20154652_DongJaeLee_assignment_02_assignment_02_b

September 17, 2020

Linear supervised regression

0. Import library

Import library

[ ]:
```
# Import libraries

# math library
import numpy as np

# visualization library
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png2x','pdf')
import matplotlib.pyplot as plt

# machine learning library
from sklearn.linear_model import LinearRegression

# 3d visualization
from mpl_toolkits.mplot3d import axes3d

# computational time
import time
```

1. Load dataset

Load a set of data pairs $\{x_i, y_i\}_{i=1}^n$ where x represents label and y represents target.

[ ]:
```
# import data with numpy
data = np.loadtxt('profit_population.txt', delimiter=',')
```

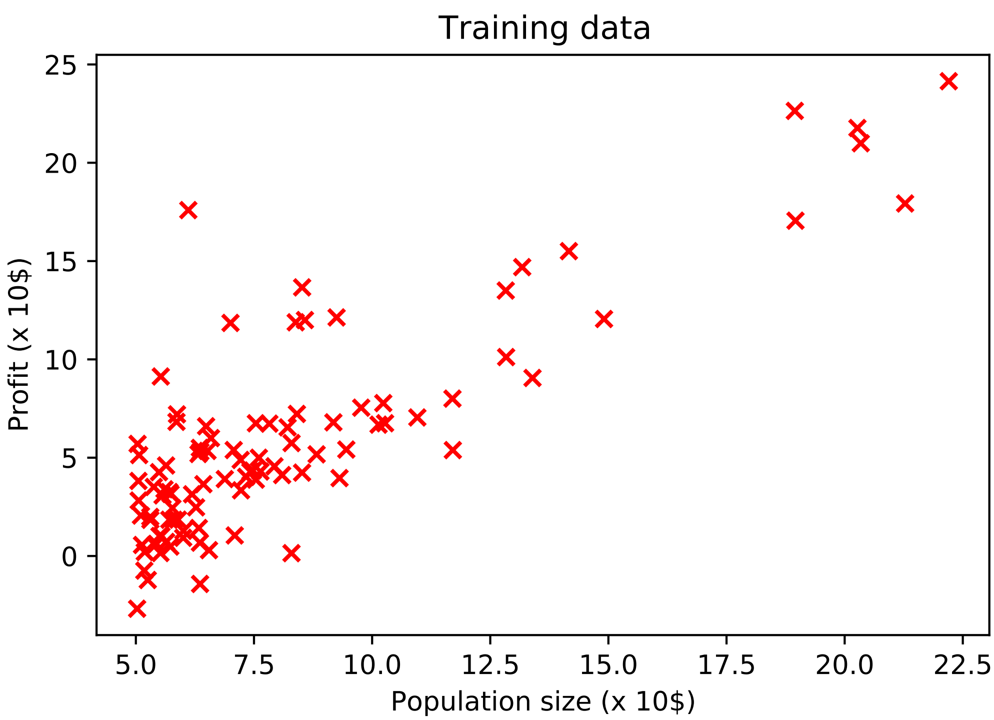
2. Explore the dataset distribution

Plot the training data points.

[ ]:
```
x_train = data[:,0]
y_train = data[:,1]
```

```
plt.scatter(x_train, y_train, c='red', marker='x')
plt.xlabel('Population size (x 10$)')
plt.ylabel('Profit (x 10$)')
plt.title('Training data')
```

[ ]: Text(0.5, 1.0, 'Training data')



3. Define the linear prediction function

$$f_w(x) = w_0 + w_1 x$$

0.0.1 Vectorized implementation:

$$f_w(x) = Xw$$

with

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \Rightarrow \quad f_w(x) = Xw = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix}$$

Implement the vectorized version of the linear predictive function.

[ ]:
```python
# construct data matrix
X = np.hstack((np.ones((len(x_train), 1)), x_train.reshape(len(x_train), 1)))

# parameters vector
w = np.array([
  [5],
  [5]
])

# predictive function definition
def f_pred(X,w):

    f = np.dot(X,w)

    return f

# Test predicitive function
y_pred = f_pred(X,w)
```

4. Define the linear regression loss

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(f_w(x_i) – y_i \right)^2$$

0.0.2 Vectorized implementation:

$$L(w) = \frac{1}{n}(Xw - y)^T(Xw - y)$$

with

$$Xw = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Implement the vectorized version of the linear regression loss function.

