The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Siyuan Xiao

Supervisor:
Qingyao Wu

Student ID：
201721045886

Grade:
Undergraduate

December 14, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—Linear Regression and Linear Classif-ication are very basic, simple and effective ways to create statistical models for things in real life. Gradient Descent is an effective way to solve this optimizing problem, and is very popular in the field of machine learning.

I. Introduction

This report will talk about the whole experiment I have made on Linear Regression, Linear classification and Gradient Descent. Its content is organized as follow:

1) Section II contains the experiment steps.
2) Section III contains the code for the two experiments.
3) Section IV makes conclusion for the experiment result.

II. Methods and Theory

Linear Regression uses 'Housing' in LIBSVM Data, including 506 samples and each sample has 13 features.

Linear classification uses 'Australian' in LIBSVM Data, including 690 samples and each sample has 14 features.

Then the experiment will be performed by the following steps:

1) Download the dataset to local host machine.
2) Load dataset into memory.
3) Split dataset into training set and validation set.
4) Create and fill necessary data structures.
5) Write functions for calculating loss and gradient (different in regression and classification).
6) Set parameters (learning rate and the number of iterations).
7) Initiate weights (using normal distribution).
8) Calculate the gradient and update weights according to the gradient calculated in each iteration.
9) Change parameters and run again.
10) Draw plot for experiment result.

III. Experiment

Here I placed the code for the two experiments:

1) Linear Regression:

```
# created by Swain, 2017-12-13, 11:34

import numpy
import matplotlib.pyplot as plot
from numpy import random
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split

#load housing_scale dataset
path = './housing_scale'
data = load_svmlight_file(path)
X = data[0]
X = X.toarray()
y = data[1]

#add a constant-bias-column to X
col = numpy.ones((X.shape[0]))
X = numpy.column_stack((X, col))

#create weight array with initial values in normal distribution
d = X.shape[1]
W = numpy.random.normal(size=d);

#split dataset into training data and test data
X_train, X_vali, y_train, y_vali = train_test_split(X, y, test_size=0.2, random_state=1)

#define loss function
def loss(X, W, y):
    y_predict = numpy.dot(X, W)
    diff = y - y_predict
    return numpy.dot(diff, diff.T) / (2 * X.shape[0])

#define gradient function
def grad(X, W, y):
    y_predict = numpy.dot(X, W)
    diff = y - y_predict
    return - numpy.dot(diff, X) / X.shape[0]

#parameters: learning rate and #iteration
lrs = [0.05, 0.1, 0.2, 0.4]
iteration = 100

#used to save results
loss_train = []
loss_vali = []

for i in range(0, len(lrs)):
    #reset W
    W = W_init
    loss_train.append(numpy.zeros(iteration))
    loss_vali.append(numpy.zeros(iteration))
    for j in range(0, iteration):
        #calculate loss on both training and validation datasets
        loss_train[i][j] = loss(X_train, W, y_train)
        loss_vali[i][j] = loss(X_vali, W, y_vali)
        #update weight according to gradient
        W = W - grad(X_train, W, y_train) * lrs[i]

for i in range(0, len(lrs)):
    plot.plot(loss_vali[i], label="lr = " + str(lrs[i]))
plot.legend()
plot.xlabel("Iteration")
plot.ylabel("Validation Loss")
plot.title("Linear Regression")
plot.show()
```

2) Linear Classification:

The only difference between these two experiments is the loss function and its corresponding gradient function.

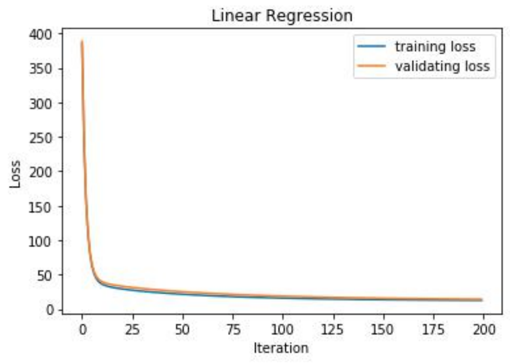
```
#define loss function (hinge loss)
def loss(X, W, y, _lambda):
    y_predict = numpy.dot(X, W)
    diff = numpy.ones(y.shape[0]) - y * y_predict
    W_0 = W.copy()
    W_0[len(W)-1] = 0
    return numpy.sum(diff) / X.shape[0] + numpy.dot(W_0,W_0.T) / 2 * _lambda

#define gradient function
def grad(X, W, y, _lambda):
    y_predict = numpy.dot(X, W)
    diff = numpy.ones(y.shape[0]) - y * y_predict
    y[diff <= 0] = 0
    W_0 = W.copy()
    W_0[len(W)-1] = 0
  return -numpy.dot(y,X) / X.shape[0] + W_0 * _lambda
```

