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　王圣杰

学　　号　201530612873

邮　　箱　2256989051@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 14 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 2 日

3. 报告人: 王圣杰

4. 实验目的:

⑴对比理解梯度下降和随机梯度下降的区别与联系。

⑵对比理解逻辑回归和线性分类的区别与联系。

⑶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。

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 G。
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 LNAG，LRMSProp，LAdaDelta 和 LAdam。
7. 重复步骤 4-6 若干次，**画出 LNAG，LRMSProp，LAdaDelta 和 LAdam 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 G。

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

8. 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 LNAG，LRMSProp，LAdaDelta 和 LAdam。
9. 重复步骤 4-6 若干次，**画出 LNAG，LRMSProp，LAdaDelta 和 LAdam 随迭代次数的变化图**。

7. 代码内容:

逻辑回归和随机梯度下降

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

#下面是对数据进行处理的代码
data1 = load_svmlight_file("a9a.txt")
X_train, y_train = data1[0], data1[1]

data2=load_svmlight_file("a9a_t.txt")
X_test,y_test=data2[0],data2[1]

y_train[y_train==-1]=0
y_test[y_test==-1]=0

X_train=X_train.dot(np.eye(123))
X_train=np.column_stack((X_train,np.ones((32561,1))))

X_test=X_test.dot(np.eye(122))
X_test=np.column_stack((X_test,np.zeros((16281,1))))
X_test=np.column_stack((X_test,np.ones((16281,1))))

y_train=y_train.reshape(32561,1)
y_test=y_test.reshape(16281,1)

m_train=X_train.shape[0]
m_test=X_test.shape[0]

#定义 sigmoid 函数
def sigmoid(z):
```

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

    #定义预测准确率函数
    def predict(w,X,y):
        right=0
        p=X.dot(w)
        for i in range(X.shape[0]):
            if p[i]>=0:
                p[i]=1
            else:
                p[i]=0

        for j in range(X.shape[0]):
            if p[j]==y[j]:
                right=right+1
        P=right/X.shape[0]
        return P

    #定义学习率
    alpha=0.1
    #初始化所有参数，这里全部采用全零初始化
    w_SGD=np.zeros((124,1))
    G_SGD=np.zeros((124,1))
    D_SGD=np.zeros((124,1))
    L_SGD=np.zeros((1000,1))
    P=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((300,124))
        z_y=np.zeros((300,1))
        for j in range(300):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        for k in range(124):
            G_SGD[k]=((sigmoid(z_x.dot(w_SGD))-z_y).T).dot(z_x[:,k])
        G_SGD=G_SGD/300
        D_SGD=-G_SGD
        w_SGD=w_SGD+alpha*D_SGD
        for l in range(m_test):

L_SGD[i]=L_SGD[i]-(y_test[l]*np.log(sigmoid((X_test[l,:].reshape(1,124)).
dot(w_SGD)))+(1-y_test[l])*np.log(1-sigmoid((X_test[l,:].reshape(1,124)).d
ot(w_SGD))))
```

```
            L_SGD[i]=L_SGD[i]/m_test
            P[i]=predict(w_SGD,X_test,y_test)
    print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
    print("Lowest loss:  iteration %d,  loss %f" %(L_SGD.argmin()+1, L_SGD.min()))
    x=np.arange(0,1000,1)
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_SGD,label='L_SGD')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
    plt.ylabel('Loss')
    plt.show()

    #定义学习率和一些超参数
    alpha=0.1
    gamma=0.9
    #初始化所有参数，这里全部采用全零初始化
    w_NAG=np.zeros((124,1))
    G_NAG=np.zeros((124,1))
    v_NAG=np.zeros((124,1))
    L_NAG=np.zeros((1000,1))
    P=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((300,124))
        z_y=np.zeros((300,1))
        for j in range(300):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        for k in range(124):

G_NAG[k]=((sigmoid(z_x.dot(w_NAG-gamma*v_NAG))-z_y).T).dot(z_x[:,k])
        G_NAG=G_NAG/300
        v_NAG=gamma*v_NAG+alpha*G_NAG
        w_NAG=w_NAG-v_NAG
        for l in range(m_test):

L_NAG[i]=L_NAG[i]-(y_test[l]*np.log(sigmoid((X_test[l,:].reshape(1,124)).dot(w_NAG)))+(1-y_test[l])*np.log(1-sigmoid((X_test[l,:].reshape(1,124)).dot(w_NAG))))
        L_NAG[i]=L_NAG[i]/m_test
```

```
            P[i]=predict(w_NAG,X_test,y_test)
        print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
        print("Lowest loss:  iteration %d,  loss %f" %(L_NAG.argmin()+1, L_NAG.min()))
        x=np.arange(0,1000,1)
        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        plt.plot(x,L_NAG,label='L_NAG')
        plt.legend(loc='upper right')
        plt.xlabel('Num of iterations')
        plt.ylabel('Loss')
        plt.show()

        #定义学习率和一些超参数
        alpha=0.001
        gamma=0.9
        epsilon=0.00000001
        #初始化所有参数，这里全部采用全零初始化
        w_RMS=np.zeros((124,1))
        G_RMS=np.zeros((124,1))
        g_RMS=np.zeros((124,1))
        L_RMS=np.zeros((1000,1))
        P=np.zeros((1000,1))
        #迭代 1000 次
        for i in range(1000):
            z_x=np.zeros((300,124))
            z_y=np.zeros((300,1))
            for j in range(300):
                r = np.random.randint(m_train)
                z_x[j,:] = X_train[r,:]
                z_y[j]=y_train[r]
            for k in range(124):
                g_RMS[k]=((sigmoid(z_x.dot(w_RMS))-z_y).T).dot(z_x[:,k])
            g_RMS=g_RMS/300
            G_RMS=gamma*G_RMS+(1-gamma)*g_RMS*g_RMS
            w_RMS=w_RMS-alpha/np.sqrt(G_RMS+epsilon)*g_RMS
            for l in range(m_test):

L_RMS[i]=L_RMS[i]-(y_test[l]*np.log(sigmoid((X_test[l,:].reshape(1,124)).dot(w_RMS)))+(1-y_test[l])*np.log(1-sigmoid((X_test[l,:].reshape(1,124)).dot(w_RMS))))
            L_RMS[i]=L_RMS[i]/m_test
            P[i]=predict(w_RMS,X_test,y_test)
        print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1,
```

```
P.max()))
    print("Lowest loss:  iteration %d,  loss %f" %(L_RMS.argmin()+1,
L_RMS.min()))
    x=np.arange(0,1000,1)
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_RMS,label='L_RMS')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
    plt.ylabel('Loss')
    plt.show()

    #定义一些超参数
    gamma=0.95
    epsilon=0.000001
    #初始化所有参数，这里全部采用全零初始化
    w_AdaD=np.zeros((124,1))
    G_AdaD=np.zeros((124,1))
    g_AdaD=np.zeros((124,1))
    deltaT=np.zeros((124,1))
    deltaW=np.zeros((124,1))
    L_AdaD=np.zeros((1000,1))
    P=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((300,124))
        z_y=np.zeros((300,1))
        for j in range(300):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        for k in range(124):

g_AdaD[k]=((sigmoid(z_x.dot(w_AdaD))-z_y).T).dot(z_x[:,k])
        g_AdaD=g_AdaD/300
        G_AdaD=gamma*G_AdaD+(1-gamma)*g_AdaD*g_AdaD
        deltaW=-np.sqrt((deltaT+epsilon)/(G_AdaD+epsilon))*g_AdaD
        w_AdaD=w_AdaD+deltaW
        deltaT=gamma*deltaT+(1-gamma)*deltaW*deltaW
        for l in range(m_test):

L_AdaD[i]=L_AdaD[i]-(y_test[l]*np.log(sigmoid((X_test[l,:].reshape(1,124)
).dot(w_AdaD)))+(1-y_test[l])*np.log(1-sigmoid((X_test[l,:].reshape(1,124))
.dot(w_AdaD))))
        L_AdaD[i]=L_AdaD[i]/m_test
```

```
            P[i]=predict(w_AdaD,X_test,y_test)
        print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
        print("Lowest loss:  iteration %d,  loss %f" %(L_AdaD.argmin()+1, L_AdaD.min()))
        x=np.arange(0,1000,1)
        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        plt.plot(x,L_AdaD,label='L_AdaD')
        plt.legend(loc='upper right')
        plt.xlabel('Num of iterations')
        plt.ylabel('Loss')
        plt.show()

        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        plt.plot(x,L_SGD,label='L_SGD')
        plt.plot(x,L_NAG,label='L_NAG')
        plt.plot(x,L_RMS,label='L_RMS')
        plt.plot(x,L_AdaD,label='L_AdaD')
        plt.plot(x,L_Adam,label='L_Adam')
        plt.legend(loc='upper right')
        plt.xlabel('Num of iterations')
        plt.ylabel('Loss')
        plt.show()
```

