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　廖瑞杰

学　　号　20130612170

邮　　箱　287462612@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 2 日

3. 报告人:廖瑞杰

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。

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 -*-\n",
"from sklearn.datasets import load_svmlight_file\n",
"import numpy as np\n",
"import matplotlib.pyplot as plt\n",
"\n",
"def sigmoid(z):\n",
"        return 1 /(1 + np.exp(-z))\n",
```

```
        "\n",
        "def loss(weight,X,y):\n",
        "       z = np.matmul(X, weight)\n",
        "          loss_ = -np.mean(y * np.log(sigmoid(z)) + (1 - y) * np.log(1 - sigmoid(z)))\n",
        "       return loss_\n",
        "\n",
        "def grad(weight,X,y):\n",
        "       z = np.matmul(X, weight)\n",
        "       h = sigmoid(z)\n",
        "       error = h - y\n",
        "       gradient = np.matmul(X.transpose(), error) / y.shape[0]\n",
        "       return gradient\n",
        "\n",
        "def SGD(weight,X,y):\n",
        "       learning_rate = 0.01\n",
        "       weight -= learning_rate * grad(weight,X,y)\n",
        "       return weight\n",
        "\n",
        "def NAG(weight,X,y,NAG_v):\n",
        "       learning_rate = 0.01\n",
        "       gamma = 0.9\n",
        "       gradient = grad(weight - gamma * NAG_v,X,y)\n",
        "       next_NAG_vector = gamma * NAG_v + learning_rate * gradient\n",
        "       weight -= next_NAG_vector\n",
        "       return weight,next_NAG_vector\n",
        "\n",
        "def RMSProp(weight,X,y,RMSProp_Gt):\n",
        "       gamma = 0.9\n",
        "       epsilon = 10e-8\n",
        "       learning_rate = 0.005\n",
        "       gradient = grad(weight,X,y)\n",
        "            RMSProp_Gt = gamma * RMSProp_Gt + (1 - gamma) * gradient**2\n",
        "            weight -= learning_rate * gradient / np.sqrt(RMSProp_Gt + epsilon)\n",
        "       return weight,RMSProp_Gt\n",
        "\n",
        "def AdaDelta(weight,X,y,AdaDelta_vector_Gt,AdaDelta_vector_t):\n",
        "       gamma = 0.95\n",
        "       epsilon = 10e-6\n",
        "       gradient = grad(weight,X,y)\n",
        "        AdaDelta_vector_Gt = gamma * AdaDelta_vector_Gt + (1-gamma) * gradient**2\n",
```

```
"            delta=-1 * gradient * np.sqrt(AdaDelta_vector_t + epsilon) / np.sqrt(AdaDelta_vector_Gt + epsilon)\n",
"       weight += delta\n",
"         AdaDelta_vector_t = gamma * AdaDelta_vector_t + (1 -gamma) * delta**2\n",
"       return weight,AdaDelta_vector_Gt,AdaDelta_vector_t\n",
"\n",
"def Adam(weight,X,y,Adam_vector_m,Adam_vector_Gt,t):\n",
"       beta = 0.9\n",
"       gamma = 0.999\n",
"       epsilon = 10e-8\n",
"       learning_rate = 0.1\n",
"       t += 1\n",
"       gradient = grad(weight,X,y)\n",
"       Adam_vector_m = beta * Adam_vector_m + (1-beta) * gradient\n",
"         Adam_vector_Gt = gamma * Adam_vector_Gt + (1 - gamma) * gradient**2\n",
"       alpha = learning_rate * np.sqrt(1 - gamma**t) / (1 - beta**t)\n",
"          weight -= alpha * Adam_vector_m / np.sqrt(Adam_vector_Gt + epsilon)\n",
"       return weight,Adam_vector_m,Adam_vector_Gt,t\n",
"\n",
"def batch(batch_count,X,y,data_size):\n",
"       if (1 + batch_count) * batch_size <= data_size:\n",
"                  return X[batch_count * batch_size:(batch_count + 1) * batch_size],y[batch_count * batch_size:(batch_count + 1) * batch_size]\n",
"       else:\n",
"              return X[batch_count * batch_size:data_size],y[batch_count * batch_size:data_size]\n",
"\n",
"\n",
"X_train, y_train = load_svmlight_file(\"a9a1\")\n",
"data_size,features=X_train.shape\n",
"X_train = X_train.toarray()\n",
"X_train = np.c_[np.ones(len(X_train)), X_train]\n",
"for i in range(0, len(y_train)):\n",
"       if y_train[i] == -1:\n",
"              y_train[i] = 0\n",
"X_test, y_test = load_svmlight_file(\"a9a2\",n_features = features)\n",
"X_test = X_test.toarray()\n",
"X_test = np.c_[np.ones(len(X_test)), X_test]\n",
"for i in range(0, len(y_test)):\n",
"       if y_test[i] == -1:\n",
"              y_test[i] = 0\n",
```

