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　陈安妮

学　　号　201530611142

邮　　箱　1412209084@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 08 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 02 日

3. 报告人:陈安妮

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，样本值为{-1，+1}。a9a 为训练集，a9a.t 为验证集。本次实验使用 sklearn 库的 load_svmlight_file 函数读取数据。

6. 实验步骤:

6.1 逻辑回归与随机梯度下降

1. 使用 sklearn 库的 load_svmlight_file 函数分别读取实验训练集 a9a 和验证集 a9a.t。
2.逻辑回归模型参数全零初始化。
3.选择 Loss 函数及对其求导,求得部分样本对 Loss 函数的梯度 G。
4.使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta 和 Adam）。
5.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 L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$ 和 L_{Adam}。

6.重复步骤 4-6 若干次，画出 L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$ 和 L_{Adam} 随迭代次数的变化图。

6.2 线性分类与随机梯度下降

1. 使用 sklearn 库的 load_svmlight_file 函数分别读取实验训练集 a9a 和验证集 a9a.t。
2.逻辑回归模型参数全零初始化。
3.选择 Loss 函数及对其求导,求得部分样本对 Loss 函数的梯度 G。
4.使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta 和 Adam）。
5.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 L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$ 和 L_{Adam}。

6.重复步骤 4-6 若干次，画出 L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$ 和 L_{Adam} 随迭代次数的变化图。

7. 代码内容:

7.1 逻辑回归与随机梯度下降

RegressionExperiment.ipynb

```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix, hstack

#获取文件数据
def get_data(file, n_features=None):
    if n_features==None:
        inputs,labels = datasets.load_svmlight_file(file)
    else:
        inputs,labels = datasets.load_svmlight_file(file,n_features=n_features)

    one = np.ones((inputs.shape[0],1))
    inputs = hstack([inputs, csr_matrix(one)]).toarray()
    labels = np.array(labels).reshape(inputs.shape[0],1)
    labels[labels==-1] = 0
    return inputs, labels

#the sigmoid function
def sigmoid(x):
    return 1.0 / (1 + np.exp(-x))

#标记
def sign(x,threshold):
    if x>=threshold:
        return 1
    else:
        return 0

# NAG
def
NAG(X_train,Y_train,X_test,Y_test,gamma=0.9,threshold=0.5,learning_rate=0.1,batch_size=64,e
poch=500):
    gradients = np.zeros((X_train.shape[1],1))
    w = np.random.rand(X_train.shape[1], 1)
    momentum = np.zeros((X_train.shape[1], 1))
```

```
        epoch_set = []
        l_nag = []
        a_nag = []
        #training
        for i in range(epoch+1):
            batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
            for j in range(batch_step,batch_step+batch_size):
                gradients = gradients+((sigmoid(np.dot(X_train[j],(w-gamma*momentum)))-Y_train[j])*X_train[j]).reshape((X_train.shape[1],1))
            gradients = gradients/batch_size
            momentum=gamma*momentum+learning_rate*gradients
            w = w - momentum
            epoch_set.append(i)
            loss = 0.0
            res = 0
            for k in range(X_test.shape[0]):
                h = sigmoid(np.dot(X_test[k],(w-gamma*momentum)))
                loss =loss+(Y_test[k]*np.log(h)+(1-Y_test[k])*np.log(1-h))
                if sign(h,threshold) == Y_test[k]:
                    res=res+1
            accuracy = res/X_test.shape[0]
            loss =-loss/X_test.shape[0]
            a_nag.append(accuracy)
            l_nag.append(loss)
        return a_nag,l_nag,epoch_set

    #RMSProp
    def RMSProp(X_train,Y_train,X_test,Y_test,gamma=0.9,epsilon=1e-10,threshold=0.5,learning_rate=0.1,batch_size=64,epoch=500):
        gradients = np.zeros((X_train.shape[1],1))
        w = np.random.rand(X_train.shape[1], 1)
        G = np.zeros((X_train.shape[1],1))
        epoch_set = []
        l_rmsp = []
        a_rmsp = []
        # training
        for i in range(epoch+1):
            batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
            for j in range(batch_step,batch_step+batch_size):
                gradients = gradients+((sigmoid(np.dot(X_train[j],w))-Y_train[j])*X_train[j]).reshape((X_train.shape[1],1))
            gradients = gradients/batch_size
```

```
            G = gamma*G+np.multiply((1-gamma)*gradients,gradients)
            w = w-np.multiply(learning_rate/(np.sqrt(G+epsilon)),gradients)
            epoch_set.append(i)
            loss=0.0
            res=0
            for k in range(X_test.shape[0]):
                h = sigmoid(np.dot(X_test[k],w))
                loss = loss+(Y_test[k]*np.log(h)+(1-Y_test[k])*np.log(1-h))
                if sign(h,threshold)==Y_test[k]:
                    res = res+1
            accuracy = res/X_test.shape[0]
            loss = -loss/X_test.shape[0]
            a_rmsp.append(accuracy)
            l_rmsp.append(loss)
        return a_rmsp,l_rmsp

    #AdaDelta
    def
AdaDelta(X_train,Y_train,X_test,Y_test,gamma=0.9,epsilon=1e-10,threshold=0.5,dx=0.001,batch
_size=64,epoch=500):
        gradients = np.zeros((X_train.shape[1],1))
        w = np.random.rand(X_train.shape[1], 1)
        G = np.zeros((X_train.shape[1],1))
        epoch_set = []
        l_adad = []
        a_adad = []
        # training
        for i in range(epoch+1):
            batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
            for j in range(batch_step,batch_step+batch_size):
                gradients                                                                    =
gradients+((sigmoid(np.dot(X_train[j],w))-Y_train[j])*X_train[j]).reshape((X_train.shape[1],1))
            gradients = gradients/batch_size
            G=gamma*G+np.multiply((1-gamma)*gradients,gradients)
            dw = -np.multiply((np.sqrt(dx+epsilon))/(np.sqrt(G+epsilon)),gradients)
            w = w+dw
            dx=gamma*dx+np.multiply((1-gamma)*dw,dw)
            epoch_set.append(i)
            loss=0.0
            res=0
            for k in range(X_test.shape[0]):
                h = sigmoid(np.dot(X_test[k],w))
                loss = loss+(Y_test[k]*np.log(h)+(1-Y_test[k])*np.log(1-h))
                if sign(h,threshold)==Y_test[k]:
```

```
                    res = res+1
            accuracy = res/X_test.shape[0]
            loss= -loss/X_test.shape[0]
            a_adad.append(accuracy)
            l_adad.append(loss)
        return a_adad,l_adad

    #Adam
    def
Adam(X_train,Y_train,X_test,Y_test,beta=0.9,gamma=0.999,epsilon=1e-8,threshold=0.5,learning
_rate=0.01,batch_size=64,epoch=500):
        gradients = np.zeros((X_train.shape[1],1))
        w = np.random.rand(X_train.shape[1], 1)
        G = np.zeros((X_train.shape[1],1))
        moments = np.zeros((X_train.shape[1], 1))
        epoch_set = []
        l_adam = []
        a_adam = []
        # training
        for i in range(epoch+1):
            batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
            for j in range(batch_step,batch_step+batch_size):
                gradients                                                                    =
gradients+((sigmoid(np.dot(X_train[j],w))-Y_train[j])*X_train[j]).reshape((X_train.shape[1],1))
            gradients = gradients/batch_size
            moments = beta*moments+(1-beta)*gradients
            G = gamma*G+np.multiply((1-gamma)*gradients,gradients)
            #alpha = learning_rate*(np.sqrt(1-gamma))/(1-beta)
            alpha = learning_rate
            w = w-alpha*moments/(np.sqrt(G+epsilon))
            epoch_set.append(i)
            loss=0.0
            res=0
            for k in range(X_test.shape[0]):
                h = sigmoid(np.dot(X_test[k],w))
                loss = loss+(Y_test[k]*np.log(h)+(1-Y_test[k])*np.log(1-h))
                if sign(h,threshold)==Y_test[k]:
                    res = res+1
            accuracy = res/X_test.shape[0]
            loss = -loss/X_test.shape[0]
            a_adam.append(accuracy)
            l_adam.append(loss)
        return a_adam,l_adam
```

```
# 画图
def plot_loss(epoch_set,l_nag,l_rmsp,l_adad,l_adam):
    plt.figure(figsize=(10,15))
    plt.subplot(211)
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.plot(epoch_set, l_nag,color='red',label='NAG',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, l_rmsp,color='blue',label='RMSProp',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, l_adad,color='green',label='AdaDelta',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, l_adam,color='yellow',label='Adam',linewidth=1.5,linestyle='-')
    plt.legend(loc='upper right')
    plt.show()

def plot_accuracy(epoch_set,a_nag,a_rmsp,a_adad,a_adam):
    plt.figure(figsize=(10,15))
    plt.subplot(211)
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.plot(epoch_set, a_nag,color='red',label='NAG',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_rmsp,color='blue',label='RMSProp',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_adad,color='green',label='AdaDelta',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_adam,color='yellow',label='Adam',linewidth=1.5,linestyle='-')
    plt.legend(loc='upper right')
    plt.show()

if __name__ == "__main__":
    x_train,y_train = get_data('a9a')
    x_test,y_test = get_data('a9a.t',x_train.shape[1]-1)
    a_nag,l_nag,epoch_set = NAG(x_train,y_train,x_test,y_test)
    a_rmsp,l_rmsp = RMSProp(x_train,y_train,x_test,y_test)
    a_adad,l_adad = AdaDelta(x_train,y_train,x_test,y_test)
    a_adam,l_adam = Adam(x_train,y_train,x_test,y_test)

    plot_loss(epoch_set,l_nag,l_rmsp,l_adad,l_adam)
    plot_accuracy(epoch_set,a_nag,a_rmsp,a_adad,a_adam)
```

7.2 线性分类与随机梯度下降

ClassificationExperiment.ipynb

```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

```
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix, hstack

#获取文件数据
def get_data(file, n_features=None):
    if n_features==None:
        inputs,labels = datasets.load_svmlight_file(file)
    else:
        inputs,labels = datasets.load_svmlight_file(file,n_features=n_features)

    one = np.ones((inputs.shape[0],1))
    inputs = hstack([inputs, csr_matrix(one)]).toarray()
    labels = np.array(labels).reshape(inputs.shape[0],1)
    return inputs, labels

#标记
def sign(x,threshold):
    if x > threshold:
        return 1
    elif x < threshold:
        return -1
    else:
        return 0

