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　苏越

学　　号　201530612774

邮　　箱　1090211523@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 9 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: 苏越

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。

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L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 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。**

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）**。
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L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 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:

逻辑回归：

```
def loss_function(X, y ,theta):
        y_prediction = sigmod(X.dot(theta))
        loss = -1./X.shape[0] * (y*np.log(y_prediction)+ (1-y)*np.log(1-y_prediction)).sum()
        return loss

def gradient(X, y, theta):
        gradient = 1./X.shape[0] * np.dot(X.transpose(), sigmod(X.dot(theta))-y)
        return gradient
```

线性分类：

```
def gradient(X_train, Y_train, theta, C):
    index =  (1 - Y_train * X_train.dot(theta) < 0)
    Y = Y_train.copy()
    Y[index] = 0
    epsilon_gradient = -np.dot(X_train.transpose(), Y)
    gradient = theta + C * epsilon_gradient
    return gradient

def loss_f(X, Y, theta, C=1):
    epsilon_loss = 1 - Y * X.dot(theta)
    epsilon_loss[epsilon_loss<0] = 0
    loss = 0.5 * np.dot(theta.transpose(), theta).sum() + C*epsilon_loss.sum()
    return loss/X.shape[0]
```

优化算法：

NAG:

```
Velocity = optimizer_params.setdefault('Velocity', np.zeros(theta.shape))
momemtum = optimizer_params.setdefault('momemtum', 0.9)

grad = gradient(X_train, y_train, theta+ momemtum*Velocity)
Velocity = momemtum*Velocity - learning_rate*grad
theta += Velocity
```

RMSProp：

```
Velocity = optimizer_params.setdefault('Velocity', np.zeros(theta.shape))
decay_rate = optimizer_params.setdefault('decay_rate', 0.9)
epsilon = optimizer_params.setdefault('epsilon', 1e-7)

grad = gradient(X_train, y_train, theta)
Velocity = decay_rate*Velocity + (1-decay_rate)*(grad**2)
theta -= learning_rate*grad/(np.sqrt(epsilon) + Velocity)
```

AdaDelta：

```
Velocity = optimizer_params.setdefault('Velocity', np.zeros(theta.shape))
update_accumulate = optimizer_params.setdefault('update_accumulate', np.zeros(theta.shape))
decay_rate = optimizer_params.setdefault('decay_rate', 0.9)
epsilon = optimizer_params.setdefault('epsilon', 1e-7)

grad = gradient(X_train, y_train, theta)
Velocity = decay_rate*Velocity + (1-decay_rate)*(grad**2)
step_update = -(np.sqrt(update_accumulate+epsilon))*grad/(np.sqrt(Velocity+epsilon))

theta +=  step_update
update_accumulate = decay_rate*update_accumulate + (1-decay_rate)*(step_update**2)
```

Adam：

```
Velocity = optimizer_params.setdefault('Velocity', np.zeros
S = optimizer_params.setdefault('S', np.zeros(theta.shape))
beta1 =  optimizer_params.setdefault('beta1', 0.9)
beta2 = optimizer_params.setdefault('beta2', 0.999)
epsilon = optimizer_params.setdefault('epsilon', 1e-8)

grad = gradient(X_train, y_train, theta)
S = beta1*S + (1-beta1)*grad
Velocity = beta2*Velocity + (1-beta2)*(grad**2)

S_t = S/(1 - (beta1**episode))
Velocity_t = Velocity/(1- (beta2**episode))
step_update = - learning_rate * S_t/ (np.sqrt(Velocity_t) + epsilon)
theta += step_update
```

（针对逻辑回归和线性分类分别填写 8-11 内容）

8. 模型参数的初始化方法:

逻辑回归：随机初始化
线性分类：随机初始化

9.选择的 loss 函数及其导数:

逻辑回归：

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

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n}\left(h_{\mathbf{w}}(\mathbf{x}_i) - y\right)\mathbf{x}_i$$

