

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**

**Student ID** 201530613207

**E-mail 978163495@qq.com**

**Tutor**   **mingkui Tan**

**Date submitted** **2017.12 .3**

**1. Topic:** LogisticRegression& Linear Classification & Stochastic Gradient Descent

**2. Time:** 2017.12.2 PM 14:00-17:00

**3. Reporter:** peizhuoXie

**4. Purposes:**

* Further understand the principle of Logistic Regression& Linear Classification & Stochastic Gradient Descent method.
* Practicing on a small data set
* Understand the process of optimizing the super parameters

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

In both Logistic Regression& Linear Classification, we use a1a and a1a.t dataset, including 32561 / 16281 samples and 123 / 123 features respectively.

**6. Experimental steps:**

* Using the function load\_svmlignt\_file from sklearn read the data set.
* Initializing the parameters randomly.
* Using Cross-entropy loss / Hinge loss and compute all the samples’ loss.
* Update the loss function by methods = ['SGD', 'NAG', 'RMSProp', 'Adadelta', 'ADAM']  
  Repeat the above steps for some times and draw the figure of loss-iteration.

**7. Code:**

from sklearn import datasets as ds  
import numpy as np  
from numpy import random  
import matplotlib.pyplot as plt  
import os  
  
# C = 1  
feature\_size = 123  
  
  
def sigmoid(input\_):  
 return 1 / (1 + np.exp(-1.0 \* input\_))  
  
  
def compute\_loss(x, y, w):  
 # x.shape = [batch\_size, feature\_size + 1], y.shape = [batch\_size, 1], w.shape = [feature\_size + 1, 1]  
 logits = np.matmul(x, w)  
 loss = -np.mean(y \* np.log(sigmoid(logits)) + (1 - y) \* np.log(1 - sigmoid(logits)))  
 return loss  
  
  
def compute\_gradient(x, y, w):  
 # x.shape = [batch\_size, feature\_size + 1], y.shape = [batch\_size, 1], w.shape = [feature\_size + 1, 1]  
 logits = np.matmul(x, w)  
 output = sigmoid(logits)  
 error = output - y  
 gradient = np.matmul(x.transpose(), error) / len(y)  
 return gradient # gradient.shape = [feature\_size + 1, 1], the same as w  
  
  
global\_list = {'NAG\_momentum': np.zeros([feature\_size + 1, 1]), 'RMS\_expectation': 1, 'ADAM\_s': 0, 'ADAM\_r': 0,  
 'ADADELDA\_r': 0, 'ADADELDA\_e': 0}  
  
  
def optimizer(method, parameter\_list, x, y, w):  
 global global\_list  
 if method == 'SGD':  
 # sgd\_para\_list = {'learning\_rate':0.01}  
 lr = parameter\_list['learning\_rate']  
 w -= lr \* compute\_gradient(x, y, w)  
 if method == 'NAG':  
 # nag\_para\_list = {'miu':0.9, 'learning\_rate':0.01}  
 lr = parameter\_list['learning\_rate']  
 miu = parameter\_list['miu']  
 momentum = global\_list['NAG\_momentum']  
 gradient = compute\_gradient(x, y, w - momentum \* lr \* miu)  
 momentum = momentum \* lr + gradient  
 w -= lr \* momentum  
 if method == 'RMSProp':  
 # rmsprop\_para\_list = {'delta':10e-7, 'rho':0.95, 'learning\_rate':0.1}  
 lr = parameter\_list['learning\_rate']  
 expectation = global\_list['RMS\_expectation']  
 rho = parameter\_list['rho']  
 delta = parameter\_list['delta']  
 gradient = compute\_gradient(x, y, w)  
 norm = np.sum(gradient\*\*2)  
 expectation = rho \* expectation + (1 - rho) \* norm  
 global\_list['RMS\_expectation'] = expectation  
 w -= lr \* gradient / (np.sqrt(expectation) + delta)  
 if method == 'ADAM':  
 # adam\_para\_list = {'delta':10e-8, 'rho1':0.9, 'rho2':0.999, 'learning\_rate':0.1}  
 delta = parameter\_list['delta']  
 rho1 = parameter\_list['rho1']  
 rho2 = parameter\_list['rho2']  
 lr = parameter\_list['learning\_rate']  
 s = global\_list['ADAM\_s']  
 r = global\_list['ADAM\_r']  
 gradient = compute\_gradient(x, y, w)  
 s = rho1 \* s + (1 - rho1) \* gradient  
 r = rho2 \* r + (1 - rho2) \* gradient \* gradient  
 s\_hat = s / (1 - rho1)  
 r\_hat = r / (1 - rho2)  
 w -= (lr \* s\_hat) / (np.sqrt(r\_hat) + delta)  
 global\_list['ADAM\_s'] = s  
 global\_list['ADAM\_r'] = r  
 if method == 'Adadelta':  
 # adadelta\_para\_list = {'delta':10e-7, 'miu':0.9, 'learning\_rate':10}  
 r = global\_list['ADADELDA\_r']  
 e = global\_list['ADADELDA\_e']  
 miu = parameter\_list['miu']  
 delta = parameter\_list['delta']  
 lr = parameter\_list['learning\_rate']  
 grad = compute\_gradient(x, y, w)  
 r = miu \* r + (1 - miu) \* grad \* grad  
 w -= (lr \* grad \* np.sqrt(e + delta)) / (np.sqrt(r + delta))  
 e = miu \* e + (1 - miu) \* e \* e  
 global\_list['ADADELDA\_r'] = r  
 global\_list['ADADELDA\_e'] = e  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x\_train, y\_train = ds.load\_svmlight\_file('./a9a')  
 x\_test, y\_test = ds.load\_svmlight\_file('./a9a.t')  
  