IV. CONCLUSION

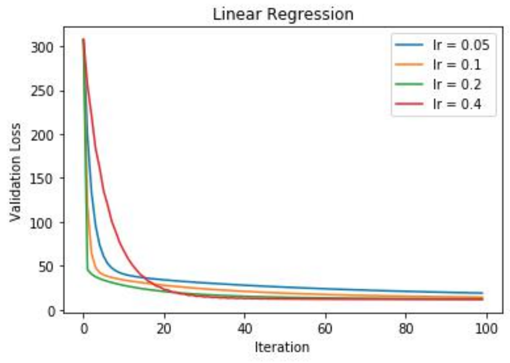
Here is the experiment result gained:

1) Linear Regression

Training loss and validation loss when using 0.05 learning rate for 200 iterations.

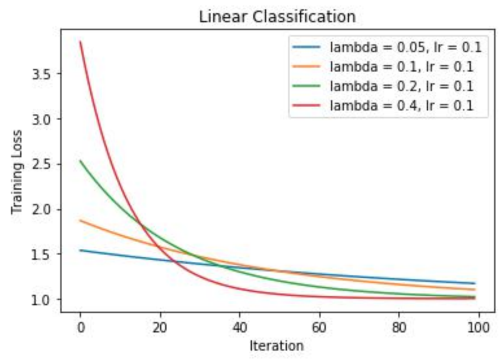


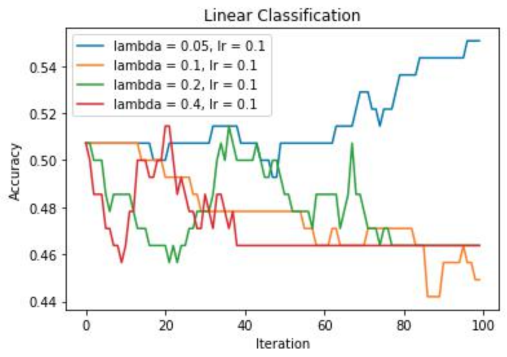
Validation loss in 4 different learning rates for 100 iterations.



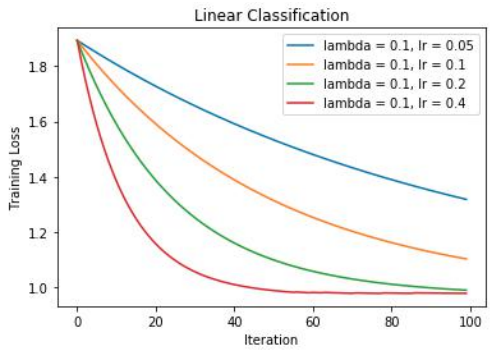
2) Linear Classification

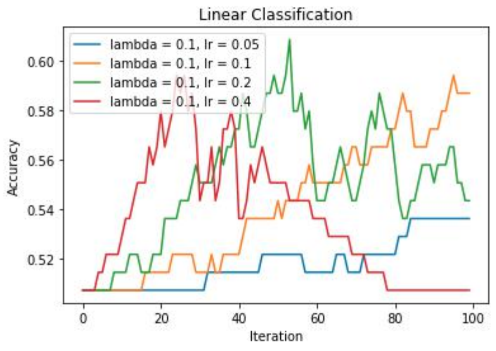
Set constant learning rate and variable lambda.





Set constant lambda and variable learning rate.





Then we can draw a conclusion according to the two experiments:

1) It will converge slowly when learning rate is small.
2) It will make larger loss when learning rate is too large.
3) An appropriate lambda can influence loss significantly.