華南理工大學

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Yang Zhang

Supervisor:
Mingkui Tan

Student ID：201630666370

Grade:

Undergraduate

October 14, 2018

Linear Regression and Stochastic Gradient Descent

***Abstract*— This experiment is based on a small scale dataset, using linear regression ， closed-form solution and Stochastic gradient descent, in order to help us realize the process of optimization and adjusting parameters.**

I. Introduction

This experiment includes two parts, Linear Regression and Shochastic Gradient Descent. All two parts are based on the similar dataset. We first load the data and spilt it. Then initialize linear model parameters and use right formulas to compute Loss under both train section and test section.

II. Methods and Theory

In closed-form solution of Linear Regression, we first split the dataset and initialize linear model parameters by setting all parameter into zero. The most important thing in this experiment is closed-form solution. Close-form solution for Linear Regression is below.

$$\begin{aligned}
\mathcal{L}_D(\mathbf{w}) &= \frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^\top(\mathbf{y}-\mathbf{X}\mathbf{w}) \\
&= \frac{1}{2}(\mathbf{y}^\top\mathbf{y}-2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}+\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}) \\
\frac{\partial\mathcal{L}_D(\mathbf{w})}{\partial\mathbf{w}} &= \frac{1}{2}\left(\frac{\partial\mathbf{y}^\top\mathbf{y}}{\partial\mathbf{w}}-\frac{\partial 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}}{\partial\mathbf{w}}+\frac{\partial\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}}{\partial\mathbf{w}}\right) \\
&= \frac{1}{2}(-2\mathbf{X}^\top\mathbf{y}+(\mathbf{X}^\top\mathbf{X}+(\mathbf{X}^\top\mathbf{X})^\top)\mathbf{w}) \\
&= -\mathbf{X}^\top\mathbf{y}+\mathbf{X}^\top\mathbf{X}\mathbf{w}
\end{aligned}$$

Then we can conclude the optimal parameter w.

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

Then we should look in to loss function. loss function (mean squared error):

$$L_D(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - y'_i)^2$$

derivatives:

$$\frac{\partial L_D(w)}{\partial w} = -X^T y + X^T X w$$

In Linear Regression and Stochastic Gradient Descent, we also initialize linear model parameters by setting all parameter into zero. In the experiment, we use the below to update.

$$\mathbf{w}' \to \mathbf{w} - \eta\frac{\partial\mathcal{L}_D(\mathbf{w})}{\partial\mathbf{w}}$$

η is learning rate, a hyper-parameter that we can adjust.

III. Experiment

A. Dataset

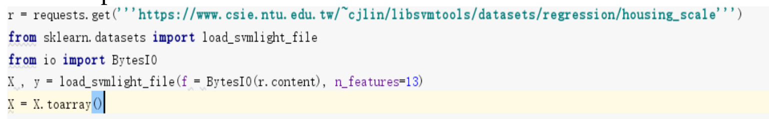
All two experiment use same dataset, which is Housing in LIBSVM Data, including 506 samples and each sample has 13 features.

B. Implementation

Closed-form solution of Linear Regression:

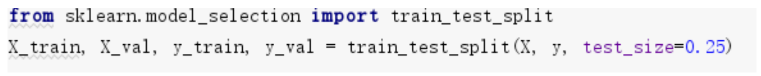
1. Load the experiment data.
2. Divide dataset.
3. Initialize linear model parameters by setting all parameter into zero,.
4. Select a Loss function and calculate the value of the Loss function of the training set, denoted as loss.
5. Get the formula of the closed-form solution.
6. Get the value of parameter W by the closed-form solution, and update the parameter W.
7. Get the Loss ， loss_train under the training set and loss_cal by validating under validation set.
8. Output the value of Loss, loss_train and loss-val.

