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　　林兆桃

学　　号　　201530612309

邮　　箱　　821575189@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：林兆桃

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。a9a.test 缺一列属性。

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:

```
# -*- coding: utf-8 -*-
from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt
```

```
def sigmoid(z):
    return 1 /(1 + np.exp(-z))

def loss(weight,X,y):
    z = np.matmul(X, weight)
    loss_ = -np.mean(y * np.log(sigmoid(z)) + (1 - y) * np.log(1 - sigmoid(z)))
    return loss_

def grad(weight,X,y):
    z = np.matmul(X, weight)
    h = sigmoid(z)
    error = h - y
    gradient = np.matmul(X.transpose(), error) / y.shape[0]
    return gradient

def SGD(weight,X,y):
    learning_rate = 0.01
    weight -= learning_rate * grad(weight,X,y)
    return weight

def NAG(weight,X,y,NAG_v):
    learning_rate = 0.01
    gamma = 0.9
    gradient = grad(weight - gamma * NAG_v,X,y)
    next_NAG_vector = gamma * NAG_v + learning_rate * gradient
    weight -= next_NAG_vector
    return weight,next_NAG_vector

def RMSProp(weight,X,y,RMSProp_Gt):
    gamma = 0.9
    epsilon = 10e-8
    learning_rate = 0.005
    gradient = grad(weight,X,y)
    RMSProp_Gt = gamma * RMSProp_Gt + (1 - gamma) * gradient**2
    weight -= learning_rate * gradient / np.sqrt(RMSProp_Gt + epsilon)
    return weight,RMSProp_Gt

def AdaDelta(weight,X,y,AdaDelta_vector_Gt,AdaDelta_vector_t):
    gamma = 0.95
    epsilon = 10e-6
    gradient = grad(weight,X,y)
    AdaDelta_vector_Gt = gamma * AdaDelta_vector_Gt + (1-gamma) *
gradient**2
    delta=-1 * gradient * np.sqrt(AdaDelta_vector_t + epsilon) /
```

```
np.sqrt(AdaDelta_vector_Gt + epsilon)
        weight += delta
        AdaDelta_vector_t = gamma * AdaDelta_vector_t + (1 -gamma) * delta**2
        return weight,AdaDelta_vector_Gt,AdaDelta_vector_t

    def Adam(weight,X,y,Adam_vector_m,Adam_vector_Gt,t):
        beta = 0.9
        gamma = 0.999
        epsilon = 10e-8
        learning_rate = 0.1
        t += 1
        gradient = grad(weight,X,y)
        Adam_vector_m = beta * Adam_vector_m + (1-beta) * gradient
        Adam_vector_Gt = gamma * Adam_vector_Gt + (1 - gamma) * gradient**2
        alpha = learning_rate * np.sqrt(1 - gamma**t) / (1 - beta**t)
        weight -= alpha * Adam_vector_m / np.sqrt(Adam_vector_Gt + epsilon)
        return weight,Adam_vector_m,Adam_vector_Gt,t

    def 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]


    X_train, y_train = load_svmlight_file("a9a1")
    data_size,features=X_train.shape
    X_train = X_train.toarray()
    X_train = np.c_[np.ones(len(X_train)), X_train]
    for i in range(0, len(y_train)):
        if y_train[i] == -1:
            y_train[i] = 0
    X_test, y_test = load_svmlight_file("a9a2",n_features = features)
    X_test = 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])
```

```
optimizer=["SGD","NAG","RMSProp","AdaDelta","Adam"]
NAG_v = np.zeros([features + 1, 1])
RMSProp_Gt = np.zeros([features + 1,1])
AdaDelta_vector_Gt = np.zeros([features + 1,1])
AdaDelta_vector_t = np.zeros([features + 1,1])
Adam_vector_m = np.zeros([features+1,1])
Adam_vector_Gt = np.zeros([features+1,1])
t = 0
batch_size = 64

for index,j in enumerate(optimizer):
    weight = np.random.rand(features + 1, 1)
    iteration = []
    error = []
    for i in range(0, int(data_size / batch_size ) + 1):
        iteration.append(i)
        X,y = batch(i,X_train,y_train,data_size)
        if j == "SGD":
            weight = SGD(weight,X_train,y_train)
        elif j == "NAG":
            weight,NAG_v = NAG(weight,X_train,y_train,NAG_v)
        elif j == "RMSProp":
            weight,RMSProp_Gt =
RMSProp(weight,X_train,y_train,RMSProp_Gt)
        elif j == "AdaDelta":
            weight,AdaDelta_vector_Gt,AdaDelta_vector_t =
AdaDelta(weight,X_train,y_train,AdaDelta_vector_Gt,AdaDelta_vector_t)
        elif j == "Adam":
            weight,Adam_vector_m,Adam_vector_Gt,t =
Adam(weight,X_train,y_train,Adam_vector_m,Adam_vector_Gt,t)
        error.append(loss(weight,X_test,y_test))
    plt.plot(iteration, error, label=j)
plt.xlabel('iteration')
plt.ylabel('loss')
plt.legend()
plt.show()
```