线性分类：

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} a_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

$$\frac{\partial f(\mathbf{w}, b)}{\mathbf{w}} = \begin{cases} \mathbf{w}^\top - C\mathbf{y}^\top\mathbf{X} & 1| - y_i(\mathbf{w}^\top\mathbf{x}_i + b) >= 0 \\ \mathbf{w}^\top & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < 0 \end{cases}$$

10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

逻辑回归：

超参数选择：learning_rate 0.001　epoch 2000

threshold 0.5

预测结果（最佳结果）：　无优化算法：0.679463797806

NAG: 0.379842727849

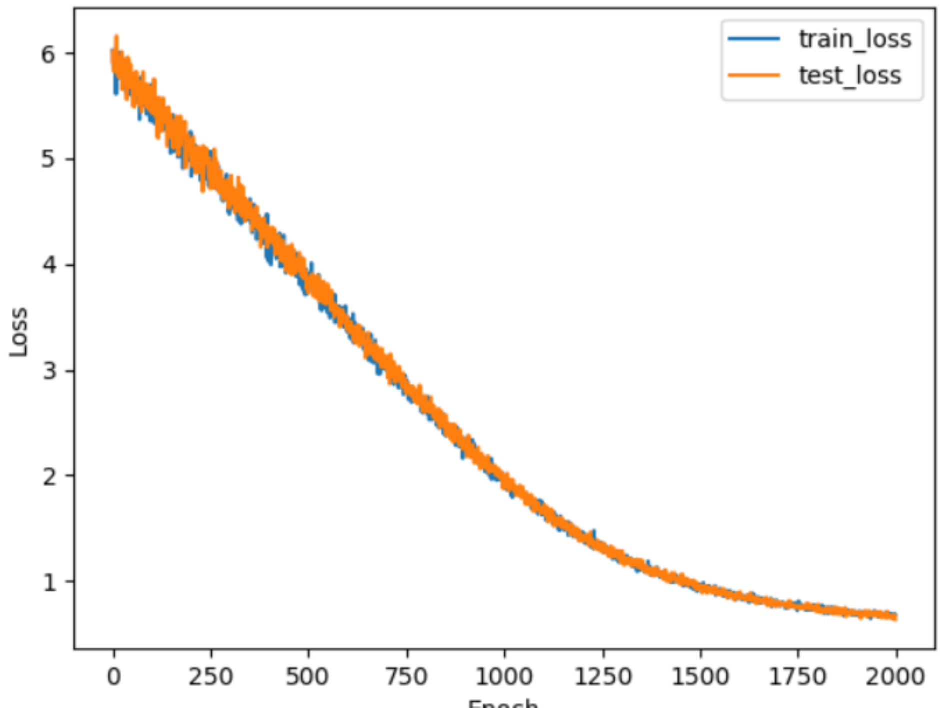