# NAG
def
NAG(X_train,Y_train,X_test,Y_test,gamma=0.9,c=0.01,threshold=0,learning_rate=0.01,batch_siz
e=64,epoch=500):
    gradients = np.zeros((X_train.shape[1],1))
    w = np.random.rand(X_train.shape[1], 1)
    momentum = np.zeros((X_train.shape[1], 1))
    epoch_set = []
    l_nag = []
    a_nag = []
    #training
    for i in range(epoch+1):
        batch_step = random.randint(0,X_train.shape[0]-batch_size-1)
        for j in range(batch_step,batch_step+batch_size):
            theta = 1-Y_train[j]*np.dot(X_train[j],(w-gamma*momentum))
            if theta < 0:
                gradients = gradients+(w-gamma*momentum)
            else:
                gradients                                                    =
```

```
gradients+w-gamma*momentum-c*(Y_train[j]*X_train[j]).reshape((X_train.shape[1],1))
                gradients = gradients/batch_size
                momentum = gamma*momentum+learning_rate*gradients
                w = w - momentum
                epoch_set.append(i)
                hinge_loss = 0.0
                res = 0
                for k in range(X_test.shape[0]):
                    Y_prediction = np.dot(X_test[k] ,(w-gamma*momentum))
                    theta = max((1.0 - Y_test[k]*Y_prediction),0)
                    hinge_loss = hinge_loss+theta
                    if sign(Y_prediction,threshold) == Y_test[k]:
                        res = res+1
                accuracy = res/X_test.shape[0]
                loss = np.sum(np.square(w-gamma*momentum))/2+c*hinge_loss/X_test.shape[0]
                a_nag.append(accuracy)
                l_nag.append(loss)
            return a_nag,l_nag,epoch_set


        #RMSProp
        def
RMSProp(X_train,Y_train,X_test,Y_test,gamma=0.9,c=0.001,epsilon=1e-10,threshold=0,learning
_rate=0.1,batch_size=64,epoch=500):
            gradients = np.zeros((X_train.shape[1],1))
            w = np.random.rand(X_train.shape[1], 1)
            G = np.zeros((X_train.shape[1],1))
            epoch_set = []
            l_rmsp = []
            a_rmsp = []
            # training
            for i in range(epoch+1):
                batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
                for j in range(batch_step,batch_step+batch_size):
                    theta = 1-Y_train[j]*np.dot(X_train[j],w)
                    if theta < 0:
                        gradients = gradients+w
                    else:
                        gradients                                                    =
gradients+w-c*(Y_train[j]*X_train[j]).reshape((X_train.shape[1],1))
                gradients = gradients/batch_size
                G = gamma*G+np.multiply((1-gamma)*gradients,gradients)
                w = w-np.multiply(learning_rate/(np.sqrt(G+epsilon)),gradients)
                epoch_set.append(i)
```

```
                hinge_loss=0.0
                res=0
                for k in range(X_test.shape[0]):
                    Y_prediction = np.dot(X_test[k] ,w)
                    theta = max((1.0 - Y_test[k]*Y_prediction),0)
                    hinge_loss = hinge_loss+theta
                    if sign(Y_prediction,threshold) == Y_test[k]:
                        res = res+1
                accuracy = res/X_test.shape[0]
                loss = np.sum(np.square(w))/2+c*hinge_loss/X_test.shape[0]
                a_rmsp.append(accuracy)
                l_rmsp.append(loss)
            return a_rmsp,l_rmsp

        #AdaDelta
        def
AdaDelta(X_train,Y_train,X_test,Y_test,gamma=0.9,c=0.01,epsilon=1e-10,threshold=0,dx=0.001
,batch_size=64,epoch=500):
            gradients = np.zeros((X_train.shape[1],1))
            w = np.random.rand(X_train.shape[1], 1)
            G = np.zeros((X_train.shape[1],1))
            epoch_set = []
            l_adad = []
            a_adad = []
            # training
            for i in range(epoch+1):
                batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
                for j in range(batch_step,batch_step+batch_size):
                    theta = 1-Y_train[j]*np.dot(X_train[j],w)
                    if theta < 0:
                        gradients = gradients+w
                    else:
                        gradients                                                    =
gradients+w-c*(Y_train[j]*X_train[j]).reshape((X_train.shape[1],1))
                gradients = gradients/batch_size
                G=gamma*G+np.multiply((1-gamma)*gradients,gradients)
                dw = -np.multiply((np.sqrt(dx+epsilon))/(np.sqrt(G+epsilon)),gradients)
                w = w+dw
                dx=gamma*dx+np.multiply((1-gamma)*dw,dw)
                epoch_set.append(i)
                hinge_loss=0.0
                res=0
                for k in range(X_test.shape[0]):
                    Y_prediction = np.dot(X_test[k] ,w)
```

```
                    theta = max((1.0 - Y_test[k]*Y_prediction),0)
                    hinge_loss = hinge_loss+theta
                    if sign(Y_prediction,threshold) == Y_test[k]:
                        res = res+1
                accuracy = res/X_test.shape[0]
                loss = np.sum(np.square(w))/2+c*hinge_loss/X_test.shape[0]
                a_adad.append(accuracy)
                l_adad.append(loss)
            return a_adad,l_adad