线性分类和随机梯度下降

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

#下面是对数据进行处理的代码
data1 = load_svmlight_file("a9a.txt")
X_train, y_train = data1[0], data1[1]

data2=load_svmlight_file("a9a_t.txt")
X_test,y_test=data2[0],data2[1]

X_train=X_train.dot(np.eye(123))
X_train=np.column_stack((X_train,np.ones((32561,1))))

X_test=X_test.dot(np.eye(122))
X_test=np.column_stack((X_test,np.zeros((16281,1))))
X_test=np.column_stack((X_test,np.ones((16281,1))))
```

```
y_train=y_train.reshape(32561,1)
y_test=y_test.reshape(16281,1)

m_train=X_train.shape[0]
m_test=X_test.shape[0]

#定义损失函数
def loss(w,X,y,C):
    M_test=1-y*(X.dot(w))
    M_test[M_test<0]=0
    loss=sum(w*w)/2+C*sum(M_test)/X.shape[0]
    return loss
#定义计算梯度的函数
def grad(w,X,y,C):
    M_train=1-y*(X.dot(w))
    minus_y=-y
    minus_y[M_train<0]=0
    g=w+C*((X.T).dot(minus_y))/16
    g[-1]=C*sum(minus_y)/16
    return g
#定义计算预测准确度的函数
def predict(w,X,y):
    right=0
    p=X.dot(w)
    for i in range(X.shape[0]):
        if p[i]>=0:
            p[i]=1
        else:
            p[i]=-1
    for j in range(X.shape[0]):
        if p[j]==y[j]:
            right=right+1
    P=right/X.shape[0]
    return P

#定义学习率和一些超参数
C=100
alpha=0.0005
#初始化所有参数，这里全部采用全零初始化
w_SGD=np.zeros((124,1))
g_SGD=np.zeros((124,1))
L_SGD=np.zeros((1000,1))
P=np.zeros((1000,1))
#迭代 1000 次
```

```
    for i in range(1000):
        z_x=np.zeros((16,124))
        z_y=np.zeros((16,1))
        for j in range(16):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        g_SGD=grad(w_SGD,z_x,z_y,C)
        w_SGD=w_SGD-alpha*g_SGD
        L_SGD[i]=loss(w_SGD,X_test,y_test,C)
        P[i]=predict(w_SGD,X_test,y_test)
    print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
    print("Lowest  loss:  iteration  %d,  loss  %f"  %(L_SGD.argmin()+1,
L_SGD.min()))
    x=np.arange(0,1000,1)
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_SGD,label='L_SGD')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
    plt.ylabel('Loss')
    plt.show()

    #定义学习率和一些超参数
    C=100
    alpha=0.0001
    gamma=0.9
    #初始化所有参数，这里全部采用全零初始化
    w_NAG=np.zeros((124,1))
    g_NAG=np.zeros((124,1))
    v_NAG=np.zeros((124,1))
    L_NAG=np.zeros((1000,1))
    P=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((16,124))
        z_y=np.zeros((16,1))
        for j in range(16):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        g_NAG=grad(w_NAG-gamma*v_NAG,z_x,z_y,C)
        v_NAG=gamma*v_NAG+alpha*g_NAG
        w_NAG=w_NAG-v_NAG
        L_NAG[i]=loss(w_NAG,X_test,y_test,C)
```

```
            P[i]=predict(w_NAG,X_test,y_test)
        print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
        print("Lowest  loss:  iteration  %d,  loss  %f"  %(L_NAG.argmin()+1,
L_NAG.min()))
        x=np.arange(0,1000,1)
        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        plt.plot(x,L_NAG,label='L_NAG')
        plt.legend(loc='upper right')
        plt.xlabel('Num of iterations')
        plt.ylabel('Loss')
        plt.show()

