INT 6151: Statistical Machine Learning - Homework 1

Vũ Quang Sơn

March 2024

1 Gradient Descent

Let $f : \mathbb{R}^d \to \mathbb{R}$, start at some points $x^{(0)} = x_1^{(0)}, ..., x_d^{(0)}$ with k = 0. At the k step we have updated rule:

$$x_i^{k+1} \leftarrow x_i^k - \lambda \partial_{x_i} f(x^k)$$

with $\partial_{x_i} f : \mathbb{R}^d \to \mathbb{R}$ is the derivative of the function with respect to the ith coordinate, $\partial_{x_i} f(x^k)$ is the value at x^k and λ is the learning rate.

1.1

Let $f(x) = x^2 - x + 6$ and $\lambda = 0.05$.
First, calculate $\partial_x f$. Then starting from $x^{(0)} = 1 (k = 0)$, apply the gradient descent with $k = 1, 2, 3$.

We have first derivative of $f(x)$ is $\partial_x f = 2x - 1$
Initializing $k = 0, x^{(0)} = 1 \Rightarrow f(x^{(0)}) = 6$

- After updating $k = 0$:

  $\partial_x f(x^{(0)}) = 2 * 1 - 1 = 1$

  $x^{(1)} = 1 - 0.05 * 1 = 0.95$

  $f(x^{(1)}) = 5.9525$

- After updating $k = 1$:

  $\partial_x f(x^{(1)}) = 2 * 0.95 - 1 = 0.9$

  $x^{(2)} = 0.95 - 0.05 * 0.9 = 0.905$

  $f(x^{(2)}) = 5.914025$

- After updating $k = 2$:

  $\partial_x f(x^{(2)}) = 2 * 0.905 - 1 = 0.81$

$x^{(3)} = 0.905 - 0.05 * 0.81 = 0.8645$

$f(x^{(3)}) = 5.88286025$

- After updating $k = 3$:

  $\partial_x f(x^{(3)}) = 2 * 0.8645 - 1 = 0.729$

  $x^{(4)} = 0.8645 - 0.05 * 0.729 = 0.82805$

  $f(x^{(4)}) = 5.8576168025$

1.2

Let $f(x) = (x_1 - x_2^2 - 3)^2 + (x_1 - x_2 - 1)^2$ and $\lambda = 0.2$. First, calculate $\partial_{x_i} f$. Then starting from $x^{(0)} = (-0.5, -0.5)(k = 0)$, apply the gradient descent with $k = 1, 2$. How many local maxima of f? Could you find them?

The first derivative of $f(x)$:

- $\partial_{x_1} f = 2(x_1 - x_2^2 - 3) + 2(x_1 - x_2 - 1)$
- $\partial_{x_2} f = -4x_2(x_1 - x_2^2 - 3) - 2(x_1 - x_2 - 1)$

Initialing $x^{(0)} = (-0.5, -0.5), k = 0, learning_rate = 0.2, f(x^{(0)})$

- After $k = 0$:

  $\partial_{x_1} f(x_0) = -9.5$

  $\partial_{x_2} f(x_0) = -5.5$

  $x^{(1)} = [1.4, 0.6]$

  $f(x^{(1)}) = 15.0625$

- After $k = 1$:

  $\partial_{x_1} f(x_1) = -4.32$

  $\partial_{x_2} f(x_1) = 5.104$

  $x^{(2)} = [2.264, -0.4208]$

  $f(x^{(2)}) = 3.672$

- After $k = 2$:

  $\partial_{x_1} f(x_2) = 1.543$

  $\partial_{x_2} f(x_2) = -4.906$

  $x^{(3)} = [1.955, 0.560]$

  $f(x^{(3)}) = 2.002$

- After $k = 3$:

