The Experiment Report of Machine Learning



**SUBJECT:**SOFTWARE ENGINEERING

December 14, 2017

Grade:

Postgraduate

Student ID：

201721045886

Supervisor:

Qingyao Wu

Author:

Siyuan Xiao

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

[[1]](#footnote-0)

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.

# 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.

# 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.

# Experiment

Here I placed the code for the two experiments:

1. Linear Regression:

|  |
| --- |
| 1. # created by Swain, 2017-12-13, 11:34 2. import numpy 3. import matplotlib.pyplot as plot 4. from numpy import random 5. from sklearn.datasets import load\_svmlight\_file 6. from sklearn.model\_selection import train\_test\_split 7. #load housing\_scale dataset 8. path = './housing\_scale' 9. data = load\_svmlight\_file(path) 10. X = data[0] 11. X = X.toarray() 12. y = data[1] 13. #add a constant-bias-column to X 14. col = numpy.ones((X.shape[0])) 15. X = numpy.column\_stack((X, col)) 16. #create weight array with initial values in normal distribution 17. d = X.shape[1] 18. W = numpy.random.normal(size=d); 19. #split dataset into training data and test data 20. X\_train, X\_vali, y\_train, y\_vali = train\_test\_split(X, y, test\_size=0.2, random\_state=1) 21. #define loss function 22. def loss(X, W, y): 23. y\_predict = numpy.dot(X, W) 24. diff = y - y\_predict 25. return numpy.dot(diff, diff.T) / (2 \* X.shape[0]) 26. #define gradient function 27. def grad(X, W, y): 28. y\_predict = numpy.dot(X, W) 29. diff = y - y\_predict 30. return - numpy.dot(diff, X) / X.shape[0] 31. #parameters: learning rate and #iteration 32. lrs = [0.05, 0.1, 0.2, 0.4] 33. iteration = 100 34. #used to save results 35. loss\_train = [] 36. loss\_vali = [] 37. for i in range(0, len(lrs)): 38. #reset W 39. W = W\_init 40. loss\_train.append(numpy.zeros(iteration)) 41. loss\_vali.append(numpy.zeros(iteration)) 42. for j in range(0, iteration): 43. #calculate loss on both training and validation datasets 44. loss\_train[i][j] = loss(X\_train, W, y\_train) 45. loss\_vali[i][j] = loss(X\_vali, W, y\_vali) 46. #update weight according to gradient 47. W = W - grad(X\_train, W, y\_train) \* lrs[i] 48. for i in range(0, len(lrs)): 49. plot.plot(loss\_vali[i], label="lr = " + str(lrs[i])) 50. plot.legend() 51. plot.xlabel("Iteration") 52. plot.ylabel("Validation Loss") 53. plot.title("Linear Regression") 54. plot.show() |

1. Linear Classification:

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

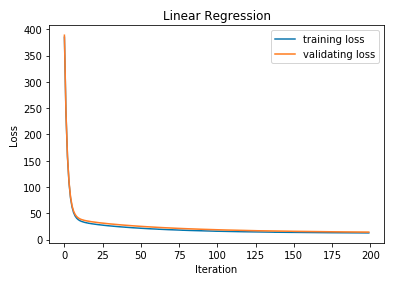
|  |
| --- |
| 1. #define loss function (hinge loss) 2. def loss(X, W, y, \_lambda): 3. y\_predict = numpy.dot(X, W) 4. diff = numpy.ones(y.shape[0]) - y \* y\_predict 5. W\_0 = W.copy() 6. W\_0[len(W)-1] = 0 7. return numpy.sum(diff) / X.shape[0] + numpy.dot(W\_0,W\_0.T) / 2 \* \_lambda 8. #define gradient function 9. def grad(X, W, y, \_lambda): 10. y\_predict = numpy.dot(X, W) 11. diff = numpy.ones(y.shape[0]) - y \* y\_predict 12. y[diff <= 0] = 0 13. W\_0 = W.copy() 14. W\_0[len(W)-1] = 0   return -numpy.dot(y,X) / X.shape[0] + W\_0 \* \_lambda |

# conclusion

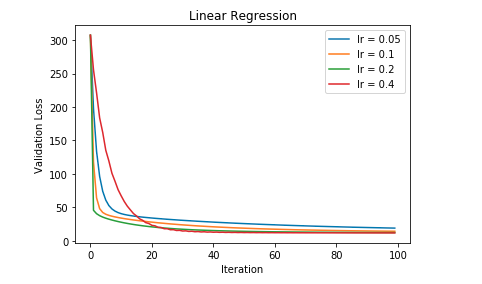
Here is the experiment result gained:

1. Linear Regression

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

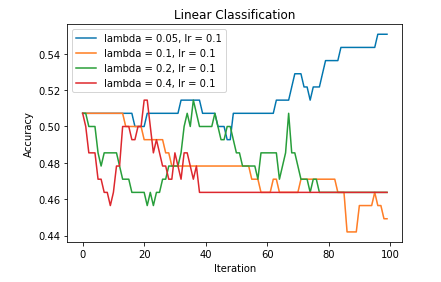
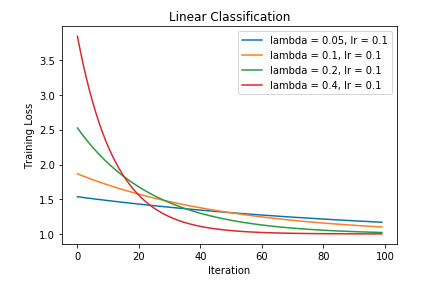


Validation loss in 4 different learning rates for 100 iterations.

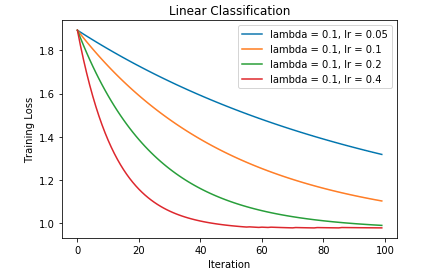


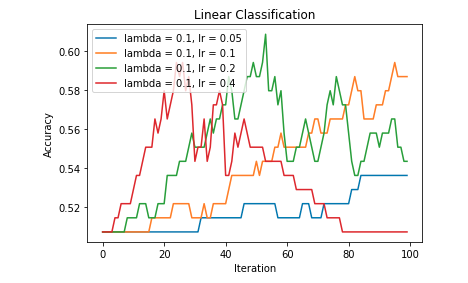
1. Linear Classification

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.

1. [↑](#footnote-ref-0)