        #定义学习率和一些超参数
        C=100
        alpha=0.001
        gamma=0.9
        epsilon=0.00000001
        #初始化所有参数，这里全部采用全零初始化
        w_RMS=np.zeros((124,1))
        G_RMS=np.zeros((124,1))
        g_RMS=np.zeros((124,1))
        L_RMS=np.zeros((1000,1))
        P=np.zeros((1000,1))
        #迭代 1000 次
        for i in range(1000):
            z_x=np.zeros((300,124))
            z_y=np.zeros((300,1))
            for j in range(300):
                r = np.random.randint(m_train)
                z_x[j,:] = X_train[r,:]
                z_y[j]=y_train[r]
            g_RMS=grad(w_RMS,z_x,z_y,C)
            G_RMS=gamma*G_RMS+(1-gamma)*g_RMS*g_RMS
            w_RMS=w_RMS-alpha/np.sqrt(G_RMS+epsilon)*g_RMS
            L_RMS[i]=loss(w_RMS,X_test,y_test,C)
            P[i]=predict(w_RMS,X_test,y_test)
        print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
        print("Lowest  loss:  iteration  %d,  loss  %f"  %(L_RMS.argmin()+1,
L_RMS.min()))
        x=np.arange(0,1000,1)
        plt.rcParams['figure.figsize'] = (10.0, 8.0)
        plt.plot(x,L_RMS,label='L_RMS')
        plt.legend(loc='upper right')
        plt.xlabel('Num of iterations')
```

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

    #定义学习率和一些超参数
    C=100
    gamma=0.95
    epsilon=0.000001
    #初始化所有参数，这里全部采用全零初始化
    w_AdaD=np.zeros((124,1))
    G_AdaD=np.zeros((124,1))
    g_AdaD=np.zeros((124,1))
    deltaT=np.zeros((124,1))
    deltaW=np.zeros((124,1))
    P=np.zeros((1000,1))
    L_AdaD=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((300,124))
        z_y=np.zeros((300,1))
        for j in range(300):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        g_AdaD=grad(w_AdaD,z_x,z_y,C)
        G_AdaD=gamma*G_AdaD+(1-gamma)*g_AdaD*g_AdaD
        deltaW=-np.sqrt((deltaT+epsilon)/(G_AdaD+epsilon))*g_AdaD
        w_AdaD=w_AdaD+deltaW
        deltaT=gamma*deltaT+(1-gamma)*deltaW*deltaW
        L_AdaD[i]=loss(w_AdaD,X_test,y_test,C)
        P[i]=predict(w_AdaD,X_test,y_test)
    print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
    print("Lowest  loss:  iteration  %d,  loss  %f"  %(L_AdaD.argmin()+1,
L_AdaD.min()))
    x=np.arange(0,1000,1)
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_AdaD,label='L_AdaD')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
    plt.ylabel('Loss')
    plt.show()

    #定义学习率和一些超参数
    C=100
    eta=0.002
```

```
    gamma=0.999
    belta=0.9
    epsilon=0.00000001
    #初始化所有参数，这里全部采用全零初始化
    P=np.zeros((1000,1))
    w_Adam=np.zeros((124,1))
    g_Adam=np.zeros((124,1))
    mt=np.zeros((124,1))
    G_Adam=np.zeros((124,1))
    L_Adam=np.zeros((1000,1))
    #迭代 1000 次
    for i in range(1000):
        z_x=np.zeros((300,124))
        z_y=np.zeros((300,1))
        for j in range(300):
            r = np.random.randint(m_train)
            z_x[j,:] = X_train[r,:]
            z_y[j]=y_train[r]
        g_Adam=grad(w_Adam,z_x,z_y,C)
        mt=belta*mt+(1-belta)*g_Adam
        G_Adam=gamma*G_Adam+(1-gamma)*g_Adam*g_Adam
        alpha=eta*np.sqrt(1-gamma)/(1-belta)
        w_Adam=w_Adam-alpha*mt/np.sqrt(G_Adam+epsilon)
        L_Adam[i]=loss(w_Adam,X_test,y_test,C)
        P[i]=predict(w_Adam,X_test,y_test)
    print("Best result:  iteration %d,  accuracy %f" %(P.argmax()+1, P.max()))
    print("Lowest  loss:  iteration  %d,  loss  %f"  %(L_Adam.argmin()+1,
L_Adam.min()))
    x=np.arange(0,1000,1)
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_Adam,label='L_Adam')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
    plt.ylabel('Loss')
    plt.show()

    plt.rcParams['figure.figsize'] = (10.0, 8.0)
    plt.plot(x,L_SGD,label='L_SGD')
    plt.plot(x,L_NAG,label='L_NAG')
    plt.plot(x,L_RMS,label='L_RMS')
    plt.plot(x,L_AdaD,label='L_AdaD')
    plt.plot(x,L_Adam,label='L_Adam')
    plt.legend(loc='upper right')
    plt.xlabel('Num of iterations')
```

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

逻辑回归和随机梯度下降 8-11：

8.模型参数的初始化方法:

全零初始化。

9.选择的 loss 函数及其导数:

Loss 函数： $L = \frac{1}{m}\sum_{n} -[y_n \ln f(x_n) + (1 - y_n)(1 - \ln f(x_n))]$

其中 $f(x_n) = \sigma(z) = \frac{1}{1+e^{-z}}$， $z = \sum w_i x_i + b$

梯度： w_i 的梯度 $G = \sum_{n}^{m} -(y_i - f(x_n))x_i^n$

（对于 w14(b)而言 x=1）

则整个向量 $\mathbf{w}$ 的梯度向量 $\mathbf{G} = \frac{1}{m}\mathbf{X}^T(\mathbf{Xw} - \mathbf{y})$

10.实验结果和曲线图:

超参数选择：

SGD 的学习率 α 取 0.1

NAG 的学习率 α 取 0.1, γ 取 0.9

RMSProp 的学习率 α 取 0.001, γ 取 0.9, ε 取 1e-8

AdaDelta 的 γ 取 0.95, ε 取 1e-6

Adam 的学习率 α 取 0.002, γ 取 0.999, ε 取 1e-8, β 取 0.9

预测结果（最佳结果）：

SGD：

```
Best result:  iteration 997,  accuracy 0.848351
Lowest loss:  iteration 998,  loss 0.330177
```

NAG：

```
Best result:  iteration 699,   accuracy 0.852098
Lowest loss:  iteration 413,   loss 0.323873
```

RMSProp：

```
Best result:  iteration 988,   accuracy 0.849948
Lowest loss:  iteration 999,   loss 0.328353
```

AdaDelta：

```
Best result:  iteration 819,   accuracy 0.852650
Lowest loss:  iteration 911,   loss 0.323587
```

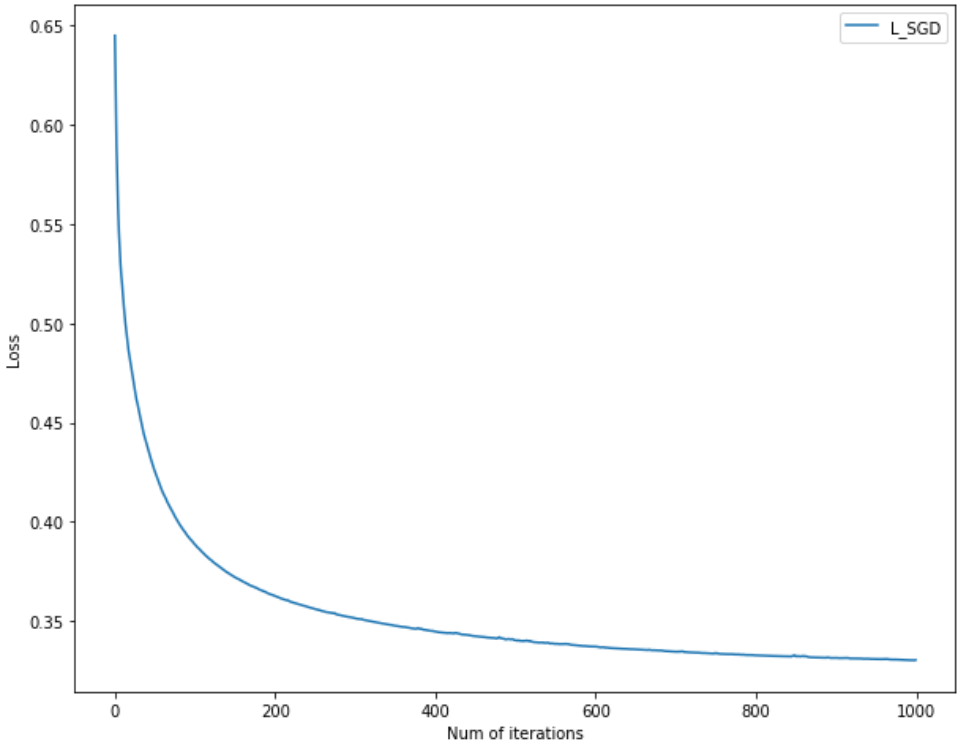
Adam：

```
Best result:  iteration 990,   accuracy 0.848904
Lowest loss:  iteration 1000,   loss 0.331318
```