```
"y_train = y_train.reshape([len(y_train), 1])\n",
"y_test = y_test.reshape([len(y_test), 1])      \n",
"\n",
"\n",
"optimizer=[\"SGD\",\"NAG\",\"RMSProp\",\"AdaDelta\",\"Adam\"]\n",
"NAG_v = np.zeros([features + 1, 1])\n",
"RMSProp_Gt = np.zeros([features + 1,1])\n",
"AdaDelta_vector_Gt = np.zeros([features + 1,1])\n",
"AdaDelta_vector_t = np.zeros([features + 1,1])\n",
"Adam_vector_m = np.zeros([features+1,1])\n",
"Adam_vector_Gt = np.zeros([features+1,1])\n",
"t = 0\n",
"batch_size = 64\n",
"\n",
"for index,j in enumerate(optimizer):\n",
"    weight = np.random.rand(features + 1, 1)\n",
"    iteration = []\n",
"    error = []\n",
"    for i in range(0, int(data_size / batch_size ) + 1):\n",
"        iteration.append(i)\n",
"        X,y = batch(i,X_train,y_train,data_size)\n",
"        if j == \"SGD\":\n",
"            weight = SGD(weight,X_train,y_train)\n",
"        elif j == \"NAG\":\n",
"            weight,NAG_v = NAG(weight,X_train,y_train,NAG_v)\n",
"        elif j == \"RMSProp\":\n",
"                                                    weight,RMSProp_Gt      =
RMSProp(weight,X_train,y_train,RMSProp_Gt)\n",
"        elif j == \"AdaDelta\":\n",
"                                weight,AdaDelta_vector_Gt,AdaDelta_vector_t  =
AdaDelta(weight,X_train,y_train,AdaDelta_vector_Gt,AdaDelta_vector_t)\n",
"        elif j == \"Adam\":\n",
"                                    weight,Adam_vector_m,Adam_vector_Gt,t   =
Adam(weight,X_train,y_train,Adam_vector_m,Adam_vector_Gt,t)\n",
"        error.append(loss(weight,X_test,y_test))\n",
"    plt.plot(iteration, error, label=j)\n",
"plt.xlabel('iteration')\n",
"plt.ylabel('loss')\n",
"plt.legend()\n",
"plt.show()"
```

线性分类：

```
"# -*- coding: utf-8 -*-\n",
"from sklearn.datasets import load_svmlight_file\n",
"import numpy as np\n",
```

```
"import matplotlib.pyplot as plt\n",
"\n",
"def sigmoid(z):\n",
"    return 1 /(1 + np.exp(-z))\n",
"\n",
"def loss(weight,X,y):\n",
"    C = 0.9\n",
"    hinge_loss_sum = 0.\n",
"        hinge_loss_sum = sum(np.maximum(0, (1 - y * np.matmul(X,weight))))\n",
"    loss_ = np.matmul(weight.T, weight)[0][0] /2 + (hinge_loss_sum * C) /y.shape[0]\n",
"    return loss_\n",
"\n",
"def grad(weight,X,y):\n",
"    C = 0.9\n",
"    gw = np.zeros((124,1))\n",
"    temp = 1 - y * np.matmul(X,weight)\n",
"    temp = np.maximum(temp /np.abs(temp),0)\n",
"    y = y * temp\n",
"    gw = -np.matmul(X.T,y)\n",
"    return (C * gw) + weight\n",
"\n",
"def SGD(weight,X,y):\n",
"    learning_rate = 0.01\n",
"    weight -= learning_rate * grad(weight,X,y)\n",
"    return weight\n",
"\n",
"def NAG(weight,X,y,NAG_v):\n",
"    learning_rate = 0.01\n",
"    gamma = 0.9\n",
"    gradient = grad(weight - gamma * NAG_v,X,y)\n",
"    next_NAG_vector = gamma * NAG_v + learning_rate * gradient\n",
"    weight -= next_NAG_vector\n",
"    return weight,next_NAG_vector\n",
"\n",
"def RMSProp(weight,X,y,RMSProp_Gt):\n",
"    gamma = 0.9\n",
"    epsilon = 10e-8\n",
"    learning_rate = 0.005\n",
"    gradient = grad(weight,X,y)\n",
"        RMSProp_Gt = gamma * RMSProp_Gt + (1 - gamma) * gradient**2\n",
"        weight -= learning_rate * gradient / np.sqrt(RMSProp_Gt +
```