[ ]:
```python
# loss function definition
def loss_mse(y_pred,y):

    diff = y_pred - y.reshape(len(y), 1)
    loss = np.dot(diff.T, diff) / len(y)

    return loss


# Test loss function
y = y_train# label
```

```
y_pred = f_pred(X,w)# prediction

loss = loss_mse(y_pred,y)
```

5. Define the gradient of the linear regression loss

0.0.3 Vectorized implementation: Given the loss

$$L(w) = \frac{1}{n}(Xw - y)^T(Xw - y)$$

The gradient is given by

$$\frac{\partial}{\partial w}L(w) = \frac{2}{n}X^T(Xw - y)$$

Implement the vectorized version of the gradient of the linear regression loss function.

[ ]:
```
# gradient function definition
def grad_loss(y_pred,y,X):

    diff = y_pred - y.reshape(len(y), 1)
    loss = np.dot(diff.T, diff) / len(y)
    grad = (np.dot(X.T, diff)*2) / len(y)
    return grad


# Test grad function
y_pred = f_pred(X,w)
grad = grad_loss(y_pred,y,X)
```

6. Implement the gradient descent algorithm

- Vectorized implementation:

$$w^{k+1} = w^k - \tau\frac{2}{n}X^T(Xw^k - y)$$

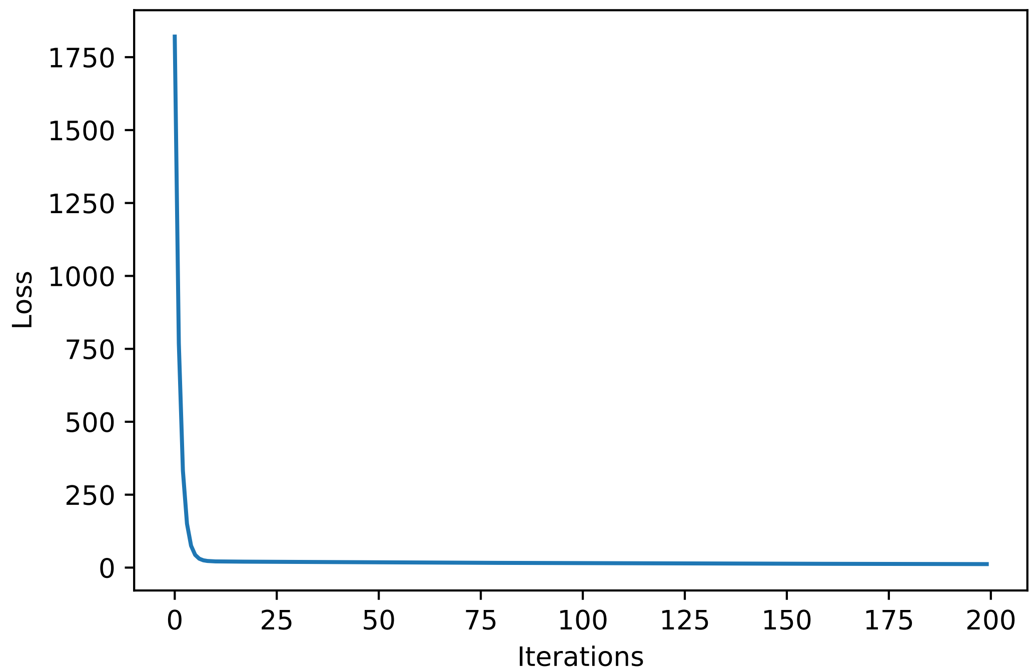
0.0.4 Implement the vectorized version of the gradient descent function.