loss 曲线图：

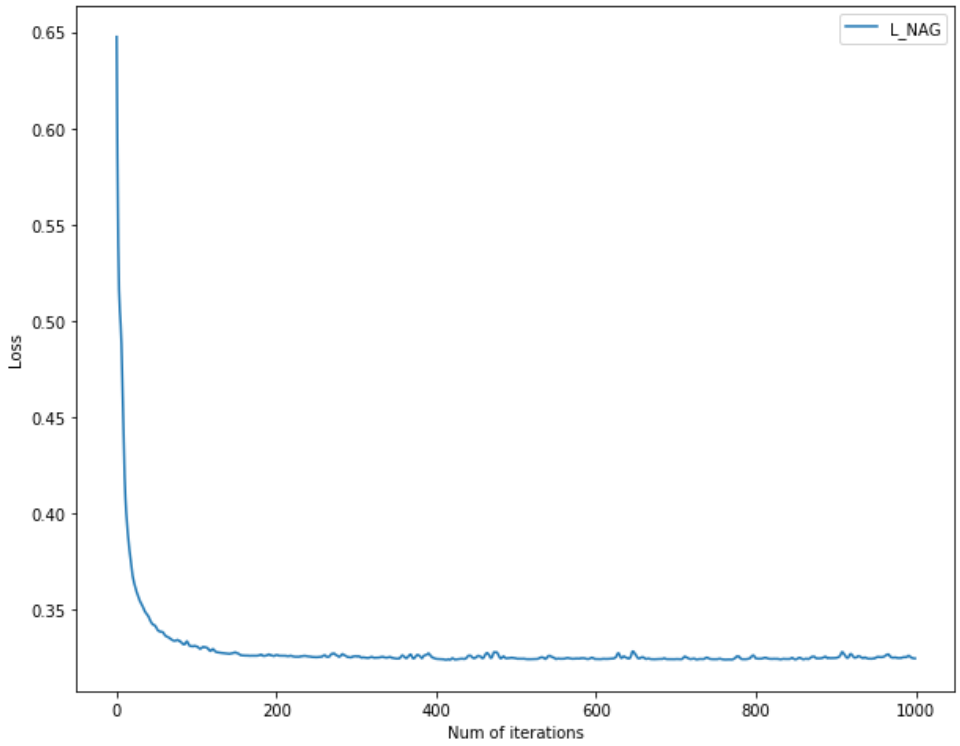
SGD：

```
Best result:  iteration 997,   accuracy 0.848351
Lowest loss:  iteration 998,   loss 0.330177
```



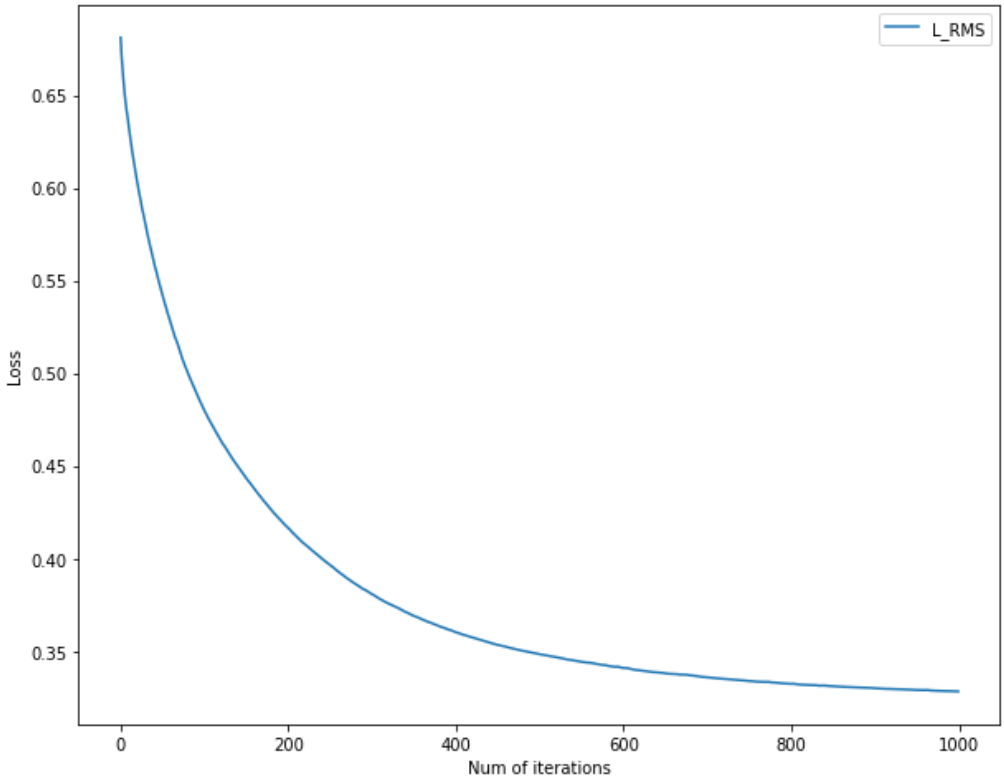
NAG：

```
Best result:  iteration 699,   accuracy 0.852098
Lowest loss:  iteration 413,   loss 0.323873
```