```
epsilon)\n",
        "       return weight,RMSProp_Gt\n",
        "\n",
        "def AdaDelta(weight,X,y,AdaDelta_vector_Gt,AdaDelta_vector_t):\n",
        "       gamma = 0.95\n",
        "       epsilon = 10e-6\n",
        "       gradient = grad(weight,X,y)\n",
        "        AdaDelta_vector_Gt = gamma * AdaDelta_vector_Gt + (1-gamma) *
gradient**2\n",
        "           delta=-1 * gradient * np.sqrt(AdaDelta_vector_t + epsilon) /
np.sqrt(AdaDelta_vector_Gt + epsilon)\n",
        "       weight += delta\n",
        "         AdaDelta_vector_t = gamma * AdaDelta_vector_t + (1 -gamma) *
delta**2\n",
        "       return weight,AdaDelta_vector_Gt,AdaDelta_vector_t\n",
        "\n",
        "def Adam(weight,X,y,Adam_vector_m,Adam_vector_Gt,t):\n",
        "       beta = 0.9\n",
        "       gamma = 0.999\n",
        "       epsilon = 10e-8\n",
        "       learning_rate = 0.1\n",
        "       t += 1\n",
        "       gradient = grad(weight,X,y)\n",
        "       Adam_vector_m = beta * Adam_vector_m + (1-beta) * gradient\n",
        "          Adam_vector_Gt = gamma * Adam_vector_Gt + (1 - gamma) *
gradient**2\n",
        "       alpha = learning_rate * np.sqrt(1 - gamma**t) / (1 - beta**t)\n",
        "          weight -= alpha * Adam_vector_m / np.sqrt(Adam_vector_Gt +
epsilon)\n",
        "       return weight,Adam_vector_m,Adam_vector_Gt,t\n",
        "\n",
        "def batch(batch_count,X,y,data_size):\n",
        "       if (1 + batch_count) * batch_size <= data_size:\n",
        "                  return X[batch_count * batch_size:(batch_count + 1) *
batch_size],y[batch_count * batch_size:(batch_count + 1) * batch_size]\n",
        "       else:\n",
        "              return X[batch_count * batch_size:data_size],y[batch_count *
batch_size:data_size]\n",
        "\n",
        "\n",
        "X_train, y_train = load_svmlight_file(\"a9a1\")\n",
        "data_size,features=X_train.shape\n",
        "X_train = X_train.toarray()\n",
        "X_train = np.c_[np.ones(len(X_train)), X_train]\n",
```

