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:
Xia Shu

Supervisor:
Qingyao Wu

Student ID：
201721046227

Grade:
Graduate

December 15, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract- This project is the implement of Linear Regression, Linear Classification and Gradient Descent in small scale dataset. Through the training of data we can get a algorithm which make a reasonable forecast for the input data.

I. INTRODUCTION

In this project, we aim to have a further understand of Linear Regression, Linear Classification and Gradient Descent, conduct some experiments under small scale dataset, and realize the process of optimization and adjusting parameters.

II. METHODS AND THEORY

The project includes two parts.

Part one is the implement of Linear Regression. Simple linear regression describes the linear relationship between a predictor variable, plotted on the x-axis, and a response variable, plotted on the y-axis. We assume the relationship can be represent by the model function below:

$$y = w^0 x^0 + w^1 x^1 + \cdots + w^n x^n + b$$

We define the loss function to calculate the goodness of the function as below:

$$L(w,b) = \sum_{n=1}^{n} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

The best function f^* should be the one which can make the loss function be the minimum, that means we need to pick a group of best argument include w and b to earn minimal loss. We choose the gradient descent to solve this problem.

$$f^* = arg\min_f L(f)$$

$$w^*, b^* = arg\min_{w,b} L(w,b)$$

$$= arg\min_{w,b} \sum_{n=1}^{n} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

We pick an initial value w^0 randomly, then compute

$$w^{i+1} \leftarrow w^i - \eta \frac{dL}{dw}|_{w=w^i}$$

$$b^{i+1} \leftarrow b^i - \eta \frac{dL}{db}|_{b=b^i}$$

in every iteration to update the value of w and b. In here, the gradient is like this:

$$\frac{\partial L}{\partial w} = \sum_{n=1}^{n} 2\left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)\left(-x_{cp}^n\right)$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^{n} 2\left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)(-1)$$

In every iteration, we can see the change of function cost, according to the result, we can change the argument like initial value of w, b and η to get less cost.

Part two is the implement of Linear Classification.

Linear Classification is used to forecast the class of input data. Giving training data (x_i, y_i) for i = 1..n with $x_i \in R^m$ and $y_i \in \{-1,1\}$, learn a classfier f(x) such that

$$\text{f(x)} \begin{cases} \geq 0 & y_i = +1 \\ \leq 0 & y_i = -1 \end{cases}$$

We define the Hingo Loss as below:

$$\text{Hinge loss} = \max(0,1 - y_i(w^T x_i + \text{b}))$$

The optimization problem becomes:

$$\min_{w,b} \frac{||w||^2}{2} + \text{C}\sum_{i=1}^{n} \max(0,1 - y_i(w^T x_i + \text{b}))$$

An optimization problem can be considered in two ways, primal problem and dual problem. In this project, we used the primal problem of basic SVM. So we have

$$g_w(x_i) = \begin{cases} -y_i x_i & 1 - y_i(w^T x_i + b) \geq 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases}$$

$$g_b(x_i) = \begin{cases} -y_i & 1 - y_i(w^T x_i + b) \geq 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases}$$

$$\nabla_w L(w,b) = w + \frac{C}{n}\sum_{i=1}^{n} g_w(x_i)$$

$$\nabla_b L(w,b) = \frac{C}{n}\sum_{i=1}^{n} g_b(x_i)$$

$$\text{w} = \text{w} - \eta\nabla_w L(w,b)$$

$$\text{b} = \text{b} - \eta\nabla_b L(w,b)$$

Then we use the gradient descent as same as linear regression to update the argument until get a good result.

III. EXPERIMENTS

Part one – Linear Regression

A. Dataset

Linear Regression uses Housing in LIBSVM Data, including 506 samples and each sample has 13 features. The data be divided into training set and validation set.

B. Implementation

The implement step of Linear Regression is as follow:

1. import the python package that need in the experiment.
2. Load the experiment data with the function of load_svmlight_file in sklearn library.

```
def get_data():
    data=load_svmlight_file("housing_scale")
    return data[0],data[1]
x,y=get_data();
```

3. Divide dataset into training set and validation set using train_test_split function.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,te
st_size=0.3)
```

4. Initialize linear model parameters

TABLE 1
Initialize argument

argument	value
w	[0,0,0, ... ,0]
b	0

```
    w=np.zeros(x.shape[1])
    b=0
```

5. Choose loss function and derivation
   The loss function and derivation had been introduced in part III.

```
def lossFunction(x,y,w,b):
    cost=np.sum(np.square(x*w+b-
y))/(2*x.shape[0])
    return cost
def derivation(x,y,w,b):
    wd=x.T.dot(x.dot(w)+b-y)/x.shape[0]
    bd=np.sum(x*w+b-y)/x.shape[0]
    return wd,bd
```

6. Calculate gradient toward loss function from all samples.
   Denote the opposite direction of gradient as Update model.
   Get the loss under the training set and by validating under validation set.
   Repeat several times

```
for i in range(num_iters):
    trainCost[i]=hingelossFunction(x_train,y_train,
    w,b,c)
```

```
    validateCost[i]=hingelossFunction(x_test,y_test
    ,w,b,c)
        Gw,Gb=derivation(x_train,y_train,w,b,c);
        Dw=-Gw
        Db=-Gb
        w=w+delta*Dw
        b=b+delta*Db
```

7. Drawing graph of as well as with the number of iterations.