        #Adam
        def
Adam(X_train,Y_train,X_test,Y_test,beta=0.9,gamma=0.999,c=0.01,epsilon=1e-8,threshold=0,lea
rning_rate=0.01,batch_size=64,epoch=500):
            gradients = np.zeros((X_train.shape[1],1))
            w = np.random.rand(X_train.shape[1], 1)
            G = np.zeros((X_train.shape[1],1))
            moments = np.zeros((X_train.shape[1], 1))
            epoch_set = []
            l_adam = []
            a_adam = []
            # training
            for i in range(epoch+1):
                batch_step=random.randint(0,X_train.shape[0]-batch_size-1)
                for j in range(batch_step,batch_step+batch_size):
                    theta = 1-Y_train[j]*np.dot(X_train[j],w)
                    if theta < 0:
                        gradients = gradients+w
                    else:
                        gradients                                                    =
gradients+w-c*(Y_train[j]*X_train[j]).reshape((X_train.shape[1],1))
                gradients = gradients/batch_size
                moments = beta*moments+(1-beta)*gradients
                G = gamma*G+np.multiply((1-gamma)*gradients,gradients)
                #alpha = learning_rate*(np.sqrt(1-gamma))/(1-beta)
                alpha = learning_rate
                w = w-alpha*moments/(np.sqrt(G+epsilon))
                epoch_set.append(i)
                hinge_loss=0.0
                res=0
                for k in range(X_test.shape[0]):
                    Y_prediction = np.dot(X_test[k] ,w)
                    theta = max((1.0 - Y_test[k]*Y_prediction),0)
                    hinge_loss = hinge_loss+theta
```

```
                if sign(Y_prediction,threshold) == Y_test[k]:
                    res = res+1
            accuracy = res/X_test.shape[0]
            loss = np.sum(np.square(w))/2+c*hinge_loss/X_test.shape[0]
            a_adam.append(accuracy)
            l_adam.append(loss)
        return a_adam,l_adam


# 画图
def plot_loss(epoch_set,l_nag,l_rmsp,l_adad,l_adam):
    plt.figure(figsize=(10,15))
    plt.subplot(211)
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.plot(epoch_set,l_nag,color='red',label='NAG',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set,l_rmsp,color='blue',label='RMSProp',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set,l_adad,color='green',label='AdaDelta',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set,l_adam,color='yellow',label='Adam',linewidth=1.5,linestyle='-')
    plt.legend(loc='upper right')
    plt.show()

def plot_accuracy(epoch_set,a_nag,a_rmsp,a_adad,a_adam):
    plt.figure(figsize=(10,15))
    plt.subplot(211)
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.plot(epoch_set, a_nag,color='red',label='NAG',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_rmsp,color='blue',label='RMSProp',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_adad,color='green',label='AdaDelta',linewidth=1.5,linestyle='-')
    plt.plot(epoch_set, a_adam,color='yellow',label='Adam',linewidth=1.5,linestyle='-')
    plt.legend(loc='upper right')
    plt.show()

if __name__ == "__main__":
    x_train,y_train = get_data('a9a')
    x_test,y_test = get_data('a9a.t',x_train.shape[1]-1)
    a_nag,l_nag,epoch_set = NAG(x_train,y_train,x_test,y_test)
    a_rmsp,l_rmsp = RMSProp(x_train,y_train,x_test,y_test)
    a_adad,l_adad = AdaDelta(x_train,y_train,x_test,y_test)
    a_adam,l_adam = Adam(x_train,y_train,x_test,y_test)

