



华南理工大学

South China University of Technology

《机器学习》课程实验报告

学 院 软件学院

专 业 软件工程

组 员 王煜

学 号 201530612927

邮 箱 1628021431@qq.com

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 9 日

3. 报告人: 王煜

4. 实验目的:

1. 对比理解梯度下降和随机梯度下降的区别与联系。
2. 对比理解逻辑回归和线性分类的区别与联系。
3. 进一步理解 SVM 的原理并在较大数据上实践。

5. 数据集以及数据分析:

实验使用的是 [LIBSVM Data](#) 中的 [a9a](#) 数据, 包含 32561 /

16281(testing)个样本, 每个样本有 123/123 (testing)个属性。

其中, 测试集只有 122 个属性, 我们将第 123 个属性初始化为 0, 同时要增加一列 1.

6. 实验步骤:

逻辑回归与随机梯度下降

1. 读取实验训练集和验证集。
2. 逻辑回归模型参数初始化, 可以考虑全零初始化, 随机初始化或者正态分布初始化。
3. 选择 Loss 函数及对其求导, 过程详见课件 ppt。
4. 求得部分样本对 Loss 函数的梯度。
5. 使用不同的优化方法更新模型参数 (NAG, RMSProp, AdaDelta 和 Adam)。

6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。

在验证集上测试并得到不同优化方法的 *Loss* 函数值，*acc*，*f1* 和 *auc*。

线性分类与随机梯度下降

1. 读取实验训练集和验证集。

2. 支持向量机模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。

3. 选择 *Loss* 函数及对其求导，过程详见课件 *ppt*。

4. 求得部分样本对 *Loss* 函数的梯度。

5. 使用不同的优化方法更新模型参数（*NAG*，*RMSPProp*，*AdaDelta* 和 *Adam*）。

6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。

在验证集上测试并得到不同优化方法的 *Loss* 函数值，*acc*，*f1* 和 *auc*。

7. 重复步骤 4-6 若干次，画出 *Loss*，*acc*，*f1* 和 *auc* 随迭代次数的变化图。

7. 重复步骤 4-6 若干次，画出 *Loss*，*acc*，*f1* 和 *auc* 随迭代次数的变化图。

7. 代码内容:

（针对逻辑回归和线性分类分别填写 8-11 内容）

线性分类:

```
# -*- coding: utf-8 -*-
```

```
"""
```

Created on Wed Dec 13 00:10:30 2017

@author: wangyu

"""

from sklearn.datasets import load_svmlight_file

import numpy as np

import matplotlib.pyplot as plt

feature_num = 123

methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]

def sigmoid(z):

return 1/(1+np.exp(-1.0*z))

def loss_function(Weight, X,y):

L = 0

N = y.shape[0]

temp=1-y*np.matmul(X,Weight)

L = sum(np.maximum(0, temp))

```

        loss = 0.5 * np.matmul(Weight.T, Weight)[0][0] + (L *
parm.get("C"))/N

    return loss

```

```

def gradient(Weight,X,y):

    L = np.zeros((124,1))

    temp=1-y*np.matmul(X,Weight)

    temp=np.maximum(temp/np.abs(temp),0)

    y=y*temp

    L=-np.matmul(X.T,y)

    return (parm.get("C") * L) + Weight

```

```

def SGD(Weight,X,y):

    l_rate = 0.01

    Weight -= l_rate*gradient(Weight,X,y)

    return Weight

```

```

def NAG(Weight,X,y):

```

```

global list
global parm

l_rate = 0.012

#momentum=list.get('NAG')

m = np.zeros([feature_num + 1, 1])

gamma = 0.9

grad=gradient(Weight-(gamma*m),X,y)

update_ = l_rate*grad+ m*gamma

m=update_

Weight -= update_

return Weight

```

```

def RMSProp(Weight,X,y):

    #G=list.get("RMSProp")

    m=np.zeros([feature_num+1,1])

    gamma = 0.8

    Epsilon = 10e-8

    l_rate = 0.015

    grad=gradient(Weight,X,y)

    G=m+(1-gamma)*grad**2

    Weight-=l_rate*grad/np.sqrt(G+Epsilon)

