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　肖亮昱

学　　号　201530613146

邮　　箱　624198747@qq.com

指导教师　吴庆耀

提交日期　2017 年　12 月 7 日

1. 实验题目：线性回归、线性分类与梯度下降

2. 实验时间：2017 年 12 月 9 日

3. 报告人:肖亮昱

4. 实验目的:

对比理解梯度下降和随机梯度下降的区别与联系。

对比理解逻辑回归和线性分类的区别与联系。

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。请自行下载训练集和验证集。

观察数据集我们可以看出，训练集共有 123 个属性，整个数据集是一个稀疏矩阵，而测试集则有 122 个属性，按照稀疏矩阵的格式，说明测试集上第 123 个属性的值为零，故在读取数据时要对测试数据进行额外处理，即增加第 123 行属性值，其值为 0。

同时在数据集上引入一个为 1 的属性来引入 bias，这样，数据集一共具有 124 个属性

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。

7、重复步骤 4-6 若干次，画出 Loss 和随迭代次数的变化图。

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。

7、重复步骤 4-6 若干次，画出 Loss 和随迭代次数的变化图。

7. 代码内容:

逻辑回归与随机梯度下降

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
------------------------------------------------
```

```
    File Name：      LogisticRegressionWithSGD
    Description :
    Author :        Nathan
    date：          2017/12/4
-------------------------------------------------
    Change Activity:
                        2017/12/4:
-------------------------------------------------
"""
__author__ = 'Nathan'

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
import math


plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode_minus']=False #用来正常显示负号


feature_num=123
batch_size=128
SGD_methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]
parm={"SGD":{"learning rate":0.01},\
      "NAG":{"learning rate":0.01,"Gamma":0.9},\
      "RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\
      "AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\
      "Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}
temp_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 compute_loss(W, X_test,y_test):
```

```
        logits = np.matmul(X_test, W)
        loss = -np.mean(y_test * np.log(sigmoid(logits)) + (1 - y_test) * np.log(1 -
sigmoid(logits)))
        return loss

    def compute_gradient(W,X_train,y_train):
        logits = np.matmul(X_train, W)
        output = sigmoid(logits)
        error = output - y_train
        gradient = np.matmul(X_train.transpose(), error) / y_train.shape[0]
        return gradient

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

    def NAG(W,X_train,y_train):
        global temp_list
        global parm
        momentum=temp_list.get('NAG')
        #learning_rate=parm.get("NAG").get("learning_rate")
        Gamma=parm.get("NAG").get("Gamma")
        gradient=compute_gradient(W-(Gamma*momentum),X_train,y_train)
        update_momentum = momentum * Gamma+ gradient *
parm.get("NAG").get("learning rate")
        temp_list["NAG"]=update_momentum
        W-=update_momentum
        return W

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

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

```
    Gamma=parm.get("AdaDelta").get("Gamma")
    Epsilon=parm.get("AdaDelta").get("Epsilon")
    gradient=compute_gradient(W,X_train,y_train)
    EG=Gamma*EG+(1-Gamma)*gradient**2
    temp_list.get("AdaDelta")["EG"]=EG
    delta=-1*gradient*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)
    EX=Gamma*EX+(1-Gamma)*delta**2
    temp_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=temp_list.get("Adam").get("M")
    G=temp_list.get("Adam").get("G")
    t=temp_list.get("Adam").get("t")
    t=t+1
    temp_list.get("Adam")["t"]=t
    gradient=compute_gradient(W,X_train,y_train)
    M=Beta*M+(1-Beta)*gradient
    temp_list.get("Adam")["M"]=M
    G=Gamma*G+(1-Gamma)*gradient**2
    temp_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(r'C:\Users\jy\Desktop\Libsvmdata\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(r'C:\Users\jy\Desktop\Libsvmdata\a9a_test.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 LogisicRegressionModel():
        X_train, y_train, X_test, y_test, data_size, features_num = getdata()
        plt.xlabel('Iteration')
        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(compute_loss(W,X_test,y_test))
                    num+=1
            plt.plot(iter_, error, label=method)
        plt.legend()
        plt.show()

LogisicRegressionModel()
```

线性分类：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
-------------------------------------------------
    File Name：     LinearClassificationWIthSGD
    Description :
    Author :        Nathan
    date：          2017/12/9
-------------------------------------------------
    Change Activity:
                    2017/12/9:
-------------------------------------------------
"""
__author__ = 'Nathan'

__author__ = 'Nathan'

from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt


plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
```

```
    plt.rcParams['axes.unicode_minus']=False #用来正常显示负号


    feature_num=123
    batch_size=512
    SGD_methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]
    parm={"C":0.9,\
          "SGD":{"learning rate":0.01},\
          "NAG":{"learning rate":0.005,"Gamma":0.9},\
          "RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\
          "AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\
          "Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}
    temp_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 compute_loss(W, X_test,y_test):
        L = 0.
        N = y_test.shape[0]
        temp=1-y_test*np.matmul(X_test,W)
        L = sum(np.maximum(0, temp))#hinge loss
        loss =0.5 * np.matmul(W.T, W)[0][0] + (L * parm.get("C"))/N
        return loss


    def compute_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")*compute_gradient(W,X_train,y_train)
        return W


    def NAG(W,X_train,y_train):
        global temp_list
        global parm
        momentum=temp_list.get('NAG')
        #learning_rate=parm.get("NAG").get("learning_rate")
        Gamma=parm.get("NAG").get("Gamma")
        gradient=compute_gradient(W-(Gamma*momentum),X_train,y_train)
        update_momentum = momentum * Gamma+ gradient *
parm.get("NAG").get("learning rate")
        temp_list["NAG"]=update_momentum
        W-=update_momentum
        return W


