

**The Experiment Report of**

**Deep *Learning***

**College Software College**

**Subject Software Engineering**

**Members**  Tengyun Wang

**Student ID 201710106574**

**E-mail wty9391@gmail.com**

**Tutor**   **Mingkui Tang**

**Date submitted** **2017.12 .07**

1. **Topic: Linear Regression, Linear Classification and Gradient Descent**
2. **Time: 2017.12.02**

**3. Reporter: Tengyun Wang**

**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:**

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.

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:**

* **Linear Regression and Gradient Descent**

1. Load the experiment data. You can use load\_svmlight\_file function in sklearn library.
2. Devide dataset. You should divide dataset into training set and validation set using train\_test\_split function. Test set is not required in this experiment.
3. Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation: Find more detail in PPT.
5. Calculate gradient toward loss function from all samples.
6. Denote the opposite direction of gradient as .
7. Update model: . is learning rate, a hyper-parameter that we can adjust.
8. Get the loss under the training set and by validating under validation set.
9. Repeate step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

* **Linear Classification and Gradient Descent**

1. Load the experiment data.
2. Divide dataset into training set and validation set.
3. Initialize SVM model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation: Find more detail in PPT.
5. Calculate gradient toward loss function from all samples.
6. Denote the opposite direction of gradient as .
7. Update model: . is learning rate, a hyper-parameter that we can adjust.
8. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Get the loss under the trainin set and by validating under validation set.
9. Repeate step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

**7. Code:**

* **Linear Regression and Gradient Descent**

LinearRegressor.py

1. #!/usr/bin/env python3
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Fri Dec  8 11:31:25 2017
6. @author: wty
7. """
8. **import** numpy as np
9. **from** sklearn.base **import** BaseEstimator,RegressorMixin
11. **class** LinearRegressor(BaseEstimator,RegressorMixin):
12. """A Linear Regressor for ML2017-lab-01"""
14. **def** \_\_init\_\_(self, w=0, lamda=0, eta=0.1, max\_epoch=50):
15. """
16. Called when initializing the classifier
17. """
18. self.w = w
19. self.lamda = lamda
20. self.eta = eta
21. self.max\_epoch = max\_epoch
23. self.w\_history = []
25. **def** \_\_h(self,w,X):
26. **return** X.dot(w)
28. **def** h(self,X):
29. **return** self.\_\_h(self.w,X)
31. **def** \_\_error(self,w,X,Y):
32. **return** self.\_\_h(w,X) - Y
34. **def** error(self,X,Y):
35. **return** self.\_\_error(self.w,X,Y)
37. **def** L(self,X,Y):
38. **return** self.\_\_L(self.w,X,Y)
40. **def** \_\_L(self,w,X,Y):
41. num\_records,num\_features  = np.shape(X)
42. lamda = self.lamda
43. e = self.\_\_error(w,X,Y)
45. regulation\_loss = 1.0/2 \* lamda \* w.transpose().dot(w)
46. loss = 1.0/2 \* 1.0/float(num\_records) \* e.transpose().dot(e)\
47. + regulation\_loss
48. **return** loss[0][0]
50. **def** g(self,X,Y):
51. **return** self.\_\_g(self.w,X,Y)
53. **def** \_\_g(self,w,X,Y):
54. num\_records,num\_features  = np.shape(X)
55. lamda = self.lamda
56. e = self.\_\_error(w,X,Y)
57. # L2 norm
58. **return** 1.0/num\_records \* X.transpose().dot(e) \
59. + lamda \* w

