

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **Xiaoli Tang**

**Student ID 201720145044**

**E-mail 1003715945@qq.com**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017.12.05**

1. **Topic:**

Linear Regression, Linear Classification and Gradient Descent

1. **Time:** 2017-12-02

**3. Reporter:** Xiaoli Tang

**4. Purposes:**

1. Further understand of linear regression and gradient descent.
2. Conduct some experiments under small scale dataset.
3. Realize the process of optimization and adjusting parameters.

**5. Data sets and data analysis:**

1. Linear Regression uses Housing in LIBSVM Data, including 506 samples and each sample has 13 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.
2. Linear classification uses australian in LIBSVM Data, including 690 samples and each sample has 14 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.

**6. Experimental steps:**

1. Linear Regression and Gradient Descent

* Load the experiment data, using load\_svmlight\_file function in sklearn library.
* Devide dataset into training set and validation set using train\_test\_split function.
* Initialize linear model parameters, setting all parameter into zero.
* Choose loss function and derivation.
* Calculate gradient G toward loss function from all samples.
* Denote the opposite direction of gradient G as D.
* Update model.
* Get the loss under the training set and validation set.
* Repeate step 5 to 8 for several times, and drawing graph of loss with the number of iterations.

1. Linear classification

* Load the experiment data.
* Divide dataset into training set and validation set.
* Initialize linear model parameters, setting all parameter into zero.
* Choose loss function and derivation.
* Calculate gradient G toward loss function from all samples.
* Denote the opposite direction of gradient G as D.
* Update model.
* Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative.
* Repeate step 5 to 8 for several times, and drawing graph of loss with the number of iterations.

**7. Code:**

1. Linear regression
2. # Linear regression
3. import numpy **as** np
4. import matplotlib.pyplot **as** plt
5. from sklearn.datasets import load\_svmlight\_file
6. from sklearn.model\_selection import train\_test\_split
7. def get\_data():
8. data = load\_svmlight\_file('./Housing.txt')
9. return data[0],data[1]
10. def split\_data(x,y):
11. X\_train, X\_validate, Y\_train, Y\_validate = train\_test\_split(x,y,test\_size=0.2)
12. return X\_train, X\_validate, Y\_train, Y\_validate
13. # compute loss
14. def loss(w,b,data,label,num):
15. b = np.ones((num,1))\*b
16. totalLoss = (label-data\*np.transpose(w)-b)\*\*2
17. totalLoss = np.sum(totalLoss/2,axis= 0)
18. totalLoss = totalLoss / float(data.shape[0])
19. return totalLoss
20. # compute gradient and optimizer w and b
21. def optimizer(data,label,w\_current,b\_current,learning\_rate,num):
22. #w\_gradient应该是样本数\*维度数
23. b = np.ones((num,1))\*b\_current
24. w\_gradient = -1/num \* (np.transpose(label-data\*np.transpose(w\_current)-b))\*data
25. #w\_gradient = np.sum(w\_gradient,axis=0)
26. b\_gradient = -1/num \*(label-data\*np.transpose(w\_current)-b)
27. b\_gradient = np.sum(b\_gradient,axis=0)
28. new\_b = b\_current - (learning\_rate \* b\_gradient)
29. new\_w = w\_current - (learning\_rate \* w\_gradient)
30. return new\_w,new\_b
31. # read and split data
32. X,Y = get\_data()
33. Y = Y.reshape((len(Y),1))
34. X\_train, X\_validate, Y\_train, Y\_validate = split\_data(X,Y)
35. # some related numbers
36. features\_num = X.shape[1]
37. train\_num = Y\_train.shape[0]
38. validate\_num = Y\_validate.shape[0]
39. train\_loss = []
40. validate\_loss = []
41. # initialize w and other parameters
42. w = np.zeros((1,features\_num))
43. b = 0
44. learning\_rate = 0.2
45. iterations = 500
46. # record loss of train and validate date
47. for i in range(iterations):
48. train\_loss.append(loss(w,b,X\_train,Y\_train,train\_num))
49. validate\_loss.append(loss(w,b,X\_validate,Y\_validate,validate\_num))
50. w,b = optimizer(X\_train,Y\_train,w,b,learning\_rate,train\_num)
51. # plot
52. fig, ax = plt.subplots()
53. train\_loss\_line = ax.plot(range(iterations),train\_loss,label='Train\_loss')
54. validate\_loss\_line = ax.plot(range(iterations),validate\_loss,label='Validate\_loss')
55. plt.legend()
56. ax.set(xlabel='Epoch',ylabel='Loss')
57. plt.show()
58. Linear classification
59. # Linear classification
60. **import** numpy as np
61. **import** matplotlib.pyplot as plt
62. import scipy
63. from sklearn.datasets import load\_svmlight\_file
64. from sklearn.model\_selection import train\_test\_split
65. # compute hinge loss
66. def hinge\_loss(w,X,Y,C=1.0):
67. records\_num,features\_num = np.shape(X)
68. zero = np.zeros((records\_num,1))
69. margin = 1 - C \* Y \* (X.dot(w))
70. return np.max([zero,margin],axis=0)
71. # compute the gradient
72. def loss(w,X,Y,lamda=0.0,C=1.0):
73. records\_num,features\_num = np.shape(X)
74. e = hinge\_loss(w,X,Y,C)
75. regulation\_loss = 1.0/2 \* lamda \* w.transpose().dot(w)
76. loss = 1.0/float(records\_num) \* e.sum() + regulation\_loss
77. return loss[0][0]
78. # compute the gradient
79. def gradient(w,X,Y,lamda=0.0,C=1.0):
80. records\_num,features\_num = np.shape(X)
81. e = hinge\_loss(w,X,Y,C)
82. indicator = np.zeros((records\_num,1))
83. indicator[np.nonzero(e)] = 1
84. return - 1.0/float(records\_num) \* C \* X.transpose().dot(Y \* indicator).sum(axis=1).reshape((features\_num,1)) \
85. + lamda \* w

