1 Linear Regression

2 Introduction

Linear regression is a popular supervised learning algorithm used for predicting a continuous response variable. It is based on the assumption that there exists a linear relationship between the independent variables (also known as features or predictors) and the dependent variable (also known as the response variable or target).

In this article, we will implement a simple linear regression model using gradient descent in Python. We will use the scikit-learn library for evaluating the model's performance and the Matplotlib library for visualizing the results.

3 Linear Regression with Gradient Descent

Gradient descent is a popular optimization algorithm used for finding the optimal parameters of a machine learning model. It works by iteratively adjusting the parameters in the direction of the negative gradient of the cost function until convergence.

The cost function for linear regression is the mean squared error (MSE), which measures the average squared difference between the predicted and actual values. The objective of the model is to minimize the MSE by adjusting the parameters (also known as weights or coefficients).

The mathematical formula for linear regression can be expressed as follows:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon \tag{1}$$

where y is the dependent variable, β_0 is the intercept, β_1 to β_n are the coefficients, x_1 to x_n are the independent variables, and ϵ is the error term.

The coefficients can be estimated using the following formula:

$$\beta = (X^T X)^{-1} X^T y \tag{2}$$

where X is the matrix of independent variables and y is the vector of dependent variables.

However, this formula is computationally expensive for large datasets. Instead, we can use gradient descent to iteratively update the coefficients until convergence.

The gradient descent algorithm can be expressed as follows:

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} MSE(\beta) \tag{3}$$

where β_j is the j-th coefficient, α is the learning rate (also known as step size), and $\frac{\partial}{\partial \beta_j} MSE(\beta)$ is the partial derivative of the MSE with respect to β_j.

We can use batch gradient descent, which updates the coefficients using the entire dataset, or stochastic gradient descent, which updates the coefficients

using a single data point at a time. In this article, we will use batch gradient descent.

4 Implementation

We will implement a LinearRegression class that can be used to train and predict on data. The class takes two hyperparameters: learning rate and number of iterations.

Here's the implementation of the LinearRegression class in Python:

```
import numpy as np

class LinearRegression:
def init(self, learning_rate=0.01, n_iters=1000):
self.learning_rate = learning_rate
self.n_iters = n_iters
self.weights = None
self.bias = None

\begin{verbatim}
def fit(self, X, y):
# Initialize weights and bias
n_samples, n_features = X.shape
self.weights = np.zeros(n_features)
self.bias = 0

python
Copy code
    # Gradient descent
    for i in range(self.n_iters):
        # Predicted values
        y_pred = np.dot(X, self.weights) + self.bias

        # Cost function
        cost = np.mean((y_pred - y) ** 2)

        # Gradient calculation
        dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
        db = (1 / n_samples) * np.sum(y_pred - y)

        # Update weights and bias
        self.weights -= self.learning_rate * dw
        self.bias -= self.learning_rate * db

        # Print progress
```

```
        if i % 100 == 0:
            print(f"Iteration {i}, Cost: {cost}")

def predict(self, X):
    # Predicted values
    y_pred = np.dot(X, self.weights) + self.bias
    return y_pred
```

The fit() method performs gradient descent on the training data to learn the optimal weights and bias. The predict() method predicts the output for new data using the learned weights and bias.

5 Usage

To use the LinearRegression class, simply import it from $linear_regression.py$ and create an instance of it with desired hyperparameters:

```
from linear_regression import LinearRegression

regressor = LinearRegression(learning_rate=0.01, n_iters=1000)
```

Then, train the model on your data using the fit() method:

```
regressor.fit(X_train, y_train)
```

Finally, predict the output for new data using the predict() method:

```
predictions = regressor.predict(X_test)
```

You can evaluate the performance of the model using mean squared error and coefficient of determination (R-squared) metrics:

```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, predictions)
accu = r2_score(y_test, predictions)
print("MSE:", mse)
print("Accuracy:", accu)
```

And visualize the results using Matplotlib:

```
y_pred_line = regressor.predict(X)
fig = plt.figure(figsize=(8, 6))
m1 = plt.scatter(X_train, y_train, color=cmap(0.9), s=10)
m2 = plt.scatter(X_test, y_test, color=cmap(0.5), s=10)
plt.plot(X, y_pred_line, color="black", linewidth=2, label="Prediction")
plt.show()
```

5.1 Linear regression loss function

The linear regression loss or cost function can be derived from **the maximum likelihood estimation approach**.

