from the linear regression model.
- **Orange dots**: Simulated real (noisy) house prices.
- **MSE**: A numeric value that tells you how far off the predictions are on average.

2.Logistic Regression:

Logistic Regression is a statistical method used to model binary classification problems — where the output is either 0 or 1 (e.g., "yes" or "no", "spam" or "not spam").

Unlike linear regression which predicts continuous values, logistic regression predicts probabilities, which are then mapped to binary outcomes.

Logistic Regression does not provide a direct value like the actual price. Instead, it gives a probability between 0 and 1 that the input belongs to a certain class.

General Formula:

$\text{hat\_y} = 1 / (1 + e^{-(\beta_0 + \beta_1 x)})$

Where:

- hat_y: The probability that the input belongs to class 1 (e.g., house price > $200,000)
- β_0: Intercept (constant term)
- β_1: Coefficient (slope for the feature)
- x: Independent variable (number of bedrooms)
- e: Euler's number ≈ 2.718

Practical Example (House Classification):

We want to predict the probability that a house costs more than $200,000 based on the number of bedrooms.

Assume:

- $\beta_0 = -8$
- $\beta_1 = 1.5$

So the logistic regression equation becomes:

$y = 1 / (1 + e^{-(-8 + 1.5 * x)})$

Example Calculation:

If the house has **6 bedrooms**, then:

y = 1 / (1 + e^-(-8 + 1.5 * 6))
= 1 / (1 + e^(-1))
≈ 1 / (1 + 0.3679)
≈ 0.731

Therefore, the probability that the house price exceeds $200,000 is approximately 73.1%.

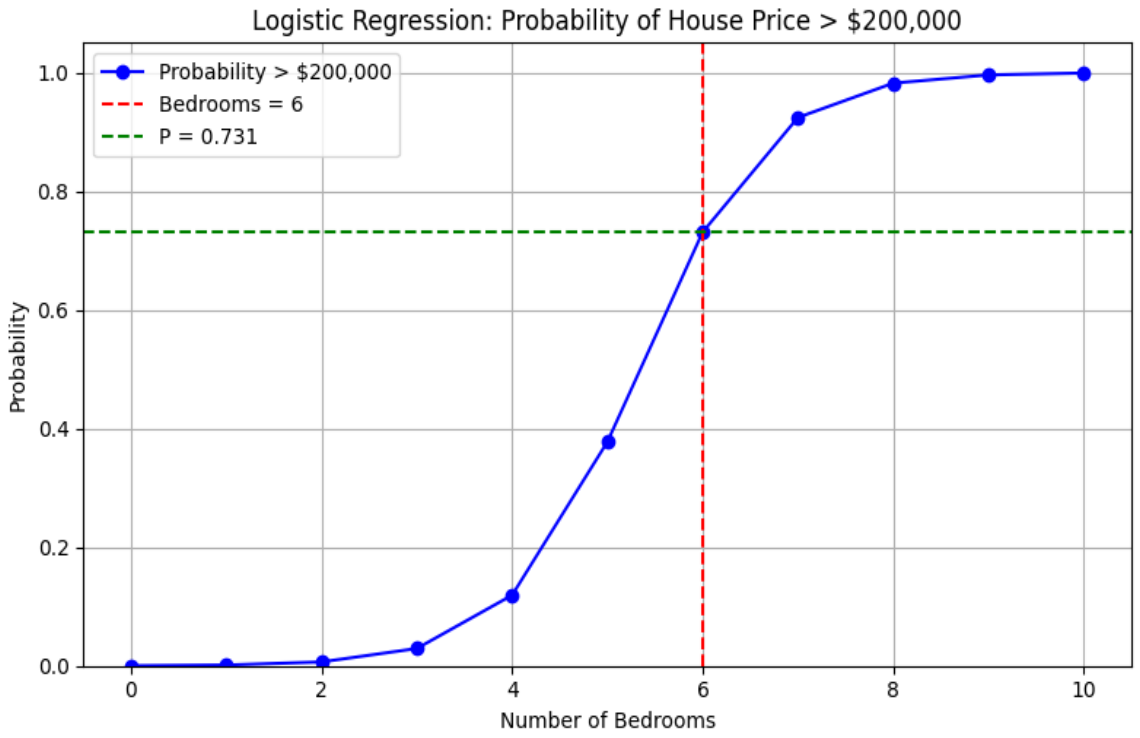
Since this probability is greater than 0.5, the model classifies the house as:

Class = 1 (Yes)

Python Implementation + Visualization:

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

# Define logistic regression function
def logistic(x, beta_0, beta_1):
    return 1 / (1 + np.exp(-(beta_0 + beta_1 * x)))

# Parameters
beta_0 = -8
beta_1 = 1.5

# Bedrooms range from 0 to 10
x_values = np.arange(0, 11)
y_probs = logistic(x_values, beta_0, beta_1)

# Predict for a specific example (6 bedrooms)
x_example = 6
y_example = logistic(x_example, beta_0, beta_1)

print(f"Predicted probability for house with {x_example} bedrooms:
{y_example:.3f}")

# Plot
plt.figure(figsize=(8, 5))
plt.plot(x_values, y_probs, marker='o', linestyle='-', color='blue',
label='Probability > $200,000')
plt.axvline(x=x_example, color='red', linestyle='--',
label=f'Bedrooms = {x_example}')
```

```
plt.axhline(y=y_example, color='green', linestyle='--', label=f'P =
{y_example:.3f}')
plt.title('Logistic Regression: Probability of House Price >
$200,000')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Probability')
plt.ylim(0, 1.05)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Log Loss – Binary Cross Entropy:

Log Loss, also known as Binary Cross-Entropy**,** is the most commonly used loss function in logistic regression. It evaluates how close the predicted probabilities are to the actual binary labels (0 or 1).