 num\_samples, num\_features = x\_train.shape  
 num\_test\_samples, num\_test\_features = x\_test.shape  
  
 x\_train = x\_train.toarray()  
 temp = np.ones(shape=[num\_samples, 1], dtype=np.float32)  
 x\_train = np.concatenate([x\_train, temp], axis=1)  
 x\_test = x\_test.toarray()  
 temp = np.zeros(shape=[num\_test\_samples, 1], dtype=np.float32)  
 temp1 = np.ones(shape=[num\_test\_samples, 1], dtype=np.float32)  
 x\_test = np.concatenate([x\_test, temp, temp1], axis=1)  
 for i in range(0, len(y\_train)):  
 if y\_train[i] == -1:  
 y\_train[i] = 0  
 for i in range(0, len(y\_test)):  
 if y\_test[i] == -1:  
 y\_test[i] = 0  
 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 = 128  
 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  
  
 methods = ['SGD', 'NAG', 'RMSProp', 'Adadelta', 'ADAM']  
 sgd\_para\_list = {'learning\_rate': 0.01}  
 nag\_para\_list = {'miu': 0.9, 'learning\_rate': 0.01}  
 rmsprop\_para\_list = {'delta': 10e-7, 'rho': 0.95, 'learning\_rate': 0.1}  
 adam\_para\_list = {'delta': 10e-8, 'rho1': 0.9, 'rho2': 0.999, 'learning\_rate': 0.1}  
 adadelta\_para\_list = {'delta': 10e-7, 'miu': 0.9, 'learning\_rate': 10}  
 para\_list = {'SGD': sgd\_para\_list, 'NAG': nag\_para\_list, 'RMSProp': rmsprop\_para\_list,  
 'Adadelta': adadelta\_para\_list, 'ADAM': adam\_para\_list}  
 for method in methods:  
 iters = []  
 test\_errors = []  
 w = np.random.rand(feature\_size+1, 1)  
 parameter\_list = para\_list[method]  
 count = 0  
 for i in range(0, 2):  
 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, parameter\_list=parameter\_list, x=feed\_dict['x'], y=feed\_dict['y'], w=w)  
 test\_errors.append(compute\_loss(x\_test, y\_test, w))  
 plt.plot(iters, test\_errors, label=method)  
  
  
plt.xlabel('Iteration')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()

from sklearn import datasets as ds  
import numpy as np  
from numpy import random  
import matplotlib.pyplot as plt  
import os  
  
C = 1  
feature\_size = 123  
  
def compute\_loss(x, y, w):  
 # x.shape = [batch\_size, feature\_size + 1], y.shape = [batch\_size, 1], w.shape = [feature\_size + 1, 1]  
 pred = np.matmul(x, w)  
 hinge\_loss = np.maximum(1 - y \* pred, 0)  
 loss = np.mean(hinge\_loss\*\*2) + C \* np.sum(w\*\*2)  
 return loss  
  
  
def compute\_gradient(x, y, w):  
 # x.shape = [batch\_size, feature\_size + 1], y.shape = [batch\_size, 1], w.shape = [feature\_size + 1, 1]  
 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 \* C \* w  
 gradient = hinge\_loss\_gradient + norm\_gradient  
 return gradient # gradient.shape = [feature\_size + 1, 1], the same as w  
  