0.0.5 Plot the loss values $L(w^k)$ with respect to iteration k the number of iterations.

[ ]:
```
# gradient descent function definition
def grad_desc(X, y, w_init, tau, max_iter):

    L_iters = np.zeros(max_iter)# record the loss values
    w_iters = np.zeros((max_iter, w_init.shape[0], w_init.shape[1]))# record
↪the parameter values
    w = w_init # initialization

    for i in range(max_iter): # loop over the iterations
```

```
        y_pred = f_pred(X, w)# linear predicition function
        grad_f = grad_loss(y_pred, y, X)# gradient of the loss
        w = w - tau*grad_f# update rule of gradient descent
        L_iters[i] = loss_mse(y_pred, y)# save the current loss value
        w_iters[i,:] = w# save the current w value

    return w, L_iters, w_iters


# run gradient descent algorithm
start = time.time()
w_init = w
tau = 0.01
max_iter = 200

w, L_iters, w_iters = grad_desc(X,y,w_init,tau,max_iter)
print('Time=',time.time() - start) # plot the computational cost
print(L_iters[-1]) # plot the last value of the loss
print(w_iters[-1]) # plot the last value of the parameter w

# plot
plt.figure(2)
plt.plot(L_iters) # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```

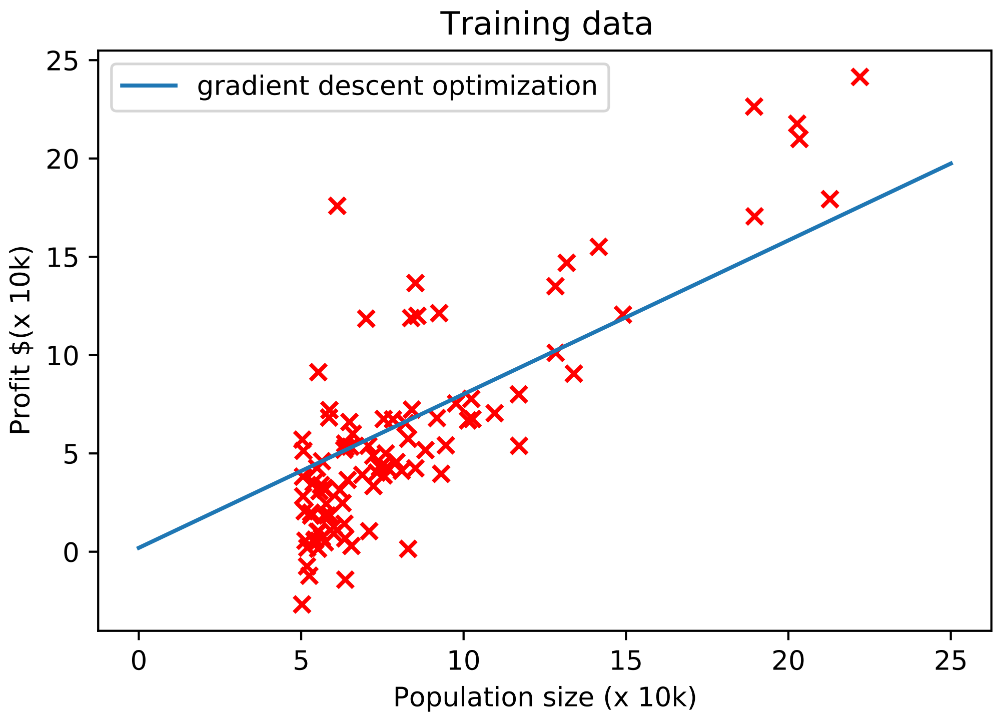
```
Time= 0.0034089088439941406
12.026186534258212
[[0.19710434]
 [0.78185893]]
```



7. Plot the linear prediction function

$$f_w(x) = w_0 + w_1 x$$

[ ]:
```
# linear regression model
x_pred = np.linspace(0,25,100) # define the domain of the prediction function
x_pred = np.hstack((np.ones((len(x_pred), 1)), x_pred.reshape(len(x_pred), 1)))
y_pred = f_pred(x_pred, w_iters[-1])# compute the prediction values within the␣
↪given domain x_pred

# plot
plt.figure(3)
plt.scatter(x_train, y_train, c='red', marker='x')
plt.plot(x_pred[:,1], y_pred, label='gradient descent optimization')
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```



