华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Jining He

Supervisor:
Qingyao Wu

Student ID:
201721045565

Grade:
Graduate

December 15, 2017

Linear Regression, Linear Classification and Gradient Descent

***Abstract*—In this experiment, we use gradient descent to optimize linear regression and linear classification on small dataset.**

I. Introduction

THIS experiment let us solve two problems. The first one requires us to predict the price of house. The second one requires us to distinguish whether or not Australians.

The Motivation of this experiment is let us further understand of linear regression and gradient descent, conduct some experiments under small scale dataset, realize the process of optimization and adjusting parameters.

II. Methods and Theory

In this experiment, we use linear regression to predict the price of house and use SVM as linear classifier to distinguish whether or not Australians.

A. Linear Regression

In linear regression, the loss function we used is:

$$J(W,b) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - (W^T x_i + b))^2 \tag{1}$$

The gradient formulas are:

$$\begin{aligned}\nabla_W J(W,b) &= \frac{1}{m}\sum_{i=1}^{m}(y_i - (x_i W + b))x_i \\ \nabla_b J(W,b) &= \frac{1}{m}\sum_{i=1}^{m}(y_i - (x_i W + b))\end{aligned} \tag{2}$$

B. Linear Classification

In linear classification, we add hinge loss to our loss function:

$$J(W,b) = \frac{||W||^2}{2} + \frac{C}{m}\sum_{i=1}^{m} max(0, 1 - y_i(W^T x_i + b)) \tag{3}$$

The gradient formulas are:

$$\begin{aligned} g_w(x_i) &= \begin{cases} -y_i x_i & 1 - y_i(w^T x_i + b) >= 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases} \\ g_b(x_i) &= \begin{cases} -y_i & 1 - y_i(w^T x_i + b) >= 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases} \\ \nabla_W J(W,b) &= W + \frac{C}{m}\sum_{i=1}^{m} g_w(x_i) \\ \nabla_b J(W,b) &= \frac{C}{m}\sum_{i=1}^{m} g_b(x_i) \end{aligned} \tag{4}$$

We update both models using:

$$\begin{aligned} W &= W - \alpha \nabla_W J(W,b) \\ b &= b - \alpha \nabla_b J(W,b) \end{aligned} \tag{5}$$

III. Experiments

A. Dataset

The datasets are from LIBSVM. Linear Regression uses Housing dataset, including 506 samples and each sample has 13 features. Linear classification uses australian dataset, including 690 samples and each sample has 14 features. We use scaled edition directly.

B. Implementation

1) Data Preprocessing: Use load_svmlight_file function in sklearn library to load the dataset. Divide dataset into training set and validation set using train_test_split function. The division ratio is 4 to 1. I also convert data to column vector.

2) Initialization: There are many ways to initialize linear model parameters. I choose to set all parameters into zero.

3) Compute Loss and Gradient: In linear regression, use formula (1) to compute loss and formula (2) to compute gradient. In linear classification, use formula (3) to compute loss and formula (4) to compute gradient.

4) Update Parameters: Use formula (5) to update parameters.

5) Repeat: Repeat step 3 and 4 until reach the end condition. Get trian loss and validation loss in every iteration.

6) Implementation Code: I use a function to initialize parameters with zeros:

```
def init_parameters_with_zeros(dim):
    W = np.zeros((dim, 1))
    b = 0
    return W, b
```

In linear regression, I use this function to compute cost and gradient:

```
def propagate(W, b, X, Y):
    m = X.shape[1]
    A = np.dot(W.T, X) + b
    cost = np.sum(np.square(A - Y)) / m
    gW = np.dot(X, (A - Y).T) * 2 / m
    gb = np.sum(2 * (A - Y)) / m
    return cost, gW, gb
```

And this function to train model:

```
def model(X_train, Y_train, X_val, Y_val,
    num_iterations=500,
    learning_rate=0.01, print_cost=False):
   train_cost_log = []
   val_cost_log = []
   W, b =
       init_parameters_with_zeros(X_train.shape[0])

   for i in range(num_iterations):
      train_cost, gW, gb = propagate(W, b,
          X_train, Y_train)
      val_cost, _, _ = propagate(W, b,
          X_val, Y_val)
      train_cost_log.append(train_cost)
      val_cost_log.append(val_cost)
      W = W - learning_rate * gW
      b = b - learning_rate * gb
      if print_cost and i % 100 == 0:
         print("After iteration %i, train
             cost: %f, val cost : %f" %(i,
             train_cost, val_cost))
   return W, b, train_cost_log,
       val_cost_log
```