```
"for i in range(0, len(y_train)):\n",
"       if y_train[i] == -1:\n",
"              y_train[i] = 0\n",
"X_test, y_test = load_svmlight_file(\"a9a2\",n_features = features)\n",
"X_test = X_test.toarray()\n",
"X_test = np.c_[np.ones(len(X_test)), X_test]\n",
"for i in range(0, len(y_test)):\n",
"       if y_test[i] == -1:\n",
"              y_test[i] = 0\n",
"y_train = y_train.reshape([len(y_train), 1])\n",
"y_test = y_test.reshape([len(y_test), 1])      \n",
"\n",
"\n",
"optimizer=[\"SGD\",\"NAG\",\"RMSProp\",\"AdaDelta\",\"Adam\"]\n",
"NAG_v = np.zeros([features + 1, 1])\n",
"RMSProp_Gt = np.zeros([features + 1,1])\n",
"AdaDelta_vector_Gt = np.zeros([features + 1,1])\n",
"AdaDelta_vector_t = np.zeros([features + 1,1])\n",
"Adam_vector_m = np.zeros([features+1,1])\n",
"Adam_vector_Gt = np.zeros([features+1,1])\n",
"t = 0\n",
"batch_size = 128\n",
"\n",
"for index,j in enumerate(optimizer):\n",
"       weight = np.random.rand(features + 1, 1)\n",
"       iteration = []\n",
"       error = []\n",
"       for i in range(0, int(data_size / batch_size ) + 1):\n",
"              iteration.append(i)\n",
"              X,y = batch(i,X_train,y_train,data_size)\n",
"              if j == \"SGD\":\n",
"                     weight = SGD(weight,X_train,y_train)\n",
"              elif j == \"NAG\":\n",
"                     weight,NAG_v = NAG(weight,X_train,y_train,NAG_v)\n",
"              elif j == \"RMSProp\":\n",
"                                                  weight,RMSProp_Gt     =
RMSProp(weight,X_train,y_train,RMSProp_Gt)\n",
"              elif j == \"AdaDelta\":\n",
"                                  weight,AdaDelta_vector_Gt,AdaDelta_vector_t  =
AdaDelta(weight,X_train,y_train,AdaDelta_vector_Gt,AdaDelta_vector_t)\n",
"              elif j == \"Adam\":\n",
"                                       weight,Adam_vector_m,Adam_vector_Gt,t  =
Adam(weight,X_train,y_train,Adam_vector_m,Adam_vector_Gt,t)\n",
"              error.append(loss(weight,X_test,y_test))\n",
```

```
"        plt.plot(iteration, error, label=j)\n",
"plt.xlabel('iteration')\n",
"plt.ylabel('loss')\n",
"plt.legend()\n",
"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：