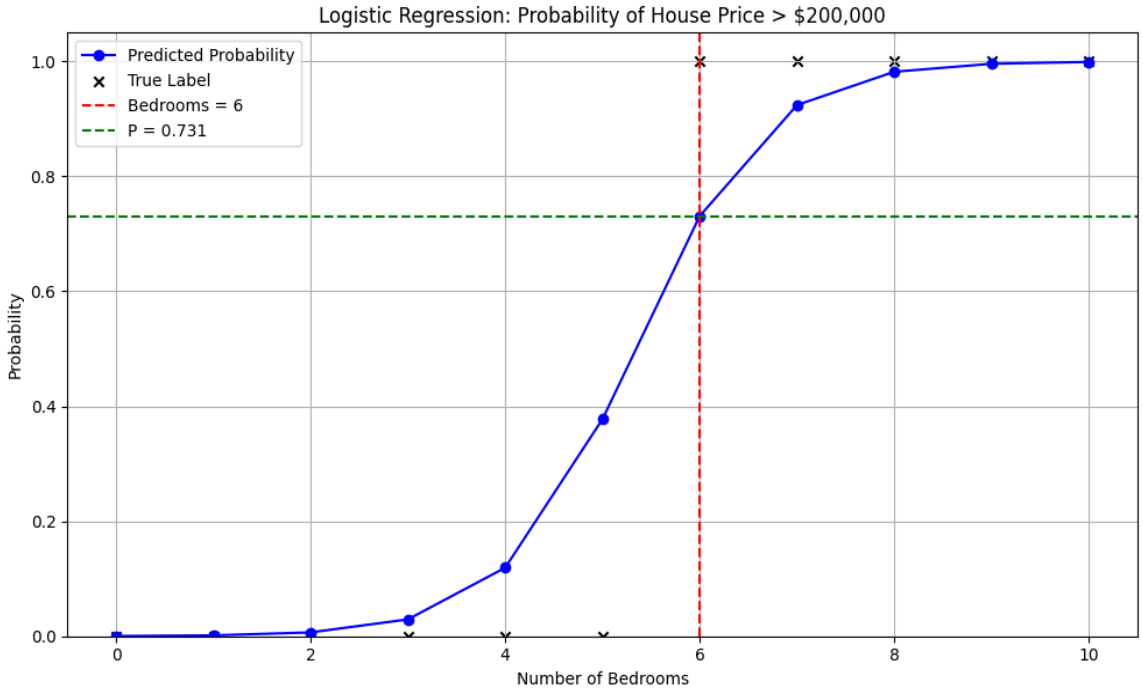
Formula:

The Binary Cross-Entropy (Log Loss) is the typical loss function:
$\text{Loss} = -(1/n) * \Sigma[y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)]$

Updated Python Code with Loss Function and Visualization:

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

# Logistic regression function
def logistic(x, beta_0, beta_1):
    return 1 / (1 + np.exp(-(beta_0 + beta_1 * x)))

# Log Loss function (Binary Cross Entropy)
def log_loss(y_true, y_pred):
    # Add epsilon to prevent log(0)
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) *
np.log(1 - y_pred))

# Parameters for logistic regression
beta_0 = -8
beta_1 = 1.5

# Bedrooms from 0 to 10
x_values = np.arange(0, 11)
y_probs = logistic(x_values, beta_0, beta_1)

# Simulate ground truth labels (e.g., these could be from a dataset)
# Let's say a house is >$200,000 if it has more than 5 bedrooms
y_true = np.array([0 if x <= 5 else 1 for x in x_values])

# Calculate log loss for each prediction
individual_losses = - (y_true * np.log(y_probs + 1e-15) + (1 -
y_true) * np.log(1 - y_probs + 1e-15))

# Total average log loss
avg_loss = log_loss(y_true, y_probs)

# Display average loss
print(f"\n Average Log Loss: {avg_loss:.4f}")
```

```
# Predict for a specific example (6 bedrooms)
x_example = 6
y_example = logistic(x_example, beta_0, beta_1)
print(f"Predicted probability for house with {x_example} bedrooms:
{y_example:.3f}")

# Plot probabilities
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_probs, marker='o', linestyle='-', color='blue',
label='Predicted Probability')
plt.scatter(x_values, y_true, color='black', marker='x', label='True
Label')
plt.axvline(x=x_example, color='red', linestyle='--',
label=f'Bedrooms = {x_example}')
plt.axhline(y=y_example, color='green', linestyle='--', label=f'P =
{y_example:.3f}')
plt.title('Logistic Regression: Probability of House Price >
$200,000')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Probability')
plt.ylim(0, 1.05)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Plot individual log losses
plt.figure(figsize=(10, 5))
plt.bar(x_values, individual_losses, color='orange',
label='Individual Log Loss')
plt.axhline(avg_loss, color='red', linestyle='--', label=f'Avg Log
Loss = {avg_loss:.4f}')
plt.title('Log Loss per Prediction')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Log Loss')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Conclusion:

Both Linear and Logistic Regression are essential techniques in supervised learning. Linear regression is suitable for problems requiring continuous value prediction, while logistic regression is best for binary classification tasks.
A solid understanding of these models lays the groundwork for mastering more complex machine learning algorithms.