62. **def** fit(self, X, Y):
63. """
64. A reference implementation of a fitting function
65. Parameters
66. ----------
67. X : array-like or sparse matrix of shape = [n\_samples, n\_features]
68. The training input samples.
69. y : array-like, shape = [n\_samples] or [n\_samples, n\_outputs]
70. The target values (class labels in classification, real numbers in
71. regression).
72. Returns
73. -------
74. self : object
75. Returns self.
76. """
77. **for** epoch **in** range(self.max\_epoch):
78. self.w = self.w - self.eta \* self.g(X,Y)
79. self.w\_history.append(self.w)
81. **return** self
83. **def** \_\_predict(self,w,X):
84. **return** self.\_\_h(w,X)
86. **def** predict(self, X):
87. """ A reference implementation of a predicting function.
88. Parameters
89. ----------
90. X : array-like of shape = [n\_samples, n\_features]
91. The input samples.
92. Returns
93. -------
94. y : array of shape = [n\_samples]
95. Returns :math:`x^2` where :math:`x` is the first column of `X`.
96. """
97. **return** self.\_\_predict(self.w,X)
99. **def** \_\_score(self,w,X,Y):
100. **return** -np.sqrt(((self.\_\_predict(w,X) - Y)\*\*2).sum()/len(Y))
102. **def** score(self, X, Y):
103. # RMSE
104. **return** self.\_\_score(self.w,X,Y)
106. **def** getLossHistory(self,X,Y):
107. **return** [self.\_\_L(w,X,Y) **for** w **in** self.w\_history]
109. **def** getScoreHistory(self,X,Y):
110. **return** [self.\_\_score(w,X,Y) **for** w **in** self.w\_history]

RegressionExperiment.py

1. # write your code here#!/usr/bin/env python3
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Sat Dec  2 09:58:38 2017
6. @author: wty
7. """
8. **import** numpy as np
9. **import** scipy
10. **import** matplotlib.pyplot as plt
11. **import** pandas as pd
13. **from** sklearn.datasets **import** load\_svmlight\_file
14. **from** sklearn.model\_selection **import** train\_test\_split,GridSearchCV
16. **import** LinearRegressor
18. data = load\_svmlight\_file("./resources/housing\_scale.txt")
19. result\_path = './results/regression\_grid\_search.csv'
20. # add interception
21. X = scipy.sparse.hstack(\
22. (scipy.sparse.csr\_matrix(np.ones((len(data[1]),1))),data[0]))
23. Y = data[1].reshape((len(data[1]),1))
25. num\_records,num\_features  = np.shape(X)
27. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(\
28. X, Y, test\_size=0.33,random\_state=5)
29. Y\_train = Y\_train.reshape((len(Y\_train),1))
30. Y\_test = Y\_test.reshape((len(Y\_test),1))
32. # initialize w
33. init\_w = np.random.normal(size=(num\_features,1))
35. max\_iterate = 100
37. param\_grid = [
38. {'lamda': [0, 0.01, 0.1, 0.5, 1], 'eta': [0.05, 0.1, 0.15, 0.2, 0.3]}
39. ]

42. rgs = GridSearchCV(LinearRegressor.LinearRegressor(init\_w,max\_epoch=max\_iterate), param\_grid,return\_train\_score=True)
43. rgs.fit(X\_train,Y\_train)
44. result = pd.DataFrame(rgs.cv\_results\_)
45. result.sort\_values('rank\_test\_score',inplace=True)
47. **print** ("Exhaustive Grid Search Result")
48. **print** ("The best estimator's parameter is",rgs.best\_params\_)
49. **print** (result.loc[:,['rank\_test\_score','mean\_test\_score','mean\_train\_score','mean\_fit\_time','params']])
50. result.to\_csv(result\_path)
51. **print** ("Result has been saved in",result\_path)
53. **print** ("Printing the best 5 models loss curves")
54. **for** i **in** range(5):
55. **print** ("Figure of",result.loc[i,'params'])
56. regressor = LinearRegressor.LinearRegressor(init\_w,max\_epoch=max\_iterate,\*\*result.loc[i,'params'])
57. regressor.fit(X\_train,Y\_train)
58. loss\_train = regressor.getLossHistory(X\_train,Y\_train)
59. loss\_test = regressor.getLossHistory(X\_test,Y\_test)
60. RMSE\_train = [-s **for** s **in** regressor.getScoreHistory(X\_train,Y\_train)]
61. RMSE\_test = [-s **for** s **in** regressor.getScoreHistory(X\_test,Y\_test)]
63. plt.figure(i)
64. plt.subplot(211)
65. train\_loss\_line = plt.plot(range(max\_iterate),loss\_train,label='train loss')
66. test\_loss\_line = plt.plot(range(max\_iterate),loss\_test,label='test loss')
67. ax=plt.gca()
68. ax.set(xlabel='Epoch', ylabel='Loss with l2 norm')
69. plt.legend()
71. plt.subplot(212)
72. train\_RMSE\_line = plt.plot(range(max\_iterate),RMSE\_train,label='train RMSE')
73. test\_RMSE\_line = plt.plot(range(max\_iterate),RMSE\_test,label='test RMSE')
74. ax=plt.gca()
75. ax.set(xlabel='Epoch', ylabel='RMSE')
76. plt.legend()
78. plt.show()

