

**The Experiment Report of**

***Deep Learning***

**College Software College**

**Subject Software Engineering**

**Members Yingting Huang**

**Student ID 201720145136**

**E-mail 757004248@qq.com**

**Tutor Prof. MingKui Tan**

**Date submitted** **2017.12.06**

1. **Topic:**

Linear Regression, Linear Classification and Gradient Descent

**2. Time:**

12/06/2017

**3. Reporter:**

Yingting Huang

**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.
2. Linear classification uses Australian in LIBSVM Data, including 690 samples and each sample has 14 features.
3. Both od the data I download has been scaled to [-1,1]. After that, I divide the data into training set, validation set. The proportion of training set is 60% while validation set accounts for 40%.

**6. Experimental steps:**

*6.1 Linear Regression and Gradient Descent*

1. Load the experiment data.
2. Divide dataset into training set and validation set.
3. Initialize linear model parameters.
4. Choose loss function and derivation.
5. Calculate gradient toward loss function from all samples.
6. Denote the opposite direction of gradient G as D.
7. Update model: .
8. Get the loss under the training set and by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

*6.2 Linear Classification and Gradient Descent*

1. Load the experiment data.
2. Divide dataset into training set and validation set.
3. Initialize SVM model parameters.
4. Choose loss function and derivation.
5. Calculate gradient toward loss function from all samples.
6. Denote the opposite direction of gradient G as D.
7. Update model: .
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 training set and by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of as well as with the number of iterations

**7. Code:**

*7.1 Linear Regression and Gradient Descent*

# coding: utf-8

# In[197]:

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

# In[198]:

# Load the experiment data

mem = Memory("./mycache")

@mem.cache

def get\_data(f):

data = load\_svmlight\_file(f)

return data

# Loss Function:

# $$ L= ||W^TX-y||^2$$

# Gradient Matrix:

# $$ \frac{\partial L}{\partial W}=2(W^TX-y)X$$

# In[199]:

def linearRegression(W, xtrain, ytrain, xtest, ytest):

"""

Inputs:

- W: A numpy array of shape (D\_in, D\_out) containing weights.

- xtrain: A numpy array of shape (N, D\_in) containing a minibatch of data.

- ytrain: A numpy array of shape (N, D\_out) containing training target value;

- xtest: A numpy array of shape (N2, D\_in) containing test data.

- ytest: A numpy array of shape (N2, D\_out) containing test target value;

Returns a tuple of:

- loss as single float

- gradient with respect to weights W; an array of same shape as W

"""

num\_train = X.shape[0]

train\_loss = 0

test\_loss =0

dW = np.zeros(W.shape) # initialize the gradient as zero

ytrain\_pred = np.dot(xtrain,W) # N by 1

ytest\_pred = np.dot(xtest,W) # N by 1

train\_loss = np.power((ytrain\_pred-ytrain),2).sum()

test\_loss = np.power((ytest\_pred-ytest),2).sum()

grad\_y= 2.0 \* (ytrain\_pred-ytrain)

dW = np.dot(xtrain.T,grad\_y)/ num\_train

return train\_loss, test\_loss, dW

# In[200]:

def predict(W , X, y ):

"""

Inputs:

- W: A numpy array of shape (D\_in, D\_out) containing weights.

- x: A numpy array of shape (N, D\_in) containing a minibatch of data.

- y: A numpy array of shape (N, D\_out) containing target value

Returns a tuple of:

- prediction of y

- root mean squared error

- coefficient of determination

- degree-of-freedom adjusted coefficient of determination

"""

N,D\_in =X.shape # N is total sample size ; D\_in is the number of predictors

y\_pred= np.dot(X ,W)

#he sum of squares due to error

SSE =np.power((y\_pred-y),2).sum()

#Mean squared error

MSE =SSE/N

#Root mean squared error

RMSE = np.sqrt(MSE)

#Total sum of squares

SST =np.power((y-np.mean(y\_train)),2).sum()

#Coefficient of determination

R2=1-SSE/SST

#Degree-of-freedom adjusted coefficient of determination

adj\_R2=1-(N-1) \*(1-R2)/(N-D\_in-1)

return y\_pred,RMSE,R2,adj\_R2

# In[201]:

# Data preparation

data = get\_data(f="D:\研一\课件\深度学习\libsvm-3.22\housing\_scale")

# Data preprocessing

X=data[0].toarray()

X=np.column\_stack((X,np.ones([X.shape[0],1]))) #add bias

Y=data[1]

Y=Y.reshape((len(Y),1))

# In[202]:

# Devide dataset

x\_train, x\_validation, y\_train, y\_validation = train\_test\_split(X, Y, test\_size=0.4, random\_state=42)

N,D\_in=x\_train.shape # D\_in is input dimension; N is batch size;