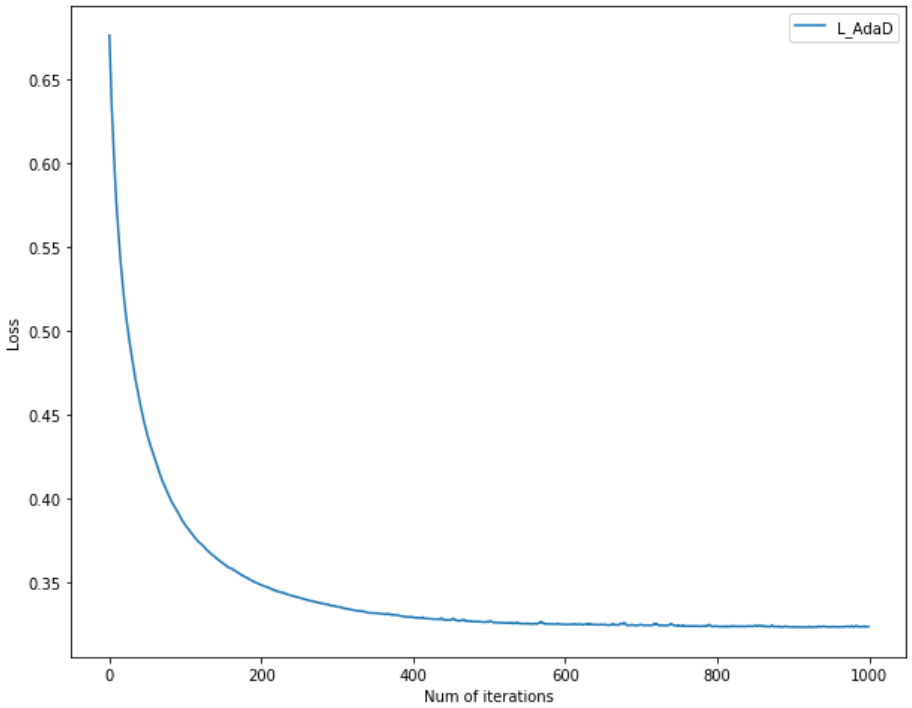
RMSProp：

```
Best result:  iteration 988,   accuracy 0.849948
Lowest loss:  iteration 999,   loss 0.328353
```



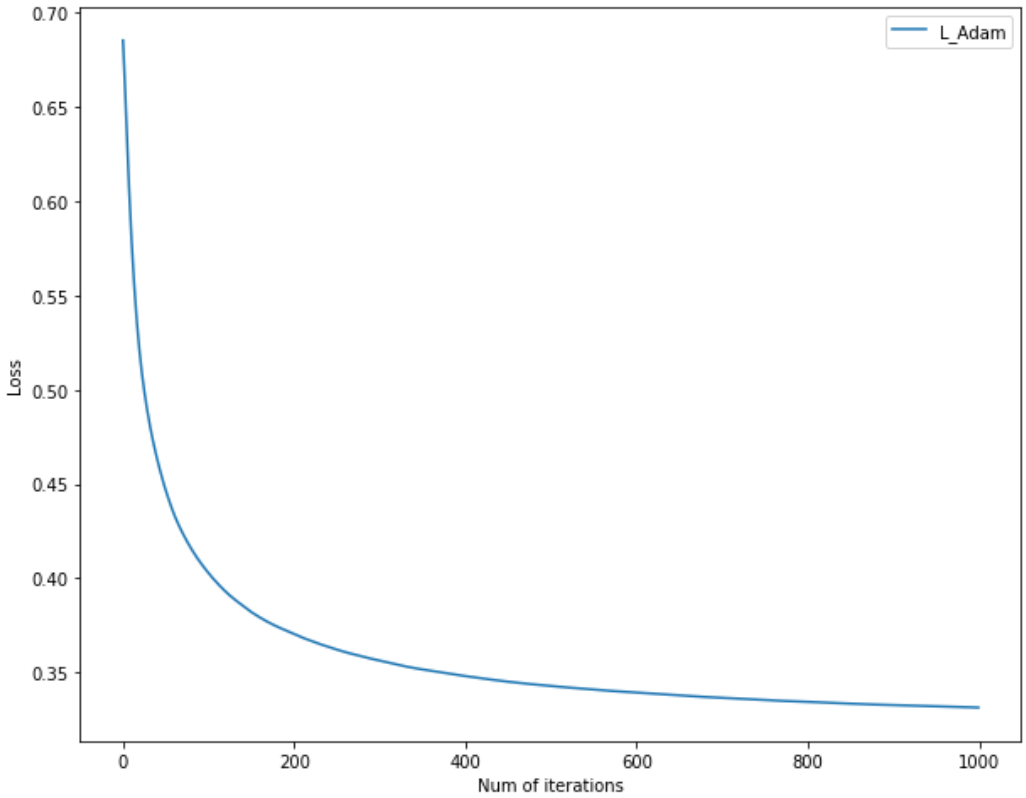
AdaDelta：

```
Best result:  iteration 819,   accuracy 0.852650
Lowest loss:  iteration 911,   loss 0.323587
```

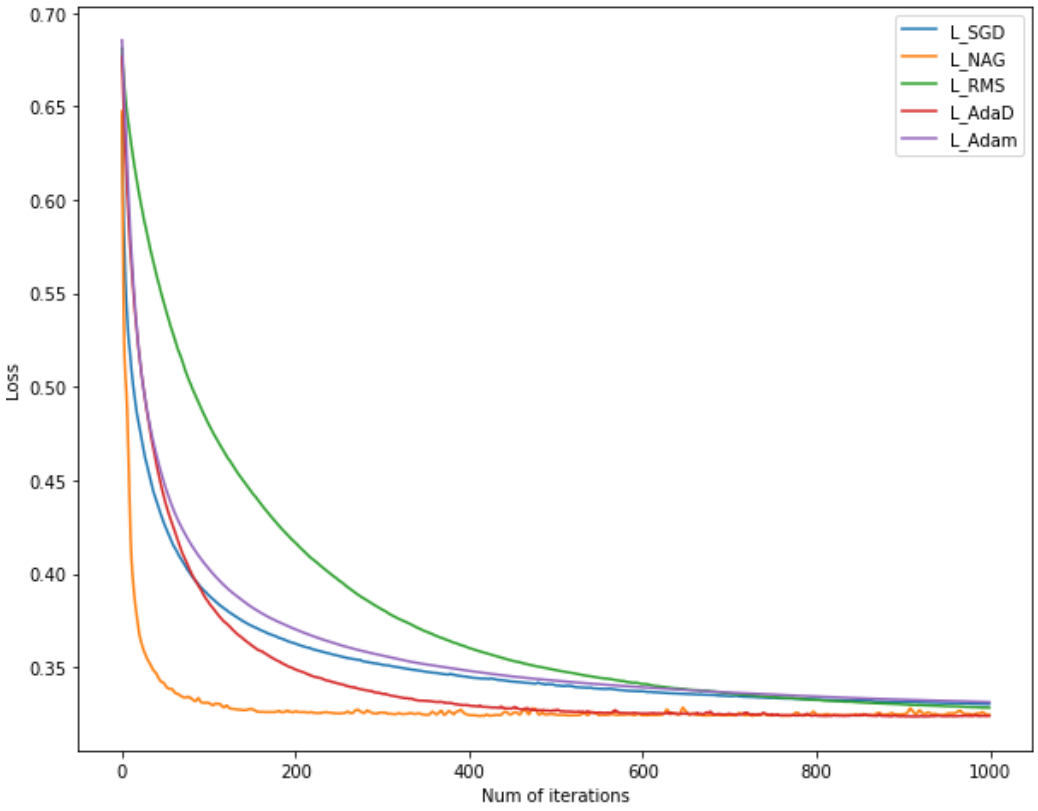


Adam：

```
Best result:  iteration 990,   accuracy 0.848904
Lowest loss:  iteration 1000,   loss 0.331318
```