* **Linear Classification and Gradient Descent**

LinearClassifier.py

1. #!/usr/bin/env python3
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Fri Dec  8 11:34:48 2017
6. @author: wty
7. """
9. **import** numpy as np
10. **from** sklearn.base **import** BaseEstimator,ClassifierMixin
12. **class** LinearClassifier(BaseEstimator,ClassifierMixin):
13. """A Linear Classifier for ML2017-lab-01"""
15. **def** \_\_init\_\_(self, w=0, lamda=0, eta=0.1, C=1.0, threshold=0.5, max\_epoch=50):
16. """
17. Called when initializing the classifier
18. """
19. self.w = w
20. self.lamda = lamda
21. self.eta = eta
22. self.C = C
23. self.threshold = threshold
24. self.max\_epoch = max\_epoch
26. self.w\_history = []
28. **def** \_\_h(self,w,X):
29. **return** X.dot(w)
31. **def** h(self,X):
32. **return** self.\_\_h(self.w,X)
34. **def** \_\_hinge\_loss(self,w,X,Y):
35. num\_records,num\_features  = np.shape(X)
36. C = self.C
37. zero = np.zeros((num\_records,1))
38. margin = 1 - C \* Y \* self.\_\_h(w,X)
39. **return** np.max([zero,margin],axis=0)
41. **def** hinge\_loss(self,X,Y):
42. **return** self.\_\_hinge\_loss(self.w,X,Y)
44. **def** L(self,X,Y):
45. **return** self.\_\_L(self.w,X,Y)
47. **def** \_\_L(self,w,X,Y):
48. num\_records,num\_features  = np.shape(X)
49. lamda = self.lamda
50. e = self.\_\_hinge\_loss(w,X,Y)
51. regulation\_loss = 1.0/2 \* lamda \* w.transpose().dot(w)
52. loss = 1.0/float(num\_records) \* e.sum()  + regulation\_loss
53. **return** loss[0][0]
55. **def** g(self,X,Y):
56. **return** self.\_\_g(self.w,X,Y)
58. **def** \_\_g(self,w,X,Y):
59. num\_records,num\_features  = np.shape(X)
60. C = self.C
61. lamda = self.lamda
62. e = self.\_\_hinge\_loss(w,X,Y)
63. indicator = np.zeros((num\_records,1))
64. indicator[np.nonzero(e)] = 1
66. **return** - 1.0/float(num\_records) \* C \
67. \* X.transpose().dot(Y \* indicator).sum(axis=1).reshape((num\_features,1)) \
68. + lamda \* w