    plot_loss(epoch_set,l_nag,l_rmsp,l_adad,l_adam)
    plot_accuracy(epoch_set,a_nag,a_rmsp,a_adad,a_adam)
```

8. 模型参数的初始化方法:

8.1 逻辑回归与随机梯度下降

8.1.1 NAG

gradients 和 momentum 采用全零初始化，w 采用随机初始化

8.1.2 RMSProp

gradients 和 G 采用全零初始化，w 采用随机初始化

8.1.3 AdaDelta

gradients 和 G 采用全零初始化，w 采用随机初始化

8.1.4 Adam

Gradients、momentum 和 G 采用全零初始化，w 采用随机初始化

8.2 线性分类与随机梯度下降

8.2.1 NAG

gradients 和 momentum 采用全零初始化，w 采用随机初始化

8.2.2 RMSProp

gradients 和 G 采用全零初始化，w 采用随机初始化

8.2.3 AdaDelta

gradients 和 G 采用全零初始化，w 采用随机初始化

8.2.4 Adam

Gradients、momentum 和 G 采用全零初始化，w 采用随机初始化

9.选择的 loss 函数及其导数:

9.1 逻辑回归与随机梯度下降

9.1.1 NAG

Loss: $J(\mathbf{w}-\gamma\mathbf{m}) = -\frac{1}{n}[\sum_{i=1}^{n} y^{(i)} log\ h_{\mathbf{w}}(x^{(i)}) + (1-y^{(i)}) \cdot log(1-h_{\mathbf{w}}(x^{(i)}))] \quad s.t. h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}-\gamma\mathbf{m})T_{\mathbf{x}}}}$

Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}-\gamma\mathbf{m}) = \frac{1}{n}\sum_{i=1}^{n}(h_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update $\mathbf{m}$: $\mathbf{m} \leftarrow \gamma\mathbf{m} + \alpha\nabla_{\mathbf{w}} J(\mathbf{w}-\gamma\mathbf{m})$

Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$

9.1.2 RMSProp

Loss: $J(\mathbf{w}) = -\frac{1}{n}[\sum_{i=1}^{n} y^{(i)} log\ h_{\mathbf{w}}(x^{(i)}) + (1-y^{(i)}) \cdot log(1-h_{\mathbf{w}}(x^{(i)}))] \quad s.t. h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}+\mathbf{b}}}$

Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma\mathbf{G} + (1-\gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{\sqrt{\mathbf{G}+\epsilon}} \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

9.1.3 AdaDelta

Loss: $J(\mathbf{w}) = -\frac{1}{n}[\sum_{i=1}^{n} y^{(i)} log\ h_{\mathbf{w}}(x^{(i)}) + (1-y^{(i)}) \cdot log(1-h_{\mathbf{w}}(x^{(i)}))] \quad s.t. h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}+\mathbf{b}}}$

Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma\mathbf{G} + (1-\gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

Update $\mathbf{dw}$: $\mathbf{dw} \leftarrow -\frac{\sqrt{dx+\epsilon}}{\sqrt{G+\epsilon}} \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{dw}$