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


    def AdaDelta(W,X_train,y_train):
        EG=temp_list.get("AdaDelta").get("EG")
        EX=temp_list.get("AdaDelta").get("EX")
        Gamma=parm.get("AdaDelta").get("Gamma")
        Epsilon=parm.get("AdaDelta").get("Epsilon")
        gradient=compute_gradient(W,X_train,y_train)
        EG=Gamma*EG+(1-Gamma)*gradient**2
        temp_list.get("AdaDelta")["EG"]=EG
        delta=-1*gradient*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)
        EX=Gamma*EX+(1-Gamma)*delta**2
        temp_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=temp_list.get("Adam").get("M")
        G=temp_list.get("Adam").get("G")
        t=temp_list.get("Adam").get("t")
        t=t+1
        temp_list.get("Adam")["t"]=t
        gradient=compute_gradient(W,X_train,y_train)
        M=Beta*M+(1-Beta)*gradient
        temp_list.get("Adam")["M"]=M
        G=Gamma*G+(1-Gamma)*gradient**2
        temp_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(r'C:\Users\jy\Desktop\Libsvmdata\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(r'C:\Users\jy\Desktop\Libsvmdata\a9a_test.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 LinearClassificationModel():
        X_train, y_train, X_test, y_test, data_size, features_num = getdata()
        plt.xlabel('Iteration')
        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(compute_loss(W,X_test,y_test))
                num+=1
        print(method+"completed!")
        plt.plot(iter_, error, label=method)
    plt.legend()
    plt.show()
```

LinearClassificationModel()

（针对线性回归和线性分类分别填写 8-12 内容）

逻辑回归与随机梯度下降：

8. 模型参数的初始化方法:

采用随机方式初始化权重 W

SGD，NAG，RMSProp，AdaDelta，Adam 的各项初始化参数均参考论文里的推荐数值进行设置。