```

```
m=G  
return Weight
```

```
def AdaDelta(Weight,X,y):  
    m = np.zeros([feature_num+1,1])  
    n = np.zeros([feature_num+1,1])  
    gamma = 0.98  
    Epsilon = 10e-6  
    grad=gradient(Weight,X,y)  
    EG=gamma*m+(1-gamma)*grad**2  
  
    delta=-1*grad*np.sqrt(n+Epsilon)/np.sqrt(EG+Epsilon)  
    EX=gamma*n+(1-gamma)*delta**2  
    m = EG  
    n = EX  
    Weight += delta  
    return Weight
```

```
def Adam(Weight,X,y):  
    beta = 0.8
```

```

gamma = 0.98

Epsilon = 10e-6

l_rate = 0.01

_m = np.zeros([feature_num+1,1])

_g = np.zeros([feature_num+1,1])

_t = 0

t=_t+1

_t=t

grad=gradient(Weight,X,y)

M=beta*_m+(1-beta)*grad

_m=M

G=gamma*_g+(1-gamma)*grad**2

_g=G

M_bias=M/(1-beta**t)

G_bias=G/(1-gamma**t)

Weight -= l_rate*M_bias/(np.sqrt(G_bias)+Epsilon)

return Weight

```

```

def opitimizer(W,X_train,y_train,method):

    if method=="SGD":

        return SGD(W,X_train,y_train)

```



```

if method=="NAG":

    return NAG(W,X_train,y_train)

if method=="RMSProp":

    return RMSProp(W,X_train,y_train)

if method=="AdaDelta":

    return AdaDelta(W,X_train,y_train)

if method=="Adam":

    return Adam(W,X_train,y_train)

```

```

def getdata():

    X_train, y_train =
load_svmlight_file("C:/Users/wangyu/Desktop/大三/机器学习/实验
/a9a.txt")

    datasize,features=X_train.shape

    X_train=np.c_[np.ones(len(X_train.toarray()))],
X_train.toarray()]

    for i in range(0, len(y_train)):

        if y_train[i] == -1:

            y_train[i] = 0

X_test,y_test=load_svmlight_file("C:/Users/wangyu/Desktop/大三/

```

机器学习/实验/a9a2.txt")

```
X_test=np.c_[X_test.toarray(),np.zeros(len(X_test.toarray()))]
```

```
X_test=np.c_[np.ones(len(X_test)),X_test]
```

```
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])
```

```
X_train,y_train=shuffle(X_train,y_train)
```

```
X_test,y_test=shuffle(X_test,y_test)
```

```
return X_train,y_train,X_test,y_test,datasize,features
```

```
def get_sub_batch(batch_count,X,y,data_size):
```

```
    if (1+batch_count)*batch_size<=data_size:
```

```
        return X[batch_count*batch_size:(batch_count + 1) * 
```

```
batch_size],y[batch_count*batch_size:(batch_count + 1) * batch_size]
```

```
    else:
```

```
        return
```

```
X[batch_count*batch_size:data_size],y[batch_count*batch_size:data_size]
```

```
def shuffle(X,y):
```

```
    rng_state = np.random.get_state()
```

```
    np.random.shuffle(X)
```

```
    np.random.set_state(rng_state)
```

```
    np.random.shuffle(y)
```

```
    return X,y
```

```
def LinearClassif():
```

```
    X_train, y_train, X_test, y_test, data_size, features_num =  
getdata()
```

```
    plt.xlabel('iters')
```

```
    plt.ylabel('Loss')
```

```
    for method in methods:
```

```
        W = np.random.rand(features_num + 1, 1)
```

```
        iter_ = []
```

```
        error = []
```

```
        num = 0
```

```
        for j in range(2):
```

```
            for i in range(0, int(data_size / batch_size ) + 1):
```

```

        iter_.append(num)

        X,y=get_sub_batch(i,X_train,y_train,data_size)

        W=optimizer(W,X,y,method)

        error.append(loss_function(W,X_test,y_test))

        num+=1

plt.plot(iter_, error, label=method)

plt.legend()

plt.show()