71. **def** fit(self, X, Y):
72. """
73. A reference implementation of a fitting function
74. Parameters
75. ----------
76. X : array-like or sparse matrix of shape = [n\_samples, n\_features]
77. The training input samples.
78. y : array-like, shape = [n\_samples] or [n\_samples, n\_outputs]
79. The target values (class labels in classification, real numbers in
80. regression).
81. Returns
82. -------
83. self : object
84. Returns self.
85. """
86. self.classes\_, \_ = np.unique(Y, return\_inverse=True)
87. **for** epoch **in** range(self.max\_epoch):
88. self.w = self.w - self.eta \* self.g(X,Y)
89. self.w\_history.append(self.w)
91. **return** self
93. **def** \_\_predict(self,w,X):
94. threshold = self.threshold
95. raw = self.\_\_h(w,X)
96. raw[raw<=threshold] = self.classes\_[0]
97. raw[raw>threshold] = self.classes\_[1]
98. **return** raw
100. **def** predict(self, X):
101. """ A reference implementation of a predicting function.
102. Parameters
103. ----------
104. X : array-like of shape = [n\_samples, n\_features]
105. The input samples.
106. Returns
107. -------
108. y : array of shape = [n\_samples]
109. Returns :math:`x^2` where :math:`x` is the first column of `X`.
110. """
111. **return** self.\_\_predict(self.w,X)
113. **def** \_\_score(self,w,X,Y):
114. num\_records,num\_features  = np.shape(X)
115. P = self.\_\_predict(w,X)
117. is\_right = P \* Y
118. is\_right[is\_right < 0] = 0
120. **return** 1.0/num\_records \* np.count\_nonzero(is\_right)
122. **def** score(self, X, Y):
123. # RMSE
124. **return** self.\_\_score(self.w,X,Y)
126. **def** getLossHistory(self,X,Y):
127. **return** [self.\_\_L(w,X,Y) **for** w **in** self.w\_history]
129. **def** getScoreHistory(self,X,Y):
130. **return** [self.\_\_score(w,X,Y) **for** w **in** self.w\_history]

ClassificationExperiment.py

1. # -\*- coding: utf-8 -\*-
3. **import** numpy as np
4. **import** scipy
5. **import** matplotlib.pyplot as plt
6. **import** pandas as pd
8. **from** sklearn.datasets **import** load\_svmlight\_file
9. **from** sklearn.model\_selection **import** train\_test\_split,GridSearchCV
11. **import** LinearClassifier
13. data = load\_svmlight\_file("./resources/australian\_scale.txt")
14. result\_path = './results/classification\_grid\_search.csv'
15. # add interception
16. X = scipy.sparse.hstack(\
17. (scipy.sparse.csr\_matrix(np.ones((len(data[1]),1))),data[0]))
18. Y = data[1].reshape((len(data[1]),1))
20. num\_records,num\_features  = np.shape(X)
22. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(\
23. X, Y, test\_size=0.33,random\_state=42)
24. Y\_train = Y\_train.reshape((len(Y\_train),1))
25. Y\_test = Y\_test.reshape((len(Y\_test),1))
27. # initialize w
28. init\_w = np.random.normal(size=(num\_features,1))
30. max\_iterate = 250
32. param\_grid = [
33. {'lamda': [0, 0.1, 0.5],
34. 'eta': [0.01, 0.02, 0.05, 0.1],
35. 'C': [1, 2],
36. 'threshold': [0.4,0.5,0.6]}
37. ]

