

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**  陈星宇

**Student ID 201530611289**

**E-mail 549278303@qq.com**

**Tutor**   **谭明奎**

**Date submitted** **2017. 12 .15**

**1. Topic:Logistic Regression, Linear Classification and Stochastic Gradient Descent**

**2. Time: 2017-12-02**

**3. Reporter:陈星宇**

**4. Purposes:**

（1）Compare and understand the difference between gradient descent and stochastic gradient descent.

（2）Compare and understand the differences and relationships between Logistic regression and linear classification.

（3）Further understand the principles of SVM and practice on larger data.

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

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

1. **Experimental steps:**

**6.1 Logistic Regression and Stochastic Gradient Descent**

1. Load the training set and validation set.

（2） Initialize logistic regression model parameters, you can consider initializing zeros, random numbers or normal distribution.

（3） Select the loss function and calculate its derivation.

（4） Calculate gradient G toward loss function from partial samples.

（5） Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

（6） Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss LNAG ，LRMSProp，LAdaDelta and LAdam .

（7） Repeat step 4 to 6 for several times, and drawing graph of LNAG ，LRMSProp，LAdaDelta and LAdam with the number of iterations.

**6.2 Linear Classification and Stochastic Gradient Descent**

（1） Load the training set and validation set.

1. Initialize SVM model parameters, you can consider initializing zeros, random numbers or normal distribution.

（3） Select the loss function and calculate its derivation.

（4） Calculate gradient G toward loss function from partial samples.

（5） Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

（6） Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss LNAG ，LRMSProp，LAdaDelta and LAdam .

（7） Repeat step 4 to 6 for several times, and drawing graph of LNAG ，LRMSProp，LAdaDelta and LAdam with the number of iterations.

**7. Code:**

**7.1 Logistic Regression and Stochastic Gradient Descent**

def sigmoid(z):

s = 1/(1+np.exp(-z.A1))

s = np.array(list(map(lambda x : x,s)))

return s

def loss(X,y,w,lamb):

h = sigmoid(X\*w)

m = y.shape[0]

y\_temp = np.array(list(map(lambda x:1 if x[0]==1 else 0,y.A)))

J = -y\_temp\*np.log(h)-(1-y\_temp)\*np.log(1-h)

w\_reg = w

w\_reg[0]=0

return J.sum()/m+lamb/(2\*m)\*((w\_reg.A\*\*2).sum())

def gradient(X,y,w,lamb):

m = y.shape[0]

y\_temp = np.array(list(map(lambda x:1 if x[0]==1 else 0,y.A)))

h\_y = np.matrix(sigmoid(X\*w)-y\_temp)

dJ = X.T\*(h\_y.T)/m

w\_reg = w

w\_reg[0] = 0

g = dJ+w\_reg\*lamb/m

return g

def gradientDecent(X,y,w,alpha,lamb,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

print("origin train loss:%f"%loss(X,y,w,lamb))

train\_loss\_history.append(loss(X,y,w,lamb))

print("origin validation loss:%f"%loss(val\_x,val\_y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

for i in range(num\_rounds):

random = list(range(0,X.shape[0]))

np.random.shuffle(random)

random = random[0:100]

gdx = X[random]

gdy = y[random]

w = w - gradient(gdx,gdy,w,lamb)\*alpha

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

return w,train\_loss\_history,val\_loss\_history

def NAG(X,y,w,alpha,lamb,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

print("NAG begin")

r = 0.9

v = np.zeros(w.shape)

for i in range(num\_rounds):

random = list(range(0,X.shape[0]))

np.random.shuffle(random)

random = random[0:100]

gdx = X[random]

gdy = y[random]

\_w = w - r\*v

gt = gradient(gdx,gdy,\_w,lamb)

v = r\*v + alpha\*gt

w = w - v

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

return w,train\_loss\_history,val\_loss\_history

def RMSProp(X,y,w,alpha,lamb,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

print("RMSProp begin")

r = 0.99

v = np.zeros(w.shape)

e = 1e-8

for i in range(num\_rounds):

random = list(range(0,X.shape[0]))

np.random.shuffle(random)

random = random[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,lamb)

gt\_2 = np.matrix((gt.A)\*\*2)

v = r\*v + (1-r)\*gt\_2

w = w - alpha/np.sqrt(v+e).A\*(gt.A)

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

return w,train\_loss\_history,val\_loss\_history

def AdaDelta(X,y,w,alpha,lamb,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

print("AdaDelta begin")

r = 0.95

v = np.zeros(w.shape)

e = 1e-4

t = np.matrix(np.zeros(w.shape))

for i in range(num\_rounds):

random = list(range(0,X.shape[0]))

np.random.shuffle(random)

random = random[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,lamb)

gt\_2 = np.matrix((gt.A)\*\*2)

v = r\*v + (1-r)\*gt\_2

dw = -(np.sqrt(t+e).A/np.sqrt(v+e).A)\*gt.A

w = w + dw

t = r\*t+(1-r)\*(t.A\*t.A)

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

return w,train\_loss\_history,val\_loss\_history

def Adam(X,y,w,alpha,lamb,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

print("Adam begin")

r = 0.999

v = np.zeros(w.shape)

e = 1e-8

m = np.zeros(w.shape)

b1 = 0.9

for i in range(num\_rounds):

random = list(range(0,X.shape[0]))

np.random.shuffle(random)

random = random[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,lamb)

gt\_2 = np.matrix((gt.A)\*\*2)

v = r\*v + (1-r)\*gt\_2

m = b1\*m + (1-b1)\*gt.A

alp = alpha\*np.sqrt(1-r)/(1-b1)

w = w - alp\*m/np.sqrt(v+e).A

train\_loss\_history.append(loss(X,y,w,lamb))

val\_loss\_history.append(loss(val\_x,val\_y,w,lamb))

return w,train\_loss\_history,val\_loss\_history

**7.2 Linear Classification and Stochastic Gradient Descent**

def loss(X,y,w,C):

m = y.shape[0]

hinge = sum(list(map(lambda x:max(0,x[0]),(1-y.A\*(X\*w).A))))

w\_2 = sum(w.A\*\*2)[0]

return (0.5\*w\_2+C\*hinge)/m

def gradient(X,y,w,C):

m = y.shape[0]

