

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **盛栋铭**

**Student ID 201530612699**

**E-mail 913951422@qq.com**

**Tutor**   **谭明奎**

**Date submitted** **2017. 12 . 3**

1. **Topic:** **Linear Regression, Linear Classification and Gradient Descent**
2. **Time: December 2nd, 2017**
3. **Reporter: 盛栋铭**
4. **Purposes:**
5. Further understand of linear regression and gradient descent.
6. Conduct some experiments under small scale dataset.
7. Realize the process of optimization and adjusting parameters.
8. **Data sets and data analysis:**

Linear Regression uses **Housing**(scaled edition) in LIBSVM Data, including 506 samples and each sample has 13 features.

Linear classification uses **australian**(scaled edition) in LIBSVM Data, including 690 samples and each sample has 14 features.

1. **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 mean value *G* of gradient toward loss function from all samples.
6. Denote the opposite direction of gradient *G* as *G’*.
7. Update model: . *η* is learning rate, a hyper-parameter that we can adjust.
8. Get the loss *Ltrain* under the training set and*Lvalidation* by validating under validation set.
9. Repeate step 5 to 8 for several times, and **drawing graph of** *Ltrain* **as well as** *Lvalidation* **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 mean value *G* of gradient toward loss function from all samples.
6. Denote the opposite direction of gradient *G* as *G’* .
7. Update model: . *η* 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 *Ltrain* under the trainin set and *Lvalidation* by validating under validation set.
9. Repeate step 5 to 8 for several times, and **drawing graph of** *Ltrain* **as well as** *Lvalidation* **with the number of iterations**.
10. **Code:**

*Linear Regression and Gradient Descent*

import pandas as pd

import numpy as np

from sklearn import datasets as ds

from sklearn.model\_selection import train\_test\_split

#load datasets

X, y = ds.load\_svmlight\_file('data/housing\_scale')

X = X.todense()

#train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

X\_train = np.array(X\_train)

X\_train.shape, y\_train.shape

#param init:all zeros

def zeroInit():

    return np.zeros(X\_train.shape[1])

# define l2 loss

def L2loss(y\_true, y\_pred):

    return 1/2 \* np.sum(np.square((y\_true - y\_pred)))

#cal gradient

def gradient(X, y, w):

    return -np.dot(X.T, y) + np.dot(np.dot(X.T, X), w)

#initialize w

w = zeroInit()

learning\_rate = 0.0015

iter\_num = 50

#store losses

train\_loss\_history = []

test\_loss\_history = []

#start training process

for i in range(iter\_num):

    train\_loss = L2loss(y\_train, np.dot(X\_train, w))

    train\_loss\_history.append(train\_loss)

    test\_loss = L2loss(y\_test, np.dot(X\_test, w))

    test\_loss\_history.append(test\_loss)

    print('iter ' + str(i) + ':', train\_loss, test\_loss)

#     print('train L2loss', L2loss(y\_train, np.dot(X\_train, w)))

#     print('test L2loss', L2loss(y\_test, np.dot(X\_test, w)))

    w -= learning\_rate \* gradient(X\_train, y\_train, w)

#plotting block

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel('iteration number')

plt.ylabel('loss')

plt.plot(range(iter\_num), train\_loss\_history,'r', label='train loss')

plt.plot(range(iter\_num), test\_loss\_history,'b',label='test loss')

plt.legend()

plt.grid()

plt.show()

*Linear Classification and Gradient Descent*

import pandas as pd

import numpy as np

from sklearn import datasets as ds

from sklearn.model\_selection import train\_test\_split

#load datasets

X, y = ds.load\_svmlight\_file('data/australian\_scale')

X = X.todense()

#train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

X\_train = np.array(X\_train)

X\_train.shape, y\_train.shape

#param init:all zeros

def zeroInit():

    #w, b

    return np.zeros(X\_train.shape[1]), np.zeros(X\_train.shape[0])

# define loss

def loss(X, y, w, b, C):

    return 1/2 \* np.sum(np.square(w)) + C \* np.sum(hingeLoss(X, y, w, b))

#define hinge loss

def hingeLoss(X, y, w, b):

    #cannot use np.max here: two arrays

    return np.maximum(0, 1-np.multiply(y, np.dot(X, w)+b))

#cal gradients

def gradient\_w(X, y, w, b, C):

    margin = 1-np.multiply(y, np.dot(X, w)+b)

    minus\_y = -y

    minus\_y[margin<0] = 0

    return w + C \* np.dot(X.T, minus\_y)

def gradient\_b(X, y, w, b, C):

    margin = 1-np.multiply(y, np.dot(X, w)+b)

    minus\_y = -y

    minus\_y[margin<0] = 0

    return C \* np.sum(minus\_y)

#param init:all zeros

def zeroInit():

    #w, b

    return np.zeros(X\_train.shape[1]), np.zeros(X\_train.shape[0])

# define loss

def loss(X, y, w, b, C):

    return 1/2 \* np.sum(np.square(w)) + C \* np.sum(hingeLoss(X, y, w, b))