The linear classification code is very similar to the linear regression code except loss function and gradient function. Moreover, we can compute predict accuracy in linear classification. Use this function to compute cost and gradient:

```
def propagate(W, b, X, Y):
   C = 0.9
   A = np.dot(W.T, X) + b
   cost = np.sum(np.square(W)) / 2 + C *
       np.sum(np.maximum(0, 1 - Y *
       (np.dot(W.T, X) + b))) / m
   filt = (1 - Y * (np.dot(W.T, X) + b))>0
   gW = W - C * np.dot(Y * filt, X.T).T / m
   gb = -C * np.sum(Y * filt * b) / m
   return cost, gW, gb
```

Use this function to predict:

```
def predict(W, b, X):
   m = X.shape[1]
   A = np.dot(W.T, X) + b
   Y_prediction = np.zeros((1, m))
   Y_prediction[A >= 0] = 1
   Y_prediction[A < 0] = -1
   return Y_prediction
```

And use this function to train model:

```
def model(X_train, Y_train, X_val, Y_val,
    num_iterations=200,
    learning_rate=0.01, print_cost=False):
   m = X_train.shape[0]
   train_cost_log = []
   val_cost_log = []
   train_accuracy_log = []
   val_accuracy_log = []
   W, b = init_parameters_with_zeros(m)
   for i in range(num_iterations):
      train_cost,gW,gb =
          propagate(W,b,X_train,Y_train)
      val_cost,_,_=
          propagate(W,b,X_val,Y_val)
      Y_train_prediction =
          predict(W,b,X_train)
      Y_val_prediction = predict(W,b,X_val)
      train_accuracy = 100 -
          np.mean(np.abs(Y_train_prediction
          - Y_train) / 2) * 100
      val_accuracy = 100 -
          np.mean(np.abs(Y_val_prediction
          - Y_val) / 2) * 100
      train_cost_log.append(train_cost)
      val_cost_log.append(val_cost)
      train_accuracy_log.append(train_accuracy)
      val_accuracy_log.append(val_accuracy)
      W = W - learning_rate * gW
      b = b - learning_rate * gb
      if print_cost and i % 10 == 0:
         print("After iteration %i, train
             cost: %f, val cost : %f train
             accuracy: %f %%, test
             accuracy: %f %%" % (i,
             train_cost, val_cost,
             train_accuracy, val_accuracy))
   return W, b, train_cost_log,
       val_cost_log, train_accuracy_log,
       val_accuracy_log
```

7) Result: In Linear regression, I record the loss for each iteration. From the Fig. 1 we can find that, both train loss and validation loss decrease as the number of iterations increases. As the number of iterations increases, the loss function changes slowly.

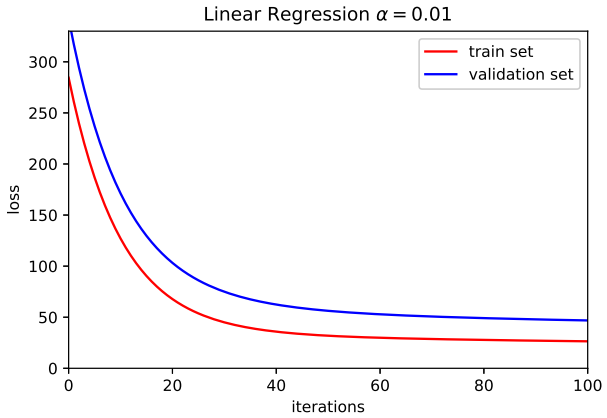

Fig. 1. The curve of the linear regression loss function with the number of iterations.

From the Fig. 2 we can see that, the slower the learning rate, the slower the loss function converges.

In Linear classification, I also plot loss function on the number of iterations with different learning rate. Moreover, I plot the curve of accuracy.

From the Fig. 3 and Fig. 4 we can see that, as the number of iterations increases, the trend of the loss function decreases and the trend of accuracy increases. From the Fig. 5 and Fig. 6 we can see that, the smaller the learning rate, the slower the loss function and accuracy change.

