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 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。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，，和。

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

7. 重复步骤 *4-6* 若干次，**画出，，和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）

逻辑回归：

```
# -*- 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



methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]
parm={"C":0.6,\
        "SGD":{"learning rate":0.01},\
        "NAG":{"learning rate":0.008,"Gamma":0.8},\
        "RMSProp":{"learning
rate":0.015,"Gamma":0.8,"Epsilon":10e-8},\
        "AdaDelta":{"Gamma":0.98,"Epsilon":10e-6},\
        "Adam":{"Beta":0.8,"Gamma":0.98,"learning
rate":0.01,"Epsilon":10e-6}}
list = {"NAG":np.zeros([feature_num + 1, 1]),\
              "RMSProp":np.zeros([feature_num+1,1]),\
```

```
"AdaDelta":{"EG":np.zeros([feature_num+1,1]),"EX":np.zeros([feature
_num+1,1])},\

"Adam":{"M":np.zeros([feature_num+1,1]),"G":np.zeros([feature_num
+1,1]),"t":0}}

    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(W,X_train,y_train):
        L_dW = np.zeros((124,1))
```

```
        temp=1-y_train*np.matmul(X_train,W)
        temp=np.maximum(temp/np.abs(temp),0)

        y=y_train*temp
        L_dW=-np.matmul(X_train.T,y)
        return (parm.get("C") * L_dW) + W


    def SGD(W,X_train,y_train):
        W-=parm.get("SGD").get("learning
rate")*gradient(W,X_train,y_train)
        return W


    def NAG(W,X_train,y_train):
        global list
        global parm
        momentum=list.get('NAG')
        #learning_rate=parm.get("NAG").get("learning_rate")
        Gamma=parm.get("NAG").get("Gamma")
        grad=gradient(W-(Gamma*momentum),X_train,y_train)
        update_momentum   =   momentum   *   Gamma+   grad   *
```

```
parm.get("NAG").get("learning rate")
        list["NAG"]=update_momentum
        W-=update_momentum
        return W


    def RMSProp(W,X_train,y_train):
        G=list.get("RMSProp")
        Gamma =parm.get("RMSProp").get("Gamma")
        Epsilon=parm.get("RMSProp").get("Epsilon")
        learning_rate=parm.get("RMSProp").get("learning rate")
        grad=gradient(W,X_train,y_train)
        G=G+(1-Gamma)*grad**2
        list["RMSProp"]=G
        W-=learning_rate*grad/np.sqrt(G+Epsilon)
        return W


    def AdaDelta(W,X_train,y_train):
        EG=list.get("AdaDelta").get("EG")
        EX=list.get("AdaDelta").get("EX")
        Gamma=parm.get("AdaDelta").get("Gamma")
```

```
    Epsilon=parm.get("AdaDelta").get("Epsilon")
    grad=gradient(W,X_train,y_train)
    EG=Gamma*EG+(1-Gamma)*grad**2
    list.get("AdaDelta")["EG"]=EG
    delta=-1*grad*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)
    EX=Gamma*EX+(1-Gamma)*delta**2
    list.get("AdaDelta")["EX"]=EX
    W+=delta
    return W


def Adam(W,X_train,y_train):
    Beta=parm.get("Adam").get("Beta")
    Gamma=parm.get("Adam").get("Gamma")
    Epsilon=parm.get("Adam").get("Epsilon")
    learning_rate=parm.get("Adam").get("learning rate")
    M=list.get("Adam").get("M")
    G=list.get("Adam").get("G")
    t=list.get("Adam").get("t")
    t=t+1
    list.get("Adam")["t"]=t
    grad=gradient(W,X_train,y_train)
```

```
        M=Beta*M+(1-Beta)*grad
        list.get("Adam")["M"]=M
        G=Gamma*G+(1-Gamma)*grad**2
        list.get("Adam")["G"]=G
        M_bias=M/(1-Beta**t)
        G_bias=G/(1-Gamma**t)
        W-=learning_rate*M_bias/(np.sqrt(G_bias)+Epsilon)
        return W


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
```

```python
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=opitimizer(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()

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
from sklearn import preprocessing

from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


#feature_num=123
batch_size=128
SGD_methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]
