

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**   **赵云潇**

**学 号 201530613757**

**邮 箱**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12 月 15 日**

## 1. 实验题目: 逻辑回归、线性分类与随机梯度下降

## 2. 实验时间：2017年 12 月 2日

## 3. 报告人:赵云潇

## 4. 实验目的:

对比理解梯度下降和随机梯度下降的区别与联系。

对比理解逻辑回归和线性分类的区别与联系。

进一步理解SVM的原理并在较大数据上实践。

## 数据集以及数据分析：

实验使用的是LIBSVM Data的中的a9a数据，包含32561 / 16281(testing)个样本，每个样本有123/123 (testing)个属性。

## 6. 实验步骤:

## 7. 代码内容:

逻辑回归

from sklearn.datasets import load\_svmlight\_file

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

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

def get\_data():

data = load\_svmlight\_file("a9a")

x\_train = data[0].toarray()

y\_train = data[1]

for i in range(0, len(y\_train)):

if y\_train[i] == -1:

y\_train[i] = 0

temp = np.ones(shape=[32561, 1], dtype=np.float32)

x\_train = np.concatenate([x\_train, temp], axis=1)

return x\_train, y\_train

def get\_test():

data = load\_svmlight\_file("a9a.t")

x\_test = data[0].toarray()

y\_test = data[1]

for i in range(0, len(y\_test)):

if y\_test[i] == -1:

y\_test[i] = 0

temp = np.zeros(shape=[16281, 1], dtype=np.float32)

temp1 = np.ones(shape=[16281, 1], dtype=np.float32)

x\_test = np.concatenate([x\_test, temp, temp1], axis=1)

return x\_test, y\_test

def sigmoid(input\_):

return 1 / (1 + np.exp(-input\_))

def train(x\_train, y\_train, x\_test, y\_test, method, iters, test\_errors):

iterations = 100

theta = random.rand(num\_features + 1)

num\_test\_samples, num\_test\_features = x\_test.shape

if method == 'sgd':

rate = 0.01

for i in range(iterations):

output = sigmoid(np.dot(x\_train[i], theta))

error = output - y\_train[i]

theta = theta - rate \* np.dot(x\_train[i], error)

predict\_error = 0

for j in range(num\_test\_samples):

predict\_output = sigmoid(np.dot(x\_test[j], theta))

predict\_error -= y\_test[j] \* np.log(predict\_output) + (1 - y\_test[j]) \* np.log(1 - predict\_output)

iters.append(i)

test\_errors.append(predict\_error / num\_test\_samples)

if method == 'nag':

rate = 0.01

miu = 0.9

momentum = np.zeros(num\_features + 1)

for i in range(iterations):

output = sigmoid(np.dot(x\_train[i], theta - rate \* miu \* momentum))

error = output - y\_train[i]

grad = np.dot(x\_train[i], error)

momentum = momentum \* rate + grad

theta = theta - rate \* momentum

predict\_error = 0

for j in range(num\_test\_samples):

predict\_output = sigmoid(np.dot(x\_test[j], theta))

predict\_error -= y\_test[j] \* np.log(predict\_output) + (1 - y\_test[j]) \* np.log(1 - predict\_output)

iters.append(i)

test\_errors.append(predict\_error / num\_test\_samples)

if method == 'rmsprop':

rate = 0.1

expectation = 1

rho = 0.95

delta = 10e-7

for i in range(iterations):

output = sigmoid(np.dot(x\_train[i], theta))

error = output - y\_train[i]

grad = np.dot(x\_train[i], error)

norm = grad \* grad

expectation = rho \* expectation + (1 - rho) \* norm

theta = theta - rate \* grad / (np.sqrt(expectation) + delta)

predict\_error = 0

for j in range(num\_test\_samples):

predict\_output = sigmoid(np.dot(x\_test[j], theta))

predict\_error -= y\_test[j] \* np.log(predict\_output) + (1 - y\_test[j]) \* np.log(1 - predict\_output)

iters.append(i)

test\_errors.append(predict\_error / num\_test\_samples)

if method == 'adam':

delta = 10e-8

rho1 = 0.9

rho2 = 0.999

rate = 0.1

s = 0

r = 0

for i in range(iterations):

output = sigmoid(np.dot(x\_train[i], theta))

error = output - y\_train[i]

grad = np.dot(x\_train[i], error)

s = rho1 \* s + (1 - rho1) \* grad

r = rho2 \* r + (1 - rho2) \* grad \* grad

s\_hat = s / (1 - rho1)

r\_hat = r / (1 - rho2)

delta\_theta = (-rate \* s\_hat) / (np.sqrt(r\_hat) + delta)

theta = theta + delta\_theta

predict\_error = 0

for j in range(num\_test\_samples):

predict\_output = sigmoid(np.dot(x\_test[j], theta))

predict\_error -= y\_test[j] \* np.log(predict\_output) + (1 - y\_test[j]) \* np.log(1 - predict\_output)

iters.append(i)

test\_errors.append(predict\_error / num\_test\_samples)

if method == 'adadelta':

r = 0

e = 0

miu = 0.9

delta = 10e-7

rate = 10

for i in range(iterations):

output = sigmoid(np.dot(x\_train[i], theta))

error = output - y\_train[i]

grad = np.dot(x\_train[i], error)

r = miu \* r + (1 - miu) \* grad \* grad

delta\_theta = (-rate \* grad \* np.sqrt(e + delta)) / (np.sqrt(r + delta))

theta = theta + delta\_theta

e = miu \* e + (1 - miu) \* e \* e

predict\_error = 0

for j in range(num\_test\_samples):

predict\_output = sigmoid(np.dot(x\_test[j], theta))

predict\_error -= y\_test[j] \* np.log(predict\_output) + (1 - y\_test[j]) \* np.log(1 - predict\_output)

iters.append(i)

test\_errors.append(predict\_error / num\_test\_samples)

def main():

x\_train, y\_train = get\_data()

x\_test, y\_test = get\_test()

methods = ['sgd', 'nag', 'rmsprop', 'adadelta', 'adam']

for method in methods:

iters = []

test\_errors = []

train(x\_train, y\_train, x\_test, y\_test, method, iters, test\_errors)

plt.plot(iters, test\_errors, label=method)

main()

plt.xlabel('Iteration')

plt.ylabel('Loss')

plt.legend()

plt.show()

