Machine Learning Notebook

The study group organized by Professor Yuhao Ge.

Yu Zhao

Last updated: December 17, 2023

Contents

❖ Chapter 3 (3/31)

1.1 Scalar-Vector Equation Derivatives

1.1.1 Scalar Equation

$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$, where $f(\vec{y})$ is a 1×1 scalar, $\vec{y}$ is an $m \times 1$ vector.

Denominator Layout. Number of rows is the same as the denominator.

$$\frac{\partial f(\vec{y})}{\partial \vec{y}} = \begin{bmatrix} \frac{\partial f(\vec{y})}{\partial y_1} \\ \frac{\partial f(\vec{y})}{\partial y_2} \\ \vdots \\ \frac{\partial f(\vec{y})}{\partial y_m} \end{bmatrix}_{m \times 1} \tag{1}$$

Numerator Layout. Number of columns is the same as the numerator.

$$\frac{\partial f(\vec{y})}{\partial \vec{y}} = \begin{bmatrix} \frac{\partial f(\vec{y})}{\partial y_1} & \frac{\partial f(\vec{y})}{\partial y_2} & \cdots & \frac{\partial f(\vec{y})}{\partial y_m} \end{bmatrix}_{1 \times m} \tag{2}$$

1.1.2 Vector Equation

$$\vec{f}(\vec{y}) = \begin{bmatrix} f_1(\vec{y}) \\ f_2(\vec{y}) \\ \vdots \\ f_n(\vec{y}) \end{bmatrix}_{n \times 1}, \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m \times 1} \tag{3}$$

$$\frac{\partial \vec{f}(\vec{y})_{n \times 1}}{\partial \vec{y}_{m \times 1}} = \begin{bmatrix} \frac{\partial f(\vec{y})}{\partial y_1} \\ \frac{\partial f(\vec{y})}{\partial y_2} \\ \vdots \\ \frac{\partial f(\vec{y})}{\partial y_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\vec{y})}{\partial y_1} & \frac{\partial f_2(\vec{y})}{\partial y_1} & \cdots & \frac{\partial f_n(\vec{y})}{\partial y_1} \\ \frac{\partial f_1(\vec{y})}{\partial y_2} & \frac{\partial f_2(\vec{y})}{\partial y_2} & \cdots & \frac{\partial f_n(\vec{y})}{\partial y_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1(\vec{y})}{\partial y_m} & \frac{\partial f_2(\vec{y})}{\partial y_m} & \cdots & \frac{\partial f_n(\vec{y})}{\partial y_m} \end{bmatrix} \tag{4}$$

$$\text{eg.} \quad \vec{f}(\vec{y}) = \begin{bmatrix} f_1(\vec{y}) \\ f_2(\vec{y}) \end{bmatrix} = \begin{bmatrix} y_1^2 + y_2^2 + y_3 \\ y_3^2 + 2y_1 \end{bmatrix}_{2 \times 1}, \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}_{3 \times 1} \tag{5}$$

$$\frac{\partial \vec{f}(\vec{y})_{n\times 1}}{\partial \vec{y}_{m\times 1}} = \begin{bmatrix} \frac{\partial f(\vec{y})}{\partial y_1} \\ \frac{\partial f(\vec{y})}{\partial y_2} \\ \frac{\partial f(\vec{y})}{\partial y_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\vec{y})}{\partial y_1} & \frac{\partial f_2(\vec{y})}{\partial y_1} \\ \frac{\partial f_1(\vec{y})}{\partial y_2} & \frac{\partial f_2(\vec{y})}{\partial y_2} \\ \frac{\partial f_1(\vec{y})}{\partial y_3} & \frac{\partial f_2(\vec{y})}{\partial y_3} \end{bmatrix} = \begin{bmatrix} 2y_1 & 2 \\ 2y_2 & 0 \\ 1 & 2y_3 \end{bmatrix}_{3\times 2} \tag{6}$$

1.1.3 Special Cases

Common special cases in matrix differentiation.

$\frac{\partial A\vec{y}}{\partial \vec{y}} = A^T$

$\frac{\partial \vec{y}^T A\vec{y}}{\partial \vec{y}} = A\vec{y} + A^T\vec{y} = 2A\vec{y}$ (if A is symmetric.)