Suppose we have a dataset of n samples, denoted as $(x_i, y_i)_{i=1}^n$, where x_i is the i-th input or feature vector, and y_i is the corresponding output or target value. The goal of linear regression is to find a linear function $f(x_i) = w^T x_i + b$ that approximates the mapping from inputs to outputs.

We assume that **the target values are generated by adding some Gaussian noise to the true output values, that is, $y_i = f(x_i) + \epsilon_i$, where ϵ_i is a random variable following a Gaussian distribution with mean zero and variance σ^2.**

Then, the likelihood function of the model can be defined as the joint probability density function of the observed data, given the model parameters

$$\begin{aligned} L(w,b) &= p(y_1, y_2, ..., y_n | x_1, x_2, ..., x_n, w, b, \sigma^2) \\ = \prod_{i=1}^{n} p(y_i|x_i, w, b, \sigma^2) \ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^T x_i - b)^2}{2\sigma^2}\right) \end{aligned} \tag{4}$$

where $p(y_i|x_i, w, b, \sigma^2)$ is the probability density function of y_i, given x_i, w, b, and σ^2.

The maximum likelihood estimation seeks to find the values of w, b, and σ^2 that maximize the likelihood function $L(w, b)$. Instead of maximizing the product of the probabilities, we can maximize the log-likelihood function, which simplifies the calculation and avoids numerical underflow:

$$\begin{aligned} \log L(w,b) &= \sum_{i=1}^{n} \log p(y_i|x_i, w, b, \sigma^2) \\ = \sum_{i=1}^{n} \left(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y_i - w^T x_i - b)^2}{2\sigma^2}\right) \ &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - w^T x_i - b)^2 \end{aligned} \tag{5}$$

where the first term does not depend on w, b, and σ^2, and the second term is the negative sum of squared errors (SSE), which measures the discrepancy between the predicted values and the true values. **Therefore, maximizing the log-likelihood function is equivalent to minimizing the SSE, which is a common loss or cost function used in linear regression:**

$$\text{SSE} = \sum_{i=1}^{n}(y_i - w^T x_i - b)^2 \tag{6}$$

This loss function can be minimized using various optimization algorithms, such as gradient descent or normal equations, to obtain the optimal values of w and b that minimize the prediction error.

6 Additional information

6.1 The probability density function of a Gaussian distribution

The probability density function of a Gaussian distribution with mean μ and variance σ^2 is given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

When the mean μ is zero, the distribution is also called a standard normal distribution with variance σ^2.

For example, suppose we have a set of measurements of the heights of students in a class, and we assume that the heights follow a Gaussian distribution with mean 170 cm and variance 25 cm^2. Then the probability of a student having a height of 175 cm can be calculated using the above formula as:

$$f(175) = \frac{1}{\sqrt{2\pi \times 25}} \exp\left(-\frac{(175-170)^2}{2 \times 25}\right) \approx 0.129$$

This means that the probability of a student having a height of 175 cm is around 12.9

6.2 Varience

The variance formula for a Gaussian distribution with mean μ and variance σ^2 is:

$$\text{Var}(X) = \sigma^2$$

where X is a random variable following a Gaussian distribution with mean μ and variance σ^2. The variance represents the measure of how much the values of X are spread out from its mean μ.

For example, let's say we have a Gaussian distribution with mean $\mu = 5$ and variance $\sigma^2 = 4$. Then the formula for the probability density function (PDF) of the Gaussian distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Substituting the values of μ and σ^2, we get:

$$f(x) = \frac{1}{2\sqrt{\pi}} e^{-\frac{(x-5)^2}{8}}$$

This function describes the probability of observing a random variable X with values x that follow a Gaussian distribution with mean $\mu = 5$ and variance $\sigma^2 = 4$. The variance $\sigma^2 = 4$ represents how much the values of X are spread out from the mean $\mu = 5$ [1] [2].

References

[1] Andriy Burkov. *The Hundred-Page Machine Learning Book*.

[2] Trevor Hastie et al. *An Introduction to Statistical Learning*.