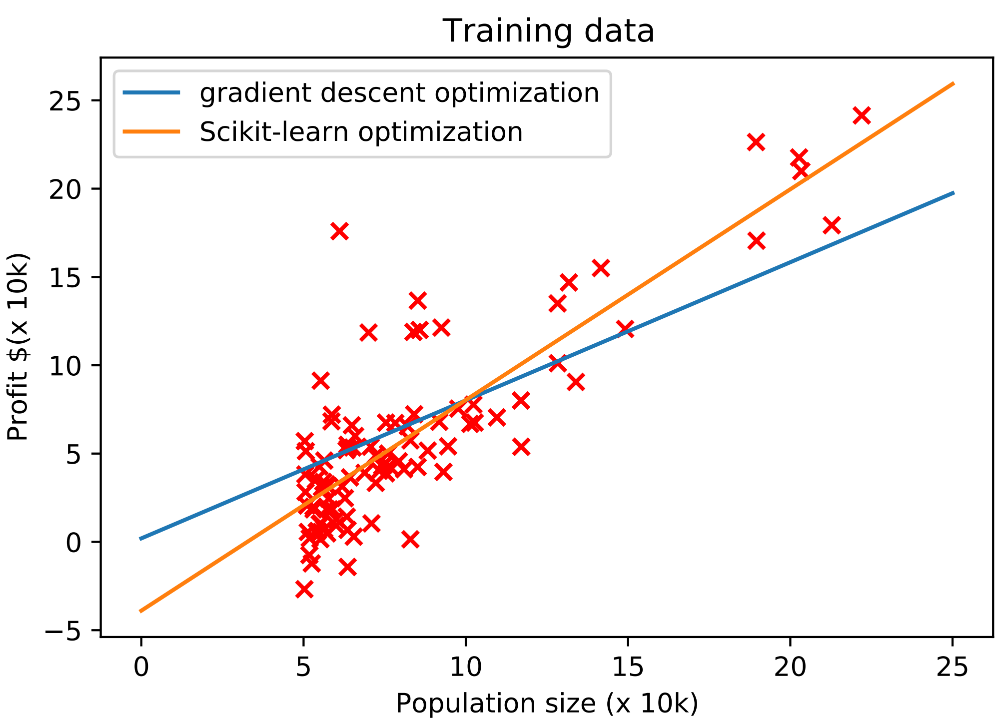
8. Comparison with Scikit-learn linear regression algorithm

0.0.6 Compare with the Scikit-learn solution

[ ]:
```
# run linear regression with scikit-learn
start = time.time()
lin_reg_sklearn = LinearRegression()
lin_reg_sklearn.fit(x_train.reshape(len(x_train), 1), y_train) # learn the␣
↪model parameters
print('Time=',time.time() - start)


# compute loss value
w_sklearn = np.zeros([2,1])
w_sklearn[0,0] = lin_reg_sklearn.intercept_
w_sklearn[1,0] = lin_reg_sklearn.coef_

print(w_sklearn)
loss_sklearn = loss_mse(np.dot(X, w_sklearn), y_train) # compute the loss from␣
↪the sklearn solution

print('loss sklearn=',loss_sklearn)
print('loss gradient descent=',L_iters[-1])
```

```
# plot
y_pred_sklearn = np.dot(x_pred, w_sklearn)# prediction obtained by the sklearn␣
↪library

plt.figure(3)

plt.scatter(x_train, y_train, c='red', marker='x')
plt.plot(x_pred[:,1], y_pred, label='gradient descent optimization')
plt.plot(x_pred[:,1], y_pred_sklearn, label='Scikit-learn optimization')
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
Time= 0.0005309581756591797
[[-3.89578088]
 [ 1.19303364]]
loss sklearn= [[8.95394275]]
loss gradient descent= 12.026186534258212
```

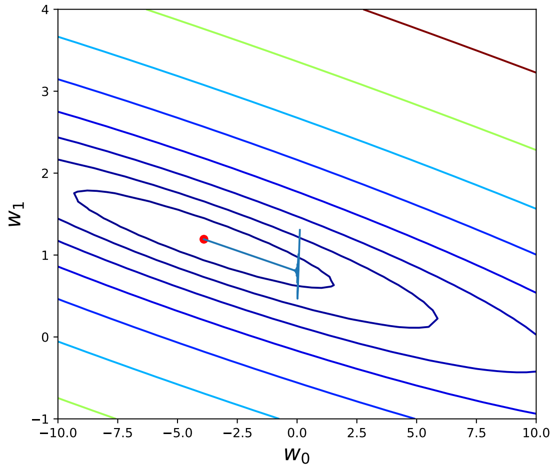


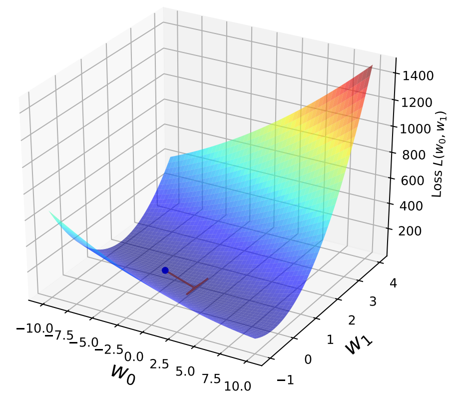
9. Plot the loss surface, the contours of the loss and the gradient descent steps