```

LinearClassif()

逻辑回归：

```

8. # -*- coding: utf-8 -*-
9. """
10.Created on Wed Dec 13 00:10:30 2017
11.
12.@author: wangyu
13. """
14.
15.from sklearn.datasets import load_svmlight_file
16.from sklearn import preprocessing
17.
18.from sklearn.model_selection import train_test_split
19.import numpy as np
20.import matplotlib.pyplot as plt
21.
22.
23.feature_num=123
24.batch_size=128
25.SGD_methods=["SGD", "NAG", "RMSProp", "AdaDelta", "Adam"]
26.
27.#定义 sigmoid 函数

```

```

28. def sigmoid(z):
29.     return 1/(1+np. exp(-1.0*z))
30.
31. #定义损失函数
32. def loss_function(Weight, X, y):
33.     l = np.matmul(X, Weight)
34.     loss = -np.mean(y * np.log(sigmoid(l)) + (1 - y) * np.log(1 -
        sigmoid(l)))
35.     return loss
36.
37. #求解梯度
38. def gradient(Weight, X, y):
39.     l = np.matmul(X, Weight)
40.     out = sigmoid(l)
41.     error = out - y
42.     grad = np.matmul(X. transpose(), error) / y. shape[0]
43.     return grad
44.
45. def SGD(Weight, X, y):
46.     l_rate = 0.008
47.     Weight -= l_rate * gradient(Weight, X, y)
48.     return Weight
49.
50. def NAG(Weight, X, y):
51.     l_rate = 0.012
52.     m = np.zeros([feature_num + 1, 1])
53.     gamma = 0.92
54.     grad = gradient(Weight - (gamma * m), X, y)
55.     update_ = l_rate * grad + m * gamma
56.     m = update_
57.     Weight -= update_
58.     return Weight
59.
60. def RMSProp(Weight, X, y):
61.     m = np.zeros([feature_num + 1, 1])
62.     gamma = 0.8
63.     Epsilon = 10e-8
64.     l_rate = 0.015
65.     grad = gradient(Weight, X, y)
66.     G = m + (1 - gamma) * grad ** 2
67.     Weight -= l_rate * grad / np. sqrt(G + Epsilon)
68.     m = G
69.     return Weight
70.

```

```

71. def AdaDelta(Weight, X, y):
72.     m = np.zeros([feature_num+1, 1])
73.     n = np.zeros([feature_num+1, 1])
74.     gamma = 0.98
75.     Epsilon = 10e-6
76.     grad=gradient(Weight, X, y)
77.     EG=gamma*m+(1-gamma)*grad**2
78.
79.     delta=-1*grad*np.sqrt(n+Epsilon)/np.sqrt(EG+Epsilon)
80.     EX=gamma*n+(1-gamma)*delta**2
81.     m = EG
82.     n = EX
83.     Weight += delta
84.     return Weight
85.
86. def Adam(Weight, X, y):
87.     beta = 0.8
88.     gamma = 0.98
89.     Epsilon = 10e-6
90.     l_rate = 0.01
91.     _m = np.zeros([feature_num+1, 1])
92.     _g = np.zeros([feature_num+1, 1])
93.     _t = 0
94.     t=_t+1
95.     _t=t
96.     grad=gradient(Weight, X, y)
97.     M=beta*_m+(1-beta)*grad
98.     _m=M
99.     G=gamma*_g+(1-gamma)*grad**2
100.    _g=G
101.    M_bias=M/(1-beta**t)
102.    G_bias=G/(1-gamma**t)
103.    Weight -= l_rate*M_bias/(np.sqrt(G_bias)+Epsilon)
104.    return Weight
105.
106.
107. def optimizer(Weight, X, y, method):
108.     if method=="SGD":
109.         return SGD(Weight, X, y)
110.     if method=="NAG":
111.         return NAG(Weight, X, y)
112.     if method=="RMSProp":
113.         return RMSProp(Weight, X, y)
114.     if method=="AdaDelta":

