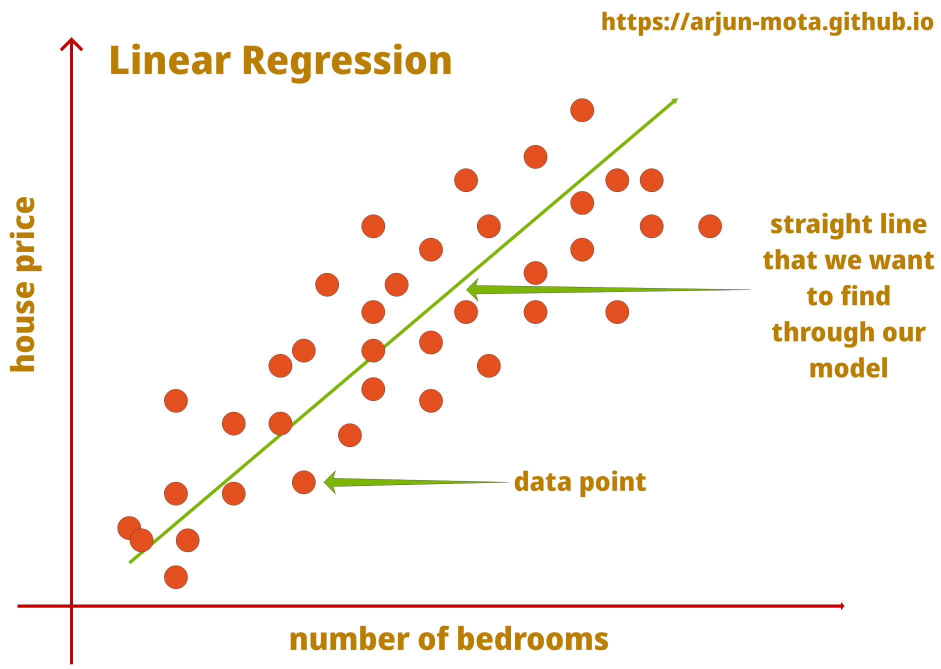
Linear Regression

By Trần Minh Dương - Learning Support

Overview

Imagine you're trying to predict house prices based on features like the size of the house, the number of bedrooms, and the location. Linear regression helps you establish a relationship between these features (inputs) and the price (output) by finding the best-fit line that minimizes the prediction error.



Linear regression offers a simple yet powerful way to model relationships in data. To find this best-fit line, we use one of two methods:

1. **Gradient Descent:** An iterative optimization process that adjusts the model step by step.
2. **Normal Equation:** A direct solution using linear algebra to calculate the optimal parameters.

In this document, we'll explore both methods in detail and understand their applications.

1. Gradient Descent

Gradient descent is an iterative optimization algorithm used to minimize the cost function in linear regression.

Key Points:

Hypothesis Function: The predicted output of linear regression is computed as follows:

- **For a single example:**

$$h_\theta(x^{(i)}) = \theta^T x^{(i)} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Here, $x^{(i)}$ is a single input example.

- **For the entire dataset (input matrix):**

$$h_\theta(X) = X\theta$$

Where:

  - $x^{(i)}$: A column vector representing the i-th example, containing n features.
    Dimension: $n \times 1$.
  - X: The input matrix containing m examples and n features.
    Dimension: $m \times n$.
  - θ: The parameter vector containing n weights (including θ_0, the bias term).
    Dimension: $n \times 1$.

Goal: Minimize the cost function (Mean Squared Error) to find the optimal parameters θ.

Mean Squared Error (MSE) Formula:

$$\boxed{J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2}$$

Where:

- m: Number of training examples in the batch.
- $h_\theta(x^{(i)})$: Predicted output for the i-th example.
- $y^{(i)}$: Actual output for the i-th example.

Steps for Batch Gradient Descent of Linear Regression:

1. Start with initial values for the parameters θ (e.g., $\theta_0 = 0, \theta_1 = 0$).
2. Compute the gradient of the cost function $J(\theta)$ with respect to each parameter:

- **Scalar form**:

$$\boxed{\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}}$$

  where:

  - $h_\theta(x^{(i)})$ is the predicted output
  - $h_\theta(x^{(i)}) - y^{(i)}$ is the error of the prediction compared to the real output
  - $x_j^{(i)}$ is the j-th feature value of the i-th training example

- **Matrix form:**

$$\boxed{\nabla J(\theta) = \frac{1}{m} X^T \left(X\theta - y \right)}$$

  where:

  - X is the input matrix of dimension m x n.
  - θ is the column vector of dimension n x 1.
  - $X\theta$ gives the vector of predicted outputs of dimension m x 1.

- y is the vector of target values of dimension m x1.