[ ]:
```python
# plot gradient descent
def plot_gradient_descent(X,y,w_init,tau,max_iter):

    def f_pred(X,w):

        f = np.dot(X,w)

        return f

    def loss_mse(y_pred,y):

        diff = y_pred - y.reshape(len(y), 1)
        loss = np.dot(diff.T, diff) / len(y)

        return loss

    # gradient descent function definition
    def grad_desc(X, y, w_init, tau, max_iter):

        L_iters = np.zeros(max_iter)# record the loss values
        w_iters = np.zeros((max_iter, w_init.shape[0], w_init.shape[1]))#␣
↪record the parameter values
        w = w_init # initialization

        for i in range(max_iter): # loop over the iterations

            y_pred = f_pred(X, w)# linear predicition function
            grad_f = grad_loss(y_pred, y, X)# gradient of the loss
            w = w - tau*grad_f# update rule of gradient descent
            L_iters[i] = loss_mse(y_pred, y)# save the current loss value
            w_iters[i,:] = w# save the current w value

        return w, L_iters, w_iters

    # run gradient descent
    w, L_iters, w_iters = grad_desc(X, y, w_init, tau, max_iter)
    # Create grid coordinates for plotting a range of L(w0,w1)-values
    B0 = np.linspace(-10, 10, 50)
    B1 = np.linspace(-1, 4, 50)

    xx, yy = np.meshgrid(B0, B1, indexing='xy')
    Z = np.zeros((B0.size,B1.size))
    # Calculate loss values based on L(w0,w1)-values
    for (i,j),v in np.ndenumerate(Z):
        tmp = np.array([xx[i,j], yy[i,j]])
        tmp = tmp.reshape(len(tmp), 1)
        Z[i,j] = loss_mse(np.dot(X, tmp), y)
```

```python
    # 3D visualization
    fig = plt.figure(figsize=(15,6))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122, projection='3d')

    # Left plot
    CS = ax1.contour(xx, yy, Z, np.logspace(-2, 3, 20), cmap=plt.cm.jet)
    ax1.scatter(w[0], w[1], c='r')
    ax1.plot(w_iters[:,0,0], w_iters[:,1,0])

    # Right plot
    ax2.plot_surface(xx, yy, Z, rstride=1, cstride=1, alpha=0.6, cmap=plt.cm.
 ↪jet)
    ax2.set_zlabel('Loss $L(w_0,w_1)$')
    ax2.set_zlim(Z.min(),Z.max())

    # plot gradient descent
    Z2 = np.zeros([max_iter])

    for i in range(max_iter):
        w0 = w_iters[i,0]
        w1 = w_iters[i,1]
        tmp_w = np.array([w0, w1])
        Z2[i] = loss_mse(np.dot(X, tmp_w), y)

    ax2.plot(w_iters[:,0,0], w_iters[:,1,0])
    ax2.scatter(w[0], w[1], c='b')

    # settings common to both plots
    for ax in fig.axes:
        ax.set_xlabel(r'$w_0$', fontsize=17)
        ax.set_ylabel(r'$w_1$', fontsize=17)
```

[ ]:
```python
# run plot_gradient_descent function
w_init = np.array([
  [0],
  [0]
])
tau = 0.01
max_iter = 50000

plot_gradient_descent(X,y,w_init,tau,max_iter)
```