```

```

115.         return AdaDelta(Weight, X, y)
116.     if method=="Adam":
117.         return Adam(Weight, X, y)
118.
119. def getdata():
120.     X_train, y_train =
        load_svmlight_file("C:/Users/wangyu/Desktop/大三/机器学习/实验
/a9a.txt")
121.     datasize, features=X_train.shape
122.     X_train=np.c_[np.ones(len(X_train.toarray()))],
        X_train.toarray()]
123.     for i in range(0, len(y_train)):
124.         if y_train[i] == -1:
125.             y_train[i] = 0
126.
        X_test,y_test=load_svmlight_file("C:/Users/wangyu/Desktop/大三/
机器学习/实验/a9a2.txt")
127.
        X_test=np.c_[X_test.toarray(), np.zeros(len(X_test.toarray()))]
128.     X_test=np.c_[np.ones(len(X_test)),X_test]
129.     for i in range(0, len(y_test)):
130.         if y_test[i] == -1:
131.             y_test[i] = 0
132.     y_train = y_train.reshape([len(y_train), 1])
133.     y_test = y_test.reshape([len(y_test), 1])
134.     X_train,y_train=shuffle(X_train,y_train)
135.     X_test,y_test=shuffle(X_test,y_test)
136.     return X_train,y_train,X_test,y_test,datasize,features
137.
138. def get_sub_batch(batch_count,X,y,data_size):
139.     if (1+batch_count)*batch_size<=data_size:
140.         return X[batch_count*batch_size:(batch_count + 1) *
            batch_size],y[batch_count*batch_size:(batch_count + 1) *
            batch_size]
141.     else:
142.         return
        X[batch_count*batch_size:data_size],y[batch_count*batch_size:da
            ta_size]
143.
144. def shuffle(X, y):
145.     rng_state = np.random.get_state()
146.     np.random.shuffle(X)
147.     np.random.set_state(rng_state)
148.     np.random.shuffle(y)

```

```

149.     return X, y
150.
151. def LogicReg():
152.     X_train, y_train, X_test, y_test, data_size, features_num =
        getdata()
153.     plt.xlabel('iters')
154.     plt.ylabel('Loss')
155.
156.     for method in SGD_methods:
157.         W = np.random.rand(features_num + 1, 1)
158.         iter_ = []
159.         error = []
160.         num = 0
161.         for j in range(2):
162.             for i in range(0, int(data_size / batch_size) + 1):
163.                 iter_.append(num)
164.
165.                 X, y = get_sub_batch(i, X_train, y_train, data_size)
166.                 W = optimizer(W, X, y, method)
167.                 error.append(loss_function(W, X_test, y_test))
168.                 num += 1
169.             plt.plot(iter_, error, label=method)
170.         plt.legend()
171.         plt.show()
172. LogicReg()

```

173. 模型参数的初始化方法:

初始化是首先随机设置，后续根据产生的图进行一步一步地调整。

最终和论文中所给的参数差不多。

模型参数见 10 中的超参数选择。

174. 选择的 loss 函数及其导数:

逻辑回归;

$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

$$J(\mathbf{w}) = -\frac{1}{n} \left[\sum_{i=1}^n y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\mathbf{w}}(\mathbf{x}_i)) \right]$$

导数：

$$\begin{aligned} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= -y \cdot \frac{1}{h_{\mathbf{w}}(\mathbf{x})} \cdot \frac{\partial h_{\mathbf{w}}(\mathbf{x})}{\partial \mathbf{w}} + (1 - y) \cdot \frac{1}{1 - h_{\mathbf{w}}(\mathbf{x})} \frac{\partial h_{\mathbf{w}}(\mathbf{x})}{\partial \mathbf{w}} \\ &= -y \cdot \frac{1}{h_{\mathbf{w}}(\mathbf{x})} \cdot \frac{\partial g(\mathbf{w}^\top \mathbf{x})}{\partial \mathbf{w}} + (1 - y) \cdot \frac{1}{1 - h_{\mathbf{w}}(\mathbf{x})} \frac{\partial g(\mathbf{w}^\top \mathbf{x})}{\partial \mathbf{w}} \\ &= \left(-\frac{xy}{h_{\mathbf{w}}(\mathbf{x})} + \frac{\mathbf{x}(1 - y)}{1 - h_{\mathbf{w}}(\mathbf{x})} \right) \cdot g(\mathbf{w}^\top \mathbf{x}) \cdot [1 - g(\mathbf{w}^\top \mathbf{x})] \\ &= (h_{\mathbf{w}}(\mathbf{x}) - y) \mathbf{x} \end{aligned}$$

