

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**  王泽众 **Wesley Wang**

**Student ID 201530612958**

**E-mail w2w.wzz@foxmail.com**

**Tutor**  **谭明奎**

**Date submitted** **2017.12 . 2**

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

**2. Time: 2017.12.2**

**3. Reporter:王泽众 Wesley Wang**

**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](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#_blank) of [LIBSVM Data](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/), including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

**6. Experimental steps:**

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient 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 ，， and .
7. Repeate step 4 to 6 for several times, and drawing graph of ，， and with the number of iterations.

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient 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 ，， and .
7. Repeate step 4 to 6 for several times, and drawing graph of ，， and with the number of iterations.

Finishing experiment report according to result: The template of report can be found in [example repository](https://github.com/chenyaofo/ML2017-lab-02).

**7. Code:**

(Fill in the contents of 8-12 respectively for linear regression and linear classification)

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

# 读取数据

data = load\_svmlight\_file(f='a9a')

# 分割数据

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data[0], data[1], test\_size=0.4, random\_state=0)

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

def \_\_init\_\_(self, turns = 50, c=0.1, learning\_rate = 0.01, batch\_size = 0.05,

silence = False, plot = True, method='AdaDelta'):

self.turns\_ = turns

self.learning\_rate\_ = learning\_rate

self.silence\_ = silence

self.plot\_ = plot

self.method\_ = method

self.c\_ = c

self.batch\_size\_ = 0.05

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

**Logisitc loss function:**

def calc\_error(self, X, y, w, b):

'''

error = 1/N\* ln(1+e^(-yn\*(wT\*xn+b))

'''

N = X.shape[0]

t1 = X.dot(w)

t = -y\*(X.dot(w)+b)

error = 1.0/N\*np.sum(np.log(1+np.exp(-y\*(X.dot(w)+b))))

return error

**SVM loss function:**

def calc\_error(self, X, y, w, b):

'''

error = 0.5||w||^2 + C\* sum (max(0, 1-yi(wTxi + b)))

'''

hinge = 1 - (X.dot(w) + b)\*y

hinge = np.array([max(0, x) for x in hinge])

error = 0.5 \* np.sum(w \*\* 2) + self.c\_ \* np.sum(hinge)

return error

different optimized methods(NAG，RMSProp，AdaDelta and Adam)

NAG

def NAG(self, X, y, w, b):

'''

y\_ = Xw + b

g\_w = (2/N)\*XT\*(y\_-y)

G\_w\_t = G\_w\_t + g^2

g\_b = (2/N)\*(y\_-y)

G\_b\_t = G\_b\_t + g^2

'''

if hasattr(self, 'v\_w\_t') == False:

n\_features = X.shape[1]

self.v\_w\_t = np.zeros([n\_features])

self.v\_b\_t = 0

g\_w, g\_b = self.calc\_gradient(X, y, w, b)

gamma = 0.9

self.v\_w\_t = gamma\*self.v\_w\_t + self.learning\_rate\_\*g\_w

self.v\_b\_t = gamma\*self.v\_b\_t + self.learning\_rate\_\*g\_b

w1 = w - self.v\_w\_t

b1 = b - self.v\_b\_t

return w1, b1

RMSProp

def RMSProp(self, X, y, w, b):

'''

y\_ = Xw + b

g\_w = (2/N)\*XT\*(y\_-y)

G\_w\_t = G\_w\_t + g^2

g\_b = (2/N)\*(y\_-y)

G\_b\_t = G\_b\_t + g^2

'''

if hasattr(self, 'G\_w\_t') == False:

self.G\_w\_t = 0

self.G\_b\_t = 0

g\_w, g\_b = self.calc\_gradient(X, y, w, b)

gamma = 0.9

self.G\_w\_t = gamma\*self.G\_w\_t + (1-gamma)\*g\_w.dot(g\_w)

self.G\_b\_t = gamma\*self.G\_b\_t + (1-gamma)\*g\_b\*g\_b

# print("G\_w\_t:", self.G\_w\_t)

# print("G\_b\_t:", self.G\_b\_t)

w1 = w - self.learning\_rate\_/((self.G\_w\_t + 10\*\*(-8))\*\*0.5)\*g\_w

b1 = b - self.learning\_rate\_/((self.G\_b\_t + 10\*\*(-8))\*\*0.5)\*g\_b

return w1, b1

AdaDelta

def AdaDelta(self, X, y, w, b):

'''

y\_ = Xw + b

delta = gamma\*delta + (1-gamma)\*g\_t.dot(g\_t)

g\_w = (2/N)\*XT\*(y\_-y)

G\_w\_t = G\_w\_t + g^2

g\_b = (2/N)\*(y\_-y)