  
global\_list = {'NAG\_momentum': np.zeros([feature\_size + 1, 1]), 'RMS\_expectation': 1, 'ADAM\_s': 0, 'ADAM\_r': 0,  
 'ADADELDA\_r': 0, 'ADADELDA\_e': 0}  
  
  
def optimizer(method, parameter\_list, x, y, w):  
 global global\_list  
 if method == 'SGD':  
 # sgd\_para\_list = {'learning\_rate':0.01}  
 lr = parameter\_list['learning\_rate']  
 w -= lr \* compute\_gradient(x, y, w)  
 if method == 'NAG':  
 # nag\_para\_list = {'miu':0.9, 'learning\_rate':0.01}  
 lr = parameter\_list['learning\_rate']  
 miu = parameter\_list['miu']  
 momentum = global\_list['NAG\_momentum']  
 gradient = compute\_gradient(x, y, w - momentum \* lr \* miu)  
 momentum = momentum \* lr + gradient  
 w -= lr \* momentum  
 if method == 'RMSProp':  
 # rmsprop\_para\_list = {'delta':10e-7, 'rho':0.95, 'learning\_rate':0.1}  
 lr = parameter\_list['learning\_rate']  
 expectation = global\_list['RMS\_expectation']  
 rho = parameter\_list['rho']  
 delta = parameter\_list['delta']  
 gradient = compute\_gradient(x, y, w)  
 norm = np.sum(gradient\*\*2)  
 expectation = rho \* expectation + (1 - rho) \* norm  
 global\_list['RMS\_expectation'] = expectation  
 w -= lr \* gradient / (np.sqrt(expectation) + delta)  
 if method == 'ADAM':  
 # adam\_para\_list = {'delta':10e-8, 'rho1':0.9, 'rho2':0.999, 'learning\_rate':0.1}  
 delta = parameter\_list['delta']  
 rho1 = parameter\_list['rho1']  
 rho2 = parameter\_list['rho2']  
 lr = parameter\_list['learning\_rate']  
 s = global\_list['ADAM\_s']  
 r = global\_list['ADAM\_r']  
 gradient = compute\_gradient(x, y, w)  
 s = rho1 \* s + (1 - rho1) \* gradient  
 r = rho2 \* r + (1 - rho2) \* gradient \* gradient  
 s\_hat = s / (1 - rho1)  
 r\_hat = r / (1 - rho2)  
 w -= (lr \* s\_hat) / (np.sqrt(r\_hat) + delta)  
 global\_list['ADAM\_s'] = s  
 global\_list['ADAM\_r'] = r  
 if method == 'Adadelta':  
 # adadelta\_para\_list = {'delta':10e-7, 'miu':0.9, 'learning\_rate':10}  
 r = global\_list['ADADELDA\_r']  
 e = global\_list['ADADELDA\_e']  
 miu = parameter\_list['miu']  
 delta = parameter\_list['delta']  
 lr = parameter\_list['learning\_rate']  
 grad = compute\_gradient(x, y, w)  
 r = miu \* r + (1 - miu) \* grad \* grad  
 w -= (lr \* grad \* np.sqrt(e + delta)) / (np.sqrt(r + delta))  
 e = miu \* e + (1 - miu) \* e \* e  
 global\_list['ADADELDA\_r'] = r  
 global\_list['ADADELDA\_e'] = e  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x\_train, y\_train = ds.load\_svmlight\_file('./a9a')  
 x\_test, y\_test = ds.load\_svmlight\_file('./a9a.t')  
  
 num\_samples, num\_features = x\_train.shape  
 num\_test\_samples, num\_test\_features = x\_test.shape  
  
 x\_train = x\_train.toarray()  
 temp = np.ones(shape=[num\_samples, 1], dtype=np.float32)  
 x\_train = np.concatenate([x\_train, temp], axis=1)  
 x\_test = x\_test.toarray()  
 temp = np.zeros(shape=[num\_test\_samples, 1], dtype=np.float32)  
 temp1 = np.ones(shape=[num\_test\_samples, 1], dtype=np.float32)  
 x\_test = np.concatenate([x\_test, temp, temp1], axis=1)  
 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 = 32  
 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  
  