线性分类：

$$\min_{\mathbf{w}, b} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

10.实验结果和曲线图：（各种梯度下降方式分别填写此项）

超参数选择：

逻辑回归：

SDG: learning_rate = 0.008

NAG: learning_rate = 0.012 gamma = 0.92

RMSProp learning_rate = 0.015 gamma = 0.8 Epsilon = 10e-8

AdaDelta gamma = 0.98 Epsilon = 10e-6

Adam learning_rate = 0.01 beta = 0.8 gamma = 0.98 Epsilon = 10e-6

线性分类：

SDG: learning_rate = 0.01

NAG: learning_rate = 0.012 gamma = 0.9

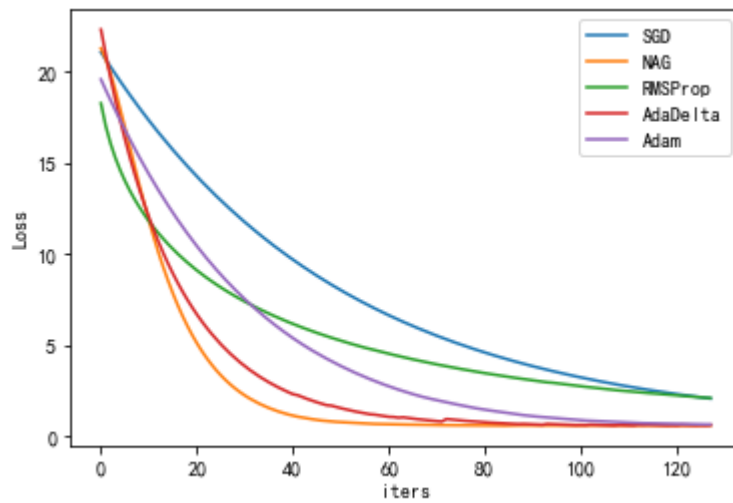
RMSProp learning_rate = 0.015 gamma = 0.8 Epsilon = 10e-8

AdaDelta gamma = 0.98 Epsilon = 10e-6

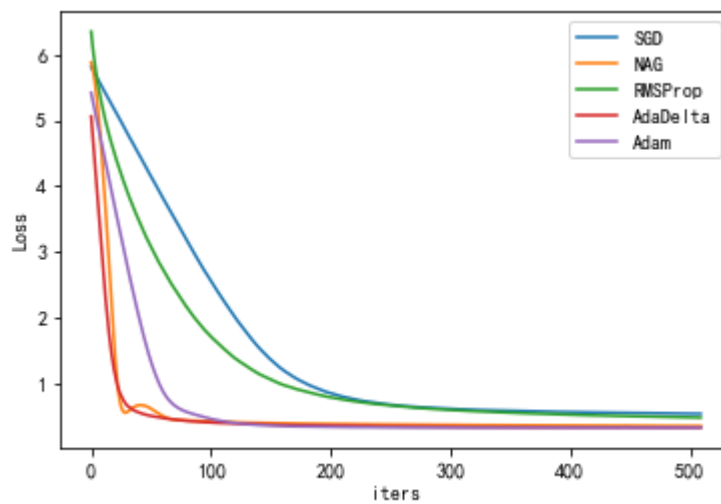
Adam learning_rate = 0.01 beta = 0.8 gamma = 0.98 Epsilon = 10e-6