五种汇总图：



11.实验结果分析:

5 种优化模型均在前 200 次迭代内就能使测试集上的 Loss 值大幅下降，然后在 200-10000 次迭代内 Loss 值缓慢地下降，最终达到 0.33 左右，此时在测试集上对应的正确率均可达 84%以上。

NAG 和 AdaDelta 模型最后的 loss 很相近，而且比另外三种模型低一些，并且两者的最高准确率达到了 85.2%，高于另外三种模型

RMSProp，SGD 和 Adam 模型的曲线比较相近，其中 RMSProp 的前 600 次迭代比另外两种稍微慢一点，600 之后基本相同，并且最后的准确率和 loss 都很相近。

线性分类和随机梯度下降 8-11：

8.模型参数的初始化方法:

全零初始化。

9.选择的 loss 函数及其导数:

Loss 函数：$L = \frac{1}{m}\sum_{m}(\max(0,1 - y^{m}(wx^{m} + b)) + \frac{1}{2}||w||^{2}$

梯度：对 wi 的梯度 $G = \frac{1}{n}\sum_{n} -\delta(y^n(w_i x^n + b)y^n x^i$

10.实验结果和曲线图:

超参数选择：

SGD 的学习率 α 取 0.0005，C=100

NAG 的学习率 α 取 0.0001, γ 取 0.9,C=100

RMSProp 的学习率 α 取 0.001, γ 取 0.9, ε 取 1e-8,C=100

AdaDelta 的 γ 取 0.95, ε 取 1e-6,C=100

Adam 的学习率 α 取 0.002, γ 取 0.999, ε 取 1e-8, β 取 0.9,C=100

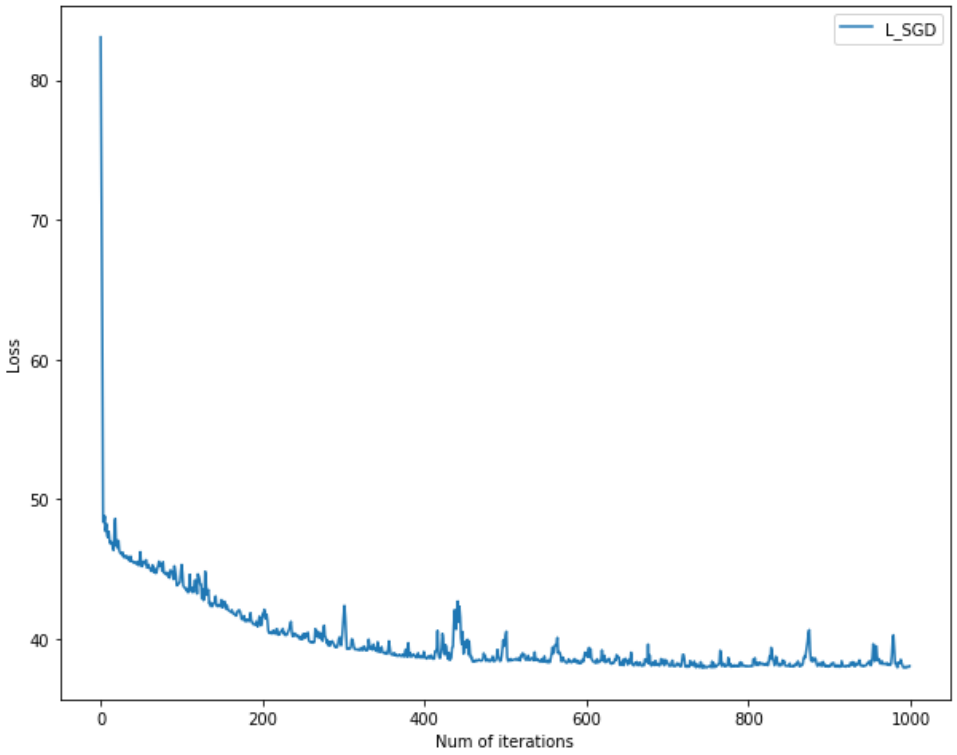
预测结果（最佳结果）：

```
SGD
    Best result:  iteration 872,  accuracy 0.845587
    Lowest loss:  iteration 744,  loss 37.945012
NAG
    Best result:  iteration 746,  accuracy 0.846140
    Lowest loss:  iteration 457,  loss 37.853306
RMSProp
    Best result:  iteration 998,  accuracy 0.848658
    Lowest loss:  iteration 690,  loss 38.435086
AdaDelta
    Best result:  iteration 740,  accuracy 0.850378
    Lowest loss:  iteration 280,  loss 38.685186
Adam
    Best result:  iteration 982,  accuracy 0.847491
    Lowest loss:  iteration 732,  loss 38.837161
```

loss 曲线图：

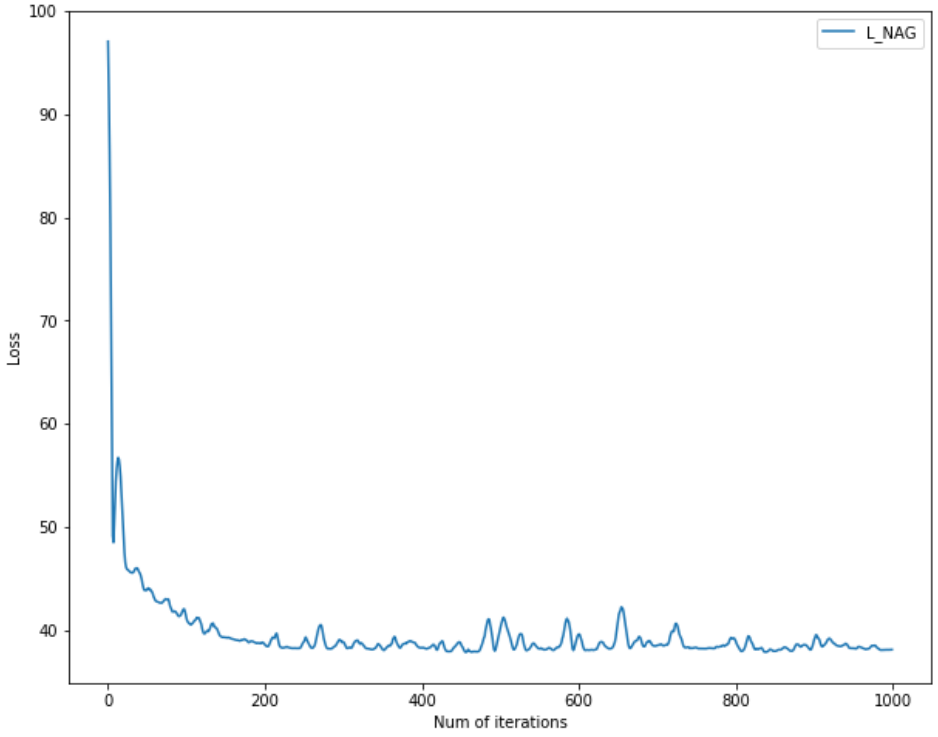
SGD:

```
Best result:  iteration 872,  accuracy 0.845587
Lowest loss:  iteration 744,  loss 37.945012
```