88. # predicted results
89. def predict(w,X,threshold=0.5):
90. raw = X.dot(w)
91. raw[raw<=threshold] = -1
92. raw[raw>threshold] = 1
93. return raw
94. # compute accuracy
95. def accuracy(w,X,Y,threshold=0.5):
96. records\_num,features\_num = np.shape(X)
97. P = predict(w,X,threshold)
98. is\_right = P \* Y
99. is\_right[is\_right < 0] = 0
100. return 1.0/records\_num \* np.count\_nonzero(is\_right)
101. # read and split data
102. data = load\_svmlight\_file("./australian.txt")
103. # add X0 to X
104. X = scipy.sparse.hstack((scipy.sparse.csr\_matrix(np.ones((len(data[1]),1))),data[0]))
105. Y = data[1].reshape((len(data[1]),1))
106. records\_num,features\_num = np.shape(X)
107. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.33,random\_state=42)
108. # initialize w and other parameters
109. w = np.random.normal(size=(features\_num,1))
110. lamda = 0.1
111. eta = 0.2
112. C = 1.0
113. threshold=0.5
114. max\_iterate = 50
115. loss\_train = []
116. loss\_test = []
117. accuracy\_train = []
118. accuracy\_test = []
119. # record loss and accuracy of train and validate date
120. for epoch in range(max\_iterate):
121. loss\_train.append(loss(w,X\_train,Y\_train,lamda,C))
122. loss\_test.append(loss(w,X\_test,Y\_test,lamda,C))
123. accuracy\_train.append(accuracy(w,X\_train,Y\_train,threshold))
124. accuracy\_test.append(accuracy(w,X\_test,Y\_test,threshold))
125. w = w - eta \* gradient(w,X\_train,Y\_train,lamda,C)
126. # plot
127. fig, ax = plt.subplots()
128. ax\_e = ax.twinx()
129. train\_loss\_line = ax.plot(range(max\_iterate),loss\_train,label='train loss')
130. test\_loss\_line = ax.plot(range(max\_iterate),loss\_test,label='test loss')
131. train\_accuracy\_line = ax\_e.plot(range(max\_iterate),accuracy\_train,'r',label='train accuracy')
132. test\_accuracy\_line = ax\_e.plot(range(max\_iterate),accuracy\_test,'g',label='test accuracy')
133. ax.set(xlabel='Epoch', ylabel='Loss with l2 norm')
134. ax\_e.set\_ylabel('Accuracy with threshold='+str(threshold))
135. ax.legend(loc=4)
136. ax\_e.legend(loc=1)
137. plt.show()

**8. Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):**

Hold-out

**9. The initialization method of model parameters:**

Set all parameter into zero in linear regression.

Initialize all parameter randomly in linear classification.