1.2 Linear Regression

Find y_1, y_2 to minimize the cost function J.

$$J = \sum_{i=1}^{n} [z_i - (y_1 + y_2 x_i)]^2$$

$$\vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}, \vec{x} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Least Squares Estimation $\quad \hat{\vec{z}} = \vec{x}\vec{y} = \begin{bmatrix} y_1 + y_2 x_1 \\ y_1 + y_2 x_2 \\ \vdots \\ y_1 + y_2 x_n \end{bmatrix}$

$$\begin{aligned} \therefore J &= (\vec{z} - \hat{\vec{z}})^T \cdot (\vec{z} - \hat{\vec{z}}) \\ &= (\vec{z} - \vec{x} \cdot \vec{y})^T \cdot (\vec{z} - \vec{x} \cdot \vec{y}) \\ &= (\vec{z}^T - (\vec{x} \cdot \vec{y})^T) \cdot (\vec{z} - \vec{x} \cdot \vec{y}) \\ &= \vec{z}^T \cdot \vec{z} - 2 \cdot (\vec{x} \cdot \vec{y})^T \cdot \vec{z} + (\vec{x} \cdot \vec{y})^T \cdot (\vec{x} \cdot \vec{y}) \end{aligned}$$

Now, taking the derivative of J with respect to $\vec{y}$ and setting it to zero:

$$\frac{\partial J}{\partial \vec{y}} = -2 \cdot \vec{x}^T \cdot \vec{z} + 2 \cdot \vec{x}^T \cdot (\vec{x} \cdot \vec{y}) = 0$$

Solving for $\vec{y}$:

$$\vec{y} = (\vec{x}^T \cdot \vec{x})^{-1} \cdot \vec{x}^T \cdot \vec{z}$$

1.3 Chain Rule

For the scalar function $J = f(y(u))$:

$$\frac{\partial J}{\partial u} = \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial u}$$

When taking the derivative of a scalar with respect to a vector $J = f(\vec{y}(\vec{u}))$:

$$\frac{\partial J}{\partial \vec{u}} = \frac{\partial \vec{y}}{\partial \vec{u}} \cdot \frac{\partial f}{\partial \vec{y}}$$

❖ Chapter 4 (4/31)

2.1 Gradient Descent Principles

Gradient descent aims to iteratively minimize a given loss or cost function.

2.1.1 Algorithm Steps:

1. Define a loss function $J(x_0)$.
2. Choose an initial point x_0 with any precision.
3. Calculate the gradient at the starting point: $\nabla J(x_0) = \frac{\partial J(x_0)}{\partial x_0}$.
4. Set a learning rate η.
5. Compute the next point $x_1 = x_0 - \eta \nabla J(x_0)$.
6. Iterate the process.
7. Choose a very small number ε.
8. Stop the iteration when the condition $|J(x_1) - J(x_0)| < \varepsilon$ is satisfied.
9. The stopping point $J(x_1)$ is the minimum value.

Key Considerations

1. The gradient is a vector pointing in a direction where moving along that direction increases $J(x)$ the fastest, and moving in the opposite direction decreases it the fastest.
2. In problems aiming to find the minimum value, the objective is to make $J(x)$ decrease rapidly.
3. The absolute value of the gradient diminishes, and the distance between adjacent x values decreases.

2.1.2 Mathematical derivation

Using Taylor expansion, expand $J(x)$ at any point x_0:

$$J(x_1) = J(x_0) + (x_1 - x_0)\frac{\partial J(x_0)}{\partial x_0}$$

How to make $J(x_1) - J(x_0) \leq 0$? One simple approach is: $x_1 - x_0 = -\frac{\partial J(x_0)}{\partial x_0}$. In practical applications, it is necessary to introduce a step size adjustment:

$x_1 - x_0 = -\eta \frac{\partial J(x_0)}{\partial x_0}$

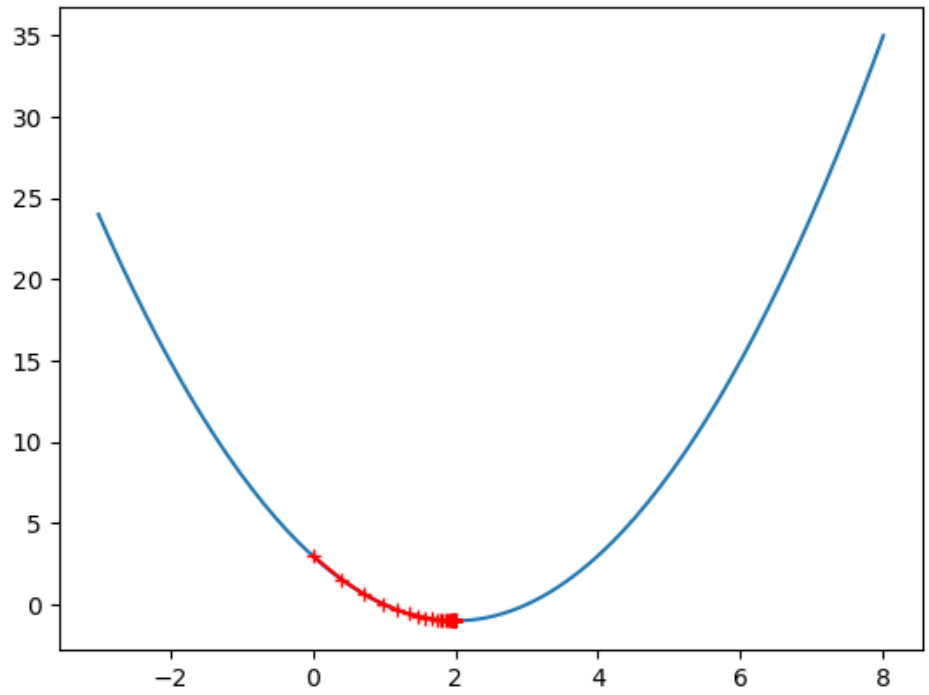
2.2 Programming Implementation in PyCharm

2.2.1 Bivariate Function

The code implementation of gradient descent for the function $y = (x-2)^2 + 1$ is as follows:

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

def dJ(theta):
    return 2 * (theta - 2)  # Calculate the gradient

def J(theta):
    try:
        return (theta - 2) ** 2 - 1
    except:
        return float('inf')

def gradient_descent(initial_theta, eta, theta_history=[],
   n_iters=1000, epsilon=1e-8):
    theta = initial_theta  # Initialize theta
    theta_history.append(initial_theta)
    i_iter = 0
    while i_iter < n_iters:
        gradient = dJ(theta)
        last_theta = theta
        theta = theta - eta * gradient
        theta_history.append(theta)
        # Termination condition
        if abs(J(theta) - J(last_theta)) < epsilon:
            break
        i_iter += 1
    return theta_history

def plot_theta_history(x, theta_history=[]):
```

```
    plt.plot(x, J(x))
    plt.plot(np.array(theta_history), J(np.array(theta_history)
   ), color='red', marker="+")
    plt.show()

# This is an example of a univariate function.
plot_x = np.linspace(-3, 8, 201)  # Prepare data
plot_y = (plot_x - 2) ** 2 - 1  # Define the function form
theta = 0.0
eta = 1
theta_history = []
theta_history = gradient_descent(0, eta)  # Call the gradient
   descent method
print("The function achieves the minimum value at x = %s!" %
   theta_history[-1])
print("The minimum value of the function is: %s" % J(
   theta_history[-1]))
plot_theta_history(plot_x, theta_history)
```



❖ Chapter 5 (5/31)

The main contents of this chapter are as follows:

(1) The principle of linear regression, i.e., why minimize the sum of squared residuals?

(2) Matrix representation formula for the sum of squared residuals in linear regression.

(3) Matrix representation formula for the derivative (or gradient) of the sum of squared residuals.

(4) Based on the above two matrices, one can write code for gradient descent to solve regression coefficients. The focus is on the iterative formula for coefficients.

(5) We validate the diversity of data used in the code. If the absolute differences of variables in X (including the intercept term) are not significant, the results are relatively accurate. However, if the differences are large, there may be cases where the absolute values of the sum of squared residuals are too large. This is because it is a sum of squares. Generally, the approach is to divide it by the length of the data. If the absolute value of the sum of squared residuals is too large, the absolute value of the gradient may also be relatively large. The general approach is to divide the gradient by the length of the data. Additionally, adjusting the learning rate to be very small may be necessary. After making these adjustments, it may lead to less accurate estimates of the intercept coefficient.

(6) Through the analysis in (5), we can understand why in future machine learning, data normalization is generally performed.

Below is my assignment code.