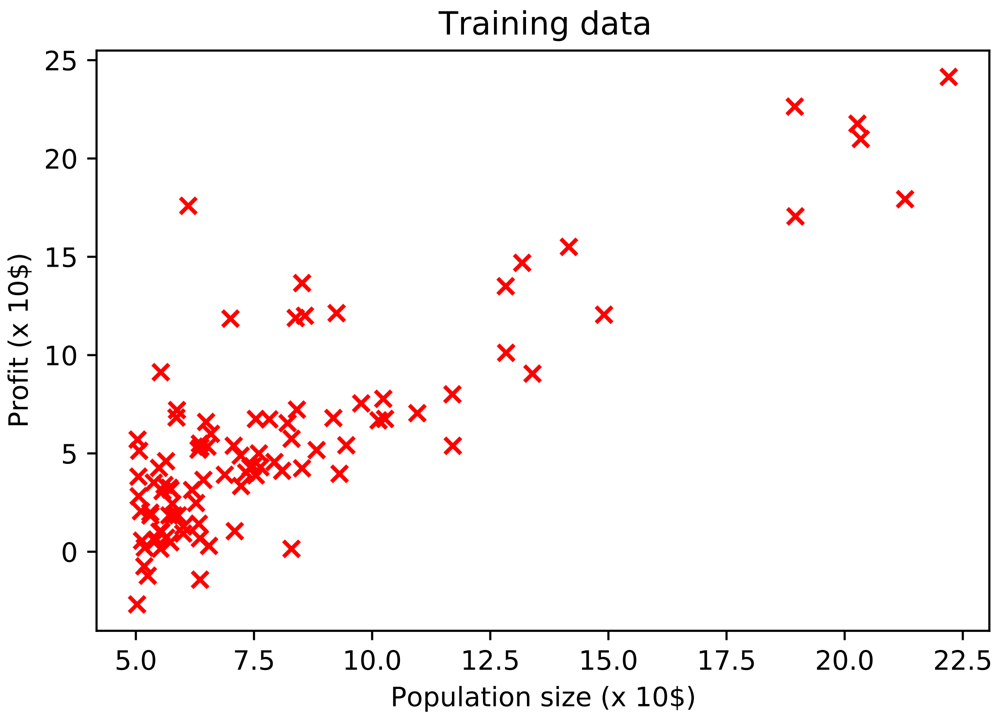


1 Output results

1.1 1. Plot the training data (1pt)

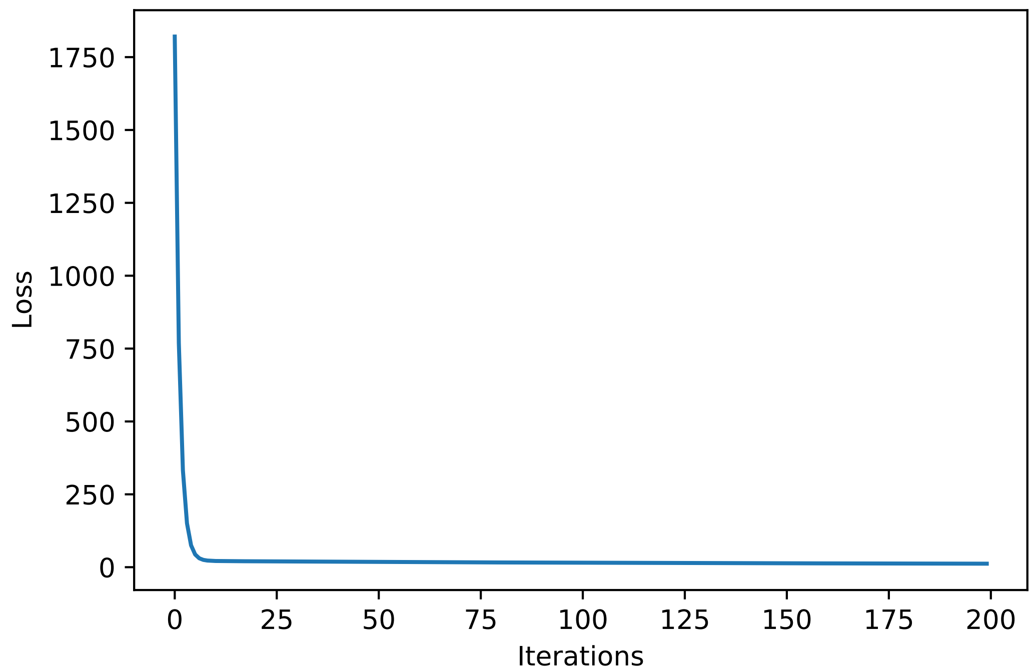
[ ]:
```
plt.scatter(x_train, y_train, c='red', marker='x')
plt.xlabel('Population size (x 10$)')
plt.ylabel('Profit (x 10$)')
plt.title('Training data')
```

[ ]:
```
Text(0.5, 1.0, 'Training data')
```



1.2 2. Plot the loss curve in the course of gradient descent (2pt)

