机器学习与数据挖掘 ~ Homework1

19335084 黄梓浩

1 实验目的和实验要求

1.1 实验目的

实现线性分类器与非线性分类器，并做对比。

1.2 实验要求

- 用 softmax 函数 实现多类分类的线性分类器；
- 通过基函数来非线性化实现的线性分类器；
- 通过正则化系数 λ =1，分别用 L1范数 和 L2范数 正则化 实现的非线性分类器；
- 通过 分类精度 ACC 来对比 四种不同分类器。

2 算法原理

2.1 多类分类

多分类的实现： 使用 softmax函数代替二分类中的 sigmoid 函数。

对一般的线性分类，样本数为 N，特征数为 M，我们有样本特征矩阵 $\tilde{\boldsymbol{X}}$，其对应的标签矩阵 Y，同时还有一个 （M+1)× K 的参数矩阵 W， 从而有线性回归 $f(\tilde{\boldsymbol{X}}) = \tilde{\boldsymbol{X}}\boldsymbol{W}$。

我们需要找到 xW 到 y 的映射，所使用的映射函数就是 softmax：

$$\underset{i}{softmax}(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

由此， softmax(XW) 能得到预测的概率矩阵，其中的 第K列向量 就相当于 第K的分类器，由此可以方便得出属于各个类别的概率。

2.2 非线性化

对原线性回归 f(x) = xw，将 x 通过基函数映射到另一值域：

$$[x_1, x_2, \cdots, x_m] \in \mathbb{R}^m \longrightarrow [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \cdots, \phi_n(\boldsymbol{x})] \in \mathbb{R}^n \triangleq \boldsymbol{\phi}(\boldsymbol{x})$$

基函数的选择可以多样化，本题中使用了 exp(x) 作为基函数

2.3 正则化

模型随着迭代次数的不断增加，会逐渐出现过度拟合的现象，即只是针对训练数据表现更好，而对于未知的数据表现更差，模型的泛化度降低。

出现这种现象的原因是，由于从训练数据中学习到过多特征，模型学习到一些无用的甚至应该丢弃的特征，所以需要通过一种方法使模型能够丢弃一些特征，控制学习的特征的数量。

正则化就是一种防止过拟合的方式，常见的有 L1正则化， L2正则化。

$$l_1 : \Omega(w) = ||w||_1 = \sum_i |w_i|$$

$$l_2 : \Omega(w) = ||w||_2^2 = \sum_i w_i^2$$

比方说，有损失函数 L（W），添加正则项 $\Omega(w)$ 后：

$$\tilde{L}(\boldsymbol{W}) = L(\boldsymbol{W}) + \lambda \|\boldsymbol{W}\|_2^2$$

通过引入正则项，能够惩罚绝对值过大的 w列向量，当 w 的绝对值接近0，可以认为其对于模型的影响能够忽略不计，从而可以控制训练中的特征数量。

2.4 梯度下降法

用损失函数 L（W），能够衡量 分类模型参数矩阵W 的好坏，当 L（W）能取到最小值，即可得到 最优的 W。两种方法来获取最优的 W：

一种是令 L（W）=0，然后求解析解，但是一般情况下，损失函数复杂且不一定存在0解，这种方法一般难以使用；另一种就是使用梯度下降法来不断迭代，得到 最优的W。

梯度下降法的原理： 有 N 个训练样本，出这些样本的损失函数 L（W），对L（W）求导，并设置一个因子r （称为学习率），最终得到迭代公式：

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - r\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

经过迭代，最终该式子能够收敛到一个局部的最小值，一般认为这就是全局最小值，此时的 W就是最优的 W。

该方法的关键点在于 $\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$ 的计算，我们使用交叉熵损失函数用于计算，交叉熵损失函数计算的是在真实类别上模型预测的概率自然对数，概率越大，L(W) 就越低。

$$L(\boldsymbol{W}) = -\frac{1}{N}\sum_{l=1}^{N}\sum_{k=1}^{K} y_k^{(l)} \ln(softmax_k(\tilde{\boldsymbol{x}}^{(l)}\boldsymbol{W}))$$

经过推导，能够得到该导数公式为：

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{1}{N}\tilde{\boldsymbol{X}}^T(softmax(\tilde{\boldsymbol{X}}\boldsymbol{W}) - \boldsymbol{Y})$$

2.5 mini-batch

在使用梯度下降法进行迭代时，是用所有训练样本计算一次梯度，再对参数进行更新。当训练样本数量过大时，每更新一次参数都要遍历一遍数据集所有样本，显然这样做的话计算量会很大。