parm={"SGD":{"learning rate":0.008},\
      "NAG":{"learning rate":0.008,"Gamma":0.92},\
      "RMSProp":{"learning rate":0.008,"Gamma":0.9,"Epsilon":10e-8},\
      "AdaDelta":{"Gamma":0.98,"Epsilon":10e-6},\
      "Adam":{"Beta":0.8,"Gamma":0.85,"learning
rate":0.008,"Epsilon":10e-8}}
list={"NAG":np.zeros([feature_num + 1, 1]),\
         "RMSProp":np.zeros([feature_num+1,1]),\

"AdaDelta":{"EG":np.zeros([feature_num+1,1]),"EX":np.zeros([feature_num+1,1])},
\

"Adam":{"M":np.zeros([feature_num+1,1]),"G":np.zeros([feature_num+1,1]),"t":0}}

#定义 sigmoid 函数
def sigmoid(z):
    return 1/(1+np.exp(-1.0*z))

#定义损失函数
def loss_function(Weight, X,y):
```

```
        l = np.matmul(X, Weight)
        loss = -np.mean(y * np.log(sigmoid(l)) + (1 - y) * np.log(1 - sigmoid(l)))
        return loss

    #求解梯度
    def gradient(Weight,X,y):
        l = np.matmul(X, Weight)
        output = sigmoid(l)
        error = output - y
        grad = np.matmul(X.transpose(), error) / y.shape[0]
        return grad

    def SGD(Weight,X,y):
        W-=parm.get("SGD").get("learning rate")*gradient(Weight,X,y)
        return Weight

    def NAG(Weight,X,y):
        global list
        global parm
        momentum=list.get('NAG')
        Gamma=parm.get("NAG").get("Gamma")
        grad=gradient(Weight-(Gamma*momentum),X,y)
        update_momentum = momentum * Gamma+ grad *
parm.get("NAG").get("learning rate")
        list["NAG"]=update_momentum
        Weight-=update_momentum
        return Weight

    def RMSProp(Weight,X,y):
        G=list.get("RMSProp")
        Gamma =parm.get("RMSProp").get("Gamma")
        Epsilon=parm.get("RMSProp").get("Epsilon")
        learning_rate=parm.get("RMSProp").get("learning rate")
        grad=gradient(W,X_train,y_train)
        G=G+(1-Gamma)*grad**2
        list["RMSProp"]=G
        Weight -= learning_rate*grad/np.sqrt(G+Epsilon)
        return Weight

    def AdaDelta(Weight,X,y):
        EG=list.get("AdaDelta").get("EG")
        EX=list.get("AdaDelta").get("EX")
        Gamma=parm.get("AdaDelta").get("Gamma")
        Epsilon=parm.get("AdaDelta").get("Epsilon")
```

```
        grad=gradient(Weight,X,y)
        EG=Gamma*EG+(1-Gamma)*grad**2
        list.get("AdaDelta")["EG"]=EG
        delta=-1*grad*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)
        EX=Gamma*EX+(1-Gamma)*delta**2
        list.get("AdaDelta")["EX"]=EX
        Weight+=delta
        return Weight

    def Adam(Weight,X,y):
        Beta=parm.get("Adam").get("Beta")
        Gamma=parm.get("Adam").get("Gamma")
        Epsilon=parm.get("Adam").get("Epsilon")
        learning_rate=parm.get("Adam").get("learning rate")
        M=list.get("Adam").get("M")
        G=list.get("Adam").get("G")
        t=list.get("Adam").get("t")
        t=t+1
        list.get("Adam")["t"]=t
        grad=gradient(Weight,X,y)
        M=Beta*M+(1-Beta)*grad
        list.get("Adam")["M"]=M
        G=Gamma*G+(1-Gamma)*grad**2
        list.get("Adam")["G"]=G
        M_bias=M/(1-Beta**t)
        G_bias=G/(1-Gamma**t)
        Weight -= learning_rate*M_bias/(np.sqrt(G_bias)+Epsilon)
        return Weight

    def opitimizer(Weight,X,y,method):
        if method=="SGD":
            return SGD(Weight,X,y)
        if method=="NAG":
            return NAG(Weight,X,y)
        if method=="RMSProp":
            return RMSProp(Weight,X,y)
        if method=="AdaDelta":
            return AdaDelta(Weight,X,y)
        if method=="Adam":
            return Adam(Weight,X,y)

    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 LogicReg():
        X_train, y_train, X_test, y_test, data_size, features_num = getdata()
        plt.xlabel('iters')
        plt.ylabel('Loss')

        for method in SGD_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=opitimizer(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()

LogicReg()
```