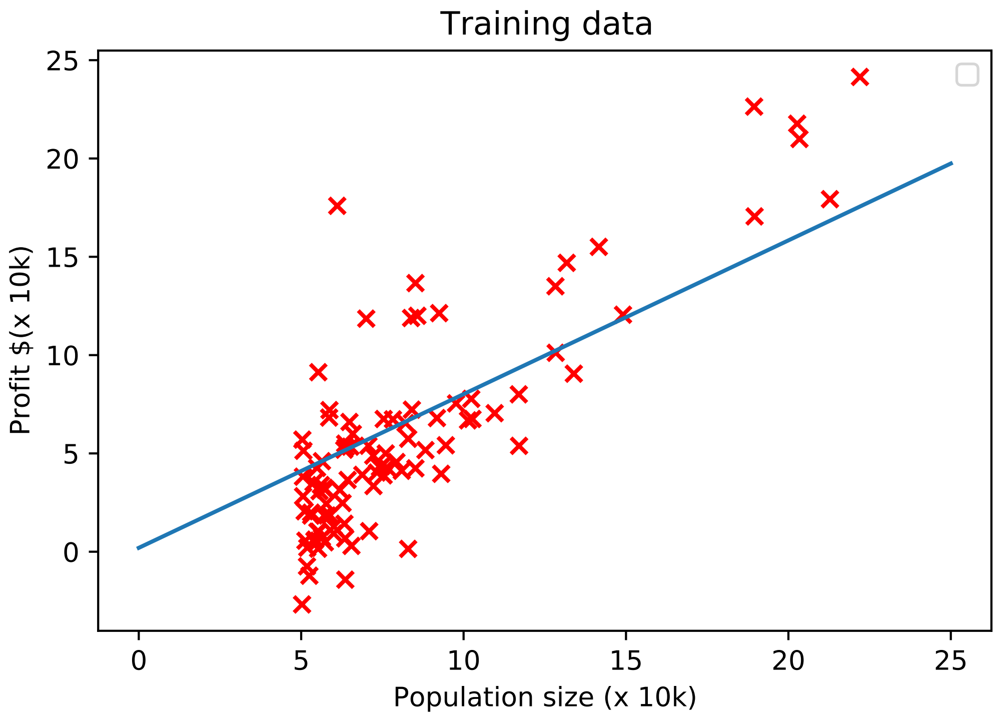
[ ]:
```
# plot
plt.figure(2)
plt.plot(L_iters) # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```



1.3 3. Plot the prediction function superimposed on the training data (2pt)

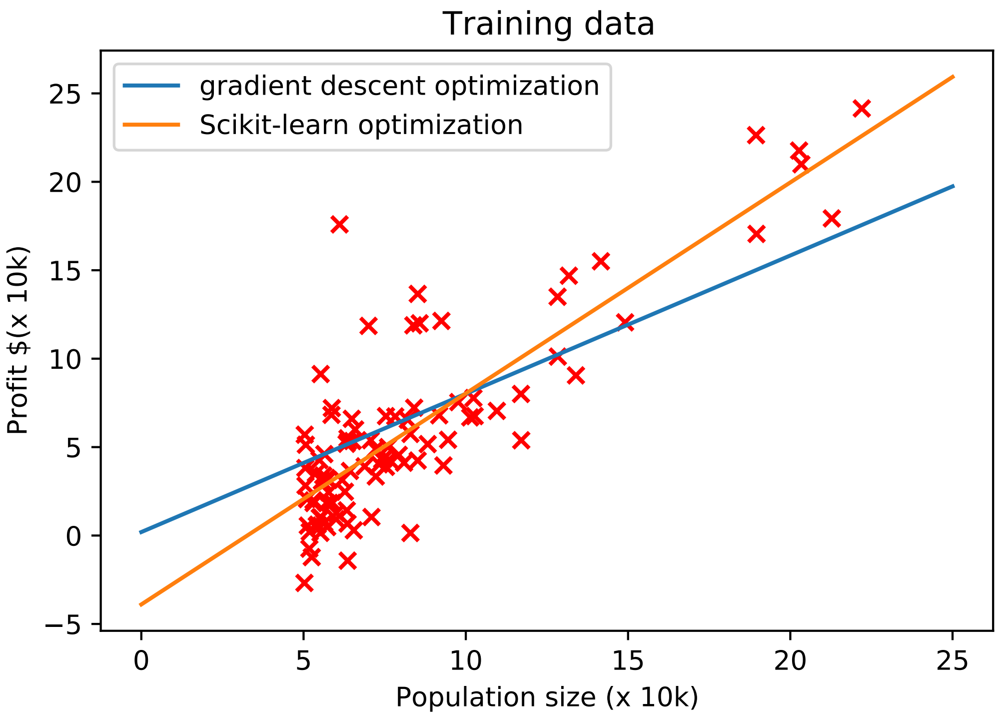
[ ]:
```
# plot
plt.figure(3)
plt.scatter(x_train, y_train, c='red', marker='x')
plt.plot(x_pred[:,1], y_pred)
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
No handles with labels found to put in legend.
```



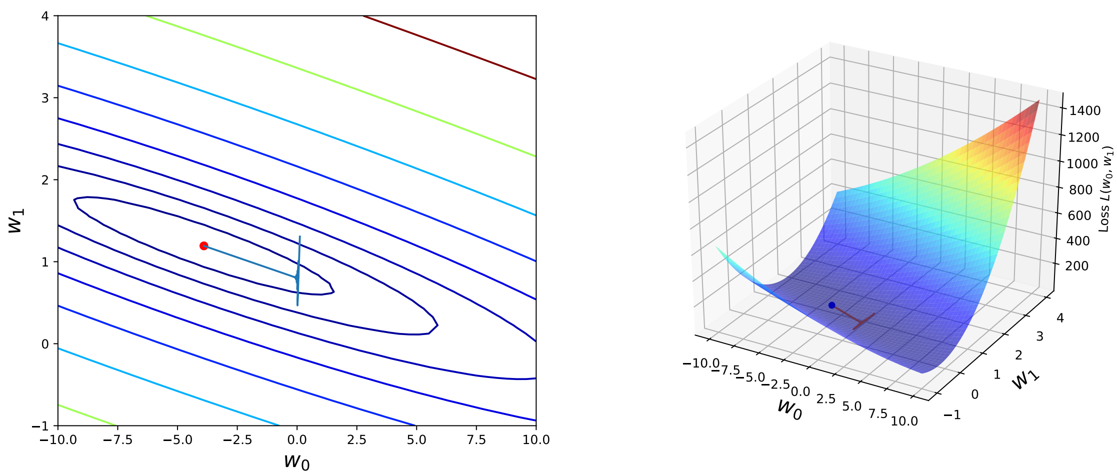
1.4 4. Plot the prediction functions obtained by both the Scikit-learn linear regression solution and the gradient descent superimposed on the training data (2pt)

[ ]:
```
plt.figure(3)
plt.scatter(x_train, y_train, c='red', marker='x')
plt.plot(x_pred[:,1], y_pred, label='gradient descent optimization')
plt.plot(x_pred[:,1], y_pred_sklearn, label='Scikit-learn optimization')
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```