一种优化的方法是 mini-batch：即在每轮训练中，将打乱的训练数据均分为若干批，按批计算梯度，更新参数，以此达到降低计算开销的目的。

2.6 标准化

每个数据都有若干个特征值。然而，各个特征的大小不同、范围不同、尺度不同，这会导致模型训练过程中梯度下降的效果不佳。

因此，我们需要进行特征缩放，把各个特征的值重新映射到一个统一的范围。

标准化是一种常用的方法，即在每一列，对每个特征值， 减去特征平均值，再除以标准差。

$$x' = \frac{x - \bar{x}}{\sigma}$$

通过这种处理，使得每一列上各特征值的均值为0，标准差为1，实现了特征的一致化。

3 代码

3.1 预处理数据集

本次实验我们使用的数据集：https://archive.ics.uci.edu/ml/datasets/Multiple+Features

该数据集适用于多变量的分类，其目的是对 0 ~ 9 的数字进行分类。

该数据集包含6个文件，每个文件包含2000行，每一行表示一个数字样本的特征，其中前200行都是数字0的特征，然后分别是 数字 1~9 的每200行的各个特征集（即每200行分为不同数字）

6个文件包括

- mfeat-fac： 216 个 profile correlations；
- mfeat-fou：76 个 字符形状的傅里叶系数；
- mfeat-kar： 64 个 Karhunen-Love 系数；
- mfeat-pix： 240 个 像素平均值；
- mfeat-zer： 47 个 泽尼克距；
- mfeat-mor：6个 形态学特征。

在这里预先使用python程序处理这些文件为 .npy 的格式文件

```
file = open("mfeat-mor",mode="r")

para_num = 6

data1_image = np.zeros((2000,para_num),dtype=np.float64)
data1_label = np.zeros(2000,dtype=np.float64)
for i in range (0, 2000):
    line = file.readline()
    line = line.strip().split()
    for j in range(0,para_num):
        line[j] = float(line[j])
    data1_image[i,:] = line[:]

    data1_label[i] = i/200
file.close()

np.save('image6.npy',data1_image)
np.save('label6.npy',data1_label)
```

可以得到 image1~ image6，label1 ~ label6的 npy文件，分别表示6个数据集的特征和标签，在学习中可以直接用 numpy库方便读取数据集。

3.2 训练集与测试集

我们需要从读取的数据集中分割出训练集和测试集，如下：

```
# 读对应序号的数据集 npy文件
data_image = np.load('mfeat/image'+str(data_num)+'.npy')
data_label = np.load('mfeat/label'+str(data_num)+'.npy')
# 打乱数据集
[row, col] = data_image.shape
state = np.random.get_state()
np.random.shuffle(data_image)
np.random.set_state(state)
np.random.shuffle(data_label)
# 按比例分配训练集和测试集
r2 = int(round(row*test_rate))
r1 = row-r2

train_image = np.zeros((r1,col))
train_label = np.zeros(r1,dtype=np.int64)
test_image = np.zeros((r2,col))
test_label = np.zeros(r2,dtype=np.int64)
for i in range(0,r1):
    train_image[i] = data_image[i]
    train_label[i] = data_label[i]
for i in range(r1,row):
    test_image[i-r1] = data_image[i]
    test_label[i-r1] = data_label[i]
```

得到训练集和测试集后，对这两个数据集都做以下处理： 对特征 添加偏置1，对标签 转换成 one-hot格式，对特征 标准化。

```
# 特征添加偏置1，同时标签转换为one-hot格式
extend = np.ones(train_image.shape[0])
train_X = np.c_[extend, train_image]
train_Y = np.zeros((train_label.size, label_num))
for i in range(train_label.size):
    train_Y[i, train_label[i]] = 1

extend = np.ones(test_image.shape[0])
test_X = np.c_[extend, test_image]
test_Y = np.zeros((test_label.size, label_num))
for i in range(test_label.size):
    test_Y[i, test_label[i]] = 1

# 特征标准化
mean_X = []
std_X = []
for i in range(train_X.shape[1]):
    mean_X.append(np.mean(train_X[:, i]))
    std_X.append(np.std(train_X[:,i]))
    if std_X[i] != 0:
```

```
            train_X[:, i] = (train_X[:,i]-mean_X[i])/std_X[i]
    for i in range(test_X.shape[1]):
        if std_X[i] != 0:
            test_X[:, i] = (test_X[:,i]-mean_X[i])/std_X[i]
```

3.3 softmax 函数

为防止 exp 函数溢出出错，计算前将所有值减去一个大值。

```
def softmax(matrix):
    bigval = matrix.max(axis = 1)-1
    bigval = np.reshape(bigval, (bigval.size, 1))
    temp = np.exp(matrix - bigval)
    col_sum = temp.sum(axis=1)
    col_sum = np.reshape(col_sum, (col_sum.size,1))
    return temp/col_sum
```