3. Update each parameter using the formula:

$$\boxed{\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \quad \text{or} \quad \theta = \theta - \alpha \nabla J(\theta)}$$

   where α is the learning rate.

4. Repeat until the cost function converges.

Advantages:

- Works well with large datasets.
- Scales to high-dimensional data.

Disadvantages:

- Requires choosing a suitable learning rate α.
- May converge slowly or get stuck in local minima if poorly initialized.

---

2. Normal Equation

The normal equation is a closed-form solution for linear regression. It directly computes the optimal parameters θ without iteration.

Key Points:

Goal: Minimize the cost function by solving:

$$\nabla J(\theta) = 0$$

$$X^T(X\theta - y) = 0$$

$$X^T X\theta = X^T y$$

- Finally, we derive the formula for θ:

$$\boxed{\theta = (X^T X)^{-1} X^T y}$$

  where:

  - X: Feature matrix.
  - y: Vector of target values.
  - X^T: Transpose of X.
  - $(X^T X)^{-1}$: Inverse of $X^T X$.

Steps:

1. Construct the feature matrix X (including the column of 1s for the bias term).
2. Compute θ using the formula above.

Advantages:

- No need for choosing a learning rate.
- Direct computation provides exact results.

Disadvantages:

- Computationally expensive for large datasets due to matrix inversion ($O(n^3)$).
- May cause error when X^TX is non-invertible or the dataset is too large to fit into memory.

Summary of Key Differences

Aspect	Gradient Descent	Normal Equation
Approach	Iterative optimization	Direct computation
Computation Cost	Scales well for large datasets	Expensive for large datasets
Parameters Required	Requires learning rate (α)	No hyperparameters
Suitability	Large datasets or high dimensions	Small to medium datasets

Exercise

Given the training data for a linear regression problem, to predict the final score of a student in Machine Learning class, based on his/her score in programming and probability class. Starting at:

- θ_0 = 0, θ_1 = 1, θ_2 = -1
- Learning rate (α) = 4

Task:

1. Calculate the coefficients after the first iteration with **batch-gradient descent**.
2. Based on the coefficients you just found, calculate the prediction score of the fifth student.

Data Table:

Student	Score in Programming	Score in Probability	Score in Machine Learning
1	18	15	15
2	15	10	11
3	16	12	13
4	19	16	17
5	17	11	???

In [2]:
```
#In this exercise, I will use the matrix approach with the numpy library in python
import numpy as np
```

In [3]:
```
#Input / Output
X = np.array([[18,15],[15,10],[16,12],[19,16]])
y = np.array([15,11,13,17])

#Parameters
```

```python
theta = np.array([0,1,-1])
alpha = 4
m = len(y) #batch size
```

In [4]:
```python
#reshape X to add an additional column of 1 to the left (bias term)
X_bias = np.hstack((np.ones((m,1)),X))
print("X_bias = \n",X_bias)
```

```
X_bias =
 [[ 1. 18. 15.]
 [ 1. 15. 10.]
 [ 1. 16. 12.]
 [ 1. 19. 16.]]
```

In [5]:
```python
def hypothesis(X,theta):
    return X @ theta # the @ here is the operator for matrix multiplication in python

predictions = hypothesis(X_bias,theta)
print(predictions)
```

```
[3. 5. 4. 3.]
```

In [6]:
```python
errors = predictions - y
print(errors)
```

```
[-12.  -6.  -9. -14.]
```

In [7]:
```python
gradient = 1/m * (X_bias.T @ errors)
print(gradient)
```

```
[ -10.25 -179.   -143.  ]
```

In [8]:
```python
theta = theta - alpha * gradient
print("Theta after one iteration of batch-gradient descent is: \n",theta)
```

```
Theta after one iteration of batch-gradient descent is:
 [ 41. 717. 571.]
```

In [9]:
```python
fifthStudent = np.array([1,17,11])

score = hypothesis(fifthStudent,theta)
print("The score of the fifth student with 17 in Programming and 11 in Probability is
```

```
The score of the fifth student with 17 in Programming and 11 in Probability is: 18511.
0
```

Normal equation approach:

In [10]:
```python
X_bias = np.hstack((np.ones((m,1)),X))

# Compute theta = (X^T . X)^(-1) . X^T . y
# np.linalg.inv(A) returns the inverse of A : A^-1
theta_normal = np.linalg.inv(X_bias.T @ X_bias) @ X_bias.T @ y
print("Theta from the normal equation:", np.round(theta_normal,2))
```

```
Theta from the normal equation: [-9.8  1.4 -0. ]
```

This document was created in Jupyter Notebook by Trần Minh Dương (tmd).

If you have any questions or notice any errors, feel free to reach out via Discord at @tmdhoctiengphap or @ICT-Supporters on the USTH Learning Support server.

Check out my GitHub repository for more projects: GalaxyAnnihilator/MachineLearning .