8. 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。
声明方法采用列表形式（是参考别人的格式，但是自己调参）
模型参数见 10 中的超参数选择。

9. 选择的 loss 函数及其导数:

逻辑回归；

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

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

导数：

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

线性分类：

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

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：c=0.6 learning_rate = 0.01

NAG: c=0.6 learning_rate =0.008 Gamma=0.8

RMSProp c=0.6 learning_rate =0.015 Gamma=0.8 Epsilon = 10e-8

AdaDelta c=0.6 Gamma=0.98 Epsilon = 10e-7

Adam c=0.6 learning_rate =0.01 beta=0.8 Gamma=0.98 Epsilon = 10e-7

线性分类：

SDG： learning_rate = 0.008

NAG: learning_rate =0.008 Gamma=0.92

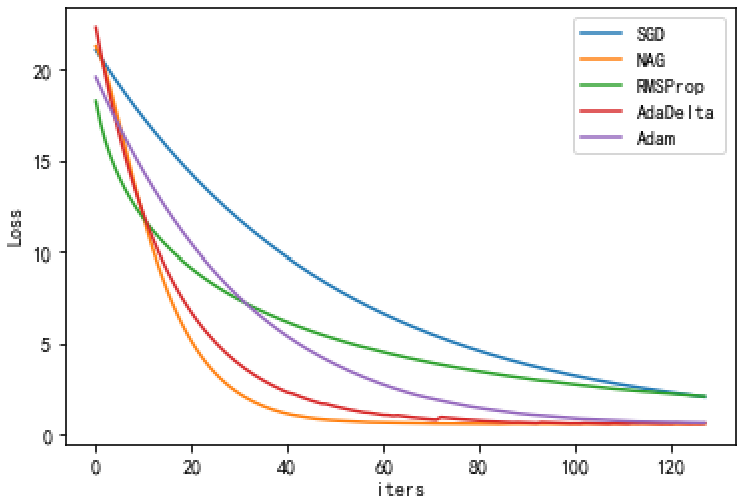
RMSProp learning_rate =0.008 Gamma=0.9 Epsilon = 10e-8

AdaDelta Gamma=0.98 Epsilon = 10e-6

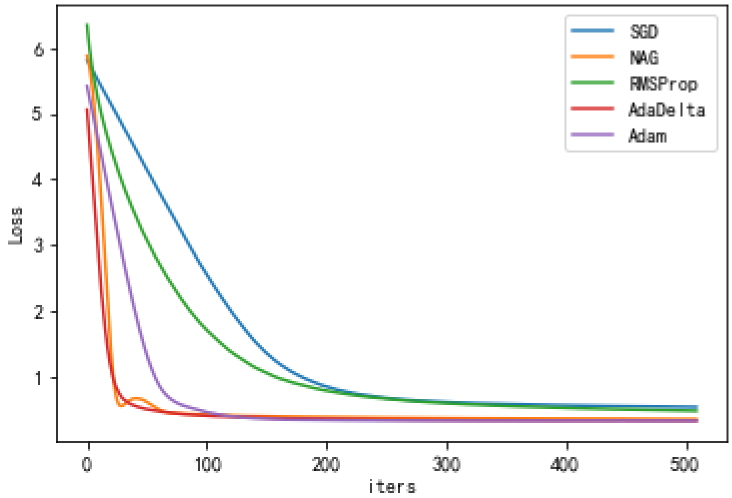
Adam learning_rate =0.008 beta=0.8 Gamma=0.85 Epsilon = 10e-8