线性分类：

```
# -*- coding: utf-8 -*-
from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt
```

```
def sigmoid(z):
    return 1 /(1 + np.exp(-z))

def loss(weight,X,y):
    C = 0.9
    hinge_loss_sum = 0.
    hinge_loss_sum = sum(np.maximum(0, (1 - y * np.matmul(X,weight))))
    loss_ = np.matmul(weight.T, weight)[0][0] /2 + (hinge_loss_sum * C)
/y.shape[0]
    return loss_

def grad(weight,X,y):
    C = 0.9
    gw = np.zeros((124,1))
    temp = 1 - y * np.matmul(X,weight)
    temp = np.maximum(temp /np.abs(temp),0)
    y = y * temp
    gw = -np.matmul(X.T,y)
    return (C * gw) + weight

def SGD(weight,X,y):
    learning_rate = 0.01
    weight -= learning_rate * grad(weight,X,y)
    return weight

def NAG(weight,X,y,NAG_v):
    learning_rate = 0.01
    gamma = 0.9
    gradient = grad(weight - gamma * NAG_v,X,y)
    next_NAG_vector = gamma * NAG_v + learning_rate * gradient
    weight -= next_NAG_vector
    return weight,next_NAG_vector

def RMSProp(weight,X,y,RMSProp_Gt):
    gamma = 0.9
    epsilon = 10e-8
    learning_rate = 0.005
    gradient = grad(weight,X,y)
    RMSProp_Gt = gamma * RMSProp_Gt + (1 - gamma) * gradient**2
    weight -= learning_rate * gradient / np.sqrt(RMSProp_Gt + epsilon)
    return weight,RMSProp_Gt

def AdaDelta(weight,X,y,AdaDelta_vector_Gt,AdaDelta_vector_t):
```

```
        gamma = 0.95
        epsilon = 10e-6
        gradient = grad(weight,X,y)
        AdaDelta_vector_Gt = gamma * AdaDelta_vector_Gt + (1-gamma) * gradient**2
        delta=-1 * gradient * np.sqrt(AdaDelta_vector_t + epsilon) / np.sqrt(AdaDelta_vector_Gt + epsilon)
        weight += delta
        AdaDelta_vector_t = gamma * AdaDelta_vector_t + (1 -gamma) * delta**2
        return weight,AdaDelta_vector_Gt,AdaDelta_vector_t

    def Adam(weight,X,y,Adam_vector_m,Adam_vector_Gt,t):
        beta = 0.9
        gamma = 0.999
        epsilon = 10e-8
        learning_rate = 0.1
        t += 1
        gradient = grad(weight,X,y)
        Adam_vector_m = beta * Adam_vector_m + (1-beta) * gradient
        Adam_vector_Gt = gamma * Adam_vector_Gt + (1 - gamma) * gradient**2
        alpha = learning_rate * np.sqrt(1 - gamma**t) / (1 - beta**t)
        weight -= alpha * Adam_vector_m / np.sqrt(Adam_vector_Gt + epsilon)
        return weight,Adam_vector_m,Adam_vector_Gt,t

    def 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]


    X_train, y_train = load_svmlight_file("a9a1")
    data_size,features=X_train.shape
    X_train = X_train.toarray()
    X_train = np.c_[np.ones(len(X_train)), X_train]
    for i in range(0, len(y_train)):
        if y_train[i] == -1:
            y_train[i] = 0
    X_test, y_test = load_svmlight_file("a9a2",n_features = features)
    X_test = 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])


    optimizer=["SGD","NAG","RMSProp","AdaDelta","Adam"]
    NAG_v = np.zeros([features + 1, 1])
    RMSProp_Gt = np.zeros([features + 1,1])
    AdaDelta_vector_Gt = np.zeros([features + 1,1])
    AdaDelta_vector_t = np.zeros([features + 1,1])
    Adam_vector_m = np.zeros([features+1,1])
    Adam_vector_Gt = np.zeros([features+1,1])
    t = 0
    batch_size = 128

    for index,j in enumerate(optimizer):
        weight = np.random.rand(features + 1, 1)
        iteration = []
        error = []
        for i in range(0, int(data_size / batch_size ) + 1):
            iteration.append(i)
            X,y = batch(i,X_train,y_train,data_size)
            if j == "SGD":
                weight = SGD(weight,X_train,y_train)
            elif j == "NAG":
                weight,NAG_v = NAG(weight,X_train,y_train,NAG_v)
            elif j == "RMSProp":
                weight,RMSProp_Gt                                    =
RMSProp(weight,X_train,y_train,RMSProp_Gt)
            elif j == "AdaDelta":
                weight,AdaDelta_vector_Gt,AdaDelta_vector_t          =
AdaDelta(weight,X_train,y_train,AdaDelta_vector_Gt,AdaDelta_vector_t)
            elif j == "Adam":
                weight,Adam_vector_m,Adam_vector_Gt,t                =
Adam(weight,X_train,y_train,Adam_vector_m,Adam_vector_Gt,t)
            error.append(loss(weight,X_test,y_test))
        plt.plot(iteration, error, label=j)
    plt.xlabel('iteration')
    plt.ylabel('loss')
    plt.legend()
    plt.show()
```

