

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**

**Student ID 201530661741**

**E-mail yimingzhao0@qq.com**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017. .**

**1. Topic: Linear regression, linear classification and gradient decent method.**

**2. Time:** 2017.12.2 AM 9:00-12:00

**3. Reporter:** Yiming Zhao

**4. Purposes:**

* Further understand the principle of linear regression and gradient decent method.
* Practicing on a small data set
* Understand the process of optimizing the super parameters

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

In the linear regression, using the Housing data set from LIBSVM Data, which include 506 samples and 13 features in each sample.

In the linear classification, using the Australian data set from LIBSVM Data, which include 690 samples and 14 features in each sample.

**6. Experimental steps:**

* Using the function load\_svmlignt\_file from sklearn read the data set.
* Using function train\_test\_split from sklearn to spilt the data into train set and validation set.
* Initializing the parameters randomly.
* Using MSE loss and compute all the samples’ loss.
* Update the loss function by gradient decent.
* Repeat the above steps for some times and draw the figure of loss-iteration.

**7. Code:**

from sklearn import datasets as ds  
from sklearn.model\_selection import train\_test\_split  
import numpy as np  
from numpy import random  
import matplotlib.pyplot as plt  
  
  
  
def train(x\_train, y\_train, x\_test, y\_test, iters\_train, loss\_train, loss\_val):  
  
 max\_iterations = 100  
 num\_samples, num\_features = x\_train.shape  
 num\_test\_samples, num\_test\_features = x\_test.shape  
 theta = random.rand(num\_features)  
 gamma = 1  
  
 lr = 0.01  
  
 for i in range(max\_iterations):  
 iters.append(i)  
 grad = 0  
 for sample in range(num\_samples):  
 output = np.dot(x\_train[sample], theta)  
 diff = y\_train[sample] - output  
 each\_grad = np.dot(x\_train[sample], diff)  
 grad += each\_grad  
 grad = (-2 \* grad) / num\_samples + gamma \* theta  
  
 theta -= lr \* grad  
  
 predict\_error = 0  
 for j in range(num\_test\_samples):  
 predict\_output = np.dot(x\_test[j], theta)  
 predict\_error += np.dot((predict\_output - y\_test[j]), (predict\_output - y\_test[j])) + 0.5 \* gamma \* np.dot(theta, theta)  
 predict\_error /= num\_test\_samples  
 print(str(i) + '\t' + str(predict\_error))  
  
 loss\_train.append(predict\_error)  
  
 train\_error = 0  
 for j in range(num\_samples):  
 predict\_output = np.dot(x\_train[j], theta)  
 train\_error += np.dot((predict\_output - y\_train[j]), (predict\_output - y\_train[j])) + 0.5 \* gamma \* np.dot(theta, theta)  
 train\_error /= num\_samples  
 print(str(i) + '\t' + str(train\_error))  
  
 loss\_val.append(train\_error)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x\_train, y\_train = ds.load\_svmlight\_file('./data/housing')  
  
 x\_train = x\_train.toarray()  
 temp = np.ones(shape=[506, 1], dtype=np.float32)  
 x\_train = np.concatenate([x\_train, temp], axis=1)  
  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_train, y\_train, test\_size=0.367, random\_state=42)  
  
 num\_samples, num\_features = x\_train.shape  
 num\_test\_samples, num\_test\_features = x\_test.shape  
  
 iters = []  
 test\_errors = []  
 train\_errors = []  
 train(x\_train, y\_train, x\_test, y\_test, iters, test\_errors, train\_errors)  
 plt.plot(iters, test\_errors, label='validation loss')  
 plt.plot(iters, train\_errors, label='training loss')  
  
 plt.xlabel('Iteration')  
 plt.ylabel('Loss')  
 plt.legend()  
 plt.show()

from sklearn import datasets as ds  
from sklearn.cross\_validation import train\_test\_split  
import numpy as np  
import matplotlib.pyplot as plt  
import os   
import matplotlib.pyplot as plt  
  
feature\_size = 14  
x, y = ds.load\_svmlight\_file("./data/australian")  
train\_x, val\_x, train\_y, val\_y = train\_test\_split(x, y, test\_size=0.3)  
  
  
train\_x = train\_x.toarray().astype(np.float32)  
temp = np.ones(shape=[len(train\_y), 1], dtype=np.float32)  
train\_x = np.concatenate([train\_x, temp], axis=1)  
val\_x = val\_x.toarray().astype(np.float32)  
temp = np.ones(shape=[len(val\_y), 1], dtype=np.float32)  
val\_x = np.concatenate([val\_x, temp], axis=1)  
train\_y = train\_y.astype(np.float32).reshape([len(train\_y), 1])  
val\_y = val\_y.astype(np.float32).reshape([len(val\_y), 1])  
  
ite = []  
train\_loss\_set = []  
val\_loss\_set = []  
  
w = np.random.rand(feature\_size + 1, 1)  
bias = np.zeros(shape=[feature\_size + 1, 1])  
bias[len(bias)-1][0] = 1.  
  