RMSProp：0.331719403837

Adadelta：0.353804335504

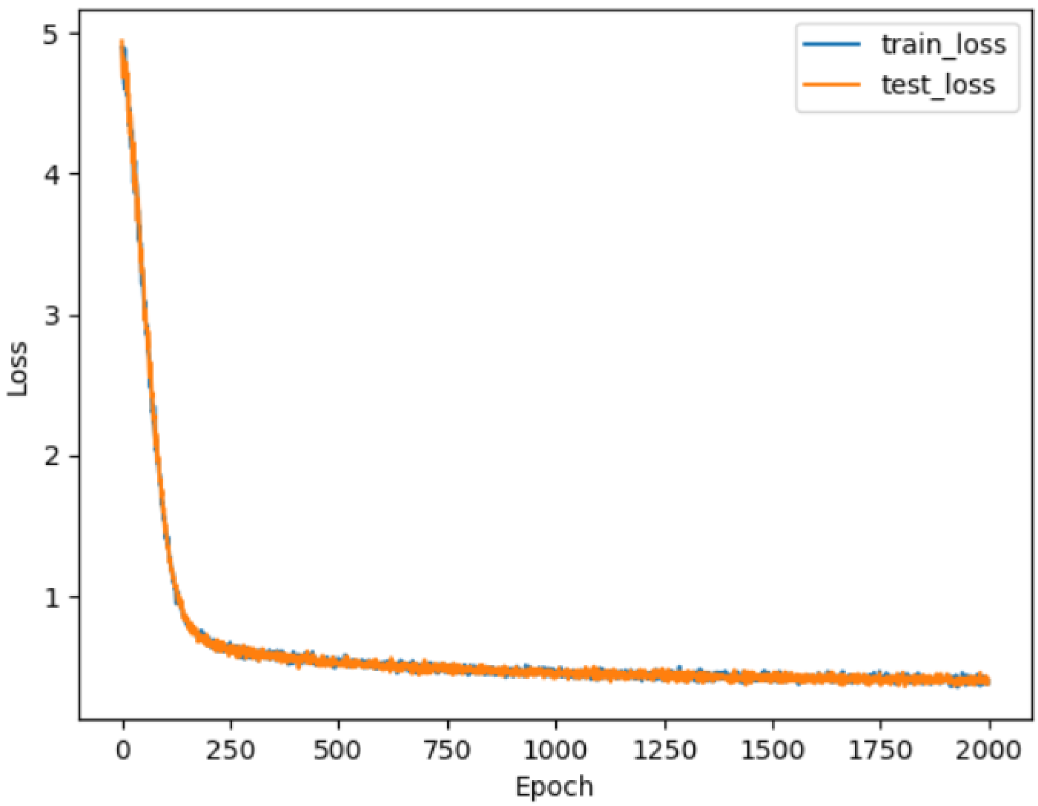
Adam：0.355391586534

loss 曲线图：

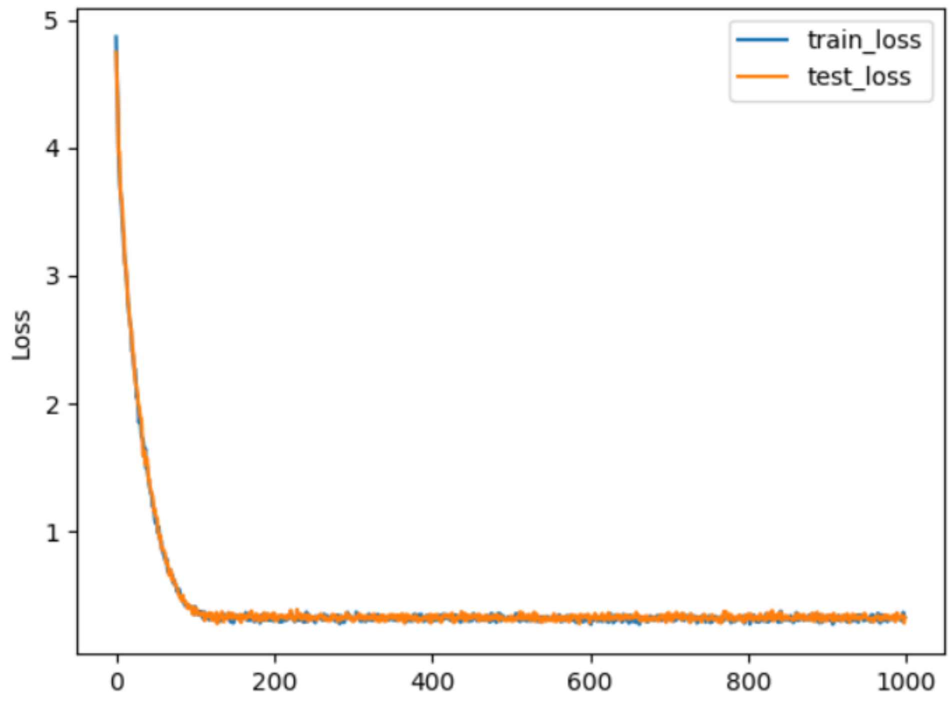
无优化算法：


train_loss
test_loss
6
5
4
3
2
1
Loss
0
250
500
750
1000
1250
1500
1750
2000

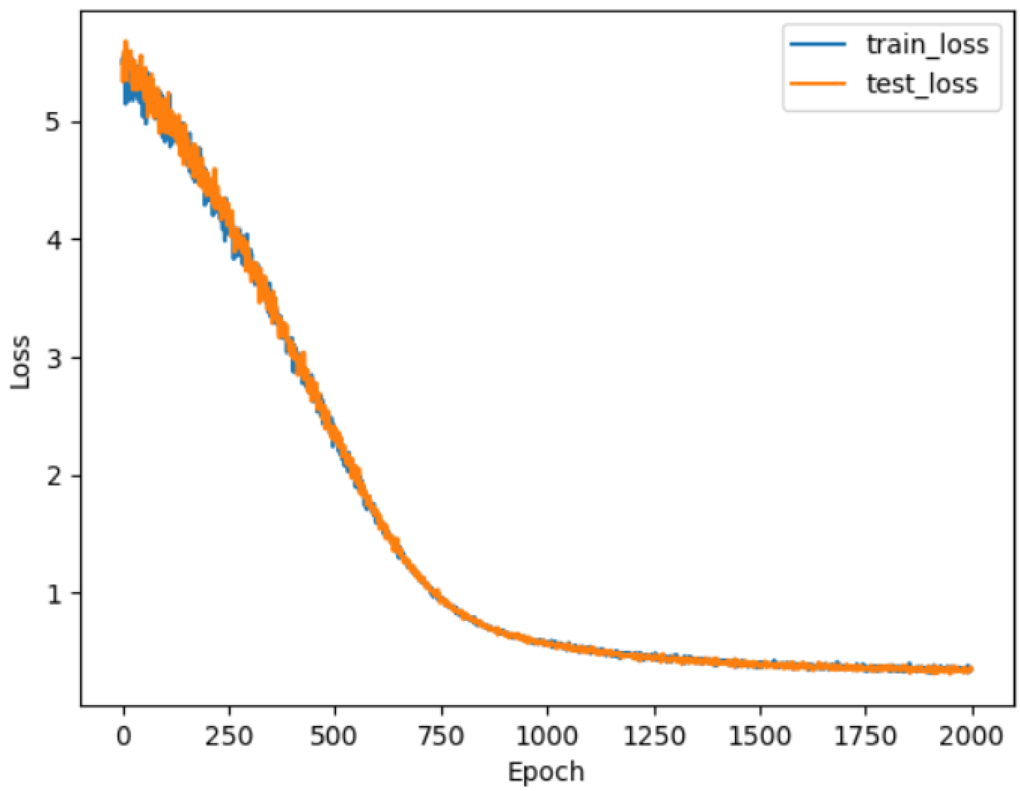
NAG:


train_loss
test_loss
5
4
3
2
1
Loss
0
250
500
750
1000
1250
1500
1750
2000
Epoch

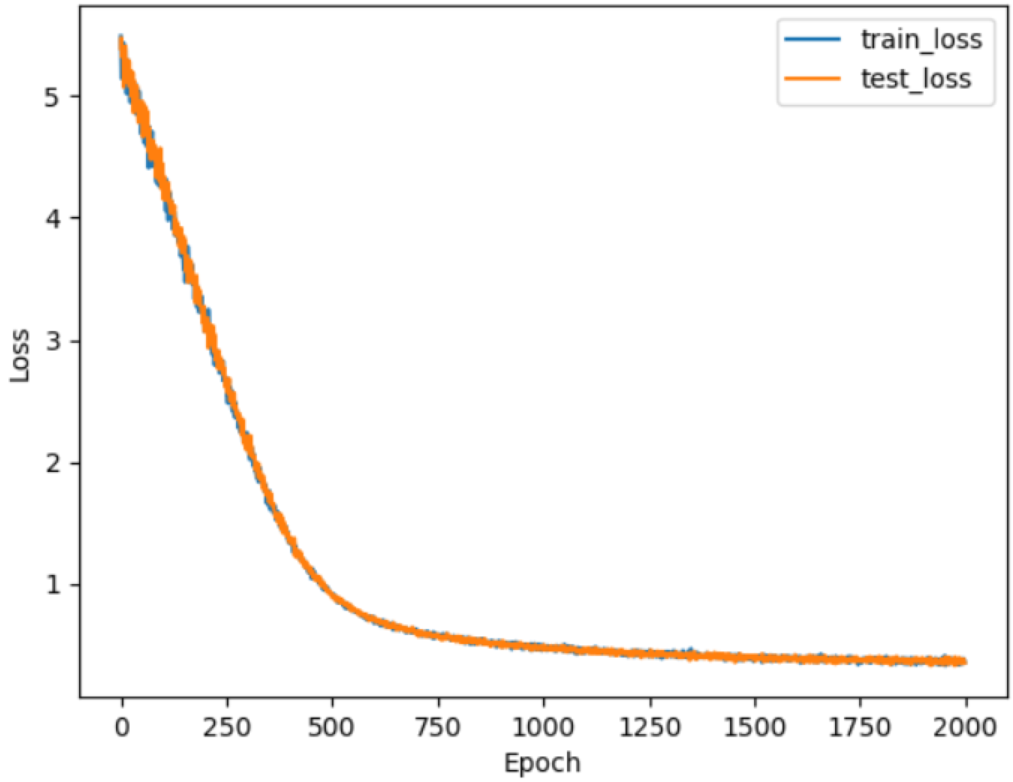
RMSProp：：



Adadelta：



Adam：



线性分类：

超参数选择：learning_rate 0.003　epoch 2000

threshold 0