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

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:（各种梯度下降方式分别填写此项）

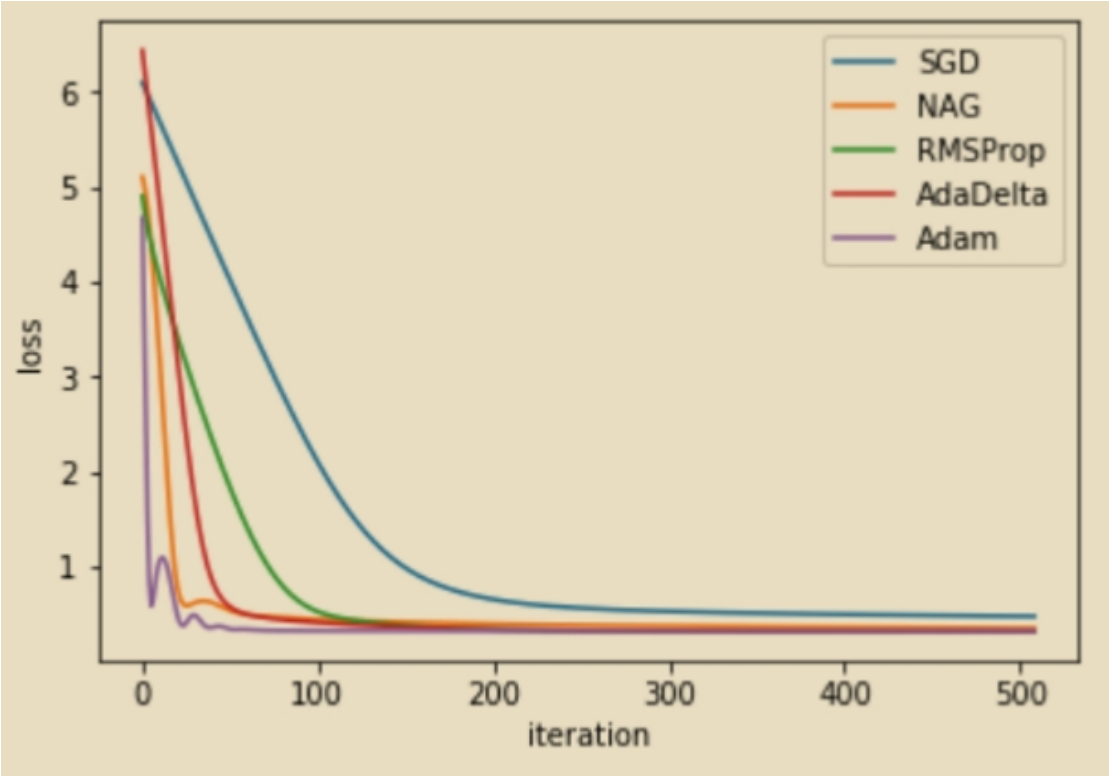
超参数选择：

SGD:learning_rate=0.01

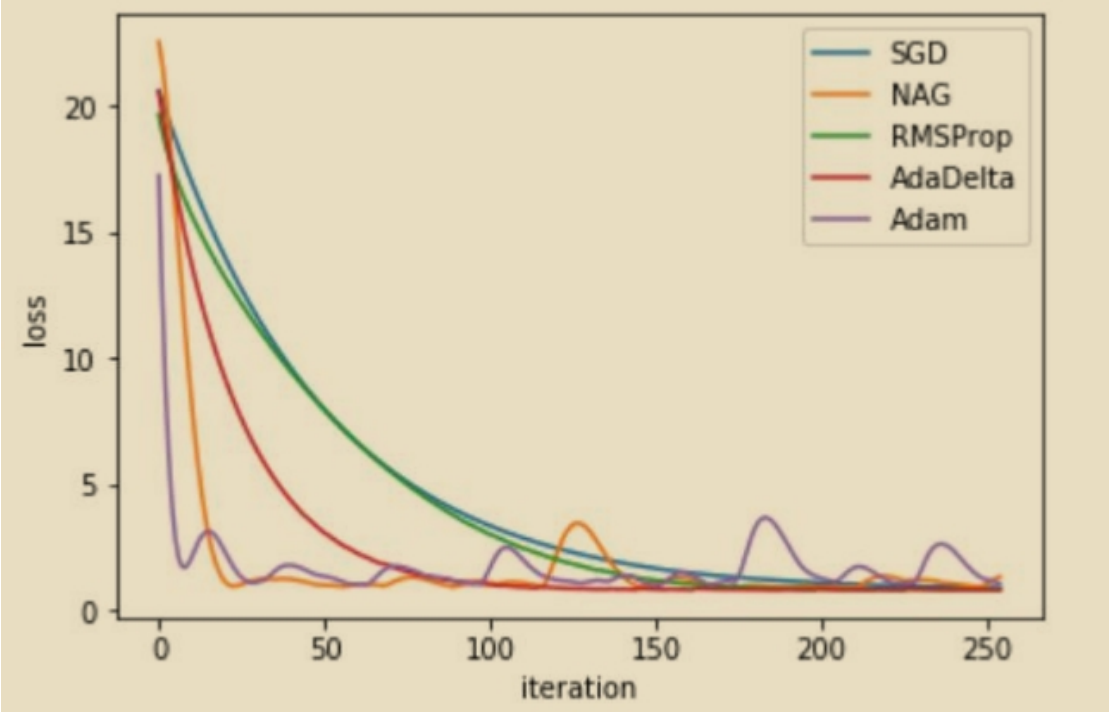
NAG：learning_rate=0.01 gamma=0.9
RMSProp: gamma=0.9 epsilon=10e-8 learning_rate=0.005
AdaDelta:gamma=0.95 epsilon=10e-6
Adam:beta=0.9 gamma=0.999 epsilon=10e-8 learning_rate=0.1
C=0.9

预测结果（最佳结果）：

loss 曲线图：



线性分类：



11.实验结果分析:

SGD 的学习速率很大，大容易震荡，小收敛很慢。人为地在训练中调节是比较困难的，也难以适应数据的特征。

NAG 利用 Momentum 预测下一步梯度

RMSprop，AdaGrad 解决 AdaGrad 学习速率趋于 0 的问题。

12.对比逻辑回归和线性分类的异同点：

均可处理二分类问题，区别为逻辑回归使用了 sigmoid 函数。

13.实验总结：

实验二让我学会了用批量下降的方式改进性能。