8. 模型参数的初始化方法:随机初始化

9.选择的 loss 函数及其导数:

逻辑回归：
loss 函数：

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

导数：

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

线性分类：
loss 函数：

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

导数：

$$g_{\mathbf{w}}(\mathbf{x}_i) = \begin{cases} -y_i\mathbf{x}_i & 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b) >= 0 \\ 0 & 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 0 \end{cases}$$

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

10.实验结果和曲线图:

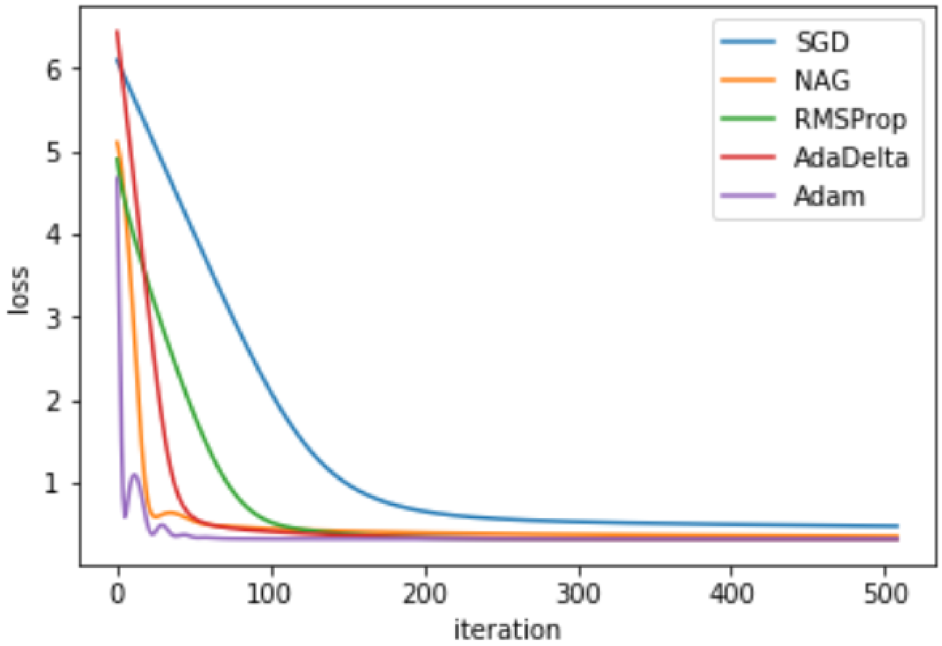
超参数选择：

SGD: learning_rate = 0.01
NAG: learning_rate = 0.01 gamma = 0.9
RMSProp: gamma = 0.9 epsilon = 10e-8 learning_rate = 0.005