40. cls = GridSearchCV(LinearClassifier.LinearClassifier(init\_w,max\_epoch=max\_iterate), param\_grid,return\_train\_score=True)
41. cls.fit(X\_train,Y\_train)
42. result = pd.DataFrame(cls.cv\_results\_)
43. result.sort\_values('rank\_test\_score',inplace=True)
44. result = result.reset\_index(drop = True)
46. **print** ("Exhaustive Grid Search Result")
47. **print** ("The best estimator's parameter is",cls.best\_params\_)
48. **print** (result.loc[:,['rank\_test\_score','mean\_test\_score','mean\_train\_score','mean\_fit\_time','params']])
49. result.to\_csv(result\_path)
50. **print** ("Result has been saved in",result\_path)
52. **print** ("Printing the best 5 models loss curves")
53. **for** i **in** range(5):
54. params = result.loc[i,'params']
55. **print** ("Figure of",params)
56. classifier = LinearClassifier.LinearClassifier(init\_w,max\_epoch=max\_iterate,\*\*params)
57. classifier.fit(X\_train,Y\_train)
58. loss\_train = classifier.getLossHistory(X\_train,Y\_train)
59. loss\_test = classifier.getLossHistory(X\_test,Y\_test)
60. accuracy\_train = classifier.getScoreHistory(X\_train,Y\_train)
61. accuracy\_test = classifier.getScoreHistory(X\_test,Y\_test)
63. plt.figure(i)
64. fig, ax = plt.subplots()
65. ax\_e = ax.twinx()
66. train\_loss\_line = ax.plot(range(max\_iterate),loss\_train,label='train loss')
67. test\_loss\_line = ax.plot(range(max\_iterate),loss\_test,label='test loss')
68. train\_accuracy\_line = ax\_e.plot(range(max\_iterate),accuracy\_train,'r',label='train accuracy')
69. test\_accuracy\_line = ax\_e.plot(range(max\_iterate),accuracy\_test,'g',label='test accuracy')
71. ax.set(xlabel='Epoch', ylabel='Loss with l2 norm')
72. ax\_e.set\_ylabel('Accuracy with threshold='+str(params['threshold']))
74. ax.legend(loc=4)
75. ax\_e.legend(loc=1)
76. plt.show()

(Fill in the contents of 8-12 respectively for linear regression and linear classification)

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

* **Linear Regression and Gradient Descent**

hold-out，Train Set: Test Set = 3 : 1, Exhaustive Grid Search

param\_grid = [

{'lamda': [0, 0.01, 0.1, 0.5, 1], 'eta': [0.05, 0.1, 0.15, 0.2, 0.3]}

]

The score is the opposite number of **RMSE**



* **Linear Classification and Gradient Descent**

hold-out，Train Set: Test Set = 3 : 1, Exhaustive Grid Search

param\_grid = [

{'lamda': [0, 0.1, 0.5],

'eta': [0.01, 0.02, 0.05, 0.1],

'C': [1, 2],

'threshold': [0.4,0.5,0.6]}

]

The score is the **accuracy** of classification model



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

* **Linear Regression and Gradient Descent**

**w = np.random.normal(size=(num\_features,1))**

* **Linear Classification and Gradient Descent**

**w = np.random.normal(size=(num\_features,1))**

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

* **Linear Regression and Gradient Descent**
* **Linear Classification and Gradient Descent**

**11. Experimental results and curve:**

* **Linear Regression and Gradient Descent**

Printing the best 5 models loss curves

Figure of {'eta': 0.05, 'lamda': 0}

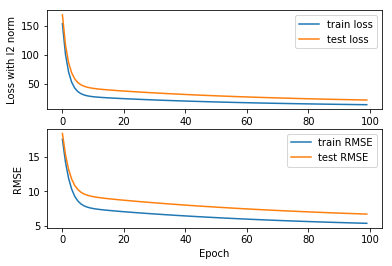


Figure of {'eta': 0.05, 'lamda': 0.01}

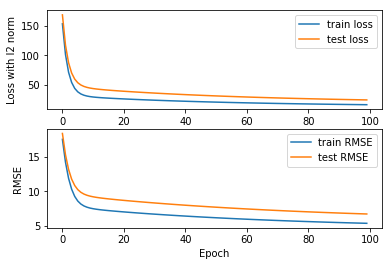


Figure of {'eta': 0.05, 'lamda': 0.1}

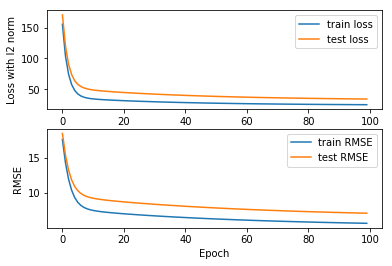


Figure of {'eta': 0.05, 'lamda': 0.5}

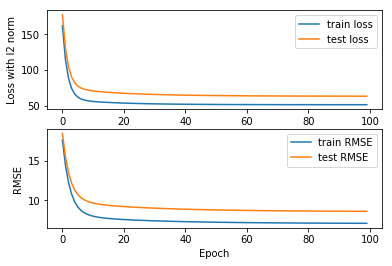
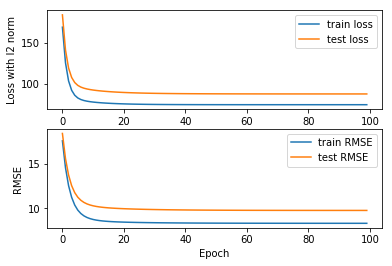