```
parm={"C":0.9,\
        "SGD":{"learning rate":0.01},\
        "NAG":{"learning rate":0.005,"Gamma":0.9},\
        "RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\
        "AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\
        "Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}
```

9.选择的 loss 函数及其导数:

$$h_w(\boldsymbol{X}) = g(\boldsymbol{w^T X}) = \frac{1}{1+e^{-\boldsymbol{w^T X}}}$$

$$\boldsymbol{J(w)} = -\frac{1}{n}\left[\sum_{i=1}^{n} y_i \log h_w(x_i) + (1-y_i)\log(1-h_w(x_i))\right]$$

其导数为：

$$\frac{\boldsymbol{\partial J(w)}}{\boldsymbol{\partial w}} = (\boldsymbol{h_w(X) - y})\boldsymbol{X}$$

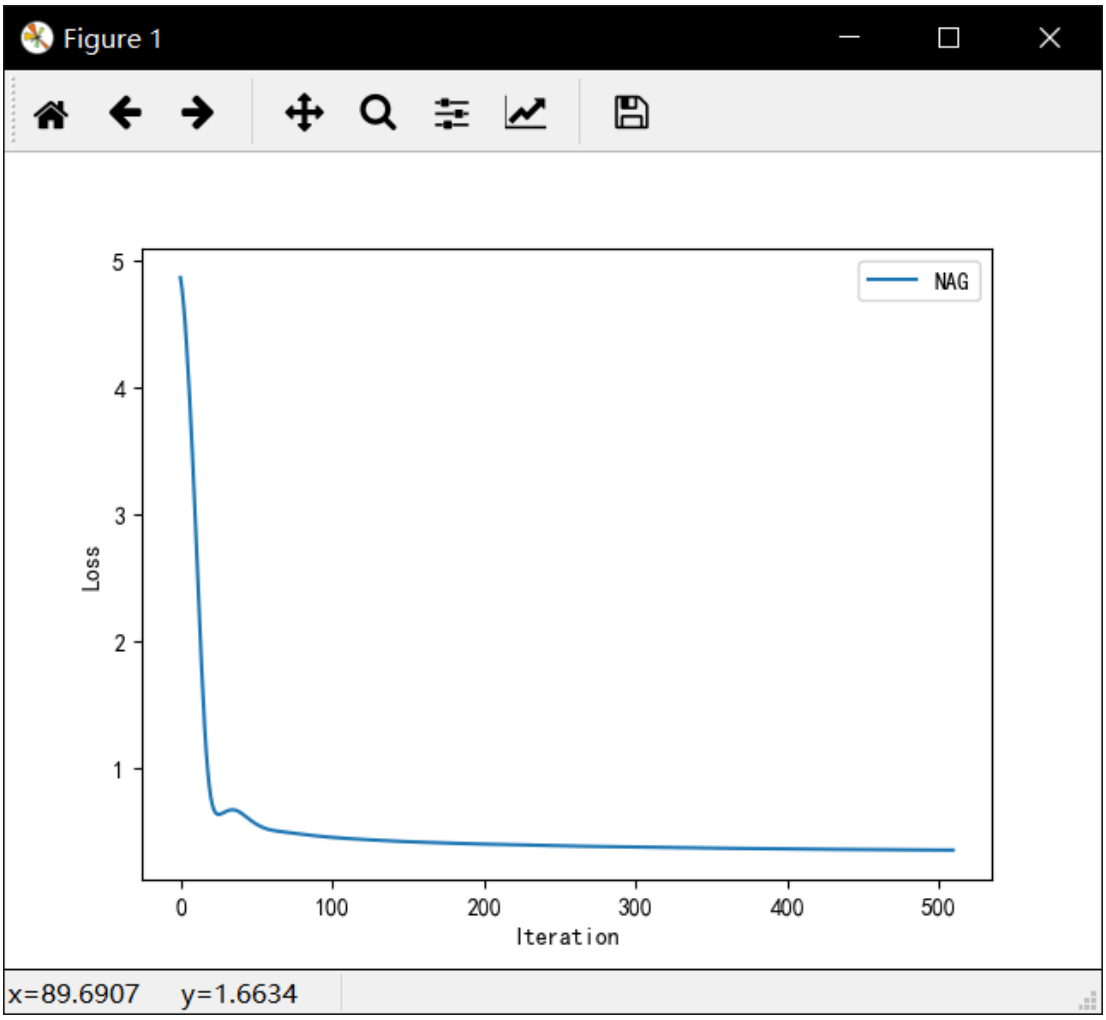
10.实验结果和曲线图:

NAG

超参数选择：**"NAG":{"learning rate":0.005,"Gamma":0.9},**

预测结果（最佳结果）：

loss 曲线图：

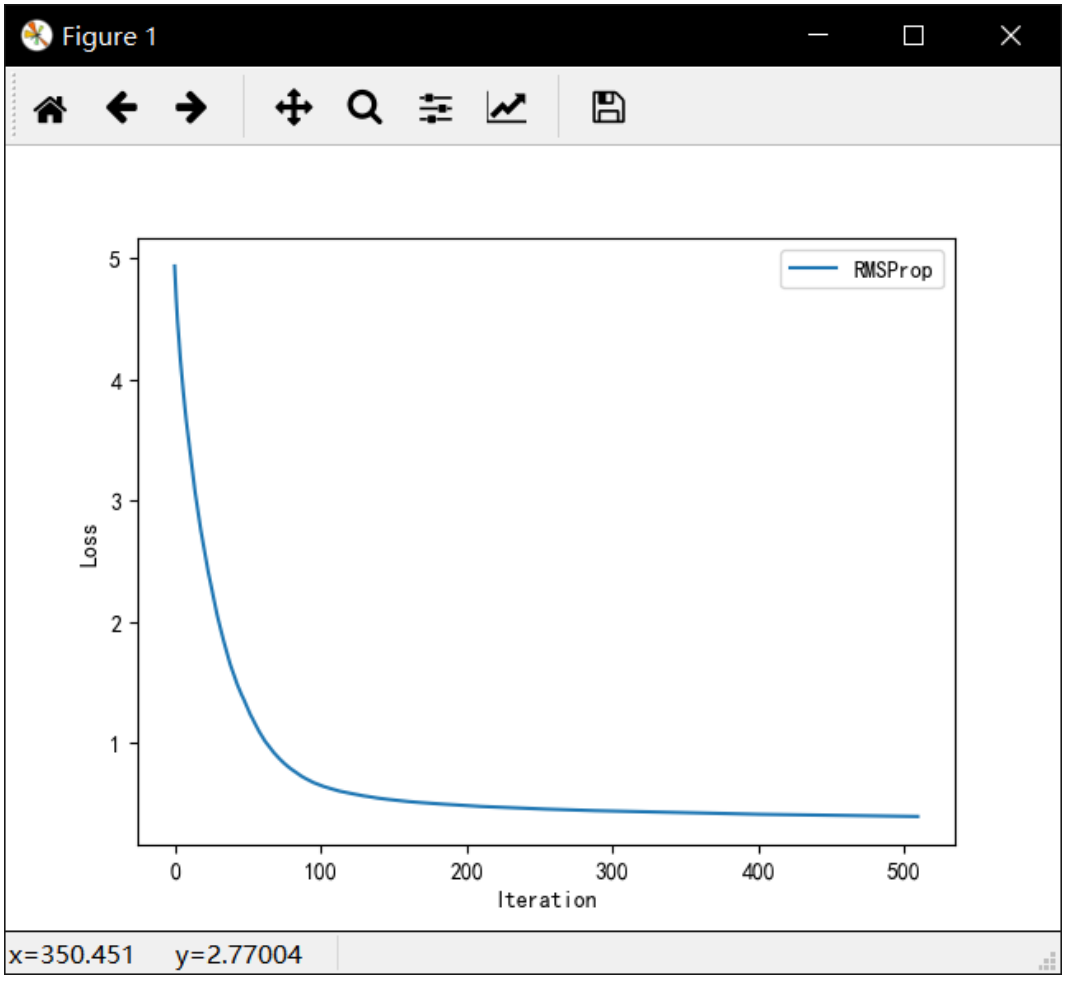


RMSProp：

超参数选择：**RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},**

预测结果（最佳结果）：

loss 曲线图：