**10. The selected loss function and its derivatives:**

1. Linear regression
2. Loss function
3. Gradient with the respect of the W
4. Linear classification
5. Loss function
6. Gradient with the respect of the W

**11. Experimental results and curve:**

## Hyper-parameter selection (η, epoch, etc.):

1. Linear regression

= 0.2, epoch = 500

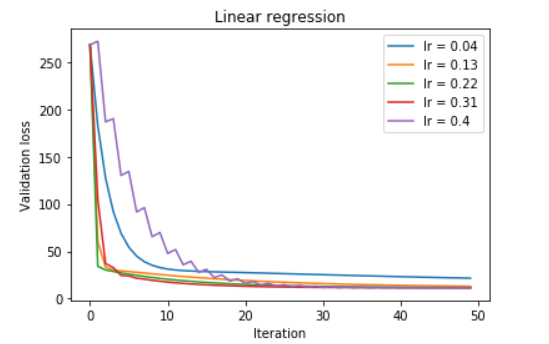
1. Linear classification

= 0.2, =0.1, epoch = 50, C = 1.0

## Assessment Results (based on selected validation):

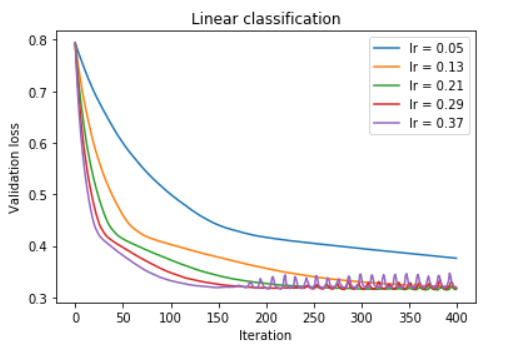
1. Linear regression

Learning rate

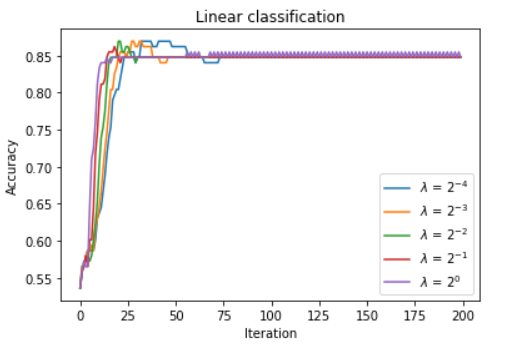


1. Linear classification

* Learning rate



* Regularization parameter

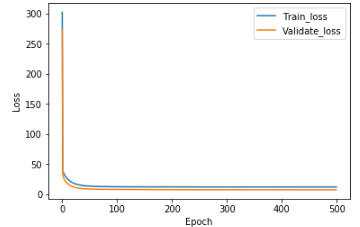


## Predicted Results (Best Results):

1. Linear regression

= 0.2, epoch = 200

Validation Loss =19.50118207

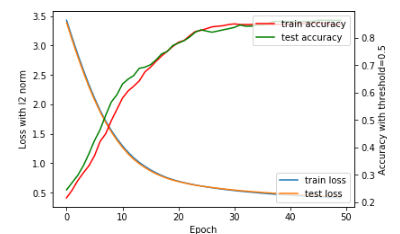


1. Linear classification

= 0.2, =0.1, epoch = 50, C = 1.0

Validation Loss = 0.42642246707271791

Accuracy = 0.8640350877192982



**12. Results analysis:**

* If the learning rate is too small, the update of the parameter W will be small. The decline of the curve is slow and the number of the iteration to reach convergence will be large.

If the learning rate is too large, the loss curve will be oscillating.

* If the regularization parameter is too small, the proportion of the regularization term will be small and the model may be easy to fall into over-fitting.

If the regularization parameter is too small, the proportion of the error term will be small and the model may be under-fitting.

**13. Similarities and differences between linear regression and linear classification:**

1. **Similarity**: Linear regression and linear classification both use the linear model.

2. **Difference:**

1) Linear regression uses Least squared loss as the loss function, but linear classification updates the parameters by Hingle loss.

2) To evaluate the linear regression, we compare the final validation loss. For linear classification, we can also evaluate the model by calculating the accuracy.

**14. Summary:**

* If the learning rate is too small, the update of the parameter W will be small. The decline of the curve is slow and the number of the iteration to reach convergence will be large.

If the learning rate is too large, the loss curve will be oscillating.

* If the regularization parameter is too small, the proportion of the regularization term will be small and the model may be easy to fall into over-fitting.

If the regularization parameter is too small, the proportion of the error term will be small and the model may be under-fitting.

* Linear regression and linear classification both use the linear model.
* Gradient decent is a valid method to optimize both regression problem and classification problem.