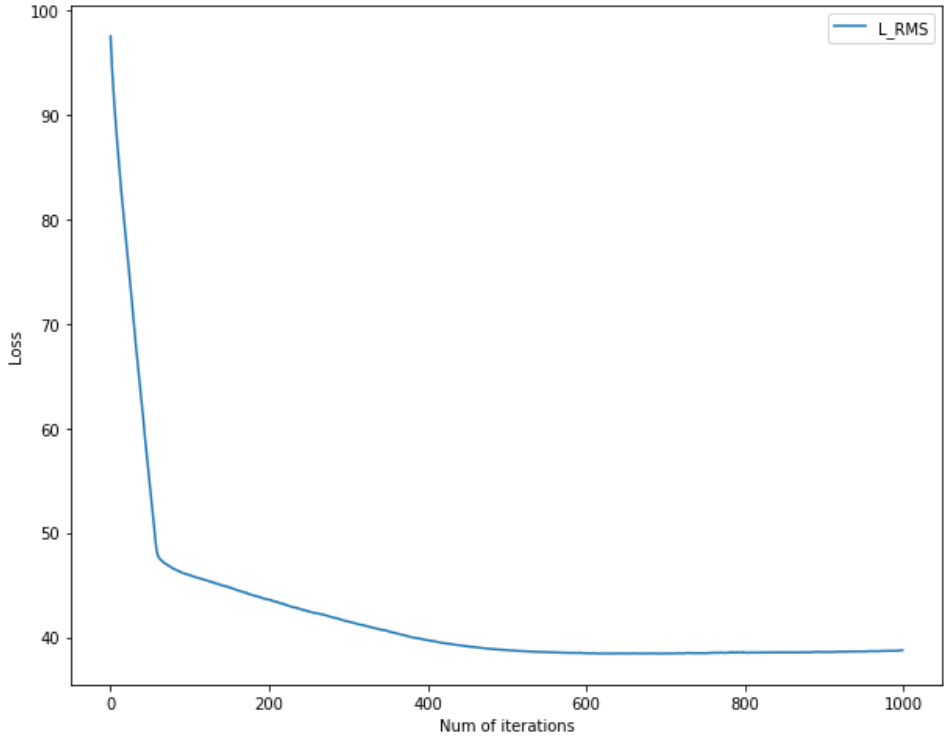
NAG:

```
Best result:  iteration 746,  accuracy 0.846140
Lowest loss:  iteration 457,  loss 37.853306
```



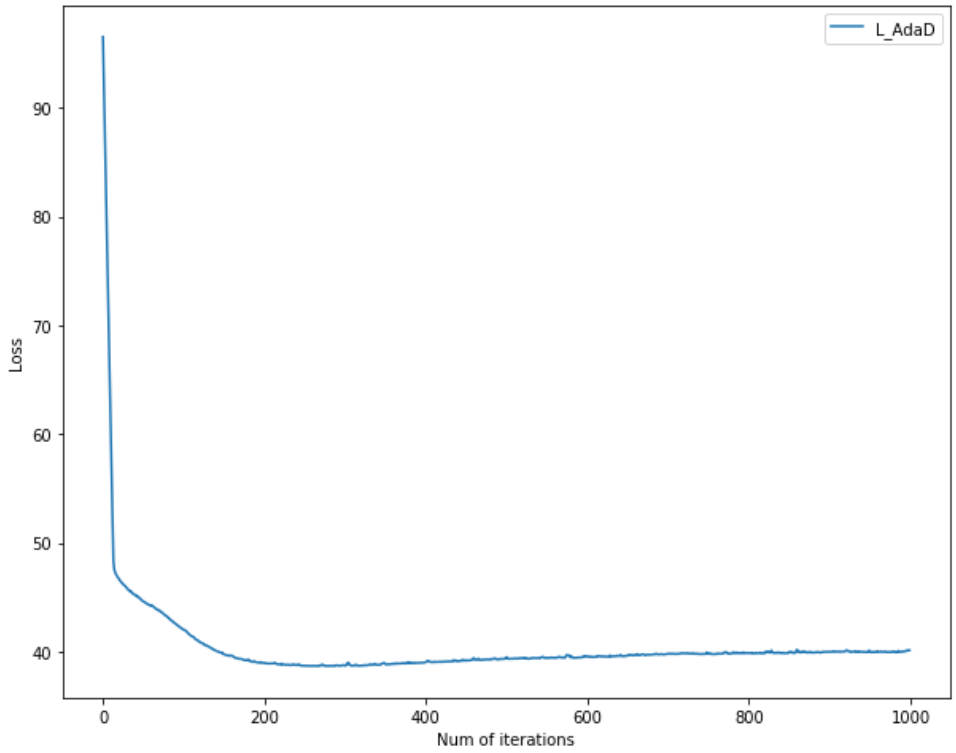
RMSProp:

```
Best result:  iteration 998,   accuracy 0.848658
Lowest loss:  iteration 690,   loss 38.435086
```



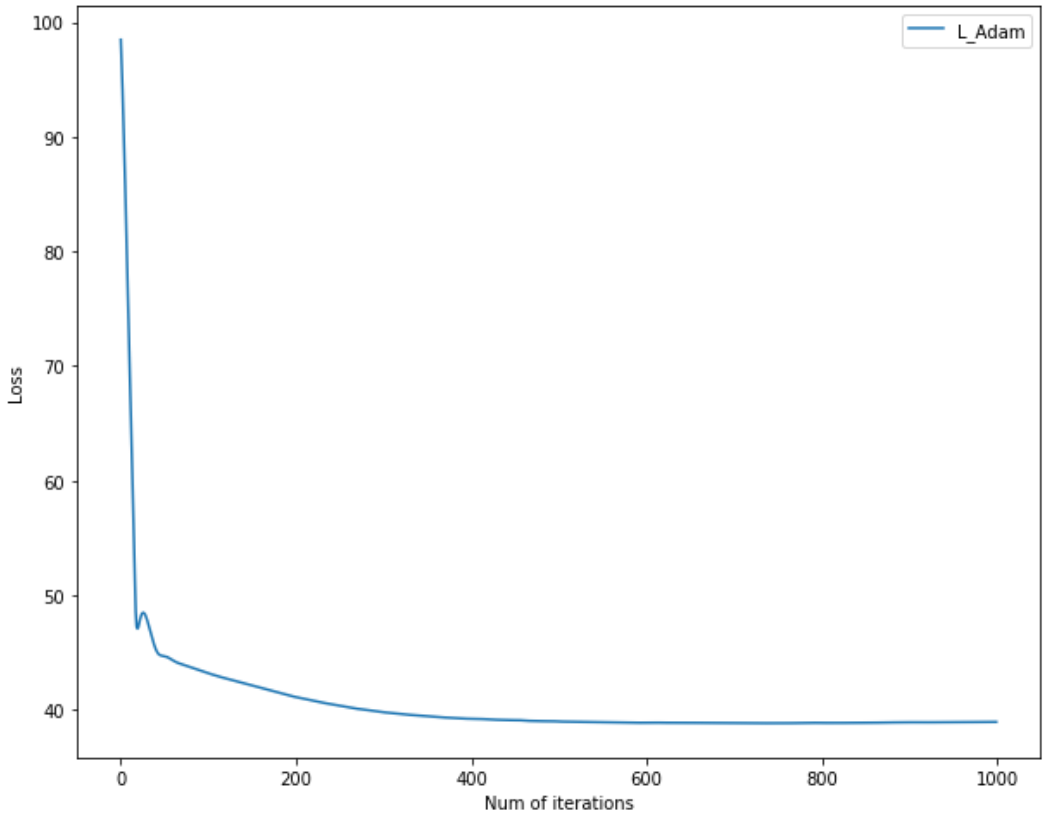
AdaDelta:

```
Best result:  iteration 740,   accuracy 0.850378
Lowest loss:  iteration 280,   loss 38.685186
```