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

indicator = 1-y.A\*((X\*w).A)

for i in range(m):

if indicator[i]>=0:

dw += w - C\*(y[i]\*X[i]).T

else:

dw += w

return dw/m

def gradientDecent(X,y,w,C,alpha,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

print("GradientDecent begin")

for i in range(num\_rounds):

random = np.random.permutation(X.shape[0])[0:100]

gdx = X[random]

gdy = y[random]

new\_w = w - gradient(gdx,gdy,w,C)\*alpha

w = new\_w

train\_loss\_history.append(loss(gdx,gdy,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

return w,train\_loss\_history,val\_loss\_history

def NAG(X,y,w,C,alpha,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

print("NAG begin")

r = 0.9

vw = np.zeros(w.shape)

for i in range(num\_rounds):

random = np.random.permutation(X.shape[0])[0:100]

gdx = X[random]

gdy = y[random]

\_w = w - r\*vw

gt = gradient(gdx,gdy,\_w,C)

vw = r\*vw + alpha\*gt

w = w - vw

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

return w,train\_loss\_history,val\_loss\_history

def RMSProp(X,y,w,C,alpha,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

print("RMSProp begin")

r = 0.9

vw = np.zeros(w.shape)

e = 1e-8

for i in range(num\_rounds):

random = np.random.permutation(X.shape[0])[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,C)

gt\_2 = np.matrix(gt\*gt)

vw = r\*vw + (1-r)\*gt\_2

w = w - 0.01/np.sqrt(vw+e).A\*(gt)

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

return w,train\_loss\_history,val\_loss\_history

def AdaDelta(X,y,w,C,alpha,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

print("AdaDelta begin")

r = 0.95

vw = np.zeros(w.shape)

e = 1e-4

tw = np.matrix(np.zeros(w.shape))

for i in range(num\_rounds):

random = np.random.permutation(X.shape[0])[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,C)

gt\_2 = np.matrix((gt)\*\*2)

vw = r\*vw + (1-r)\*gt\_2

dw = -(np.sqrt(tw+e)/np.sqrt(vw+e)).A\*gt

w = w + dw

tw = r\*tw+(1-r)\*(tw.A\*tw.A)

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

return w,train\_loss\_history,val\_loss\_history

def Adam(X,y,w,C,alpha,num\_rounds,val\_x,val\_y):

train\_loss\_history = []

val\_loss\_history = []

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(X,y,w,C))

print("Adam begin")

r = 0.99

vw = np.zeros(w.shape)

e = 1e-6

mw = np.zeros(w.shape)

b1 = 0.9

for i in range(num\_rounds):

random = np.random.permutation(X.shape[0])[0:100]

gdx = X[random]

gdy = y[random]

gt = gradient(gdx,gdy,w,C)

gt\_2 = np.matrix(gt\*\*2)

vw = r\*vw + (1-r)\*gt\_2

mw = b1\*mw + (1-b1)\*gt

alp = alpha\*np.sqrt(1-r)/(1-b1)

w = w - alp\*mw/np.sqrt(vw+e).A

train\_loss\_history.append(loss(X,y,w,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,C))

return w,train\_loss\_history,val\_loss\_history

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

**8.1 Logistic Regression and Stochastic Gradient Descent**

In this experiment I initialize the parameter w randomly in the range of (0，1)

**8.2 Linear Classification and Stochastic Gradient Descent**

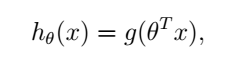
In this experiment we initialize the parameter w and b randomly in the range of (0，1).In the actual implement I merge b into w by adding a column to w and adding a column with ones to X.

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

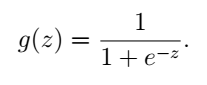
**9.1 Logistic Regression and Stochastic Gradient Descent**

**Loss Function :**

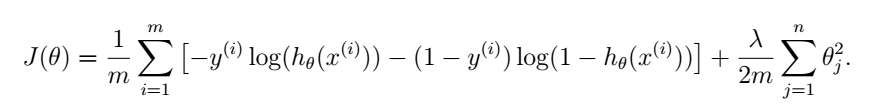
First we define our hypothesis function as



Where g(z) is the sigmiod function



Base on the hypothesis function above ,the cost function is defined as



The term at the tail of formula is the regularization term and λ is the regularization parameter .For intuition ,the loss function will calculate the loss of examples that are classified to the wrong class .

**9.2 Linear Classification and Stochastic Gradient Descent**

**10. Experimental results and curve:**

## Hyper-parameter selection:

* **Logistic Regression and Stochastic Gradient Descent**
* **Linear Classification and Stochastic Gradient Descent**

## Predicted Results (Best Results):

* **Logistic Regression and Stochastic Gradient Descent**
* **Linear Classification and Stochastic Gradient Descent**

## Loss curve:

* **Logistic Regression and Stochastic Gradient Descent**
* **Linear Classification and Stochastic Gradient Descent**

1. **Results analysis:**

**10.1 Logistic Regression and Stochastic Gradient Descent**

**10.2 Linear Classification and Stochastic Gradient Descent**

1. **Similarities and differences between logistic regression and linear classification：**

**13. Summary:**