D\_out =y\_train.shape[1] # D\_out is output dimension

# In[203]:

# Initialize parameter

W = random.random(size=(D\_in, D\_out)) # weights

eta = 0.01 # learning rate

maxIterations=1000

L\_train=[]; # train loss

L\_validation=[]; #validation loss

# In[204]:

# Iterations

for t in range(maxIterations):

# Compute loss

train\_loss, validation\_loss, grad\_W= linearRegression(W, x\_train, y\_train, x\_validation, y\_validation)

# Save the values

L\_train.append (train\_loss)

L\_validation.append (validation\_loss)

# Update weights using gradient descent

W -= eta \* grad\_W

# In[207]:

# Plot

plt.plot(L\_train,'r',label='train loss')

plt.plot(L\_validation,'b',label='validation loss')

plt.title('Loss Curve') # give plot a title

plt.xlabel('Iterations')# make axis labels

plt.ylabel('Loss')

plt.legend()

plt.show()

# In[206]:

#Assessment Result

y\_train\_pred,RMSE\_train,R2\_train,adj\_R2\_train=predict(W , x\_train, y\_train )

print('------------Assessment Results-----------\nRMSE\_train=',RMSE\_train,'\nR-squared\_train=',R2\_train,'\nadjusted R-squared\_train =',adj\_R2\_train)

#Predicted Result

y\_validation\_pred,RMSE\_validation,R2\_validation,adj\_R2\_validation=predict(W , x\_validation, y\_validation )

print('\n------------Predicted Results------------\nRMSE\_validation=',RMSE\_validation,'\nR-squared\_validation=',R2\_validation,'\nadjusted R-squared\_validation =',adj\_R2\_validation)

*7.2 Linear Classification and Gradient Descent*

# coding: utf-8

# In[1]:

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

# In[2]:

# Load the experiment data

mem = Memory("./mycache")

@mem.cache

def get\_data(f):

data = load\_svmlight\_file(f)

return data

# Loss Function:

# $$ L= \frac{1}{N}\sum\_{i}\sum\_{j\ne y\_i}[max(0,f(x\_i;W)\_j-f(x\_i;W)\_{y\_i}+\Delta]+\lambda \sum\_k\sum\_lW^2\_{k,l}$$

# Gradient Matrix:

# $$ \frac{\partial L}{\partial w\_{y\_i}}=-(\sum\_{j \ne y\_i}1(w^T\_jx\_i-w^T\_{y\_i}x\_i+\Delta>0))x\_i \quad \quad (j=y\_i)$$

# $$ \frac{\partial L}{\partial w\_j}=1(w^T\_jx\_i-w^T\_{y\_i}x\_i+\Delta>0))x\_i \quad \quad \quad \quad (j\ne y\_i)$$

# where l(x) is an indicator function：

# $$l(x==T)=1$$

# $$l(x==F)=0$$

# In[3]:

def predict(W , X, y ):

"""

Inputs:

- W: A numpy array of shape (D, C) containing weights.

- x: A numpy array of shape (N, D) containing a minibatch of training data.

- y: A numpy array of shape (N,) containing training labels; ytrain[i] = c means that xtrain[i] has label c, where 0 <= c <C.

Returns a tuple of:

- prediction of y

- accuracy of the model

"""

# the class scores for all examples

score = np.dot(X,W)

# find the index with max score in each column (the predicted class)

y\_pred= np.argmax(score, axis = 1)

# and calculate accuracy (fraction of predictions that are correct)

cmp=(y\_pred == y)

accuracy=len(cmp[cmp==True])/len(cmp)

return y\_pred,accuracy

# In[4]:

def svm(W, xtrain, ytrain, xtest, ytest, threshold, reg):

"""

Inputs:

- W: A numpy array of shape (D, C) containing weights.

- xtrain: A numpy array of shape (num\_train, D) containing a minibatch of training data.

- ytrain: A numpy array of shape (num\_train,) containing training labels; ytrain[i] = c means that xtrain[i] has label c, where 0 <= c <C.

- xtest: A numpy array of shape (num\_test, D) containing a minibatch of test data.

- ytest: A numpy array of shape (num\_test,) containing test labels

- threshold: mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative.