预测结果（最佳结果）：

逻辑回归：

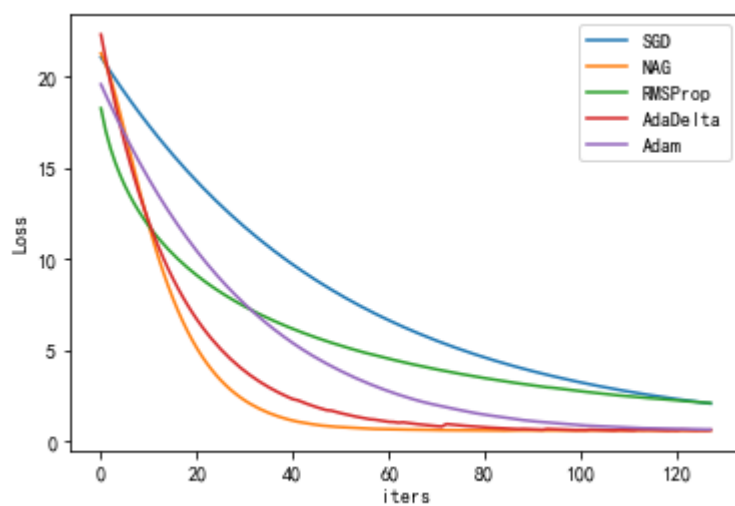
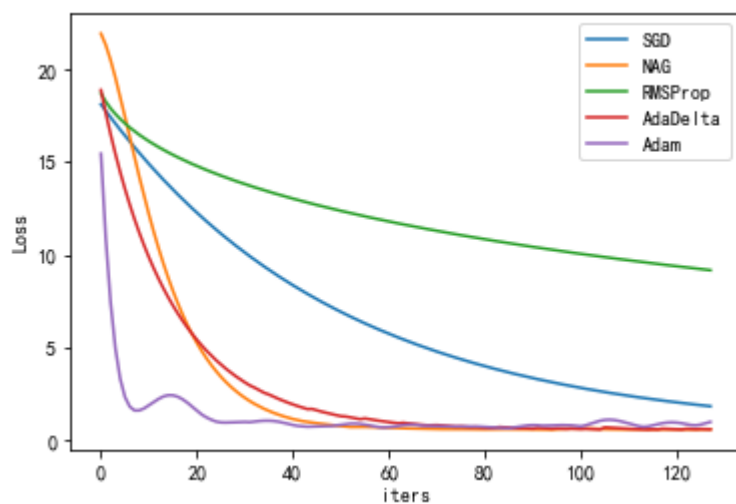