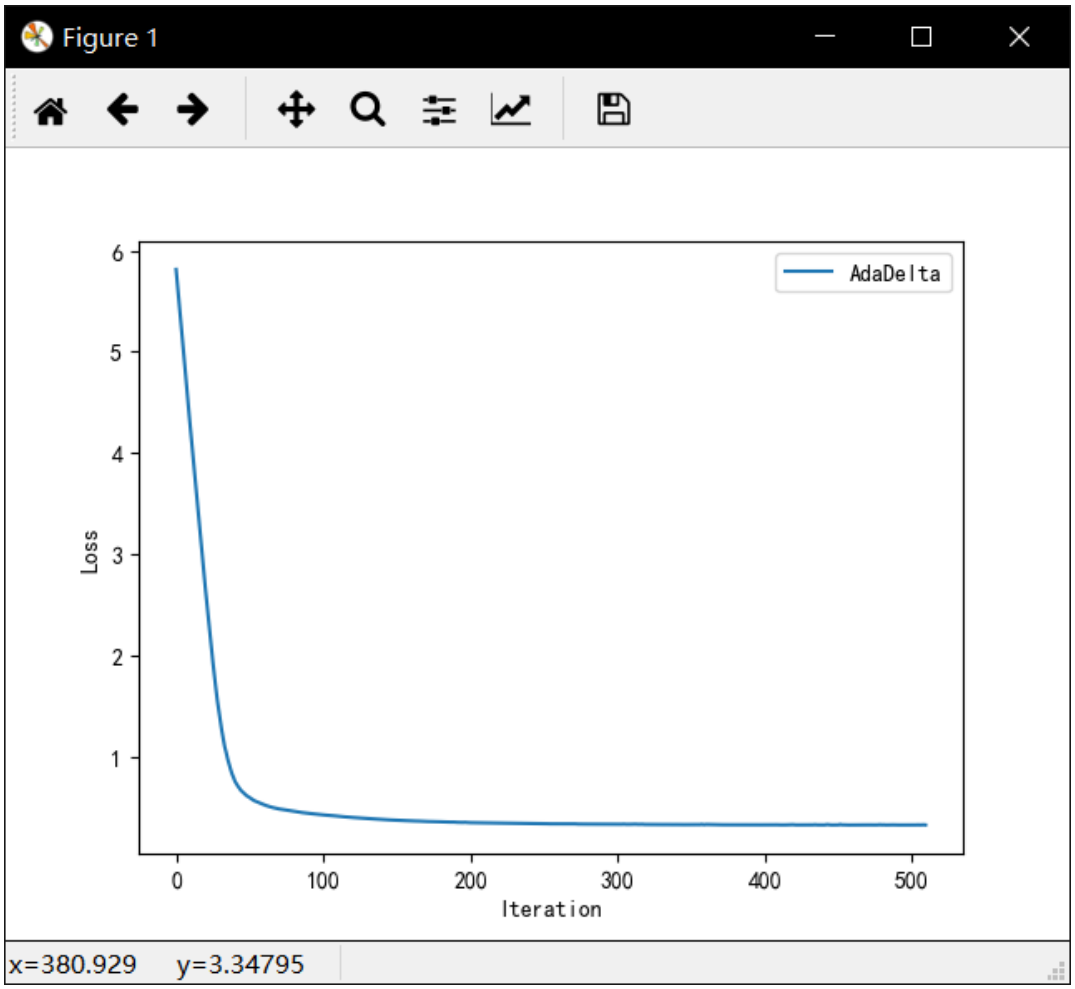


AdaDelta

超参数选择：**"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6}**

预测结果（最佳结果）：

loss 曲线图：

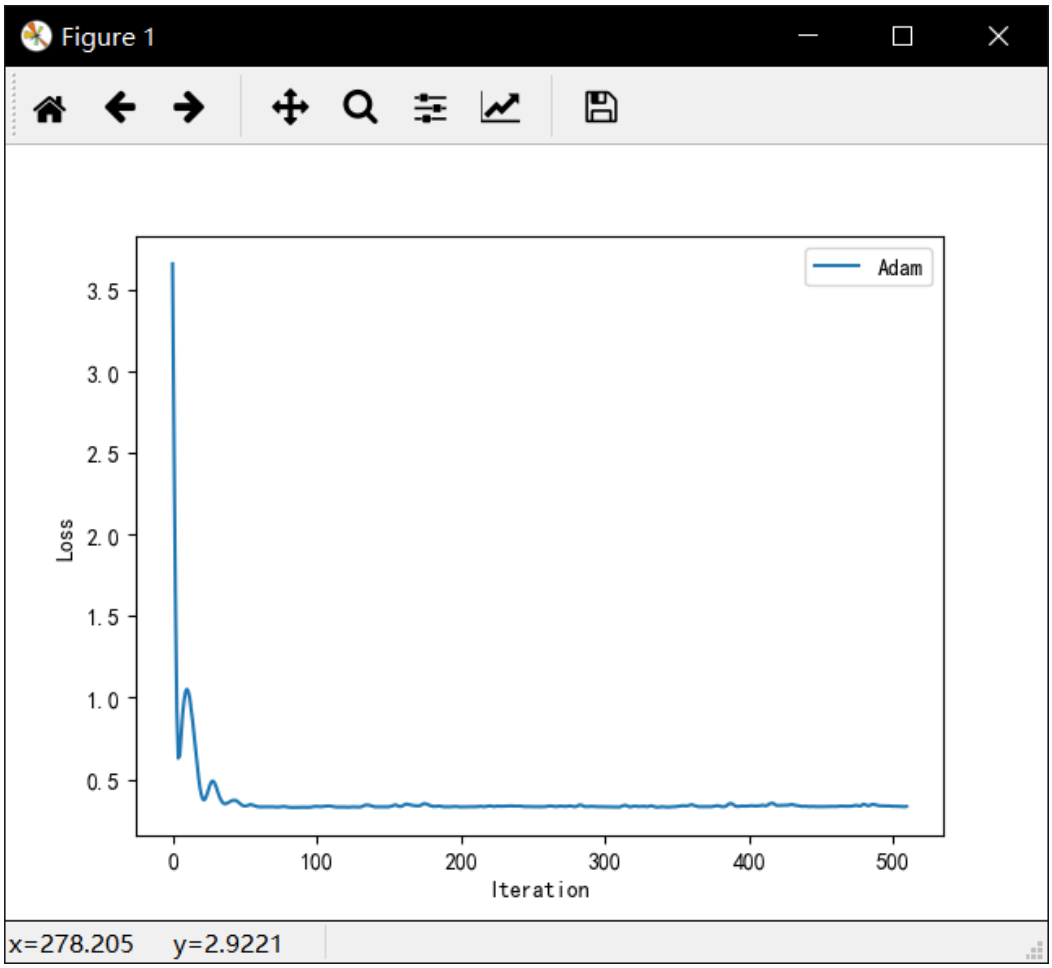


Adam

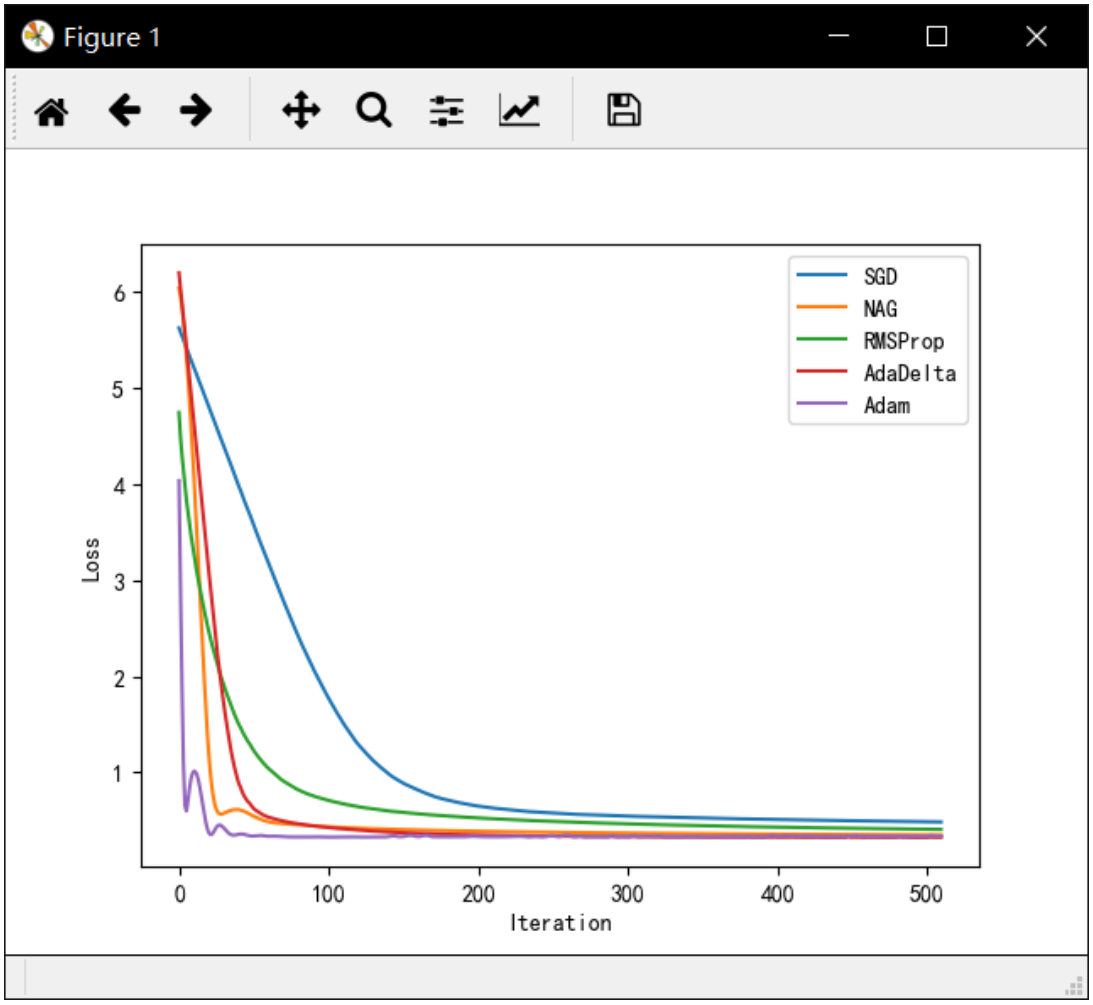
超参数选择：**"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}**

预测结果（最佳结果）：

loss 曲线图：



12.实验结果分析:



从这五条曲线我们可以看出，单纯的 SGD 的 loss 在下降速度上比较缓慢，这说明了 SGD 在面对大量数据时梯度下降的准确率略低，同时 learning rate 的选取比较困难，容易使得 loss 曲线振荡且易收敛到局部最优。NAG 引入了动量影响的使得梯度的更新更加灵活，AdaDelta 不再依赖于人为设定的全局学习率，在初期加速效果不错，可在后期在最小值附近振荡，RMSprop 可看做是 AdaDelta 的特例，但仍依赖于全局学习率，Adam 本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率，可以看出它的 loss 曲线下降的速度是最快的，Adam 的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

线性分类与随机梯度下降：

8. 选择的评估方法（留出法，交叉验证，k 折交叉验证等）：

采用留出法来切分数据集：调用 train_test_split 函数切分数据集，test_size=0.33,选取 0.33 的数据作为验证集，随机选择。

9. 模型参数的初始化方法:

采用随机方式初始化权重 W

SGD，NAG，RMSProp，AdaDelta，Adam 的各项初始化参数均参考论文里的推荐数值进行设置。