```
import numpy as np
from sklearn.model_selection import train_test_split

class LinearRegression:
    def __init__(self):
        self._theta = None
        self.intercept_ = None
        self.coef_ = None

    def fit_gd(self, X_train, y_train, eta=0.01, n_iters=10000)
   :
        X_b = np.c_[np.ones((len(X_train), 1)), X_train]
        initial_theta = np.zeros(X_b.shape[1])
```

```
        def J(theta, X_b, y):
            return ((y - X_b.dot(theta)).T @ (y - X_b.dot(theta
    ))) / len(X_b)

        def DJ(theta, X_b, y):
            return X_b.T.dot(X_b.dot(theta) - y) * 2 / len(X_b)

        def gradient_descent(X_b, y, initial_theta, eta,
    n_iters, epsilon=1e-8):
            theta = initial_theta
            for _ in range(n_iters):
                last_theta = theta
                theta = theta - eta * DJ(theta, X_b, y)

                if abs(J(theta, X_b, y) - J(last_theta, X_b, y)
    ) < epsilon:
                    break

            return theta

        self._theta = gradient_descent(X_b, y_train,
    initial_theta, eta, n_iters)
        self.intercept_ = self._theta[0]
        self.coef_ = self._theta[1:]

        return self

     def predict(self, X_predict):
        assert self._theta is not None
        assert X_predict.shape[1] == len(self._theta) - 1

        X_b = np.c_[np.ones((len(X_predict), 1)), X_predict]
        return X_b.dot(self._theta)

# Generate simulated data
np.random.seed(666)
x = 2 * np.random.random(size=100)
y = 4.0 + 3.0 * x + np.random.normal(size=100)
X = x.reshape(-1, 1)
```

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Fit using gradient descent
reg = LinearRegression()
reg.fit_gd(X_train, y_train)
print("Coefficients:", reg.coef_)
print("Intercept:", reg.intercept_)
```

❖ Chapter 6 (6/31)

Let y be a random variable with a probability density function dependent on parameters θ (a vector, possibly with multiple parameters, e.g., mean μ and variance σ). Then, the probability of observing a sequence of numbers $y_1, y_2, y_3, \ldots, y_n$ is given by:

$$L(y|\theta) = \prod_{i=1}^{n} f(y_i, \theta)$$

Here, the expression $L(\theta)$ is called the likelihood function. Given a specific set of parameters θ, it becomes a concrete numerical value. Now, consider the inverse problem where we lack information about θ but possess information about f. How can we infer θ using this information?

- We can express the likelihood function mentioned above as a function of θ:

$$L(\theta|y, f) = \prod_{i=1}^{n} f(y_i|\theta)$$

- A natural idea is that θ, which maximizes the above function, is the one we seek. Denote this optimal θ as $\hat{\theta}_{MLE}$:

$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta|y, f)$$

Solution Approach:

- To find the solution, the following equation is commonly employed:

$$\frac{\partial}{\partial\theta} L(\theta, y) = 0$$

- Another common form is expressed using the logarithm:

$$\frac{\partial}{\partial\theta} \ln(L(\theta, y)) = g(\theta) = 0$$

This expression is often referred to as the likelihood equation and is typically expressed in the form of a summation.

$$g(\theta) = \sum_{i=1}^{n} g_i(\theta)$$

where $g(\theta)$ and $g_i(\theta)$ are random vectors.

References

[1] Chapter 3. DRCAN. *Matrix Differentiation*. Link.

[2] Chapter 4. gyhccer. *Gradient Descent*. Link.