# training  
iteration = 100  
lr = 1e-6  
C = 0.1  
for i in range(0, iteration):  
 ite.append(i)  
 pred = np.matmul(train\_x, w)  
 hinge\_loss = np.maximum(1 - train\_y \* pred, 0)  
 train\_loss = np.mean(hinge\_loss \*\* 2) + C \* np.sum((w - bias) \*\* 2)  
 gradient = -np.matmul(train\_x.transpose(), hinge\_loss \* train\_y) / len(train\_y)  
 w -= lr \* (gradient + 2 \* C \* (w - w[len(w) - 1][0] \* bias))  
 train\_loss\_set.append(train\_loss)  
  
 val\_pred = np.matmul(val\_x, w)  
 val\_hinge\_loss = np.maximum(1 - val\_y \* val\_pred, 0)  
 val\_loss = np.mean(val\_hinge\_loss \*\* 2) + C \* np.sum((w - w[len(w) - 1][0] \* bias) \*\* 2)  
 val\_loss\_set.append(val\_loss)  
  
plt.plot(ite, train\_loss\_set, label='train')  
plt.plot(ite, val\_loss\_set, label='validation')  
plt.xlabel('Iteration')  
plt.ylabel('Loss')  
plt.legend()  
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.):** hold-out

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

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

**Linear Regression:**

The include the bias of the linear regression.

**Linear Classification:**

**11. Experimental results and curve:**

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

Linear regression:

* Learning rate: 0.01
* Gamma = 1
* Iteration = 100

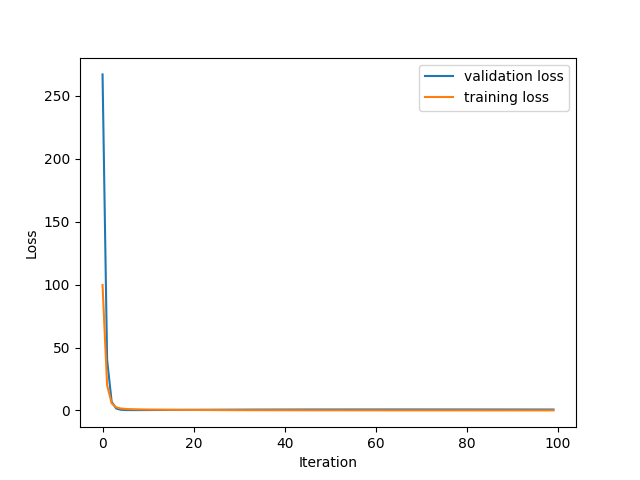
Linear classification:

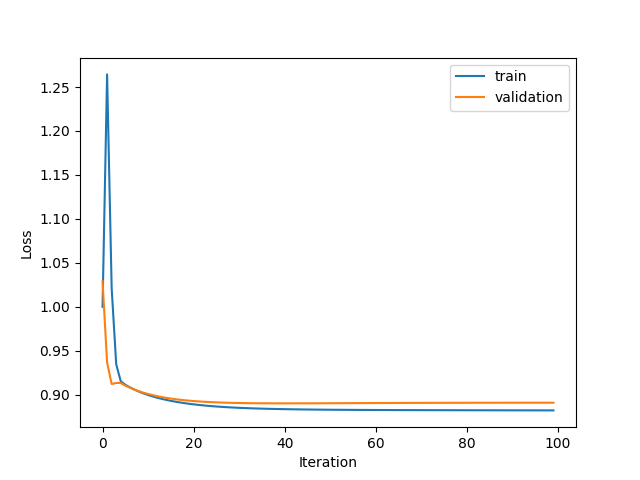
* Learning rate: 0.01
* Gamma = 0.1
* Iteration = 100

## Assessment Results (based on selected validation):

## Predicted Results (Best Results):

## Loss curve:





**12. Results analysis:**

Most experiments can be converged after 20 iterations.

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

**14. Summary:**