```
parm={"C":0.9,\
        "SGD":{"learning rate":0.01},\
        "NAG":{"learning rate":0.005,"Gamma":0.9},\
        "RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\
        "AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\
        "Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}
```

10.选择的 loss 函数及其导数:

Loss 函数L(**w**)为 margin loss 和 hinge loss 的组合

$$\boldsymbol{L(w)} = \frac{\|\boldsymbol{w}\|^2}{\mathbf{2}} + \boldsymbol{C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{g(w)}$$

$$\boldsymbol{g(w)} = \mathbf{max}(\mathbf{0}, \mathbf{1} - \boldsymbol{y_i w^T x_i})$$

其导数为：

$$\boldsymbol{g_w(x)} = \begin{cases} -\boldsymbol{y_i x_i} & 1 - \boldsymbol{y_i w^T x_i} >= 0 \\ \mathbf{0} & 1 - \boldsymbol{y_i w^T x_i} < 0 \end{cases}$$

$$\frac{\boldsymbol{\partial L(w)}}{\boldsymbol{\partial w}} = \boldsymbol{w} + \boldsymbol{C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{g_w(x_i)}$$

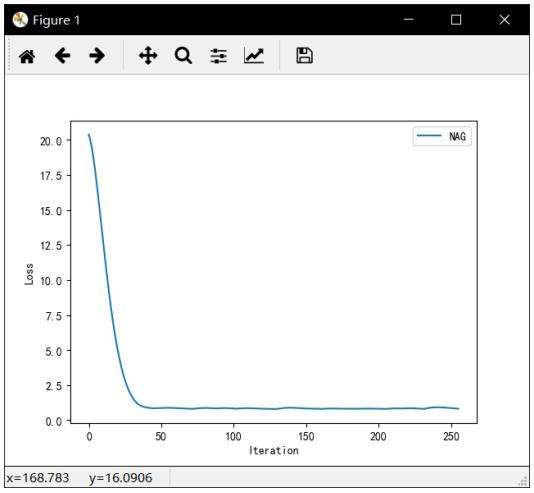
11.实验结果和曲线图:

NAG

超参数选择：**"NAG":{"learning rate":0.005,"Gamma":0.9},**

预测结果（最佳结果）：

loss 曲线图：

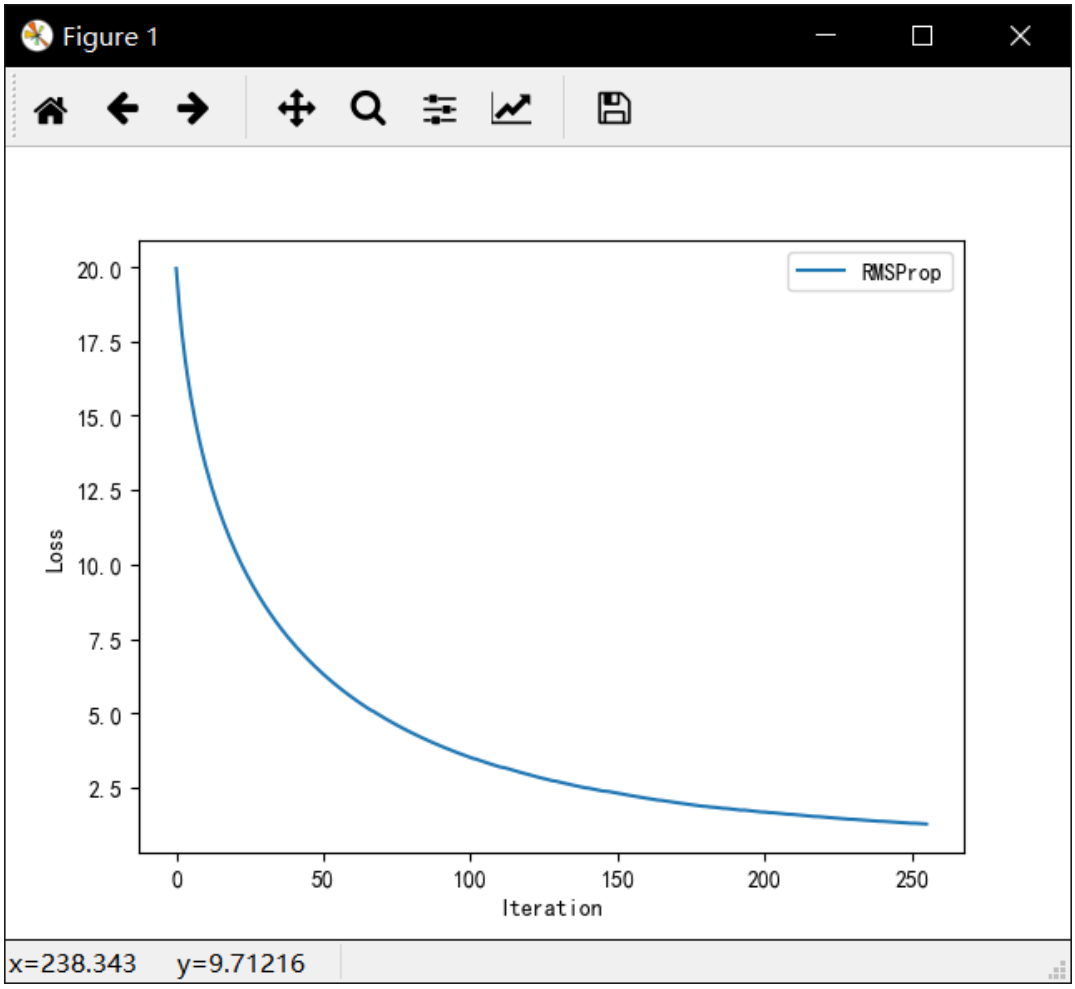