$\partial_{x_1} f(x_3) = -1.928$

$\partial_{x_2} f(x_3) = 2.257$

$x^{(4)} = [2.341, 0.109]$

$f(x^{(4)}) = 1.968$

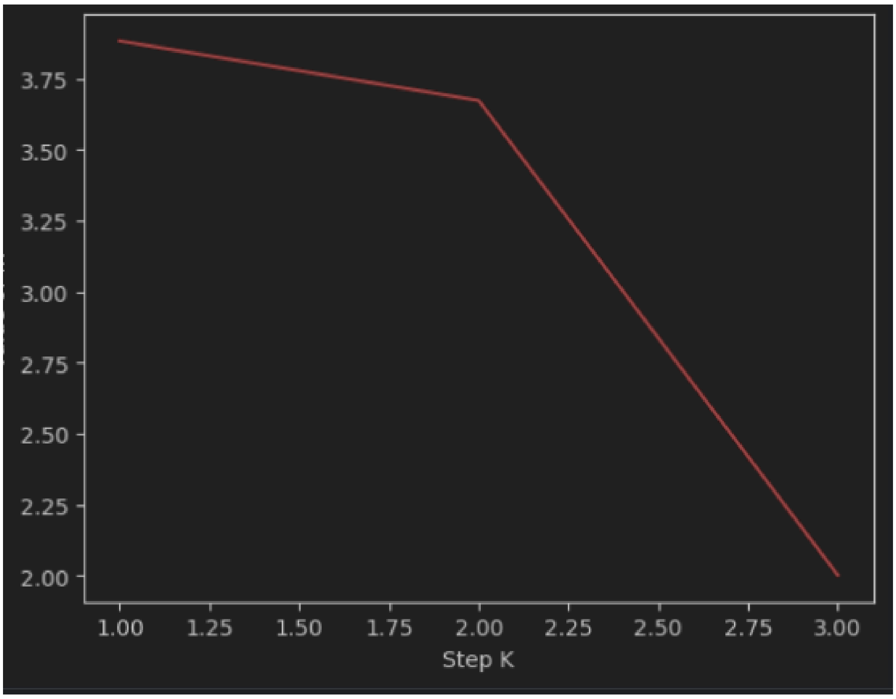

Figure 1: f(x) after every epoch

To find the function's local maximum, we need to solve:

$$\frac{\partial f}{\partial x_1} = 0$$

$$\frac{\partial f}{\partial x_2} = 0$$

The result is: $x = [\frac{19}{8}, \frac{1}{2}]$ We have the Hessian matrix of $f(x)$:

$$A = \partial_{x_1 x_1} f(x_s)$$

$$B = \partial_{x_1 x_2} f(x_s)$$

$$C = \partial_{x_2 x_2} f(x_s)$$

Because $A * C - B^2 = 44 > 0$ and $A > 0 \Rightarrow f(x)$ has one local minimum at $x = [\frac{19}{8}, \frac{1}{2}]$

1.3

Apply the gradient descent with $\lambda = 0.01$ and $\lambda = 0.05$ respectively, and draw the contour plot of $f(x)$ after 10 steps

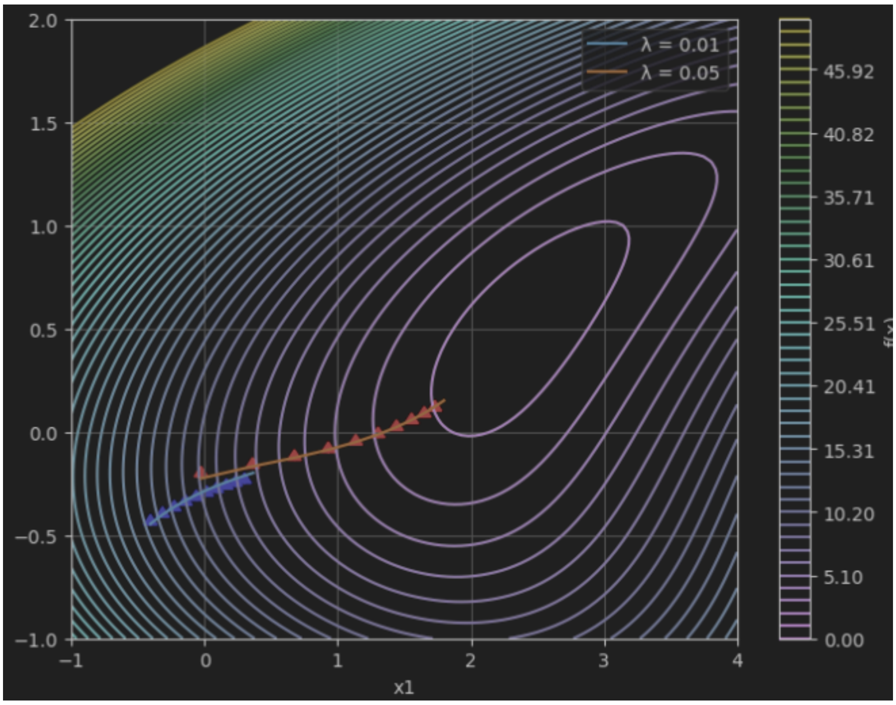

Figure 2: Gradient Descent with $\lambda = 0.01$ and $\lambda = 0.05$

Figure 2 shows that $\lambda = 0.05$ helps the function converge faster than $\lambda = 0.01$. Therefore, $\lambda = 0.05$ is the better for optimization.

