

**The Experiment Report of**

***Deep Learning***

**College Software College**

**Subject Software Engineering**

**Members**  **Yihan Zheng**

**Student ID 201720145105**

**E-mail yihanzheng7@gmail.com**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017.12 .5**

**1. Topic:** Linear Regression, Linear Classification and Gradient Descent

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

**3. Reporter:** Yihan Zheng

**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. Load the experiment data and divide the dataset into training set and validation set.
2. Initialize linear model parameters. Set all parameter into zero, initialize it randomly or with normal distribution.
3. Define the loss function of the linear regression to be Least squared loss, and the loss function of the linear classification to be Hinge loss.
4. Compute the gradient of the loss function with respect to the weight W and bias b.
5. Update the parameters W and b.
6. Repeat above steps for several times until convergence.

**7. Code:**

1. Linear regression
2. # Linear regression
3. import numpy **as** np
4. import matplotlib.pyplot **as** plt
5. from numpy import random
6. from sklearn.externals.joblib import Memory
7. from sklearn.datasets import load\_svmlight\_file
8. from sklearn.model\_selection import train\_test\_split
9. #mem = Memory("./mycache")
10. path = './housing\_scale.txt'
12. #get the dataset
13. def get\_data(path):
14. data = load\_svmlight\_file(path)
15. **return** data[0], data[1]
17. X, y = get\_data(path)
18. X = X.toarray()
19. #y = W^T \*X + b -> y = W\_extend^T \* [X,1]
20. column = np.ones(( X.shape[0]))
21. X = np.column\_stack((X,column))
23. #divide into training data **and** validation data
24. X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=24)
26. N = X\_train.shape[1]
27. W\_zeros = np.zeros(N)
28. W\_random = random.random(size=N)
29. W\_normal = np.random.normal(size=N)
31. #calculate the loss
32. def cal\_Loss(X,W,y):
33. preY = np.dot(X,W)
34. diifY = y - preY
35. Loss = np.dot(diifY,diifY.T)/(2 \* X.shape[0])
36. **return** Loss
38. #calculate the gradient
39. def cal\_G(X,W,y):
40. preY = np.dot(X,W)
41. diifY = y - preY
42. G = - np.dot(diifY,X)/ X.shape[0]
43. **return** G
45. #draw the result
46. def draw\_plot(Loss\_train,Loss\_validation):
47. plt.plot(Loss\_train,label="Loss\_train")
48. plt.plot(Loss\_validation,label="Loss\_validation")
49. plt.legend()
50. plt.xlabel("Iteration")
51. plt.ylabel("Loss")
52. plt.title("Linear regression")
53. plt.show()
55. lr = 0.2
56. iteration = 200
57. #get different kinds of initial data（W\_zeros,W\_random **or** W\_normal）
58. W = W\_normal
59. Loss\_train = np.zeros(iteration)
60. Loss\_validation = np.zeros(iteration)
61. **for** j in range(0,iteration):
62. #the training loss
63. Loss\_train[j] = cal\_Loss(X\_train,W,y\_train)
64. #the gradient of the loss **function**
65. G = cal\_G(X\_train,W,y\_train)
66. #the validation loss
67. Loss\_validation[j] = cal\_Loss(X\_validation,W,y\_validation)
68. #update the parameter W
69. W = W - G \* lr