 methods = ['SGD', 'NAG', 'RMSProp', 'Adadelta', 'ADAM']  
 sgd\_para\_list = {'learning\_rate': 0.01}  
 nag\_para\_list = {'miu': 0.9, 'learning\_rate': 0.01}  
 rmsprop\_para\_list = {'delta': 10e-7, 'rho': 0.95, 'learning\_rate': 0.1}  
 adam\_para\_list = {'delta': 10e-8, 'rho1': 0.9, 'rho2': 0.999, 'learning\_rate': 0.1}  
 adadelta\_para\_list = {'delta': 10e-7, 'miu': 0.9, 'learning\_rate': 10}  
 para\_list = {'SGD': sgd\_para\_list, 'NAG': nag\_para\_list, 'RMSProp': rmsprop\_para\_list,  
 'Adadelta': adadelta\_para\_list, 'ADAM': adam\_para\_list}  
 for method in methods:  
 iters = []  
 test\_errors = []  
 w = np.random.rand(feature\_size+1, 1)  
 parameter\_list = para\_list[method]  
 count = 0  
 for i in range(0, 1):  
 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, parameter\_list=parameter\_list, x=feed\_dict['x'], y=feed\_dict['y'], w=w)  
 test\_errors.append(compute\_loss(x\_test, y\_test, w))  
 plt.plot(iters, test\_errors, label=method)  
  
  
plt.xlabel('Iteration')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()

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

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

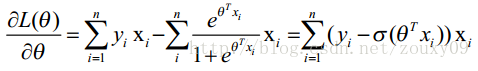
Random initialization

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

**Linear Regression:**

The include the bias of the linear regression.





**Linear Classification:**

**11. Experimental results and curve:**

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

Linear regression:

* Gamma = 0
* sgd\_para\_list = {'learning\_rate': 0.01}  
  nag\_para\_list = {'miu': 0.9, 'learning\_rate': 0.01}  
  rmsprop\_para\_list = {'delta': 10e-7, 'rho': 0.95, 'learning\_rate': 0.1}  
  adam\_para\_list = {'delta': 10e-8, 'rho1': 0.9, 'rho2': 0.999, 'learning\_rate': 0.1}  
  adadelta\_para\_list = {'delta': 10e-7, 'miu': 0.9, 'learning\_rate': 10}

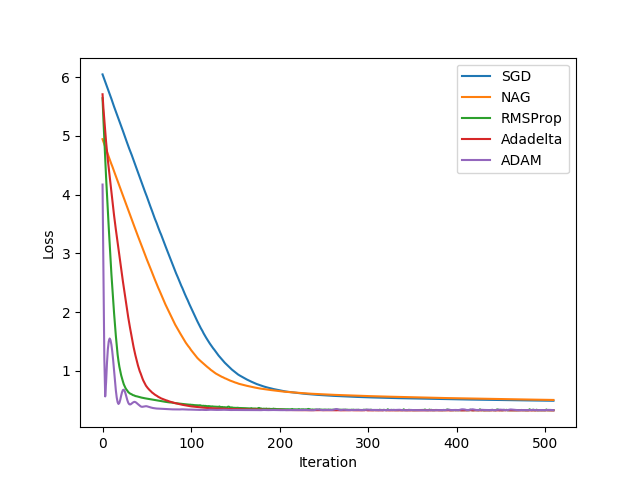
Linear classification:

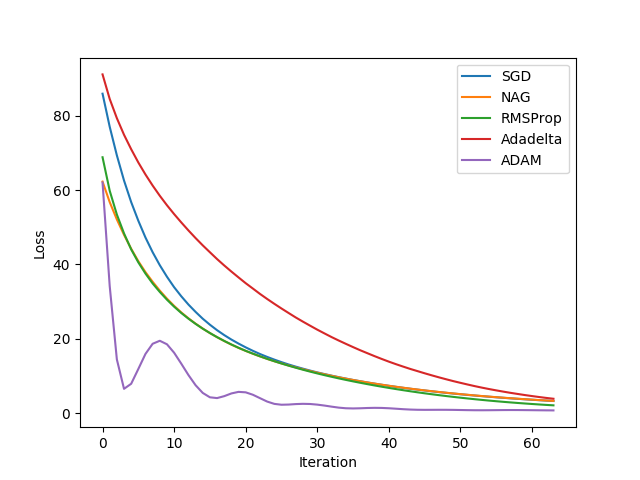
* Gamma = 0.1
* sgd\_para\_list = {'learning\_rate': 0.01}  
  nag\_para\_list = {'miu': 0.9, 'learning\_rate': 0.01}  
  rmsprop\_para\_list = {'delta': 10e-7, 'rho': 0.95, 'learning\_rate': 0.1}  
  adam\_para\_list = {'delta': 10e-8, 'rho1': 0.9, 'rho2': 0.999, 'learning\_rate': 0.1}  
  adadelta\_para\_list = {'delta': 10e-7, 'miu': 0.9, 'learning\_rate': 10}

## Assessment Results (based on selected validation):

## Predicted Results (Best Results):

## Loss curve:





**12. Results analysis:**

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

**14. Summary:**