线性分类

from sklearn.datasets import load\_svmlight\_file

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

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

import os

feature\_size = 123

bias = np.zeros(shape=[feature\_size + 1, 1])

bias[len(bias)-1][0] = 1.

def get\_data():

data = load\_svmlight\_file("a9a")

x\_train = data[0].toarray()

y\_train = data[1]

temp = np.ones(shape=[32561, 1], dtype=np.float32)

x\_train = np.concatenate([x\_train, temp], axis=1)

return x\_train, y\_train

def get\_test():

data = load\_svmlight\_file("a9a.t")

x\_test = data[0].toarray()

y\_test = data[1]

temp = np.zeros(shape=[16281, 1], dtype=np.float32)

temp1 = np.ones(shape=[16281, 1], dtype=np.float32)

x\_test = np.concatenate([x\_test, temp, temp1], axis=1)

return x\_test, y\_test

def loss(x, y, w):

pred = np.matmul(x, w)

hinge\_loss = np.maximum(1 - y \* pred, 0)

loss = np.mean(hinge\_loss \*\* 2) + np.sum((w - bias) \*\* 2)

return loss

def gradient(x, y, w):

pred = np.matmul(x, w)

hinge\_loss = np.maximum(1 - y \* pred, 0)

hinge\_loss\_gradient = -np.matmul(x.transpose(), hinge\_loss \* y) / len(y)

norm\_gradient = 2 \* (w - bias)

gradient = hinge\_loss\_gradient + norm\_gradient

return gradient

def optimizer(method, x, y, w):

global global\_list

if method == 'sgd':

rate = 0.01

print(w)

w -= rate \* gradient(x, y, w)

if method == 'nag':

rate = 0.01

miu = 0.9

momentum = 0

grad = gradient(x, y, w - momentum \* rate \* miu)

momentum = momentum \* rate + grad

w -= rate \* momentum

if method == 'rmsprop':

rate = 0.1

expectation = 1

rho = 0.95

delta = 10e-7

grad = gradient(x, y, w)

expectation = rho \* expectation + (1 - rho) \* (grad \*\* 2)

w -= rate \* grad / (np.sqrt(expectation) + delta)

if method == 'adam':

delta = 10e-7

rho1 = 0.9

rho2 = 0.999

rate = 0.1

s = 0

r = 0

grad = gradient(x, y, w)

s = rho1 \* s + (1 - rho1) \* grad

r = rho2 \* r + (1 - rho2) \* (grad \*\* 2)

s\_hat = s / (1 - rho1)

r\_hat = r / (1 - rho2)

w -= (rate \* s\_hat) / (np.sqrt(r\_hat) + delta)

if method == 'adadelta':

r = 0

e = 0

miu = 0.9

delta = 10e-7

rate = 10

grad = gradient(x, y, w)

r = miu \* r + (1 - miu) \* (grad \*\* 2)

w -= (rate \* grad \* np.sqrt(e + delta)) / (np.sqrt(r + delta))

e = miu \* e + (1 - miu) \* (w \*\* 2)

def main():

x\_train, y\_train = get\_data()

x\_test, y\_test = get\_test()

num\_samples, num\_features = x\_train.shape

num\_test\_samples, num\_test\_features = x\_test.shape

y\_train = y\_train.reshape([len(y\_train), 1])

y\_test = y\_test.reshape([len(y\_test), 1])

def shuffle\_train():

global x\_train, y\_train

rng\_state = np.random.get\_state()

np.random.shuffle(x\_train)

np.random.set\_state(rng\_state)

np.random.shuffle(y\_train)

batch\_size = 1024

data\_size = num\_samples

def feed\_data(batch\_count):

if (1 + batch\_count) \* batch\_size <= data\_size:

feed\_dict = {'x': x\_train[batch\_count \* batch\_size:(batch\_count + 1) \* batch\_size],

'y': y\_train[batch\_count \* batch\_size:(batch\_count + 1) \* batch\_size]}

else:

feed\_dict = {'x': x\_train[batch\_count \* batch\_size:data\_size],

'y': y\_train[batch\_count \* batch\_size:data\_size]}

return feed\_dict

for method in methods:

iters = []

test\_errors = []

w = np.random.rand(feature\_size+1, 1)

print(w)

count = 0

for i in range(0, 3):

shuffle\_train()

for batch\_count in range(0, int(data\_size / batch\_size) + 1):

feed\_dict = feed\_data(batch\_count)

iters.append(count)

count += 1

optimizer(method=method, x=feed\_dict['x'], y=feed\_dict['y'], w=w)

test\_errors.append(loss(x\_test, y\_test, w))

plt.plot(iters, test\_errors, label=method)

main()

plt.xlabel('Iteration')

plt.ylabel('Loss')

plt.legend()

plt.show()

## 10.实验结果和曲线图:

