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　韩松岳

学　　号　201536471115

邮　　箱　han14122@163.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 2 日

3. 报告人:韩松岳

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。

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，，和。
7.重复步骤 4-6 若干次，画出，，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，，和。
7.重复步骤 4-6 若干次，画出，，和随迭代次数的变化图。

7. 代码内容:

```python
def get_batch(X_train, y_train):
    #打乱矩阵 取batch
    random_sequence = np.arange(X_train.shape[0])
    np.random.shuffle(random_sequence)
    X_batch = np.zeros((batch_size, dimension))
    y_batch = np.zeros(batch_size)
    for i in range(batch_size):
        X_batch[i] = X_train[random_sequence[i], :]
        y_batch[i] = y_train[random_sequence[i]]
    return X_batch, y_batch
```

```python
def gradient (X, y, w):
    g = np.zeros(X.shape[1])   #存放 hinge loss 的梯度
    for i in range(X.shape[0]):   #对每一条记录迭代一次
        judge = y[i] * np.dot(X[i], w.reshape(dimension, 1))   #分段函数 判断
        if (judge < 1):               #小于1 则更新g 否则 g = g + 0
            g = g - y[i] * X[i]
    G = w + C * g              #计算梯度G
    return G
```

```python
def loss (X, y, w):
    loss = 0
    for i in range(X.shape[0]):
        judge = y[i] * np.dot(X[i], w.reshape(dimension, 1))    #分段函数 判断y * X * W与1的大小
        if (judge >= 1):
            loss = loss + 0                                     #大于1 hinge loss = 0
        else:
            loss = loss + 1 - judge                             #小于1 加上 1 - y * X * W
    loss = np.dot(w, w.reshape(dimension, 1)) / 2 + C * loss / X.shape[0]               #loss取平均
    return loss
```

```python
def NAG(w, v, g, eta=0.05, gama=0.9):
    v = gama * v + eta * g
    w = w - v
    return w, v
```

```python
def RMSProp(w, G, g, gama=0.9, eta=0.001, epsilon=1e-8):
    G = gama * G + (1 - gama) * np.dot(g, g.reshape(g.shape[0], 1))
    w = w - (eta / np.sqrt(G + epsilon)) * g
    return w, G
```

```python
def AdaDelta(w, G, g, delta, gama=0.95, epsilon=1e-6):
    G = gama * G + (1-gama) * np.dot(g, g)
    delta_w = -(np.sqrt(delta + epsilon) / np.sqrt(G + epsilon)) * g
    w = w + delta_w
    delta = gama * delta + (1-gama) * np.dot(delta_w, delta_w.reshape(delta_w.shape[0], 1))
    return w, G, delta
```

```python
def Adam(w, m, G, g, t, beta=0.9, gama=0.999, eta=0.002, epsilon=1e-8):
    m = beta * m + (1-beta) * g
    G = gama * G + (1-gama) * np.dot(g, g.reshape(g.shape[0], 1))
    a = eta * np.sqrt(1 - np.power(gama, t))
    w = w - (a / np.sqrt(G + epsilon)) * m
    t = t + 1
    return w, m, G, t
```

```python
def NAG_run(X_train, y_train, X_test, y_test, w):
    gama = 0.9
    v = np.zeros(dimension)
    NAG_loss = np.zeros(iter)    #初始化 存放NAG法的验证集loss的向量
    for i in range(iter):            #迭代
        NAG_loss[i] = loss(X_test, y_test, w)
        g = gradient(X_train, y_train, w - gama * v)
        w, v = NAG(w, v, g)
    return NAG_loss
```

```python
def RMSProp_run(X_train, y_train, X_test, y_test, w):
    G = 0
    RMSProp_loss = np.zeros(iter)    #初始化 存放NAG法的验证集loss的向量
    for i in range(iter):
        RMSProp_loss[i] = loss(X_test, y_test, w)
        g = gradient(X_train, y_train, w)
        w, G = RMSProp(w ,G, g)
    return RMSProp_loss
```

```python
def AdaDelta_run(X_train, y_train, X_test, y_test, w):
    G = 0
    delta = 0
    AdaDelta_loss = np.zeros(iter)
    for i in range(iter):
        AdaDelta_loss[i] = loss(X_test, y_test, w)
        g = gradient(X_train, y_train, w)
        w, G, delta = AdaDelta(w, G, g, delta)
    return AdaDelta_loss
```

```python
def Adam_run(X_train, y_train, X_test, y_test, w):
    m = np.zeros(dimension)
    G = 0
    t = 0
    Adam_loss = np.zeros(iter)
    for i in range(iter):
        Adam_loss[i] = loss(X_test, y_test, w)
        g = gradient(X_train, y_train, w)
        w, m, G, t = Adam(w, m, G, g, t)
    return Adam_loss
```