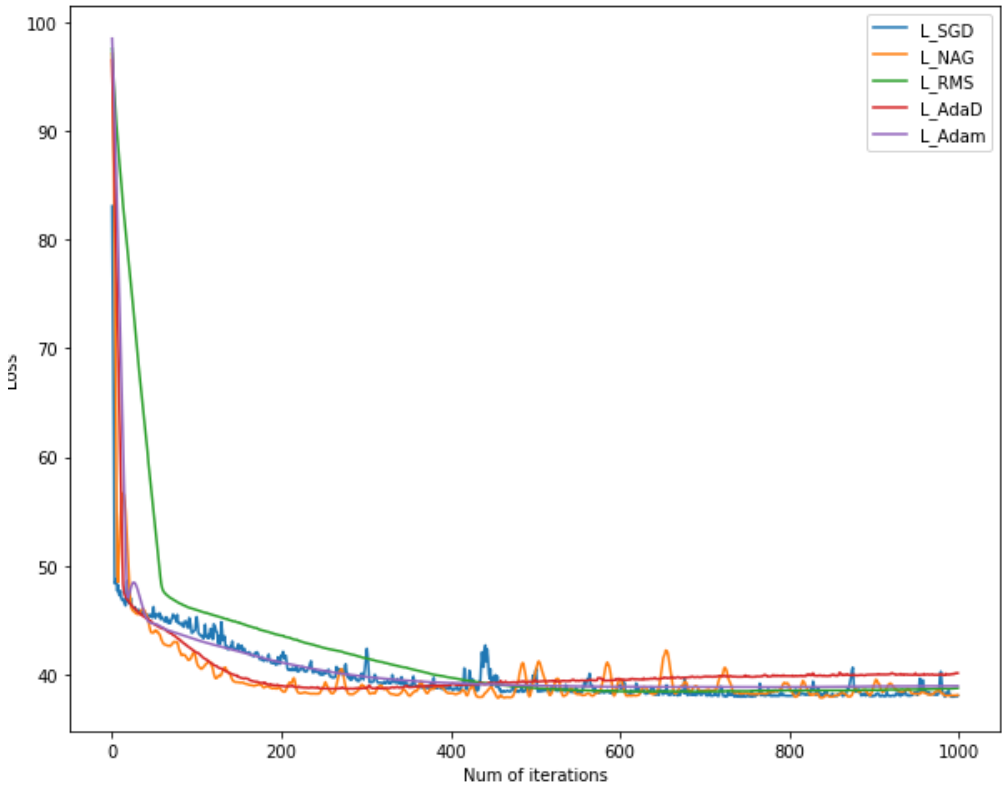


Adam;

```
Best result:  iteration 982,   accuracy 0.847491
Lowest loss:  iteration 732,   loss 38.837161
```



五种汇总图：



11.实验结果分析:

5 种优化模型均在前 200 次迭代内就能使测试集上的 Loss 值大幅下降，然后在 200-10000 次迭代内 Loss 值缓慢地下降，最终达到 38 左右，此时在测试集上对应的正确率均可达 84%以上。

AdaDelta 在测试集上最高可以达到 85%以上的准确率，而且算法也比较稳定。

以上 5 种优化模型的曲线除了 RMSProp 在整个迭代中均有不同程度的上下波动，其中 SGD 和 NAG 的波动程度比较大。

12.对比逻辑回归和线性分类的异同点：

两种方法都是常见的分类算法,从目标函数来看,区别在于逻辑回归采用的是 logistical loss,svm 采用的是 hinge loss.这两个损失函数的目的都是增加对分类影响较大的数据点的权重,减少与分类关系较小的数据点的权重.SVM 的处理方法是只考虑 support vectors,也就是和分类最相关的少数点,去学习分类器.而逻辑回归通过非线性映射,大大减小了离分类平面较远的点的权重,相对提升了与分类最相关的数据点的权重.两者的根本目的都是一样的.此外,根据需要,两个方法都可以增加不同的正则化项,如 l1,l2 等等.所以在很多实验中,两种算法的结果是很接近的

但是逻辑回归相对来说模型更简单,好理解,实现起来,特别是大规模线性分类时比较方便.而 SVM 的理解和优化相对来说复杂一些.但是 SVM 的理论基础更加牢固,有一套结构化风险最小化的理论基础,虽然一般使用的人不太会去关注.还有很重要的一点,SVM 转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算

svm 更多的属于非参数模型,而 logistic regression 是参数模型,本质不同.其区别就可以参考参数模型和非参模型的区别就好了

logic 能做的 svm 能做,但可能在准确率上有问题,svm 能做的 logic 有的做不了

13.实验总结：

通过本次实验，我深入的学习了逻辑回归以及采用 SVM 实现的线性分类模型的实现和参数调试，学习了如何利用 Jupyter Notebook 来编写模型代码，了解了 sklearn，numpy，jupyter，matplotlib 包的一些基本操作。通过自己手写代码，让我进一步理解了逻辑回归、线性分类和梯度下降的原理，并且在小规模的数据集上实践，通过绘制图像来可视化优化和调参的过程，让这种过程给自己更深的体会和收获，在试验过程总，我有几次打错了，算法的代码，都是通过绘制出的 Loss-Iterations 图观察出来的，还有在调参的过程中通过更改参数来观察图形的变化让我直观地体会到了调参的过程。通过这次试验，让我对于机器学习有了真实的接触，为我之后的实验打下了必要的的基础，并在过程中收获了对于不同问题的处理方法，积累了宝贵的经验。