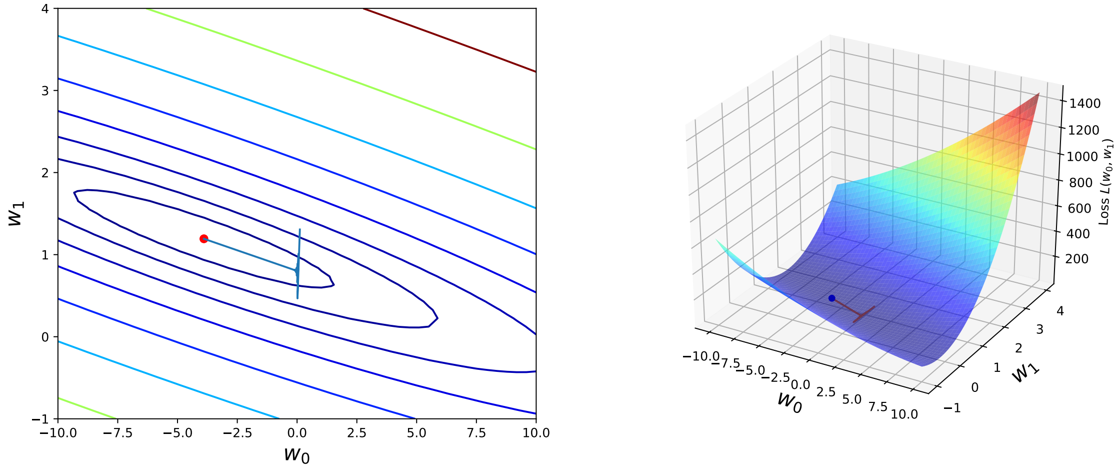
1.5 5. Plot the loss surface (right) and the path of the gradient descent (2pt)

[ ]:
```
plot_gradient_descent(X,y,w_init,tau,max_iter)
```



1.6 6. Plot the contour of the loss surface (left) and the path of the gradient descent (2pt)

[ ]:
```
plot_gradient_descent(X,y,w_init,tau,max_iter)
```



[ ]:
```
```