The vital parts of code is below:



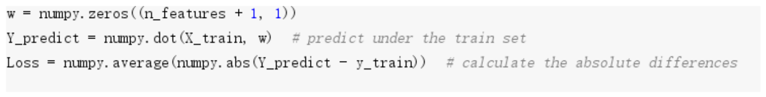
```
r = requests.get('''https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/housing_scale''')
from sklearn.datasets import load_svmlight_file
from io import BytesIO
X , y = load_svmlight_file(f = BytesIO(r.content), n_features=13)
X = X.toarray()
```

Fig. 1. Load the data



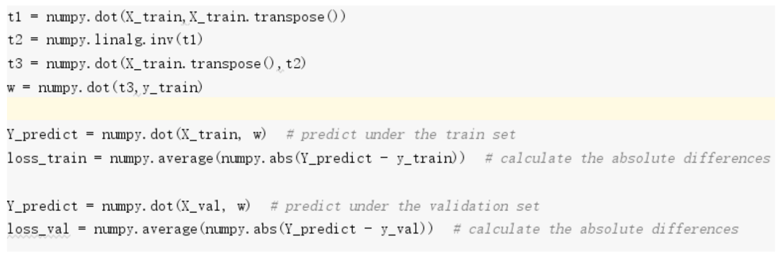
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25)
```

Fig. 2. Split the data



```
w = numpy.zeros((n_features + 1, 1))
Y_predict = numpy.dot(X_train, w)  # predict under the train set
Loss = numpy.average(numpy.abs(Y_predict - y_train))  # calculate the absolute differences
```

Fig. 3. Preprocess the data



```
t1 = numpy.dot(X_train,X_train.transpose())
t2 = numpy.linalg.inv(t1)
t3 = numpy.dot(X_train.transpose(),t2)
w = numpy.dot(t3,y_train)

Y_predict = numpy.dot(X_train, w)  # predict under the train set
loss_train = numpy.average(numpy.abs(Y_predict - y_train))  # calculate the absolute differences

Y_predict = numpy.dot(X_val, w)  # predict under the validation set
loss_val = numpy.average(numpy.abs(Y_predict - y_val))  # calculate the absolute differences
```

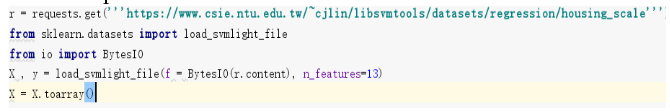
Fig. 4. Compute the result

The result of the experiment is below:
Loss is 22.37704485488127
loss_train is 5.463314267712153
loss_val is 16.950290334270967

Linear Regression and Stochastic Gradient Descent:

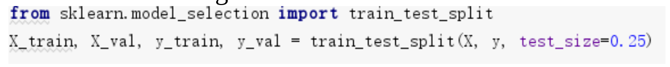
1. Load the experiment data.
2. Divide dataset.
3. Initialize linear model parameters by setting all parameter into zero.
4. Choose loss function and derivation
5. Calculate G toward loss function from each sample.
6. Denote the opposite direction of gradient G as D.
7. Update model: Wt=Wt−1+ηD. η is learning rate, a hyper-parameter that we can adjust.
8. Get the loss_train under the training set and loss_val by validating under validation set.
9. Repeat step 5 to 8 for several times, and output the value of loss_train as well as loss_val.

The vital parts of code is below:



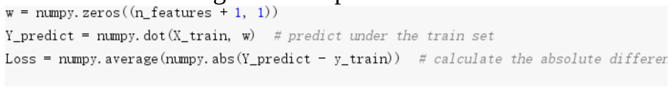
```
r = requests.get('''https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/housing_scale'''
from sklearn.datasets import load_svmlight_file
from io import BytesIO
X , y = load_svmlight_file(f = BytesIO(r.content), n_features=13)
X = X.toarray()
```

Fig. 5. Load the data



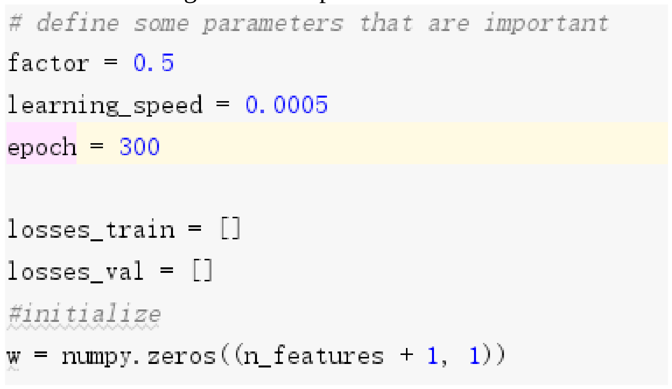
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25)
```