#define hinge loss

def hingeLoss(X, y, w, b):

    #cannot use np.max here: two arrays

    return np.maximum(0, 1-np.multiply(y, np.dot(X, w)+b))

#cal gradients

def gradient\_w(X, y, w, b, C):

    margin = 1-np.multiply(y, np.dot(X, w)+b)

    minus\_y = -y

    minus\_y[margin<0] = 0

    return w + C \* np.dot(X.T, minus\_y)

def gradient\_b(X, y, w, b, C):

    margin = 1-np.multiply(y, np.dot(X, w)+b)

    minus\_y = -y

    minus\_y[margin<0] = 0

    return C \* np.sum(minus\_y)

#initialize w, b

w, b = zeroInit()

learning\_rate = 0.003

iter\_num = 100

C = 0.1

#store losses

train\_loss\_history = []

test\_loss\_history = []

#start training process

for i in range(iter\_num):

    train\_loss = loss(X\_train, y\_train, w, b, C)

    train\_loss\_history.append(train\_loss)

    test\_loss = loss(X\_test, y\_test, w, b, C)

    test\_loss\_history.append(test\_loss)

    print('iter ' + str(i) + ':', train\_loss, test\_loss)

#     print('train L2loss', L2loss(y\_train, np.dot(X\_train, w)))

#     print('test L2loss', L2loss(y\_test, np.dot(X\_test, w)))

    w -= learning\_rate \* gradient\_w(X\_train, y\_train, w, b, C)

    b -= learning\_rate \* gradient\_b(X\_train, y\_train, w, b, C)

#plotting block

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel('iteration number')

plt.ylabel('loss')

plt.plot(range(iter\_num), train\_loss\_history,'r', label='train loss')

plt.plot(range(iter\_num), test\_loss\_history,'b',label='test loss')

plt.legend()

plt.grid()

plt.show()

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

I use hold-out method in these two labs.

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

I use zero to initialize the parameters in these two labs.

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

For linear regression:

For SVM:

1. **Experimental results and curve:**

*Linear Regression and Gradient Descent*

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

η=0.0015, epoch=50

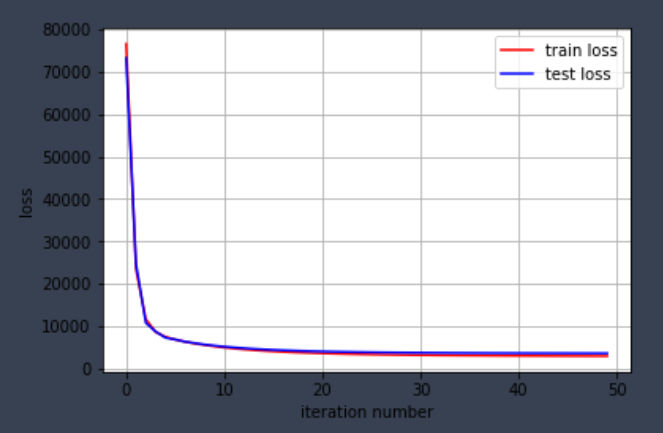
## Assessment Results (based on selected validation):

3613.0021429

## Predicted Results (Best Results):

3613.0021429 (regarding test set as the validation set)

## Loss curve:



*Linear Classification and Gradient Descent*

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

η=0.003, epoch=100, C=0.1

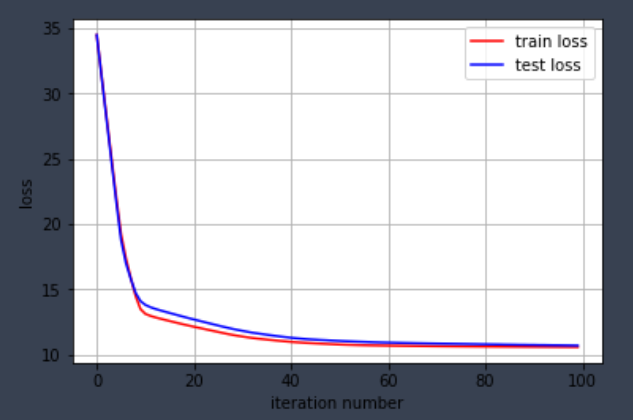
## Assessment Results (based on selected validation):

10.6830472544

## Predicted Results (Best Results):

10.6830472544 (regarding test set as the validation set)

## Loss curve:



1. **Results analysis:**

From the figures above, I find that training loss is roughly the same as testing loss, which means that training data and testing data have similar distributions. (I use a testing set which has the same size as training set.)

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

Differences:

1. Different labels:

Labels in linear regression are continuous values, while in linear classification, labels are discrete values 0 and 1.

1. Different loss functions:

In linear regression, we use mean square loss, while in linear classification, we use hinge loss.

Similarities:

1. We use gradient descent to optimize these two problems.
2. Similar models: linear models
3. **Summary:**

It’s a great lab. I learned how to use gradient descent in optimizing linear regression and linear svm. Although it was not my first time to create machine learning models, I did spend a lot of time trying to debug and speed up my code using vector operations, which is definitely a great experience.