

**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=0.3, random\_state=42)

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

X\_train.shape, y\_train.shape, X\_test.shape, y\_test.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)))

#evaluation metric

def mse(y\_true, y\_pred):

    return np.sum(np.square((y\_true - y\_pred)))/len(y\_true)

#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 = 0.001

iter\_num = 200

#store losses

train\_loss\_history = []

test\_loss\_history = []

train\_metric\_history = []

test\_metric\_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)

    train\_metric = mse(y\_train, np.dot(X\_train, w))

    train\_metric\_history.append(train\_metric)

    test\_metric = mse(y\_test, np.dot(X\_test, w))

    test\_metric\_history.append(test\_metric)

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

#     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)

print('best metric score: ', test\_metric\_history[-1])

#plotting loss

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()

#plotting metric

plt.xlabel('iteration number')

plt.ylabel('metric value')

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

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

plt.legend()

plt.grid()

plt.show()

*Linear Classification and Gradient Descent*

# coding: utf-8

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=0.3, random\_state=42)

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

#transformation trick: combine w and b together

X\_train = np.concatenate((X\_train, np.ones((X\_train.shape[0], 1))), axis=1)

X\_test = np.concatenate((X\_test, np.ones((X\_test.shape[0], 1))), axis=1)

X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape

#param init:all zeros

def zeroInit():

    #[w b]^T

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

# define loss

def loss(X, y, w, C):

    return 1/2 \* np.sum(np.square(w[:-1])) + C \* np.sum(hingeLoss(X, y, w))

# define metric

def accuracy(y\_true, y\_pred):

    return np.sum(y\_true==y\_pred) / len(y\_true)

#define hinge loss

def hingeLoss(X, y, w):

    #cannot use np.max here: two arrays

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

#cal gradients

def gradient(X, y, w, C):

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

    minus\_y = -y

    minus\_y[margin<0] = 0

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

    res[-1] -= w[-1] #b

    return res

# define prediction

def predict(X, w, threshold):

    res = np.dot(X, w)

    res[res>=threshold] = 1

    res[res<threshold] = -1

    return res

#initialize w, b

w = zeroInit()

learning\_rate = 0.0001

iter\_num = 2000

C = 0.01

threshold = 0

#store losses

train\_loss\_history = []

test\_loss\_history = []

train\_accuracy\_history = []

test\_accuracy\_history = []

#start training process

for i in range(iter\_num):

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

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

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

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

    train\_accuracy = accuracy(y\_train, predict(X\_train, w, threshold))

    train\_accuracy\_history.append(train\_accuracy)

    test\_accuracy = accuracy(y\_test, predict(X\_test, w, threshold))

    test\_accuracy\_history.append(test\_accuracy)

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

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

#plotting block

import matplotlib.pyplot as plt

get\_ipython().magic('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()

plt.xlabel('iteration number')

plt.ylabel('%')

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

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

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=0.001, epoch=200

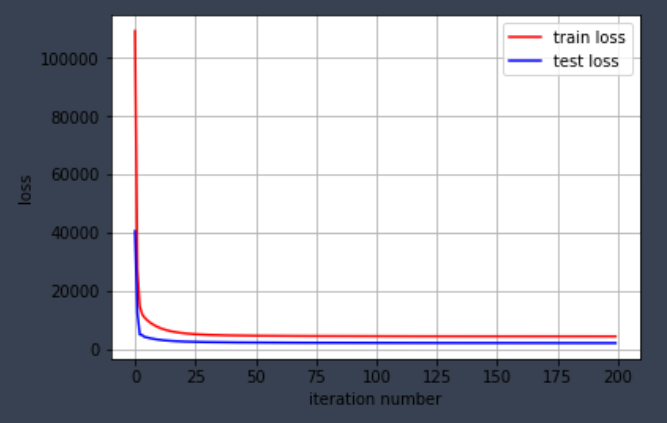
## Assessment Results (based on selected validation):

26.1400931477 (Metric: mean square error)