Fig. 6. Split the data



```
w = numpy.zeros((n_features + 1, 1))
Y_predict = numpy.dot(X_train, w)  # predict under the train set
Loss = numpy.average(numpy.abs(Y_predict - y_train))  # calculate the absolute differen
```

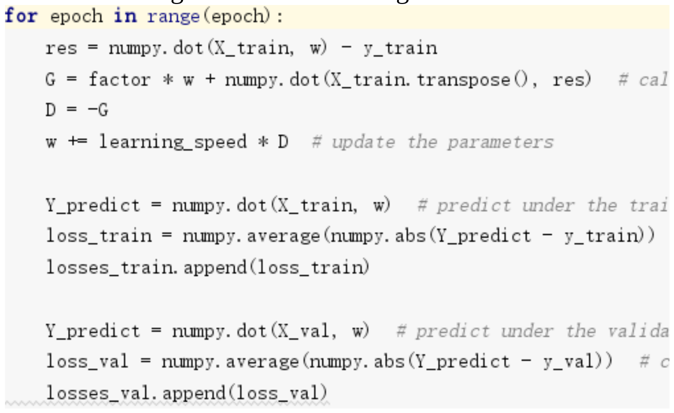
Fig. 7. Preprocess the data



```
# define some parameters that are important
factor = 0.5
learning_speed = 0.0005
epoch = 300

losses_train = []
losses_val = []
#initialize
w = numpy.zeros((n_features + 1, 1))
```

Fig. 8. Define the regular term



```
for epoch in range(epoch):
    res = numpy.dot(X_train, w) - y_train
    G = factor * w + numpy.dot(X_train.transpose(), res)  # cal
    D = -G
    w += learning_speed * D  # update the parameters

    Y_predict = numpy.dot(X_train, w)  # predict under the trai
    loss_train = numpy.average(numpy.abs(Y_predict - y_train))
    losses_train.append(loss_train)

    Y_predict = numpy.dot(X_val, w)  # predict under the valida
    loss_val = numpy.average(numpy.abs(Y_predict - y_val))  # c
    losses_val.append(loss_val)
```

Fig. 9. Compute the result

The result is below:

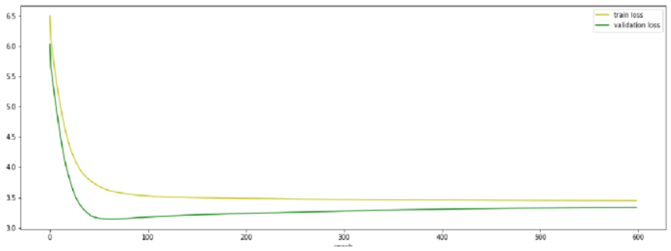


Fig. 10. Output

IV. CONCLUSION

The gradient descent is very smooth, which means that the descent process meets the requirements of this lab. In addition, the descent process of the loss of train and the loss of evaluation are similar, which means that the hyper-parameters chosen are suitable.

This experiment is really interesting! Honestly, I have never try to write codes to implement the basic methods of machine learning. In addition, this experiment does help me a lot and now I have a better understand about the meanings and effects of hyper-parameters such as learning rate and max iterations.