Update $\mathbf{dx}$: $\mathbf{dx} \leftarrow \gamma\mathbf{dx} + (1-\gamma) \cdot \mathbf{dw} \odot \mathbf{dw}$

9.1.4 Adam

Loss: $J(\mathbf{w}) = -\frac{1}{n}[\sum_{i=1}^{n} y^{(i)} log\ h_{\mathbf{w}}(x^{(i)}) + (1-y^{(i)}) \cdot log(1-h_{\mathbf{w}}(x^{(i)}))] \quad s.t.\ h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}+\mathbf{b}}}$
Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$
Update $\mathbf{m}$: $\mathbf{m} \leftarrow \beta \cdot \mathbf{m} + (1-\beta) \cdot \nabla_{\mathbf{w}} J(\mathbf{w})$
Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma \mathbf{G} + (1-\gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$
Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{\mathbf{m}}{\sqrt{\mathbf{G}+\epsilon}}$

9.2 线性分类与随机梯度下降

9.2.1 NAG

Loss: $J(\mathbf{w}-\gamma\mathbf{m}) = min_{\mathbf{w},b} \frac{\|\mathbf{w}-\gamma\mathbf{m}\|^2}{2} + \lambda\sum_{i=1}^{n}\xi_i \quad s.t.\ \xi_i = max(0, 1-y_i(x_i(\mathbf{w}-\gamma\mathbf{m})+b))$
Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}-\gamma\mathbf{m}) = \begin{cases} \mathbf{w}-\gamma\mathbf{m}+\lambda\cdot 0 & \text{if } 1-\xi_i \le 0 \\ \mathbf{w}-\gamma\mathbf{m}-\lambda(y_i x_i) & \text{if } 1-\xi_i > 0 \end{cases}$
Update $\mathbf{m}$: $\mathbf{m} \leftarrow \gamma\mathbf{m} + \alpha\nabla_{\mathbf{w}} J(\mathbf{w}-\gamma\mathbf{m})$
Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$

9.2.2 RMSProp

Loss: $J(\mathbf{w}) = min_{\mathbf{w},b} \frac{\|\mathbf{w}\|^2}{2} + \lambda\sum_{i=1}^{n}\xi_i \quad s.t.\ \xi_i = max(0, 1-y_i(x_i\mathbf{w}+b))$
Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{cases} \mathbf{w}+\lambda\cdot 0 & \text{if } 1-\xi_i \le 0 \\ \mathbf{w}-\lambda(y_i x_i) & \text{if } 1-\xi_i > 0 \end{cases}$
Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma\mathbf{G} + (1-\gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$
Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{\sqrt{\mathbf{G}+\epsilon}} \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

9.2.3 AdaDelta

Loss: $J(\mathbf{w}) = min_{\mathbf{w},b} \frac{\|\mathbf{w}\|^2}{2} + \lambda\sum_{i=1}^{n}\xi_i \quad s.t.\ \xi_i = max(0, 1-y_i(x_i\mathbf{w}+b))$
Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{cases} \mathbf{w}+\lambda\cdot 0 & \text{if } 1-\xi_i \le 0 \\ \mathbf{w}-\lambda(y_i x_i) & \text{if } 1-\xi_i > 0 \end{cases}$
Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma\mathbf{G} + (1-\gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$
Update $\mathbf{dw}$: $\mathbf{dw} \leftarrow -\frac{\sqrt{dx+\epsilon}}{\sqrt{G+\epsilon}} \odot \nabla_{\mathbf{w}} J(\mathbf{w})$
Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{dw}$
Update $\mathbf{dx}$: $\mathbf{dx} \leftarrow \gamma\mathbf{dx} + (1-\gamma) \cdot \mathbf{dw} \odot \mathbf{dw}$

9.2.4 Adam

Loss: $J(\mathbf{w}) = min_{\mathbf{w},b} \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_{i=1}^{n} \xi_i \quad s.t.\ \xi_i = max(0, 1 - y_i(x_i\mathbf{w} + b))$

Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{cases} \mathbf{w} + \lambda \cdot 0 & \text{if } 1\text{- } \xi_i \leq 0 \\ \mathbf{w} - \lambda(y_i x_i) & \text{if } 1\text{- } \xi_i > 0 \end{cases}$

Update $\mathbf{m}$: $\mathbf{m} \leftarrow \beta \cdot \mathbf{m} + (1 - \beta) \cdot \nabla_{\mathbf{w}} J(\mathbf{w})$

Update $\mathbf{G}$: $\mathbf{G} \leftarrow \gamma \mathbf{G} + (1 - \gamma) \cdot \nabla_{\mathbf{w}} J(\mathbf{w}) \odot \nabla_{\mathbf{w}} J(\mathbf{w})$

Update $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{\mathbf{m}}{\sqrt{\mathbf{G}+\epsilon}}$

10.实验结果和曲线图:

10.1 逻辑回归与随机梯度下降

10.1.1 超参数选择：

10.1.1.1 NAG

gamma=0.9,threshold=0.5,learning_rate=0.1,batch_size=64,epoch=500

10.1.1.2 RMSProp

gamma=0.9,epsilon=1e-10,threshold=0.5,learning_rate=0.1,batch_size=64,epoch=500

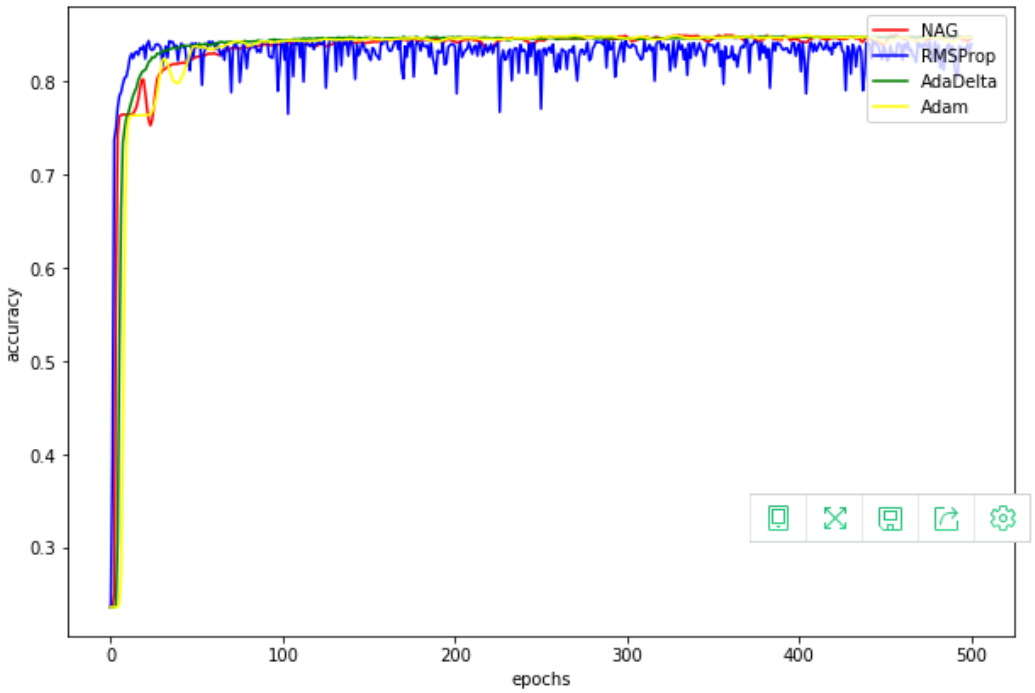
10.1.1.3 AdaDelta

gamma=0.9,epsilon=1e-10,threshold=0.5,dx=0.001,batch_size=64,epoch=500

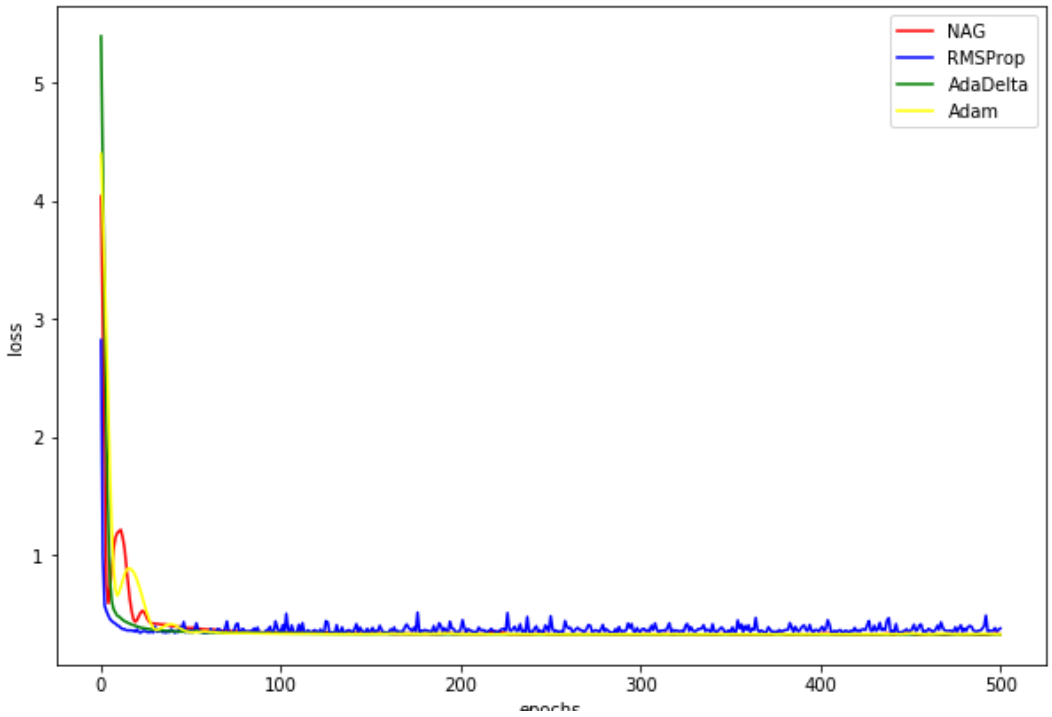
10.1.1.4 Adam

beta=0.9,gamma=0.999,epsilon=1e-8,threshold=0.5,learning_rate=0.01,
batch_size=64,epoch=500

10.1.2 预测结果（最佳结果）：



10.1.3loss 曲线图：



10.2 线性分类与随机梯度下降

10.2.1 超参数选择：

10.2.1.1 NAG

gamma=0.9,c=0.01,threshold=0,learning_rate=0.01,batch_size=64,epoch=500

10.2.1.2 RMSProp

gamma=0.9,c=0.01,epsilon=1e-10,threshold=0,learning_rate=0.1,batch_size=64,