- reg: (float) regularization strength

Returns a tuple of:

- train\_loss as single float

- test\_loss as single float

- gradient with respect to weights W; an array of same shape as W

"""

dW = np.zeros(W.shape) # initialize the gradient as zero

num\_classes = W.shape[1]

#train

train\_loss = 0

scores\_train = xtrain.dot(W) # num\_train by C

num\_train = xtrain.shape[0]

scores\_train\_correct = scores\_train[np.arange(num\_train), ytrain] # 1 by num\_train

scores\_train\_correct = np.reshape(scores\_train\_correct, (num\_train, 1)) # num\_train by 1

margins\_train = scores\_train - scores\_train\_correct + 1.0 # num\_train by C

margins\_train[np.arange(num\_train), ytrain] = 0.0

margins\_train[margins\_train <= threshold] = 0.0

train\_loss += np.sum(margins\_train) / num\_train

train\_loss += reg \* np.sum(W \* W)

margins\_train[margins\_train > threshold] = 1.0

row\_sum = np.sum(margins\_train, axis=1) # 1 by num\_train

margins\_train[np.arange(num\_train), ytrain] = -row\_sum

dW += np.dot(xtrain.T, margins\_train)/num\_train + reg \* W # D by C

#test

test\_loss = 0

scores\_test = xtest.dot(W) # num\_test by C

num\_test = xtest.shape[0]

scores\_test\_correct = scores\_test[np.arange(num\_test), ytest] # 1 by N

scores\_test\_correct = np.reshape(scores\_test\_correct, (num\_test, 1)) # N by 1

margins\_test = scores\_test - scores\_test\_correct + 1.0 # N by C

margins\_test[np.arange(num\_test), ytest] = 0.0

margins\_test[margins\_test <= threshold] = 0.0

test\_loss += np.sum(margins\_test) / num\_test

test\_loss += reg \* np.sum(W \* W)

return train\_loss, test\_loss, dW

# In[5]:

# Data preparation

data = get\_data(f="D:\研一\课件\深度学习\libsvm-3.22\Australian\_scale")

# Data preprocessing

X=data[0].toarray()

X= np.column\_stack((X,np.ones([X.shape[0],1]))) #add bias

Y=data[1]

Y=Y.reshape(len(Y),order='C')

Y=Y.astype(np.int) # float->int

Y[Y== -1]=0

# In[6]:

# Devide dataset

x\_train, x\_validation, y\_train, y\_validation = train\_test\_split(X, Y, test\_size=0.4, random\_state=42)

N,D=x\_train.shape # N is batch size; D is input dimension

C=len(list(set(y\_train))) # total number of labels

# In[7]:

# Initialize parameter

W = np.zeros((D, C)) # weights

maxIterations=1000

threshold=0 #mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative.

reg= 0.1 #regularization strength

eta = 0.01 # learning rate

L\_train=[]; # train loss

L\_validation=[]; #validation loss

# In[8]:

# Iterations

for t in range(maxIterations):

# Compute loss

train\_loss, validation\_loss, grad\_W= svm(W, x\_train, y\_train, x\_validation, y\_validation, threshold, reg)

# Save the values

L\_train.append (train\_loss)

L\_validation.append (validation\_loss)

# Update weights using gradient descent

W -= eta \* grad\_W

# In[13]:

# Plot

plt.plot(L\_train,'r',label='train loss')

plt.plot(L\_validation,'b',label='validation loss')

plt.title('Loss Curve') # give plot a title

plt.xlabel('Iterations')# make axis labels

plt.ylabel('Loss')

plt.legend()

plt.show()

# In[12]:

# Assessment Result

y\_train\_pred,training\_accuracy = predict(W , x\_train, y\_train )

print('------------Assessment Results-----------\ntraining accuracy=',training\_accuracy)

#Predicted Result

y\_validation\_pred,validation\_accuracy = predict(W , x\_validation, y\_validation )

print('\n------------Predicted Results------------\nvalidation accuracy=',validation\_accuracy)

**8. Selection of validation:**

*8.1 Linear Regression and Gradient Descent*

The evaluating method of linear regression and gradient descent I select is *simple cross validation*. I divide the data into training set, validation set randomly. And the proportion of training set is 60% while validation set accounts for 40%. At the end of the experiment, I choose the RMSE, R-squared, adjusted R-squared to assess the result of the linear model.

*8.2 Linear Classification and Gradient Descent*

The evaluating method of linear classification and gradient descent I select is *simple cross validation*. I divide the data into training set, validation set randomly. And the proportion of training set is 60% while validation set accounts for 40%. At the end of the experiment, I choose the accuracy to assess the result of the SVM model.

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

*9.1 Linear Regression and Gradient Descent*

All parameters are set into zero in the linear model.

*9.2 Linear Classification and Gradient Descent*

All parameters are set into zero in the SVM model.