```
num_iters=50;
delta=0.01
train_cost,validate_cost,w,b=linearClassify(x_trai
n,x_test,y_train,y_test,delta,num_iters)
plt.figure(1)
x=np.arange(0,num_iters,1)
plt.plot(x,train_cost,label = "Train loss")
plt.plot(x,validate_cost,label = "Validation loss")
plt.legend(loc='right')
plt.show()
```

The result:

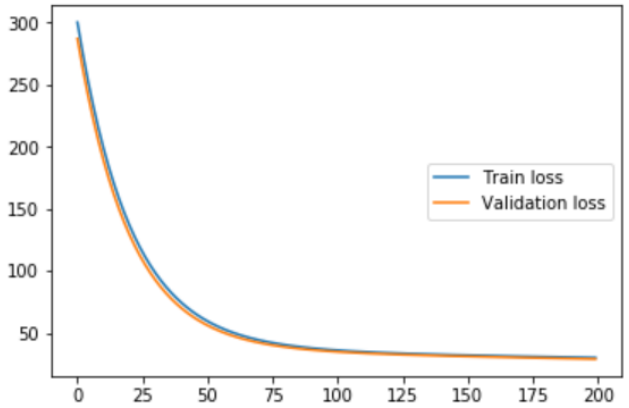

Fig. 1 The loss of different iteration

Parameters used in the experiment:

TABLE 2
Parameters

argument	value
Num_iters	50
η	0.01

Part two – Linear Classification

A. Dataset

Linear classification uses australian in LIBSVM Data, including 690 samples and each sample has 14 features. The data be divided into training set and validation set.

B. Implementation

The implement step of Linear Classification is as follow:

1. import the python package that need in the experiment.
2. Load the experiment data with the function of load_svmlight_file in sklearn library.

```
def get_data():
    data=load_svmlight_file("australian_scale")
    return data[0],data[1]
x_data,y_data=get_data();
```

3. Divide dataset into training set and validation set using train_test_split function.

```
x_train,x_test,y_train,y_test=train_test_split(x_dat
a,y_data,test_size=0.3)
```

4. Initialize linear model parameters

TABLE 3
Initialize argument

argument	value
w	[0,0,0, … ,0]
b	0

```
    w=np.zeros(x.shape[1])
    b=0
```

5. Choose loss function and derivation
   The loss function and derivation had been introduced in part III.

```
def hingelossFunction(x,y,w,b,c):
    cost=0
    for i in range(x.shape[0]):
        cost+=np.max([0,1-y[i]*(x[i]*w+b)])

cost=np.sum(np.square(w))+c*cost/(2*x.shape[0])
    return cost
def derivation(x,y,w,b,c):
    wd=np.zeros(x.shape[1])
    bd=0
    for i in range(x.shape[0]):
        if((1-y[i]*(x[i]*w+b)))>0 :
            for k in range(x.shape[1]) :
                wd[k]+=(-1)*y[i]*x[i,k]
            bd+=(-1)*y[i]
    wd=(wd*c+w)/x.shape[0]
    bd=(bd*c)/x.shape[0]
    return wd,bd
```

6. Calculate gradient toward loss function from all samples.
   Denote the opposite direction of gradient as Update model.
   Get the loss under the training set and by validating under validation set.
   Repeat several times

```
    for i in range(num_iters):

trainCost[i]=lossFunction(x_train,y_train,w,b)

validateCost[i]=lossFunction(x_test,y_test,w,b)
        Gw,Gb=derivation(x_train,y_train,w,b);
        Dw=-Gw
        Db=-Gb
        w=w+delta*Dw
        b=b+delta*Db
```

7. Drawing graph of as well as with the number of iterations.

```
num_iters=50;
delta=0.005
train_cost,validate_cost=linearClassify(x_train,x_t
est,y_train,y_test,delta,num_iters)
plt.figure(1)
x=np.arange(0,num_iters,1)
plt.plot(x,train_cost,label = "Train loss")
plt.plot(x,validate_cost,label = "Test loss")
plt.legend(loc='right')
plt.show()
```

The result:

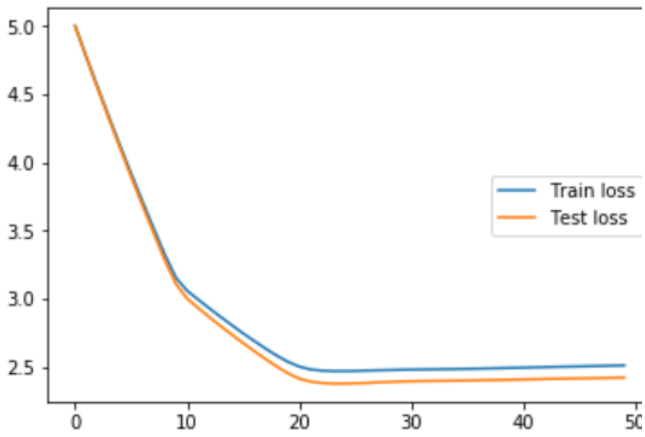

Fig. 2 The loss of different iteration

Parameters used in the experiment:

TABLE 4
Parameters

argument	value
Num_iters	50
η	0.005

IV. CONCLUTION

As we can see from the result, the cost of function is gradually decline with the iteration of argument. Finally, we can get a best function for forecast. It proves the algorithm works effective.

Through the project, I have a better understanding about Linear Regression, Linear Classification and Gradient Descent, and realized the process of optimization and adjusting parameters. In order to get better result, we should try more argument.