70. #draw the result
71. draw\_plot(Loss\_train,Loss\_validation)
72. Linear classification
73. # Linear classification
74. **import** numpy as np
75. **import** matplotlib.pyplot as plt
76. **from** numpy **import** random
77. **from** sklearn.externals.joblib **import** Memory
78. **from** sklearn.datasets **import** load\_svmlight\_file
79. **from** sklearn.model\_selection **import** train\_test\_split
80. #mem = Memory("./mycache")
81. path = './australian\_scale.txt'
83. #get the dataset
84. **def** get\_data(path):
85. data = load\_svmlight\_file(path)
86. **return** data[0], data[1]
88. X, y = get\_data(path)
89. X = X.toarray()
90. #y = W^T \*X + b -> y = W\_extend^T \* [X,1]
91. column = np.ones((X.shape[0]))
92. X = np.column\_stack((X,column))
94. #divide into training data and validation data
95. X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=24)
97. N=X\_train.shape[1]
98. W\_zeros = np.zeros(N)
99. W\_random = random.random(size=N)
100. W\_normal = np.random.normal(size=N)
102. #calculate the loss
103. **def** cal\_Loss(X,W,y,lambdal,W\_0):
104. preY = np.dot(X,W)
105. diifY = np.ones(y.shape[0]) - y \* preY
106. diifY[diifY < 0] =0
107. Loss =np.sum(diifY) / X.shape[0] + np.dot(W\_0,W\_0.T)/2\*lambdal
108. **return** Loss
110. #calculate the gradient
111. **def** cal\_G(X,W,y,lambdal,W\_0):
112. preY = np.dot(X,W)
113. diifY = np.ones(y.shape[0]) - y \* preY
114. y\_get = y.copy()
115. y\_get[diifY <= 0] =0
116. G = -np.dot(y\_get,X) / X.shape[0] + W\_0 \*lambdal
117. **return** G
119. #calculate the accuracy
120. **def** cal\_Accuracy(X,W,y):
121. preY = np.dot(X,W)
122. count = np.sum(preY \* y >0)
123. Accuracy = count / X.shape[0]
124. **return** Accuracy
126. #draw the result
127. **def** draw\_plot(Loss\_train,Loss\_validation,Accuracy):
128. fig = plt.figure()
129. ax1 = fig.add\_subplot(111)
130. ax1.plot(Loss\_train,label="Loss\_train")
131. ax1.plot(Loss\_validation,label="Loss\_validation")
132. ax1.set\_ylabel("Loss")
133. ax1.set\_xlabel("Iteration")
134. ax1.legend()
135. ax2 = ax1.twinx()
136. ax2.plot(Accuracy,label="Accuracy",color = 'r')
137. ax2.legend()
138. ax2.set\_ylabel("Accuracy")
139. ax2.set\_title("Linear classification")
140. plt.show()
142. lr = 0.04
143. iteration = 400
144. lambdal = 0.01
145. #get different kinds of initial data（W\_zeros,W\_random or W\_normal）
146. W = W\_normal
147. Loss\_train = np.zeros(iteration)
148. Loss\_validation = np.zeros(iteration)
149. Accuracy = np.zeros(iteration)
150. **for** j **in** range(0,iteration):
151. W\_0 = W.copy()
152. W\_0[N-1]= 0
153. #the training loss
154. Loss\_train[j] = cal\_Loss(X\_train,W,y\_train,lambdal,W\_0)
155. #the gradient of the loss function
156. G = cal\_G(X\_train,W,y\_train,lambdal,W\_0)
157. #the validation loss
158. Loss\_validation[j] = cal\_Loss(X\_validation,W,y\_validation,lambdal,W\_0)
159. #accuracy
160. Accuracy[j] = cal\_Accuracy(X\_validation,W,y\_validation)
161. #update the parameter W
162. W = W - G \* lr
163. #draw the result
164. draw\_plot(Loss\_train,Loss\_validation,Accuracy)