RMSProp：

超参数选择：**RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},**

预测结果（最佳结果）：

loss 曲线图：

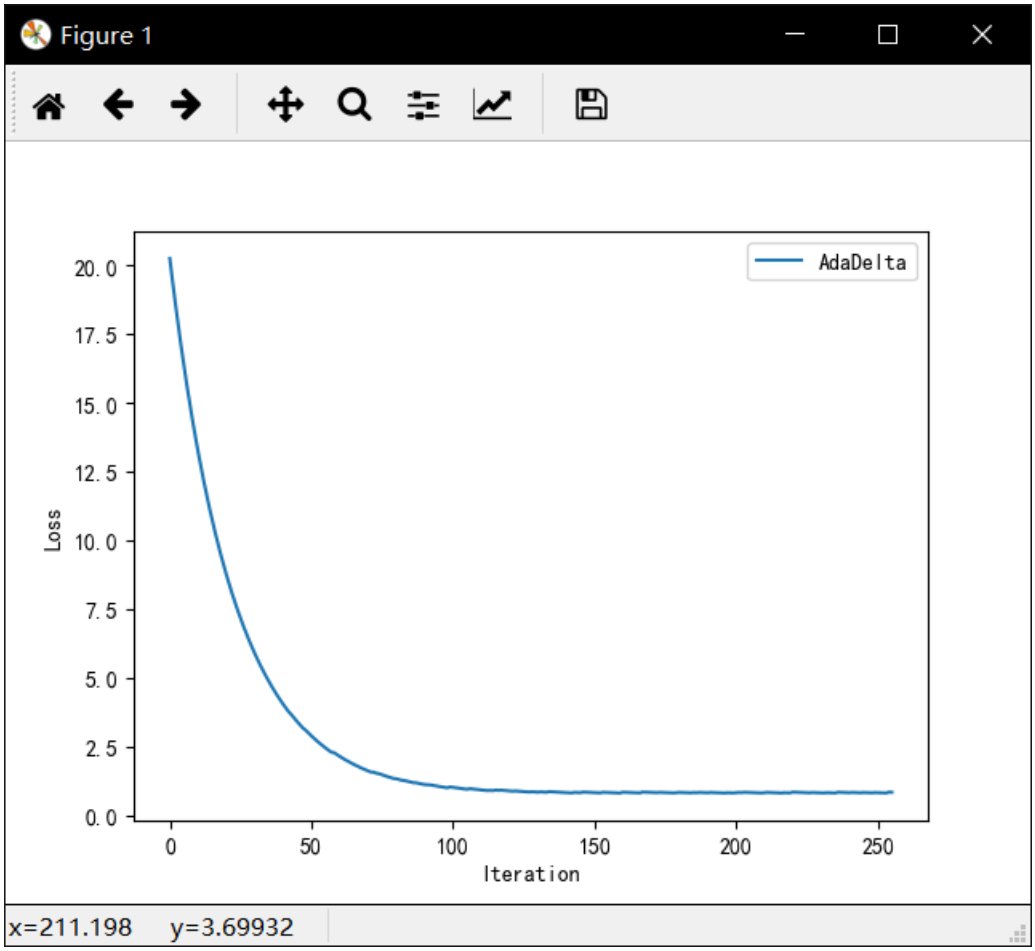


AdaDelta

超参数选择：**"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6}**

预测结果（最佳结果）：

loss 曲线图：

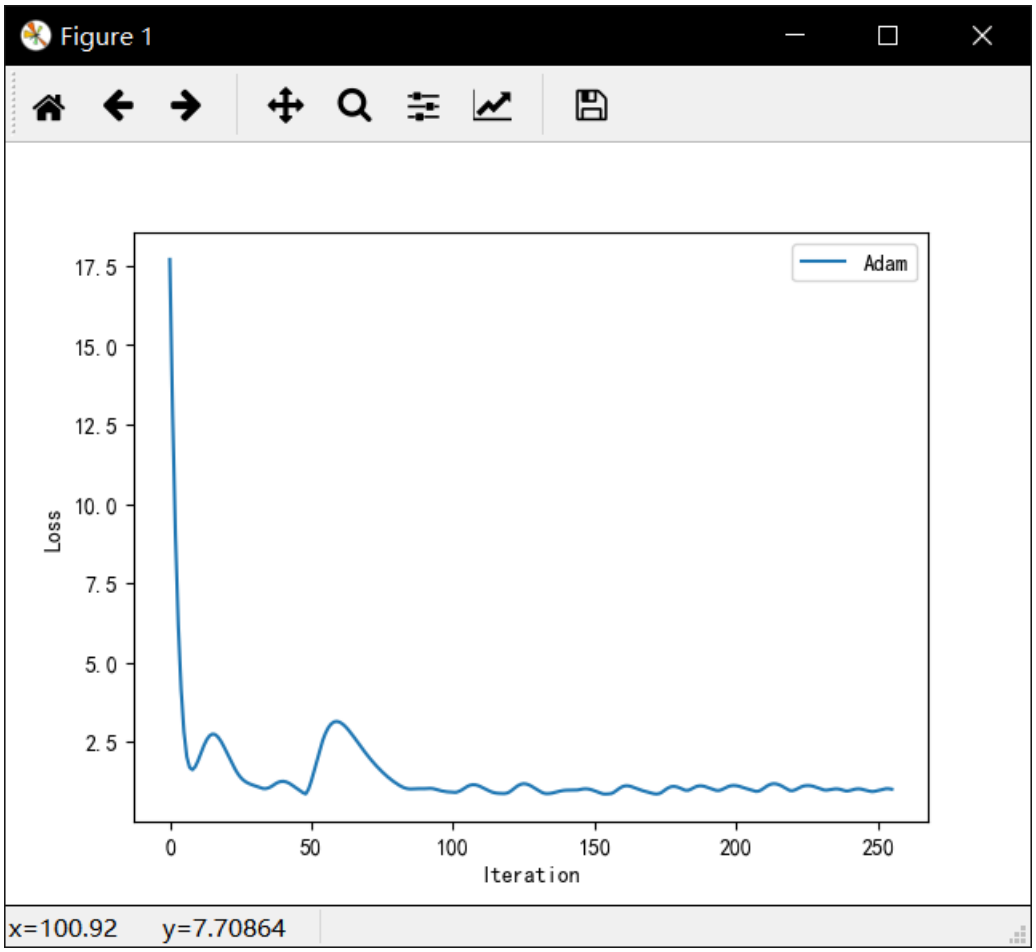


Adam

超参数选择：**"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}**

预测结果（最佳结果）：

loss 曲线图：



12.实验结果分析:

从这五条曲线我们可以看出，单纯的 SGD 的 loss 在下降速度上比较缓慢，

这说明了 SGD 在面对大量数据时梯度下降的准确率略低，同时 learning rate 的选取比较困难，容易使得 loss 曲线振荡且易收敛到局部最优。NAG 引入了动量影响的使得梯度的更新更加灵活，AdaDelta 不再依赖于人为设定的全局学习率，在初期加速效果不错，可在后期在最小值附近振荡，RMSprop 可看做是 AdaDelta 的特例，但仍依赖于全局学习率，Adam 本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam 的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

13.对比线性回归和线性分类的异同点：

线性回归和线性分类本质上都是模型的拟合，都是在最小化 loss 的过程中调整参数来拟合数据。他们的不同在于输出变量的类型，线性回归输出的是定量分析，也就是预测一个实际值，线性分类的输出是定性分析，也就是做分类判断。在最小化 loss 函数的过程中，他们的思路是相通相近的，都可以用递归下降的方式来优化权重

14.实验总结：

通过此次实验，进一步理解了线性回归，线性分类，随机梯度下降以及其他四种随机梯度下降的原理与实现方法。在大规模数据集上实践体会优化和调参的过程。同时也训练了我们对矩阵运算的使用，尤其实在大规模的数据上，要尽快能的利用 numpy 库里的运算函数而不是自己写 for 循环，这样才能充分使用计算机的性能，加快运算速度，提高训练的迭代次数。