3.4 梯度更新

使用交叉熵计算损失，对正则项，L1_lamb和 L2_lamb 分别为两种方法的 λ系数，控制这个变量是否为0，就可以选择使用哪一种正则化方法。

```
def train(W, train_X, train_Y, learning_rate):
    temp = softmax(np.dot(train_X, W)) - train_Y
    dW = (1/train_X.shape[0])*np.dot(train_X.T, temp)
    # 添加正则项，控制L1，L2的lambda至少一个为0，实现不同正则方法
    dW += L1_lamb*np.abs(W)
    dW += 2*L2_lamb*W
    W -= learning_rate*dW
    return W
```

3.5 训练过程

由于本实验只要求我们对比分类器在测试集的ACC： $ACC = \frac{1}{m}\sum_{i=1}^{m}\delta\left(f(\mathbf{x}^{(i)}), y^{(i)}\right)$

所以只要得到学习结束的ACC就可以。

```
def train_proce(train_X,train_Y,test_X,test_Y):
    # 随机初始化 W 矩阵
    W = np.random.rand(train_X.shape[1], label_num)
    # 训练
    for i in range(epochs):
        # 打乱训练集
        state = np.random.get_state()
        np.random.shuffle(train_X)
        np.random.set_state(state)
        np.random.shuffle(train_Y)
        size = int(train_X.shape[0]/K)
        for k in range(K):
```

```
            block_X = np.r_[train_X[:k*size, :], train_X[(k+1)*size:, :]]
            block_Y = np.r_[train_Y[:k*size, :], train_Y[(k+1)*size:, :]]
            # mini-batch
            for i in range(int(block_X.shape[0]/batch_size)):
                # 这一batch中的起始和结束位置
                l = i*batch_size
                r = l+batch_size
                if (r > block_X.shape[0]):
                    r = block_X.shape[0]
                W = train(W, block_X[l:r,:], block_Y[l:r,:],  learning_rate)

    return get_acc(W, test_X, test_Y)
```

计算 ACC：

```
def get_acc(W, test_X, labels_Y):
    temp = softmax(np.dot(test_X, W))
    Y = temp.argmax(axis=1)
    num = 0
    for i in range(Y.size):
        if (labels_Y[i, Y[i]] == 1):
            num += 1
    return num/Y.size
```