预测结果（最佳结果）：

逻辑回归：

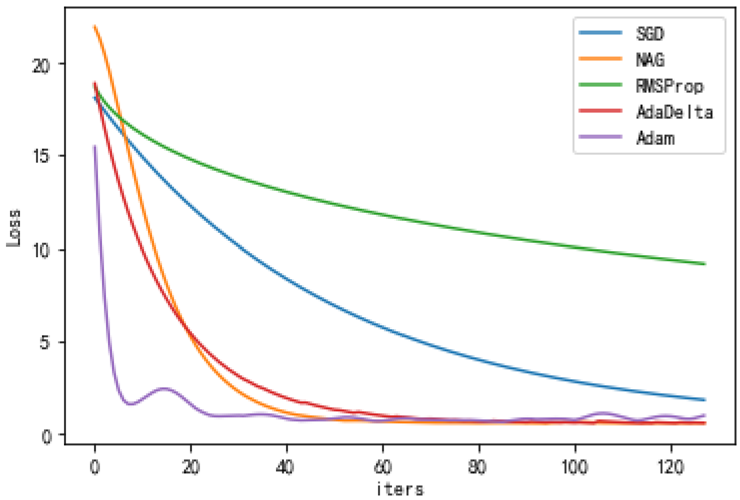


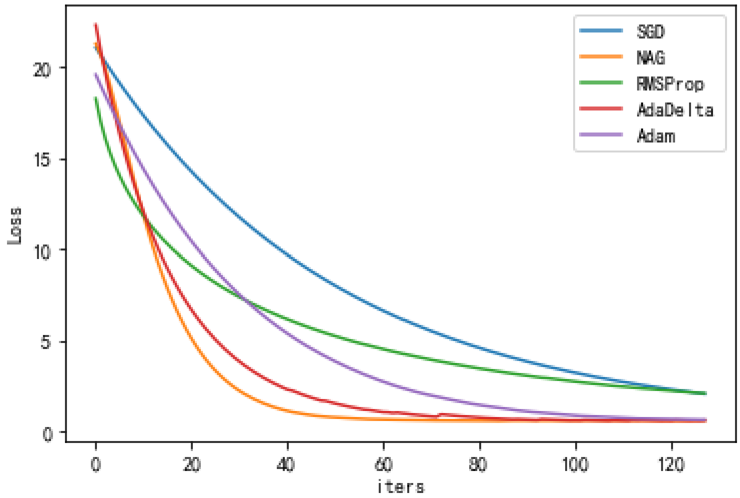
线性分类：



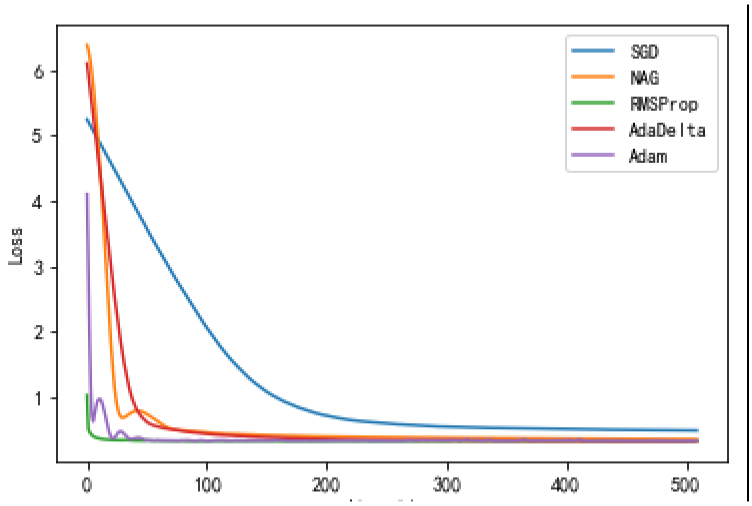
loss 曲线图：

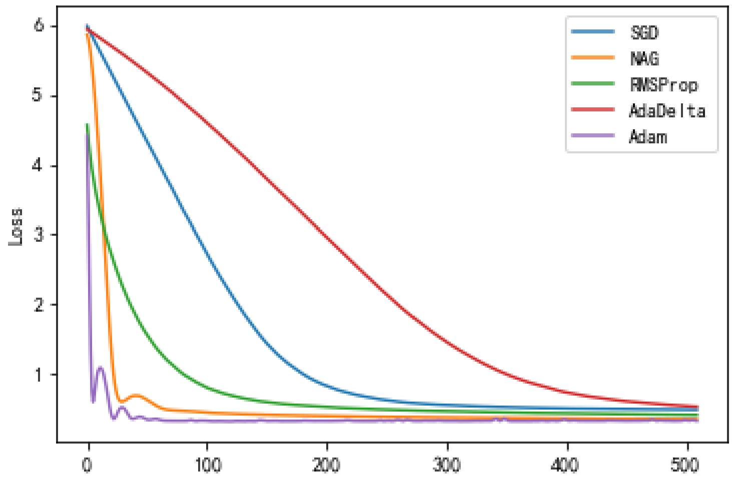
逻辑回归：（包括一些调试参数的）

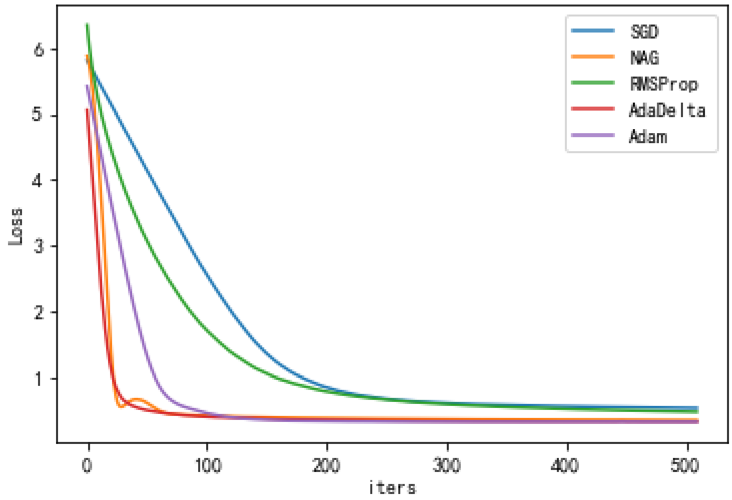




线性分类：（包括一些调参的曲线）







10. 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。

11. 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。