G\_b\_t = G\_b\_t + g^2

'''

if hasattr(self, 'G\_w\_t') == False:

self.G\_w\_t = 0

self.G\_b\_t = 0

self.delta\_w = 0

self.delta\_b = 0

N = X.shape[0]

y\_ = X.dot(w) + b

# 求梯度

# g\_w = (2/float(N))\*X.T\*(y\_ - y)

# g\_b = (2/float(N))\*np.sum(y\_-y)

g\_w, g\_b = self.calc\_gradient(X, y, w, b)

gamma = 0.95

# 求Gt

self.G\_w\_t = gamma\*self.G\_w\_t + (1-gamma)\*g\_w.dot(g\_w)

self.G\_b\_t = gamma\*self.G\_b\_t + (1-gamma)\*g\_b\*g\_b

# 求delta\_theta

d\_w = -((self.delta\_w + 10\*\*(-8))\*\*0.5)/((self.G\_w\_t + 10\*\*(-8))\*\*0.5)\*g\_w

d\_b = -((self.delta\_b + 10\*\*(-8))\*\*0.5)/((self.G\_b\_t + 10\*\*(-8))\*\*0.5)\*g\_b

# 求delta\_t

self.delta\_w = gamma\*self.delta\_w + (1-gamma)\*d\_w.dot(d\_w)

self.delta\_b = gamma\*self.delta\_b + (1-gamma)\*d\_b\*d\_b

w1 = w + d\_w

b1 = b + d\_b

return w1, b1

Adam

def Adam(self, X, y, w, b):

'''

y\_ = Xw + b

g\_w = (2/N)\*XT\*(y\_-y)

G\_w\_t = G\_w\_t + g^2

g\_b = (2/N)\*(y\_-y)

G\_b\_t = G\_b\_t + g^2

'''

if hasattr(self, 'm\_w\_t') == False:

n\_features = X.shape[1]

self.m\_w\_t = np.zeros([n\_features])

self.m\_b\_t = 0

self.G\_w\_t = 0

self.G\_b\_t = 0

self.t = 0

self.t += 1

g\_w, g\_b = self.calc\_gradient(X, y, w, b)

belta = 0.9

gamma = 0.9

# 求 m\_t

self.m\_w\_t = belta\*self.m\_w\_t + (1-belta)\*g\_w

self.m\_b\_t = belta\*self.m\_b\_t + (1-belta)\*g\_b

# 求 G\_t

self.G\_w\_t = gamma\*self.G\_w\_t + (1-gamma)\*g\_w.dot(g\_w)

self.G\_b\_t = gamma\*self.G\_b\_t + (1-gamma)\*g\_b\*g\_b

a = self.learning\_rate\_\*(1-gamma\*\*self.t)\*\*0.5/(1-belta\*\*self.t)

w1 = w - a\*self.m\_w\_t/((self.G\_w\_t + 10\*\*(-8))\*\*0.5)

b1 = b - a\*self.m\_b\_t/((self.G\_b\_t + 10\*\*(-8))\*\*0.5)

1. **Experimental results and curve:**

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

Logistic Regression

Leaning rate ： 0.01

Turns： 50

Classification

Learning rate ：0.01

Tradeoff c ： 0.01

Turns： 50

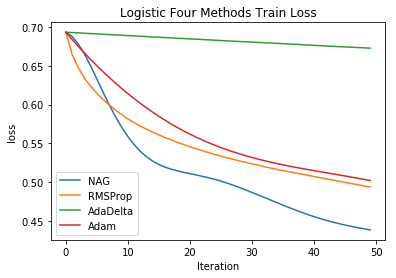
## Assessment Results (based on selected validation):

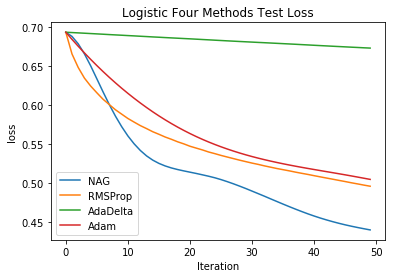
Logistic Regression：

Classification：

## Predicted Results (Best Results):

**Logistic Regression:**



****

**Classification:**

## 

1. **Results analysis:**

Regression :

With the iteration, loss gradually decreases, and finally tents to smooth.

The learning rate have a greate influence on the performance of linear regression.

Classification :

With the iteration, loss gradually decreases, and finally tents to smooth.

The learning rate have a greate influence on the performance of linear regression either. What’s more, the tradeoff coefficient also have a great influence.

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

Regression :

With the iteration, loss gradually decreases, and finally tents to smooth.

The learning rate have a greate influence on the performance of linear regression.

Classification :

With the iteration, loss gradually decreases, and finally tents to smooth.

The learning rate have a greate influence on the performance of linear regression either. What’s more, the tradeoff coefficient also have a great influence.

1. **Summary:**

Linear Regression and Linear Classification can deal with some of simple situations, and it’s ability is limited. It can’t tackle nonlinear problem.