2 Coding

Source code for generating data and model

```
def add_noise_data(input_data, input_labels, n_points, mean, scale):
    """
    Create a noise version of the input data

    Params:
        input_data: base input data
        input_labels: base input labels
        n_points: the number of needed points
        mean, scale: the Gaussian data
    """
```

```
    raw_X = []
    raw_labels = []

    noise = np.random.normal(loc=mean, scale=scale, size=(n_points, 2))
    for i in range(n_points):
        k = np.random.randint(len(input_data))

        raw_X.append([input_data[k][0] + noise[i][0],
                      input_data[k][1] + noise[i][1]])

        raw_labels.append(input_labels[k])

    return np.array(raw_X), np.array(raw_labels)

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=10000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def binary_cross_entropy_loss(self, y, y_pred):
        return -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))

    def error_calculation(self, y_pred, y_true):
        n = y_pred.shape[0]
        y_pred[np.where(y_pred >= 0.5)] = 1
        y_pred[np.where(y_pred < 0.5)] = 0
        return 1 / n * np.sum(np.square(y_pred - y_true))

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for i in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)
            current_loss = self.binary_cross_entropy_loss(y, y_predicted)
            error = self.error_calculation(y_predicted, y)
            print(f"Epoch {i + 1}: Loss = {current_loss}, weights = {self.weights}, bias
```

```
            if error == 0:
                print(f"Done")
                break

            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls
```

2.1

We have logistic function h which satisfies the condition $er\hat{r}_D(h) = 0$:

$$P\{|er\hat{r}_D(h) - err_{\mathbb{P}}(h)| \leq \epsilon\} \geq 1 - 2e^{-2n\epsilon^2}$$

$$\Leftrightarrow P\{| - err_{\mathbb{P}}(h)| \leq \epsilon\} \geq 1 - 2e^{-2n\epsilon^2}$$

$$\Leftrightarrow P\{err_{\mathbb{P}}(h) \leq \epsilon\} \geq 1 - 2e^{-2n\epsilon^2}$$

With $\epsilon = 0.1, n = 10$:

$$\Rightarrow P\{err_{\mathbb{P}}(h) \leq 0.1\} \geq 1 - 2e^{-2*10*0.1^2} = -0.637$$

The above shows that $er\hat{r}_D(h) = 0$ with $P\{err_{\mathbb{P}}(h) > 0.1 = 0\}$ or we can say this does not happen.

2.2

Seed	Number of Epoches
0	8
10	16
20	9
30	14
40	6

Table 1: The number of epochs to converge respectively to seed

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
seeds = [0, 10, 20, 30, 40]
for seed in seeds:
    print(f"Seed {seed}")
    np.random.seed(seed)
    X_noise, y_noise = add_noise_data(X, y, 10, 0., 0.2)
    model = LogisticRegression()
    model.fit(X_noise, y_noise)
```

2.3

When the sample size increases, the time for the algorithm to classify increases and can not be separable.

Samples	Seed	Number of Epoches
20	0	10
	10	21
	20	5
	30	9
	40	8
50	0	26
	10	11
	20	Failed
	30	10
	40	11
100	0	Failed
	10	Failed
	20	Failed
	30	Failed
	40	Failed
200	0	Failed
	10	Failed
	20	Failed
	30	Failed
	40	Failed
500	0	Failed
	10	Failed
	20	Failed
	30	Failed
	40	Failed

Table 2: The number of epochs to converge respectively to samples, seed

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
seeds = [0, 10, 20, 30, 40, 50]
N = [20, 50, 100, 200, 500]
for n in N:
    print("N = {0}".format(n))
    for seed in seeds:
        print(f"Seed {seed}")
        np.random.seed(seed)
        X_noise, y_noise = add_noise_data(X, y, n, 0., 0.2)
        model = LogisticRegression()
        model.fit(X_noise, y_noise)
```

2.4

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])
seeds = [0, 42, 100, 200]
N = [10, 20]
for n in N:
    print("N = {0}".format(n))
    for seed in seeds:
        print(f"Seed {seed}")
        np.random.seed(seed)
        X_noise, y_noise = add_noise_data(X, y, n, 0., 0.2)
        X_test, y_test = add_noise_data(X, y, n, 0., 0.3)
        model = LogisticRegression()
        model.fit(X_noise, y_noise)
        y_pred = model.sigmoid(np.dot(X_test, model.weights) + model.bias)
        print(f"Testing ERR: {model.error_calculation(y_test, y_pred)}")
```

Samples	Seed	Number of Epoches	Training Error	Testing Error
10	0	8	0	0.2492
	42	19	0	0.2487
	100	6	0	0.2495
	200	10	0	0.2493
20	0	10	0	0.2493
	42	9	0	0.2494
	100	12	0	0.2495
	200	11	0	0.2492

Table 3: Result of 2.3