（针对逻辑回归和线性分类分别填写 8-11 内容）

7. 模型参数的初始化方法:

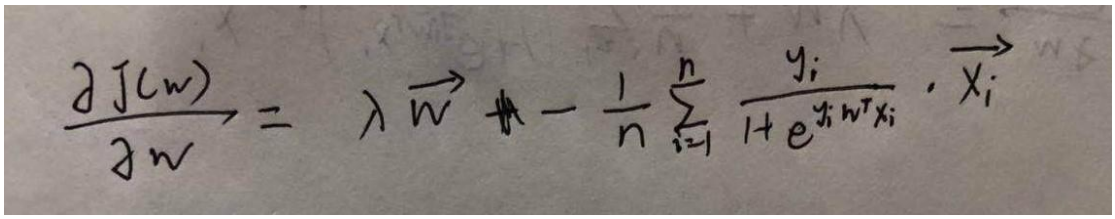
逻辑回归：正态分布初始化
线性分类：正态分布初始化

8. 选择的 loss 函数及其导数:

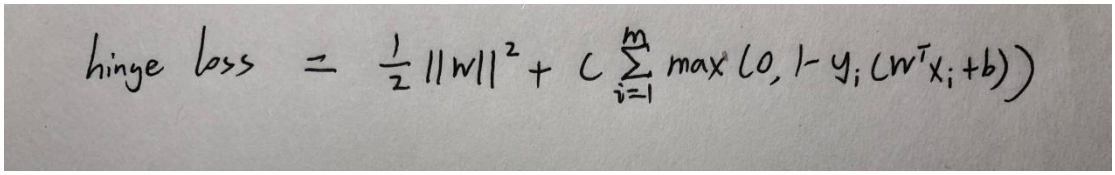
逻辑回归 loss 函数：

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i\cdot\mathbf{w}^\top\mathbf{x}_i}) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

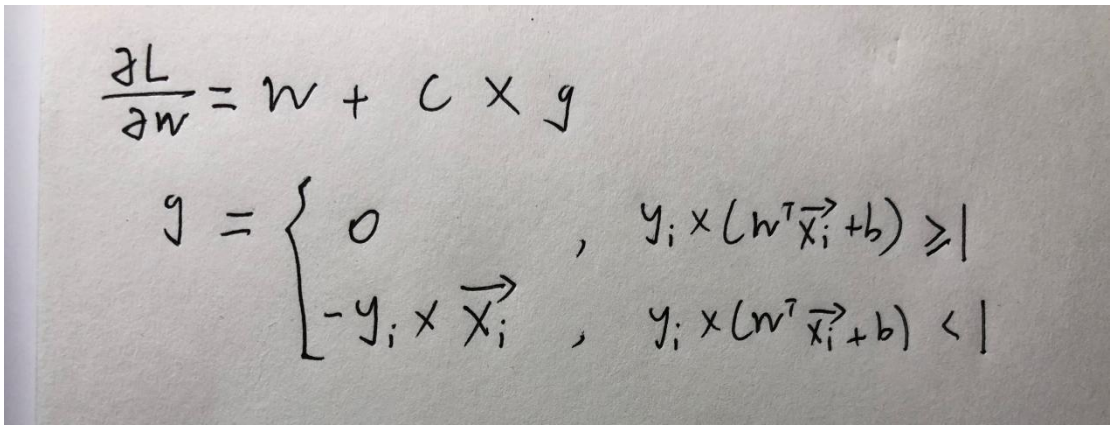
逻辑回归 loss 函数梯度：



线性分类 loss 函数：



线性分类 loss 函数梯度：



10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

NAG：

超参数选择：eta=0.05, gama=0.9

RMSProp：

超参数选择：gama=0.9, eta=0.001, epsilon=1e-8

AdaDelta:

超参数选择：gama=0.95, epsilon=1e-6

Adam:

超参数选择： beta=0.9, gama=0.999, eta=0.002, epsilon=1e-8

11.实验结果分析:

在逻辑回归中，NAG_loss 下降最快，其次是 AdaDelta, RMSProp，Adam
在 SVM 线性分类中，Adam 下降最快，其次是 RMSProp， NAG，AdaDelta

12.对比逻辑回归和线性分类的异同点：

逻辑回归，线性分类使用相同的优化方法，不同的 loss 函数

13.实验总结：

在这次实验中，我学习到了多种高级梯度下降优化方法，发现理解了不同优化方法的优缺点以及适用的环境。在今后的学习中要善于运用不同的优化算法，在必要时可以尝试自己改进。