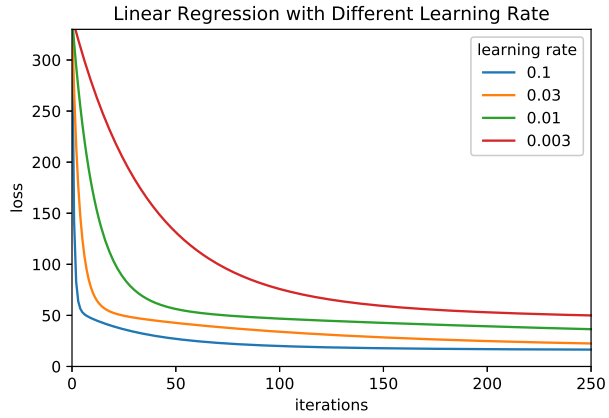

Fig. 2. The curve of the linear regression loss function with different learning rate.

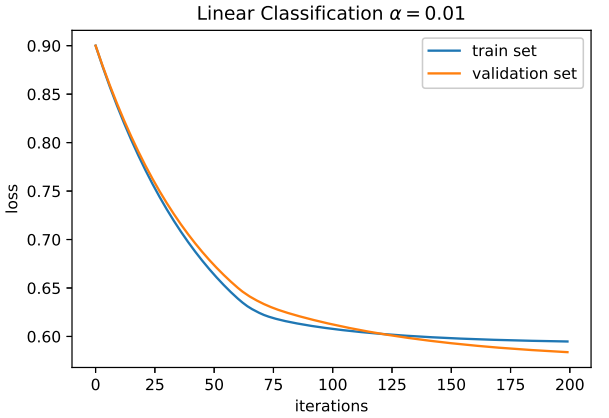

Fig. 3. The curve of the linear classification loss function with the number of iterations.

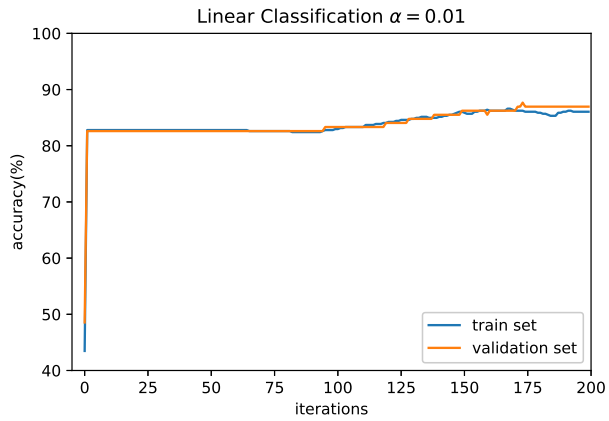

Fig. 4. The curve of the linear classification accuracy with the number of iterations.

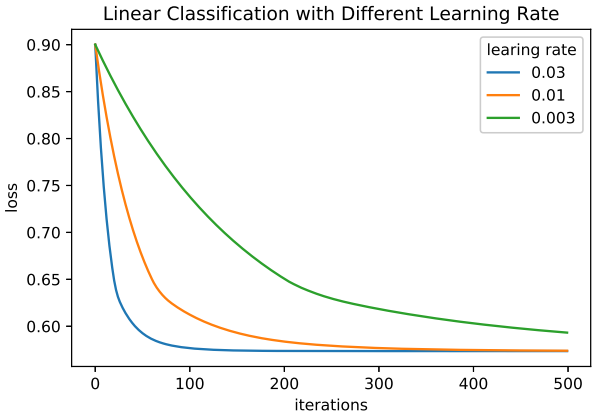

Fig. 5. The curve of the linear classification loss function with different learning rate.

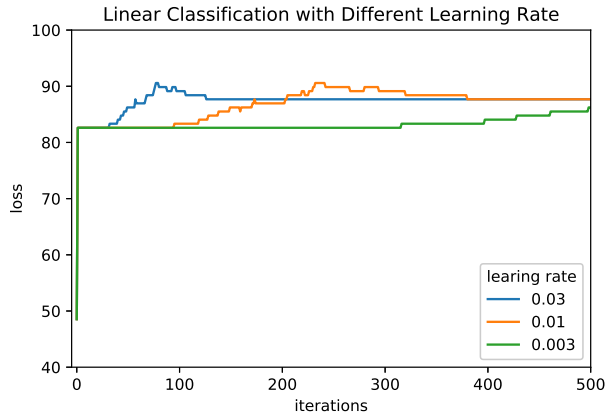

Fig. 6. The curve of the linear classification accuracy with different learning rate.

IV. Conclusion

Through this experiment, I learned how to solve problems using linear regression and linear classification, and learned how to use gradient descent to optimize the model. I learned how to adjusting hyperparameter. I learned that learning rate is an important hyperparameter, and if the learning rate is too large or too small, we can not get a good model.