预测结果（最佳结果）：　无优化算法：0.00762303880575

NAG: 0.00802919137105

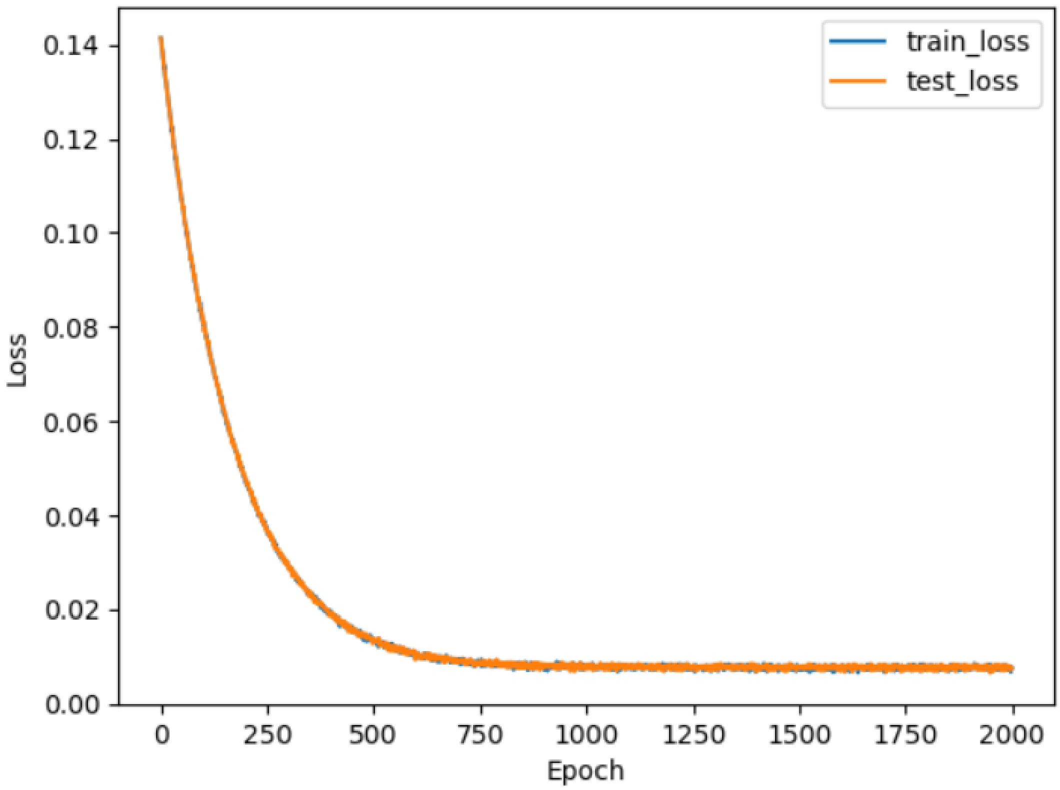
RMSProp：0.00788777036583

Adadelta：0.00790246771568

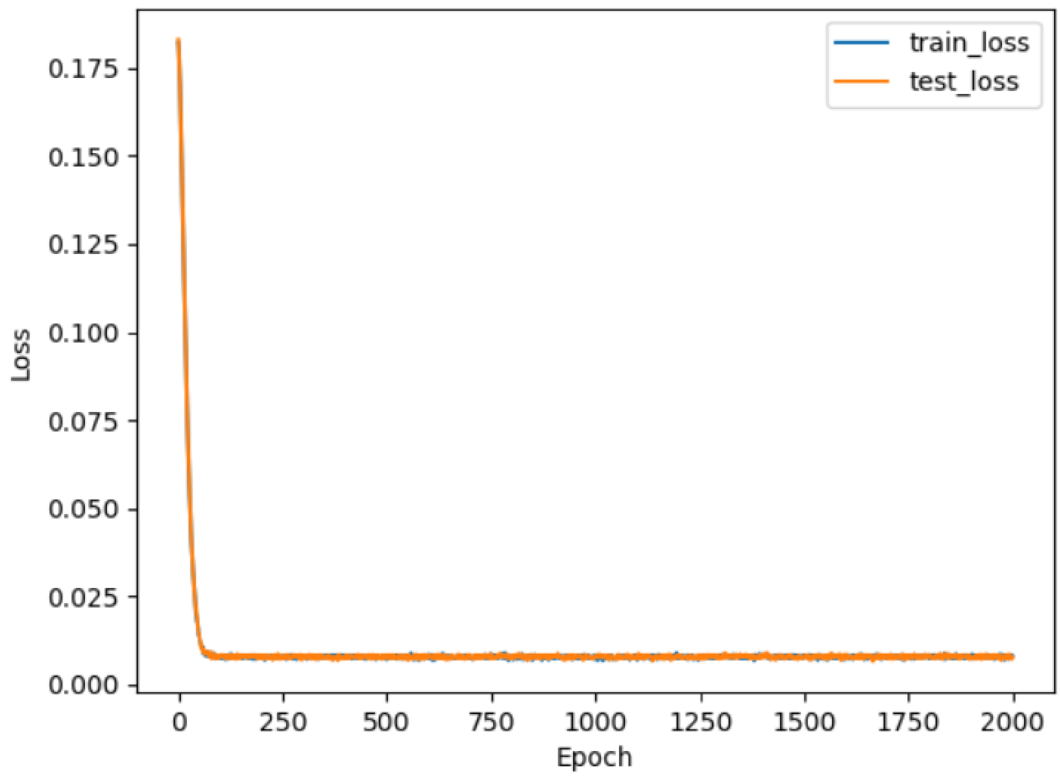
Adam：0.00790097634716

loss 曲线图：