线性分类：

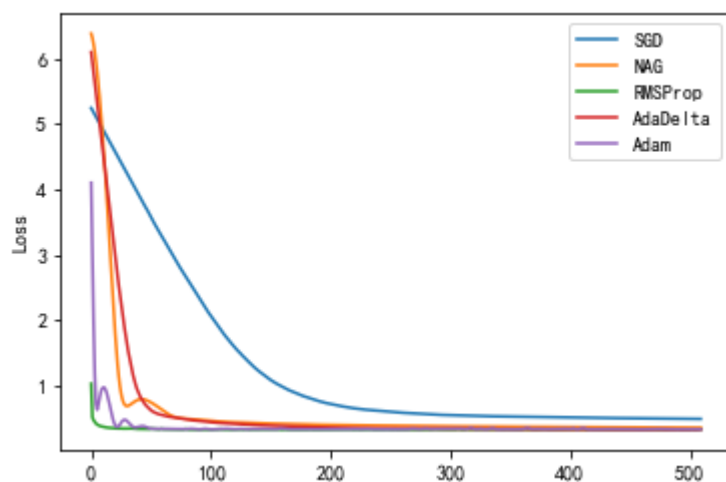


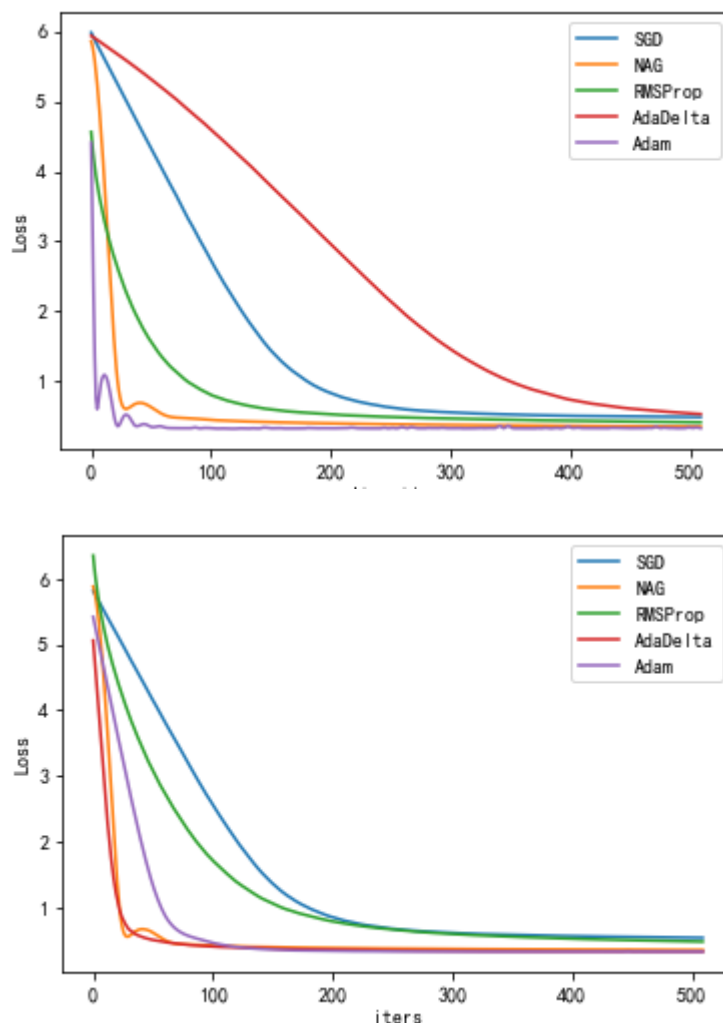
loss 曲线图：

逻辑回归：（包括一些调试参数的）



线性分类：（包括一些调参的曲线）





175. 实验结果分析:

从实验结果的图形上看，单纯的随机梯度下降方法（蓝色线）的下降速度较为缓慢，说明 SGD 效率不够高，同时我发现在调参的时候，`learning_rate` 的选取需要十分小心谨慎。

NAG 是通过动量的引入来预测结果，最终损失较小

RMSProp 相对于 SGD 较快达到平衡。

AdaDelta 无需设置学习率，可以看出经过调参之后它的下降速度达到非常大的值

Adam 利用了 AdaGrad 和 RMSProp 在稀疏数据上的优点。对初始化的偏差的修正也让 Adam 表现的更好。

176. 对比逻辑回归和线性分类的异同点:

逻辑回归的模型 是一个非线性模型，sigmoid 函数，又称逻辑回归函数。但是它本质上又是一个线性回归模型，因为除去

sigmoid 映射函数关系，其他的步骤，算法都是线性回归的。

于是问题有变成了线性回归和线性分类的异同啦。他们本质上都是对于线性回归的拟合，通过对于参数的调整，来减小 loss 的大小。

不同的是，线性回归是连续的预测值，而线性分类是对 label 做一个定性的判断。都可以通过 SGD 以及四种优化方法进行优化。

13.实验总结：

这次实验让我加深了对随机梯度下降、线性分类和逻辑回归的理解，同时也深入理解了 NAG, RMSProp, Adam, AdaDelta 这几种优化方法的了解，之前自己写一些方法，使得运行速度非常慢，通过请教才知道运用已有的方法是很简便的！这次实验也让我认识到了自己的不足，那就是处理问题思路不清晰，对于 python 的库的应用掌握不足，同时本身的打码能力较弱，这让我在本次实验中非常吃力。于是我就请教大神和同学，学习他们的写代码的习惯和方法，她们也非常认真的教了我很多。在此次实验中，我觉得我明白了这几种方法的原理以及是一种进步了，至于实现方法的思路以及具体实现过程，我觉得我必须要多加练习，才能达到要求。