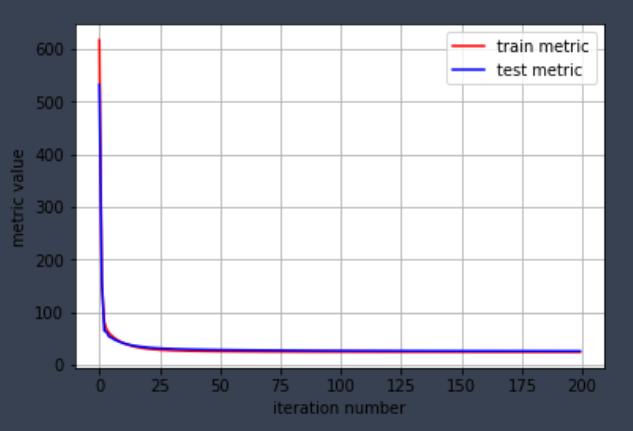
## Predicted Results (Best Results):

26.1400931477 (regarding test set as validation set)

## Loss curve:



## Metric curve(mse):



*Linear Classification and Gradient Descent*

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

η=0.0001, epoch=2000, C=0.01, threshold = 0

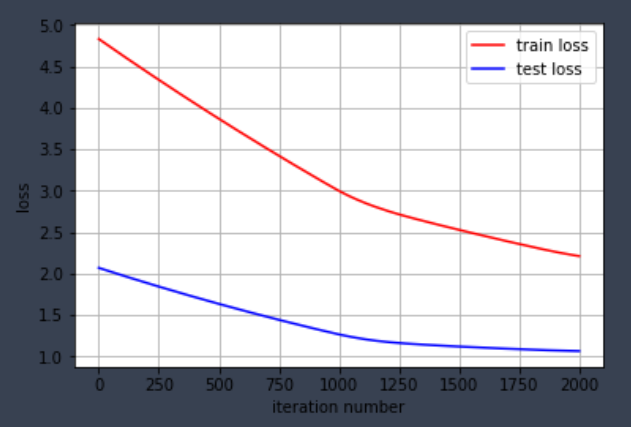
## Assessment Results (based on selected validation):

89.3719806763% (Metric: accuracy)

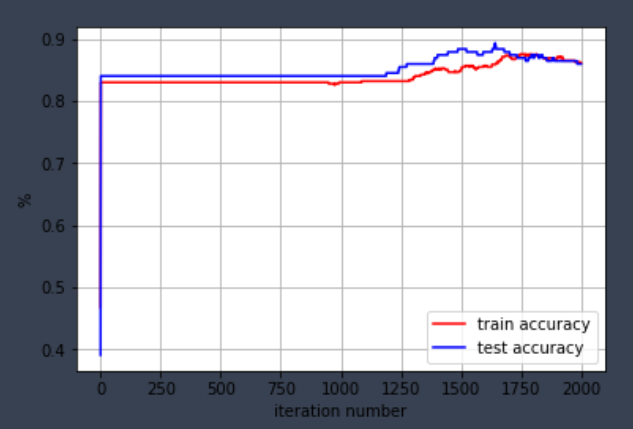
## Predicted Results (Best Results):

89.3719806763% (regarding test set as the validation set)

## Loss curve:



## Metric curve(accuracy):



1. **Results analysis:**

From the figures above, I find that the value of training metric is roughly the same as the value of testing metric, which means that training data and testing data have similar distributions.

In linear regression, the loss-curve figure shows that the training loss is higher than testing loss, due to a larger training data. (I use 70 percent of the original data as training data while the rest are left as testing data. I didn’t take average when calculating the loss.)

During the 50~1000 epochs of linear classification, accuracies of prediction remain the same even though the loss value keeps decreasing, because accuracy and hinge loss are different metrics. The decrease in hinge loss suggests a classifier with larger margin, which doesn’t necessarily mean that there will be an improvement in the accuracy of predicting the labels.

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.