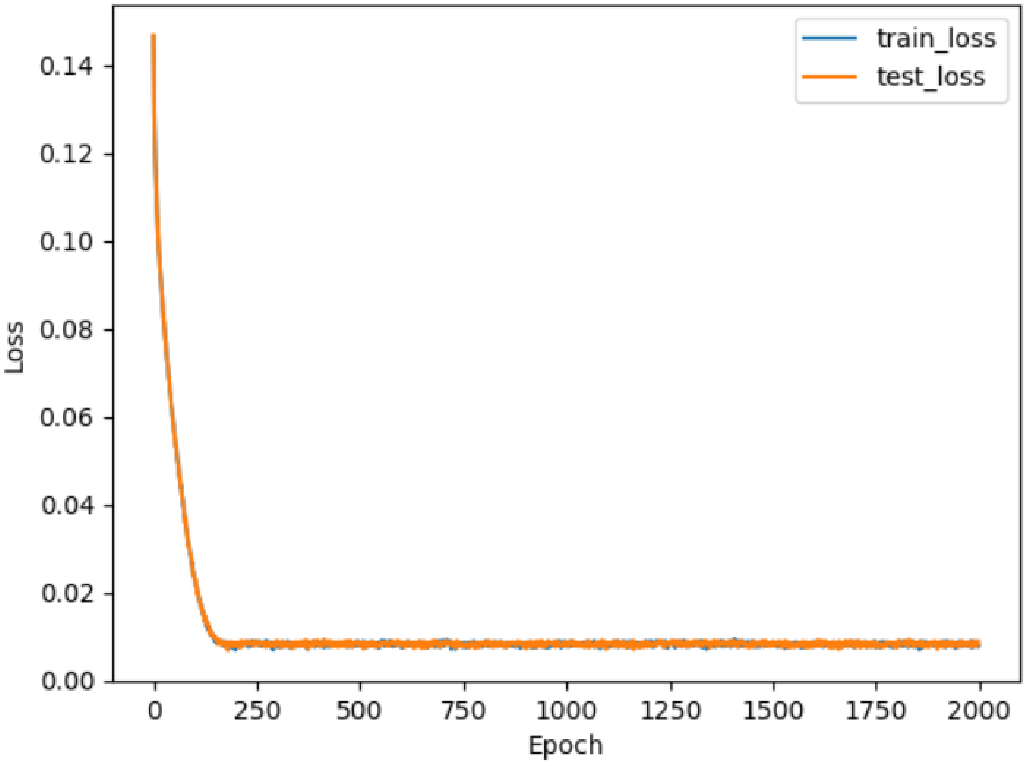
无优化算法：


train_loss
test_loss
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Loss
0
250
500
750
1000
1250
1500
1750
2000
Epoch

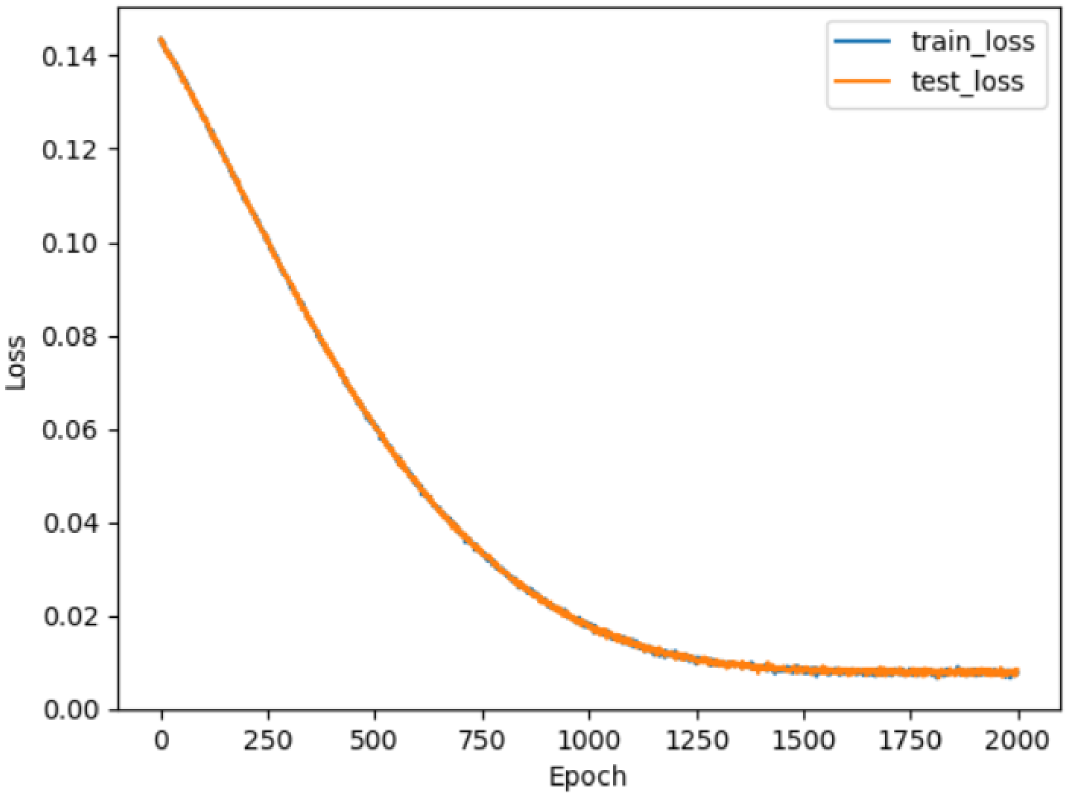
NAG:


train_loss
test_loss
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Loss
0
250
500
750
1000
1250
1500
1750
2000
Epoch

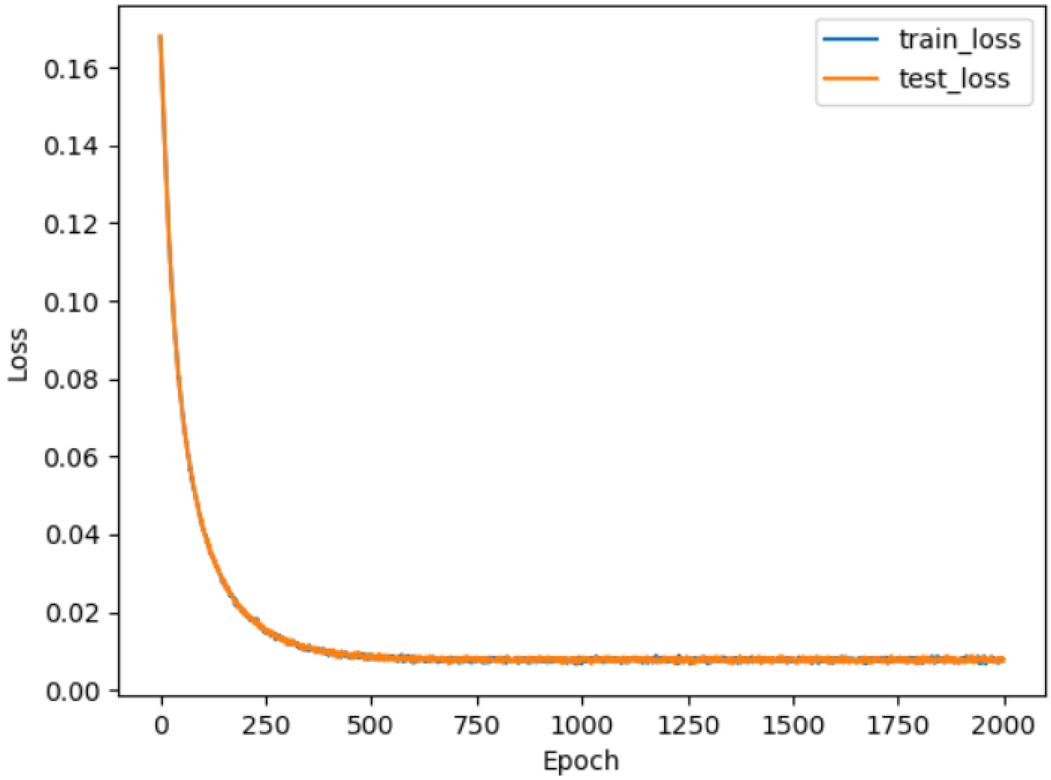
RMSProp:



Adadelta：



Adam：



11.实验结果分析

逻辑回归中 4 种优化算法结果比不用优化算法的结果好，而线性分类用不用优化算法结果差距不大。

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。svm 更多的属于非参数模型,而 logistic regression 是参数模型。

13.实验总结：

逻辑回归和线性分类都是分类算法，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。