3 结果分析

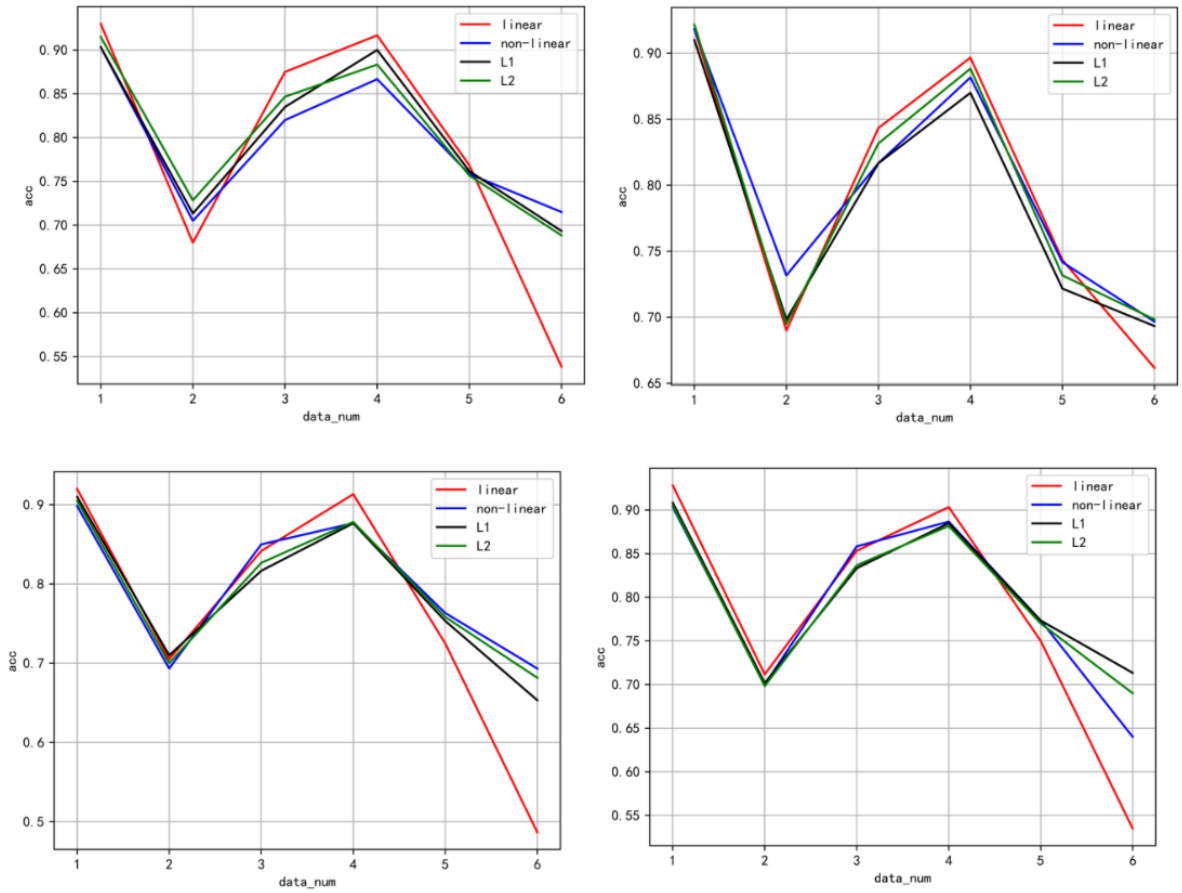
使用该训练方式，我们对不同的数据集进行测试，比较不同的分类器的ACC。

在该参数设置下进行测试，进行100轮次的学习。（非线性化的基函数为 exp(x) ）

```
test_rate = 0.3
K = 10
epochs = 100
learning_rate = 0.005
batch_size = 512
```

由于测试集划分有随机性，所以做了多次测试。

部分结果如下所示：



横坐标表示数据集序号，纵坐标表示该分类器在该数据集上的ACC。

其中 红，蓝，黑，绿线 分别代表 纯线性、非线性化、L1正则非线性化、L2正则非线性化分类器 在 6个数据集上表现的 ACC 结果。

1~6 的序号分别代表的数据集是：

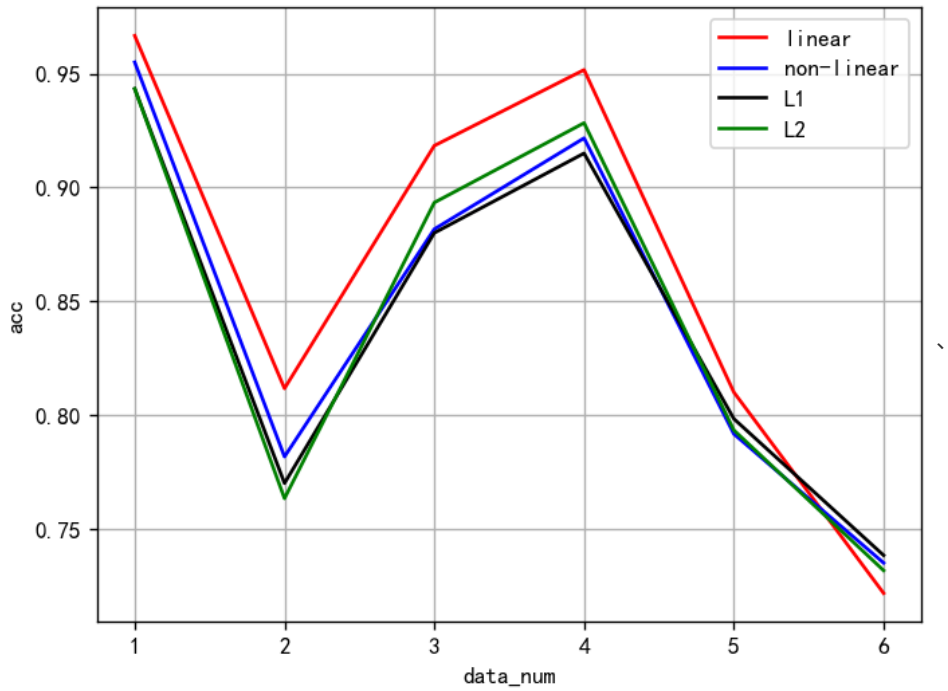
1. mfeat-fac： 216 个 profile correlations；
2. mfeat-fou：76 个 字符形状的傅里叶系数；
3. mfeat-kar： 64 个 Karhunen-Love 系数；
4. mfeat-pix： 240 个 像素平均值；
5. mfeat-zer： 47 个 泽尼克距；
6. mfeat-mor：6个 形态学特征。

可以明显发现的是，纯线性的分类方法对于特征参数较多的情况更好，但是在特征数较低的情况下，非线性化的方式更好。

总体上，L2正则化的非线性分类器比其他的分类器表现更加稳定。

为了得到更多结论，将学习轮次提升到1000次，使用以下的参数进行测试：

```
test_rate = 0.3
K = 10
epochs = 1000
learning_rate = 0.005
batch_size = 512
```



从结果发现，所有分类器的ACC都有提升，而纯线性分类器的ACC 的提升更为明显，与非线性的基函数选择有关。