o



**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **Yu Guo**

**Student ID 201530611500**

**E-mail G624786883@gmail.com**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017. 12 . 2**

**1. Topic:**

Linear regression, linear classification and gradient descent in python.

**2. Time:**

2017.12.2

**3. Reporter:**

Yu Guo

**4. Purposes:**

Achieve linear regression and linear classification with gradient descent in python.

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

Linear Regression uses Housing in LIBSVM, including 506 samples and each sample has 13 features. Linear classification uses australianin LIBSVM Data, including 690 samples and each sample has 14 features.

**6. Experimental steps:**

*Linear Regression and Gradient Descent*

1. Load the experiment data.
2. Divide dataset.
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 training loss under the training set and test loss by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of training loss as well as test loss with the number of iterations.

*Linear Classification and Gradient Descent*

1. Load the experiment data.
2. Divide dataset.
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 training loss under the training set and test loss by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of training loss as well as test loss with the number of iterations.

**7. Code:**

ClassificationExperiment.ipynb:

import numpy as np

import matplotlib.pyplot as plt

import sklearn.datasets as sd

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

def getLoss(w,x,y):

loss = 0

m = x.shape[0]

for i in range(m):

xi = np.column\_stack(([1],x.getrow(i).toarray()))

hi = xi.dot(w)

loss += (hi - y[i])\*(hi - y[i])/2

return loss[0][0]

def updateWeight(w,x,y):

m = x.shape[0]

n = x.shape[1] + 1

tempw = np.zeros((n,1))

for j in range(n):

gradient = 0

for i in range(m):

xi = np.column\_stack(([1],x.getrow(i).toarray()))

hi = xi.dot(w)

gradient += (hi - y[i])\*xi[0][j]

gradient /= m

tempw[j][0] = w[j][0] - alpha\*gradient

return tempw

data = sd.load\_svmlight\_file('housing\_scale')

X,y = data[0],data[1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=30)

alpha = 0.2

w = np.zeros((X.shape[1] + 1,1))

times = 30

trainLossList = []

testLossList = []

for t in range(times):

w = updateWeight(w, X\_train, y\_train)

trainLossList.append(getLoss(w, X\_train, y\_train))

testLossList.append(getLoss(w, X\_test, y\_test))

time = np.arange(times)

plt.plot(time, trainLossList, label='train')

plt.plot(time, testLossList, label='test')

plt.show()

RegresstionExperiment.ipynb:

import numpy as np

import matplotlib.pyplot as plt

import sklearn.datasets as sd

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

def getGradient(w,x,y):

m = x.shape[0]

n = x.shape[1]

gradient = np.zeros((n,1))

for i in range(m):

gradient += w

if (1 - (y[i]\*np.dot(w.T,x[i].T))[0])>0:

gradient += 0-C\*y[i]\*(x[i].reshape((n,1)))

gradient = gradient/m

return gradient

def getLoss(w,x,y):

loss = 0

for i in range(x.shape[0]):

loss += max(0,1 - (y[i]\*np.dot(w.T,x[i].T)[0]))

return 0.5\*(np.dot(w.T,w)[0][0])+C\*loss

def getError(w,x,y):

error = 0

m = x.shape[0]

for i in range(m):

if x[i].dot(w)[0]>-0.2:

predict = 1

else: predict = -1

if predict != y[i]: error += 1

return error

alpha = 0.05

C = 1

times = 100

data = sd.load\_svmlight\_file('australian\_scale')

X,y = data[0],data[1]

w = np.zeros((X.shape[1] + 1,1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=90)

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

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

X\_train = np.column\_stack((appendTrain,X\_train.toarray()))

X\_test = np.column\_stack((appendTest,X\_test.toarray()))

trainLossList = []

testLossList = []

for i in range(times):

gradient = getGradient(w,X\_train,y\_train)

w = w - alpha\*gradient

trainLossList.append(getLoss(w,X\_train,y\_train))

testLossList.append(getLoss(w,X\_test,y\_test))

time = np.arange(times)

plt.plot(time, trainLossList, label='train')

plt.plot(time, testLossList, label='test')

plt.show()

print(getError(w,X\_train,y\_train))

print(getError(w,X\_test,y\_test))

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

cross-validation

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

all zero initialization

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

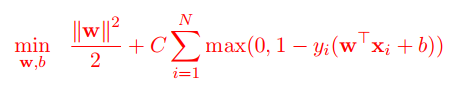
Linear regression:

loss function:

 for every parameter ,gradient\_theta(j) =

Linear classification:

loss function:

 Gradient = ||w|| + 0 yi(Wtxi+b)>=1

||w|| - Cxiyi else

**11. Experimental results and curve:**

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

## Linear regression:

## . η = 0.2

## Epoch = 50

Linear classification:

## . η = 0.05

## Epoch = 100

## C = 1

## Assessment Results (based on selected validation):

## Regression: Average deviation is 3.3

## Classification: Error rate = 12%

## Predicted Results (Best Results):

## Regression: Average deviation is 3.6

## Classification: Error rate = 15%

## Loss curve:

## Linear regression:

## Linear classification:

**12. Results analysis:**

It’s clear in loss curve that the value of loss function decrease in the process of gradient descent.

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

Similarities: Both of them use linear model. In this experiment, use gradient descent to try to find a ideal minimum.

Difference: The result of regression is continuous value, while the result of classification is discrete value.

**14. Summary:**

This experiment shows that gradient descent do work in linear regression and SVM. The loss function is going down with the number of iterations. It’s intutive to code with python