Figure of {'eta': 0.05, 'lamda': 1}



* **Linear Classification and Gradient Descent**

Printing the best 5 models loss curves

Figure of {'C': 1, 'eta': 0.05, 'lamda': 0.1, 'threshold': 0.6}

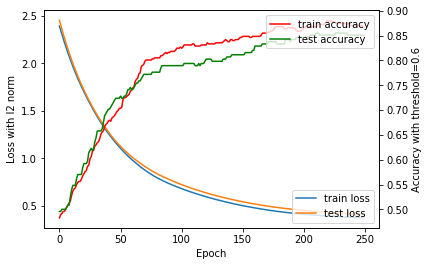


Figure of {'C': 1, 'eta': 0.1, 'lamda': 0.5, 'threshold': 0.4}

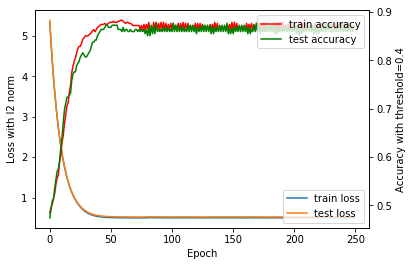


Figure of {'C': 1, 'eta': 0.1, 'lamda': 0.1, 'threshold': 0.6}

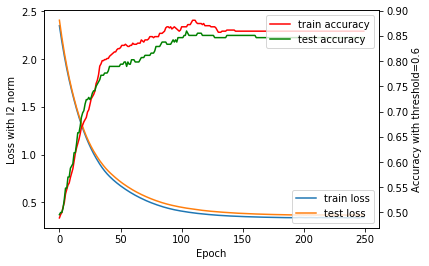


Figure of {'C': 1, 'eta': 0.05, 'lamda': 0.5, 'threshold': 0.4}

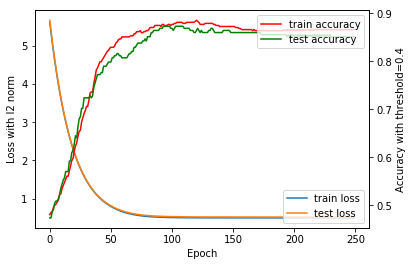
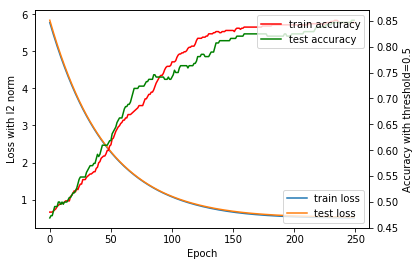


Figure of {'C': 1, 'eta': 0.02, 'lamda': 0.5, 'threshold': 0.5}



* **12. Results analysis:**

Gradient decent is a valid method to optimize both regression problem and classification problem.

At the beginning of optimization, the loss decrease quickly and the accuracy of model increase sharply. With epoch goes by, the model verges to be optimized.

The greater the learning rate is, the quicker the loss decreases. But large learning rate may lead to vibrating.

The metrics to evaluate models are important. In genera, a smaller loss leads to better metrics. But there are exceptions, see the relationship between accuracy and loss in classification experiments.

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

In linear regression, the evaluation metric is often MSE(mean square error).

In linear classification, the evaluation metric is often accuracy. And the threshold is a important factor to influence the accuracy.

**14. Summary:**

Gradient decent is a valid method to optimize both regression problem and classification problem.

There are many differences between linear regression and linear classification.