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

*10.1 Linear Regression and Gradient Descent*

1. Loss Function：

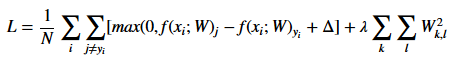


1. Gradient Matrix：

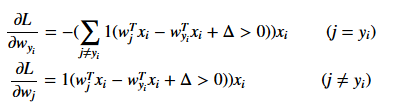
C:\Users\yingting.huang\AppData\Roaming\Tencent\Users\757004248\QQ\WinTemp\RichOle\S78~GB(9T$V$77)JE3Q39S5.png

*10.2 Linear Classification and Gradient Descent*

1. Loss Function：



1. Gradient Matrix：



Where l(x) is an indicator function：

C:\Users\yingting.huang\AppData\Roaming\Tencent\Users\757004248\QQ\WinTemp\RichOle\SB(R[H@@63ZN02N03JL@FFE.png

**11. Experimental results and curve:**

*11.1 Linear Regression and Gradient Descent*

## Hyper-parameter selection:

* *eta*, the learning rate, is set to 0.01
* *maxIterations*, the maximum number of iterations, is set to 1000

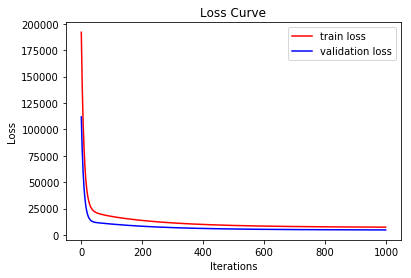
## Assessment Results (based on selected validation):

* *RMSE\_train*, the root mean squared error of training data, is equal to 4.96136623475
* *R-squared\_train*, the coefficient of determination of the training data, is equal to 0.724127965617
* *Adjusted R-squared\_train*, the degree-of-freedom adjusted coefficient of determination of the training data, is equal to 0.710717519501

## Predicted Results (Best Results):

* *RMSE\_validation*, the root mean squared error of validationing data, is equal to 4.89733305645
* *R-squared\_validation*, the coefficient of determination of the validationing data, is equal to 0.692970318205
* *Adjusted R-squared\_validation*, the degree-of-freedom adjusted coefficient of determination of the validationing data, is equal to 0.67010640573

## Loss curve:



*11.2 Linear Classification and Gradient Descent*

## Hyper-parameter selection:

* *eta*, the learning rate, is set to 0.01
* *maxIterations*, the maximum number of iterations, is set to 1000
* *threshold*, marks the sample whose predict scores greater than the threshold as positive, on the contrary as negative., is set to 0.
* *reg*, the regularization strength, is set to 0.1

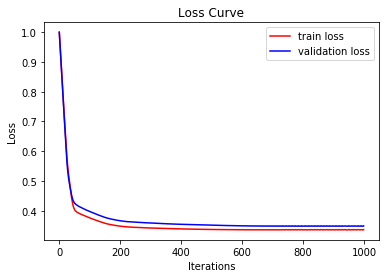
## Assessment Results (based on selected validation):

* *training\_accuracy,* the accuracy of training data, is equal to 0.857487922705314

## Predicted Results (Best Results):

* *validation\_accuracy,* the accuracy of validation data, is equal to 0.8514492753623188

## Loss curve:



**12. Results analysis:**

*12.1 Linear Regression and Gradient Descent*

From the Assessment Results in 12.1(1) and the predicted results in 12.1(2), we can see that the accuracy is high, which mean the predicting effect of the model is good.

From the loss curve, the train loss and the validation loss converge to a small number nears zero with the number of iterations. That is to say, the model we have trained is robust.

*12.2 Linear Classification and Gradient Descent*

From the Assessment Results in 11.1(1) and the predicted results in 11.1(2), we can see that the root mean squared error is little and the degree-of-freedom adjusted coefficient of determination is close to 1, which mean the predicting effect of the Model is good.

From the loss curve, the train loss and the validation loss converge to a small number with the number of iterations. That is to say, the model we have trained is robust.

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

In my opinion, linear regression and linear classification are essentially the same. Both of them are trained to fit the model and related to prediction, where regression predicts a value from a continuous set, whereas classification predicts the 'belonging' to the class. Given the following：

F: x-->y

If y is discrete/categorical variable, then this is classification problem.

If y is real number/continuous, then this is a regression problem.

**14. Summary:**

It is important to be clear when using terms like regression, classification and prediction to discriminate between the task we are performing and the method used to perform it. A classification task involves taking an input and labelling it as belonging to a given class, so the output is categorical. On the other hand, a prediction task involves predicting a continuous valued output.

Methods for achieving these tasks include regression, in which a continuous valued output is estimated (or, rather, the expected value of a distribution on a continuous variable is estimated, conditional on a given set of input values). This can be used to carry out a prediction task, as you would expect. It can also be used to carry out a classification task, for example using logistic regression to estimate the log odds of the input pattern belonging to a given class. In this case, the task is classification, the method is regression.

Classification methods simply generate a class label rather than estimating a distribution parameter. K nearest neighbor (KNN) is a good example where the task and the method are both called classification.