epoch=500

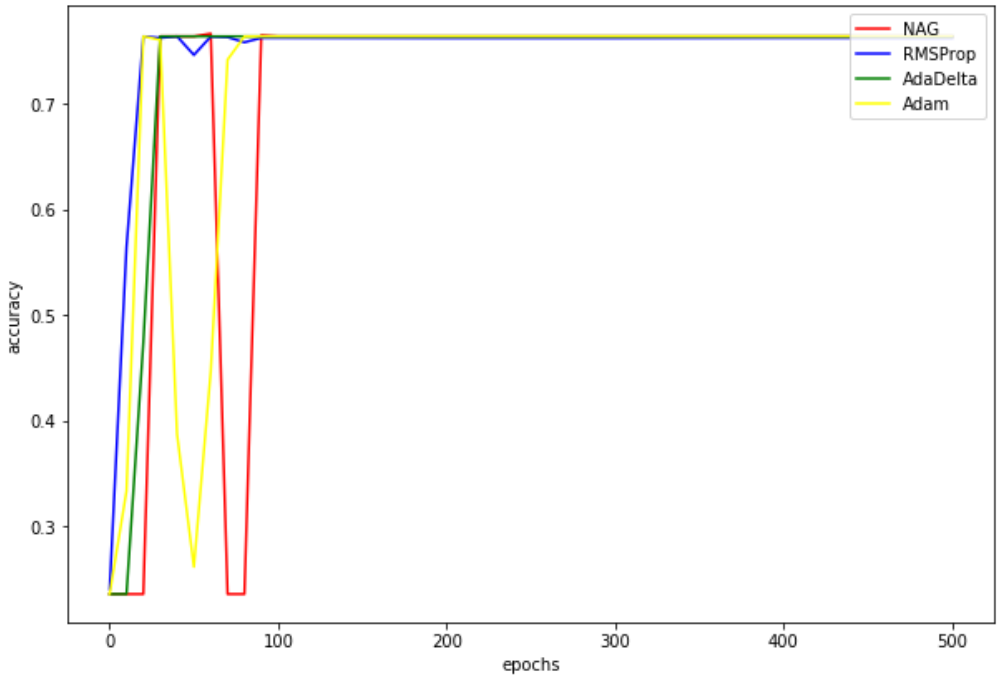
10.2.1.3 AdaDelta

gamma=0.9,c=0.01,epsilon=1e-10,threshold=0,dx=0.001,batch_size=64,epoch=500

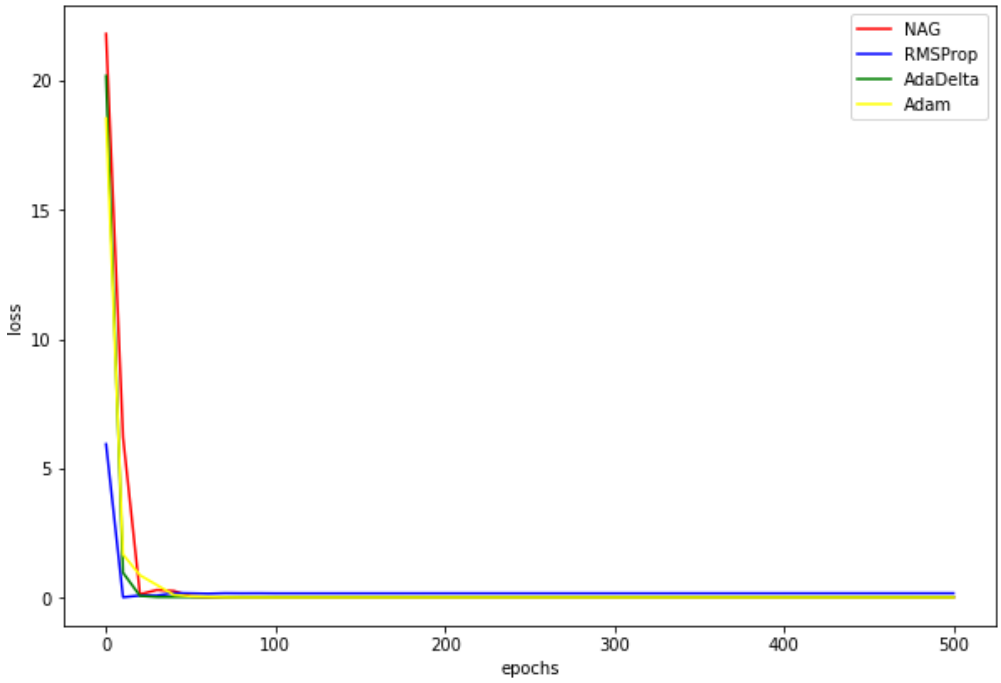
10.2.1.4 Adam

beta=0.9,gamma=0.999,c=0.01,epsilon=1e-8,threshold=0,learning_rate=0.01,
batch_size=64,epoch=500

10.2.2 预测结果（最佳结果）：



10.2.3 loss 曲线图：



11.实验结果分析:

在本次实验中，分别采用了 NAG，RMSProp，AdaDelta 和 Adam 四种不同的优化方法更新模型参数，并得到不同优化方法的 Loss 函数值。由实验结果分析可知：

1.NAG 利用 momentum 预测下一步的梯度，在梯度更新时做一个校正，避免前进太快，同时提高灵敏度。下降初期时，使用上一次参数和 momentum 更新，下降方向一致，乘上较大 γ 的能够进行很好的加速；下降中后期时，在局部最小值来回震荡的时候，gradient 接近于 0， γ 使得更新幅度增大，跳出陷阱；在梯度改变方向的时候，γ 能够减少更新，即 momentum 项能够在相关方向加速 SGD，抑制振荡，从而加快收敛。

2.RMSprop 依赖于全局学习率,是 Adagrad 算法的一种发展，也是 Adadelta 的变体，效果趋于二者之间，适合处理非平稳目标。

3.AdaDelta 算法训练初中期，加速效果不错，很快；而训练后期，反复在局部最小值附近抖动。

4.Adam 结合了 Adagrad 善于处理稀疏梯度和 RMSprop 善于处理非平稳目标的优点,对内存需求较小,为不同的参数计算不同的自适应学习率,同时也适用于大多非凸优化问题，适用于大数据集和高维空间。

对于逻辑回归的结果进行分析，可知这四种优化算法都能快速收敛，得到较小的 loss 值（0.1 左右）和较高的正确率（0.87 左右）。但是在本次的数据集上，效果最佳的是 AdaDelta，它收敛最快，loss 值最低且有较高的正确

率。

对于线性分类而言，这四种优化算法也都能快速收敛，得到较小的 loss 值和较高的正确率。本次实验采用的是 svm 算法，在本次的数据集上，效果最佳的是 RMSPro，它收敛最快，loss 值最低（0.14 左右）且有较高的正确率（0.76 左右）。

实验结果表明，在本次实践的数据集上，利用逻辑回归分类的效果比利用 svm 线性分类的效果要好。

12.对比逻辑回归和线性分类的异同点：

相同点：

1.逻辑回归和线性分类都是用于解决二分类问题，即结果集 y 的取值只有 0 和 1 的分类问题。

2. 逻辑回归和线性分类都必须选取一个合理的模型，例如 sigmoid 函数、一次函数等。

3. 逻辑回归和线性分类都必须选择一个"美好"的 loss 函数 (可以评估拟合程度，而且还是 convex 函数，这样才能求导)

4.采取多种优化方法（例如 NAG，RMSProp，AdaDelta 和 Adam 等）求出最好的模型参数。

相异点：

逻辑回归的主体还是回归操作: 回归对象是 sigmoid 函数, 它将输入映射为一个处于 0 到 1 之间的小数. 得到这个 0 到 1 之间的小数之后人为将其解读成概率, 然后根据事先设定的阈值进行分类。而线性分类是以超平面为决策边界的分类器。

13.实验总结：

本次实验使我进一步了解逻辑回归和线性分类的原理，同时也更加深刻的理解梯度下降和随机梯度下降的区别与联系。逻辑回归和线性分类都可以用于解决二分类问题，但是逻辑回归的主体依然是回归，只是回归对象是 sigmoid 函数，而线性分类，例如 SVM，则是用超平面为决策边界的分类。

SVM 方法是通过一个非线性映射 p，把样本空间映射到一个高维乃至无穷维的特征空间中(Hilbert 空间)，使得在原来的样本空间中非线性可分的问题转化为在特征空间中的线性可分的问题。简单地说，就是升维和线性化。升维，就是把样本向高维空间做映射，一般情况下这会增加计算的复杂性，甚至会引起"维数灾难"。但是在本次实验作为线性分类问题来说，SVM 把低维样本空间无法线性处理的样本集，在高维特征空间中通过一个线性超平面实现线性划分。SVM 是在高维特征空间中建立线性学习机，所以与线性模型相比，不但几乎不增加计算的复杂性，而且在某种程度上避免了"维数灾难"。

在这次实验中，我也遇到了不少难题。其中最为突出的就是调参。本次实验通过采用 NAG，RMSProp，AdaDelta 和 Adam 四种不同的优化方法更新模型参数，大大减小了调参的数量，但是，迭代次数 epoch 和初始学习率仍然需要调整，由于经验不足，调参过程还是比较艰难的，我的调参能力还需要继续加强！