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

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

normal distribution

**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 = 200

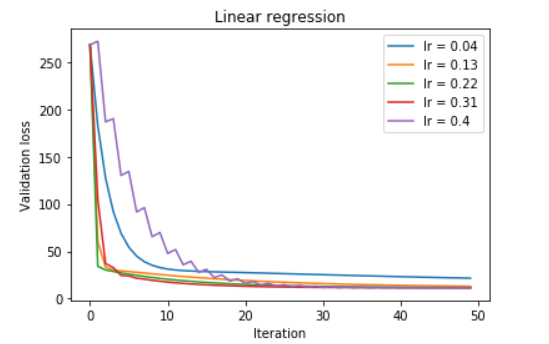
1. Linear classification

= 0.2, =0.5, epoch = 400

## Assessment Results (based on selected validation):

1. Linear regression
2. Learning rate

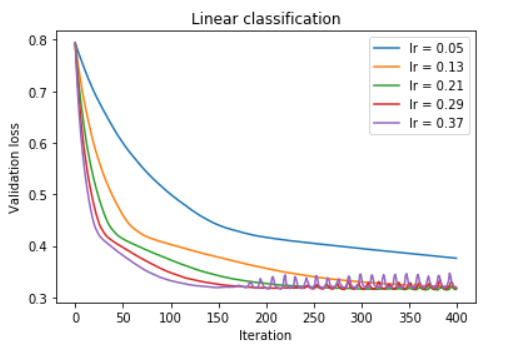
We chose 5 different learning rates [0.04, 0.13, 0.22, 0.31, 0.4] to optimize the linear regression algorithm, the results are as the following：



Result analysis:

1. 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.
2. If the learning rate is too large, the loss curve will be oscillating.
3. Linear classification
4. Learning rate

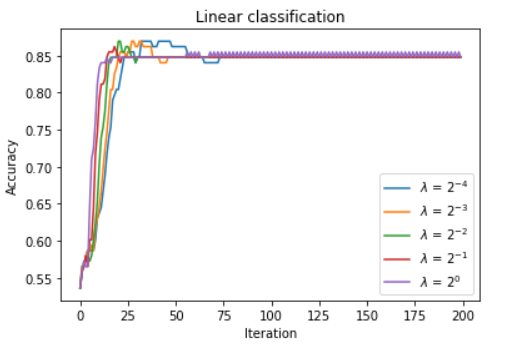
We chose 5 different learning rates [0.05, 0.13, 0.21, 0.29, 0.37] to optimize the linear regression algorithm, the results are as the following



Result analysis:

1. 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.
2. If the learning rate is too large, the loss curve will be oscillating.
3. Regularization parameter

We chose 5 different regularization parameter [, , , , ] to optimize the linear regression algorithm, the results are as the following



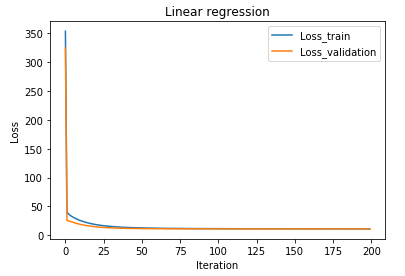
Result analysis:

1. 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.
2. If the regularization parameter is too small, the proportion of the error term will be small and the model may be under-fitting.

## Predicted Results (Best Results):

1. Linear regression

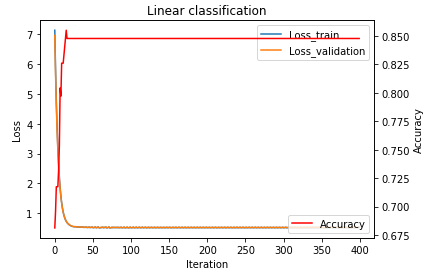
= 0.2, epoch = 200



Validation Loss = 11.29208

1. Linear classification

= 0.2, =0.5, epoch = 400

****

Validation Loss = 0.527841

Accuracy = 0.84826

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

1. Similarity: Linear regression and linear classification both use the linear model.
2. Difference:
3. Linear regression uses Least squared loss as the loss function, but linear classification updates the parameters by Hinge loss.
4. For the linear regression, we learning a model to simulate the mapping between input X and output y. But for the linear classification task, we want to find a hyper plane to separate the different target.
5. To evaluate the linear regression, we compare the validation loss. For linear classification, we can also evaluate the model by calculating the accuracy.