AdaDelta: gamma = 0.95 epsilon = 10e-6
Adam: beta = 0.9 gamma = 0.999 epsilon = 10e-8 learning_rate = 0.1
C = 0.9

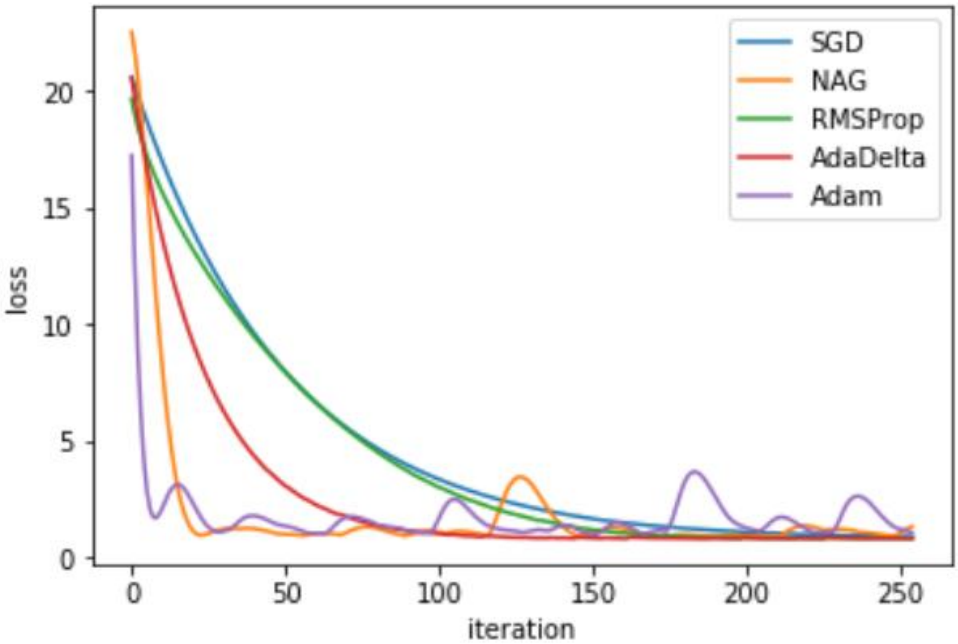
预测结果（最佳结果）：

loss 曲线图：

逻辑回归：



线性分类：



11.实验结果分析:

由 loss 曲线图，SGD 的 loss 下降最慢，学习率难以选择，大学习率容易震

荡，小学习率使 loss 收敛很慢，容易陷入不太好局部最小或者鞍点。NAG 核心思想就是利用 Momentum 预测下一步的梯度，对 SGD 实现了改进。RMSprop 可以解决 AdaGrad 中学习速率趋向 0 的问题。AdaDelta 同样可以解决 AdaGrad 中学习速率趋向 0 的问题，它避免人为设置学习速率，但有时相对比较慢。相对其他来说，Adam 快很多，但更新过程较复杂。

12.对比逻辑回归和线性分类的异同点：

logistic 回归与线性回归实际上有很多相同之处，均可以用来处理二分类问题，最大的区别就在于逻辑回归使用了 sigmoid 激活函数。

13.实验总结：

相比实验一，我更多地使用了 numpy 库里的函数，极大地简化了代码过程。同时，我了解到实验一的梯度下降会在大数据集上进行冗余计算，而实验二通过批量下降的方法改进了性能。另外，我对 NAG、Adam 等参数更新方法有了更深的理解。