机器学习导论第一次作业

韦俊林 (201928016029023)

2020 年 3 月 29 日

1 简述题

1.1 请描述对数几率回归模型（包括学习任务、学习模型、假设空间）

线性回归模型是用简单的线性方程实现对数据的拟合，然而这只能完成回归任务，并不能完成分类任务。因此，对数几率回归模型 (logistics regression) 则是利用广义线性回归的思想，期望找到一个单调可微函数 $g(x)$，利用该函数将分类任务的真实标记与线性回归模型的预测值联系起来，构建一个分类模型。例如：在 $y = w^T x + b$ 的基础上做二分类任务 $y \in \{0, 1\}$。最直观的想法是在线性模型输出再套上一个“单位阶跃函数”实现分类功能，但是该函数不连续更不可微不能满足上述对 $g(x)$ 的要求。因此，选用 $y = 1/(1 + e^{-x})$。该函数不仅可以预测出类别，还能够得到近似概率预测，这点对很多需要利用概率辅助决策的任务很有用，并且对数几率函数是任意阶可导函数，存在很好的数学性质，许多数值优化算法都可以直接用于求去最优解。

总结，对数几率回归模型就是在给定 n 个训练样本 $(x_1, y_1), ..., (x_n, y_n)$，其中 $x_i \in R^d$，$i = [1, n]$ 为 d 维样本特征，$y_i \in \{0, 1\}$ 为其对应的类别标签。Logistic 回归采用的假设函数：

$$h(x|w, b) = \frac{1}{1 + exp^{-(w^T x + b)}}$$

目标是训练模型参数 w, b，从而最小化代价函数。

1.2 请描述正则化 softmax 回归模型（包括学习任务、学习模型、假设空间）

Softmax Regression 是 Logistic 回归的推广，能以更加紧凑的方式来处理 Logistic 回归中所面临的多分类问题。因此，Softmax 回归的任务是给定 n 个训练样本 $(x_1, y_1), ..., (x_n, y_n)$，其中 $x_i \in R^d$，$i = [1, n]$ 为 d 维空间中的样本特征，$y_i \in \{1, 2, ..., c\}$ 为其对应的类别标签，训练出一个解决多分类问题的模型。给定测试样本 x，期望基于所学的假设函数能对每个类别 j 估算出概率值 $P(y = j|x)$。其以概率形式的假设函数为：

$$h(x; W) = \begin{pmatrix} P(y = 1|x; W) \\ P(y = 2|x; W) \\ ... \\ P(y = c|x; W) \end{pmatrix} = \frac{1}{\sum_{i=1}^{c} e^{w_i^T x}} = \begin{pmatrix} e^{w_1^T x} \\ e^{w_2^T x} \\ ... \\ e^{w_c^T x} \end{pmatrix} \in R^c$$

其中 x 为齐坐标表示，即 x 的维数是 $d+1$ 维，齐次坐标则对应着平移量 b，$W = [w_1, w_2, ..., w_c] \in R^{(d+1)c}$。

对应的损失函数为：

$$l(W) = -\sum_{i=1}^{n} \sum_{j=1}^{c} \delta(y_i = j) \log \frac{e^{w_j^T x_i}}{\sum_{k=1}^{c} e^{w_k^T x_i}}$$

其中 $\delta(*)$ 为指示函数。

但是通过优化求解该损失函数发现该回归模型中存在冗余的参数等问题，也就是说 Softmax 模型被过度参数化。这样会造成采用牛顿法优化会遇到数值计算的问题。因此，引入正则化项使结构风险最小化。新的学习模型：

$$l(W) = -\sum_{i=1}^{n}\sum_{j=1}^{c}\delta(y_i = j)\log\frac{e^{w_j^T x_i}}{\sum_{k=1}^{c} e^{w_k^T x_i}} + \lambda\|W\|_F^2$$

$$\frac{\partial l(W)}{\partial w_j} = \sum_{i=1}^{n} X_i(\delta(y_i = j) - P(y_i = j|x; W)) + 2\lambda w_j$$

其中 λ 是超参数，引入正则化项后，损失函数就变成了严格的凸函数，可保证得到唯一解，此时的 Hessian 矩阵变为可逆矩阵，并且因为是凸函数，梯度下降法和 L-BFGS 等算法可以保证收敛到全局最优。

1.3 请描述 ECOC 的优缺点

ECOC(Error Correcting Output Codes, 纠错输出码)，用于多分类学习任务。主要思想是通过事先分别为 k 类类别定义一串编码序列，在分类的时候，只需比较待分类样本与各串编码的差异程度。

在测试阶段，ECOC 编码对分类器的错误有一定的容忍和修正能力，并且对同一个学习任务，ECOC 编码越长，纠错能力就越强。但是对于长编码，最优编码设计是一个 NP 难问题，而且编码越长训练的时间也越长。最后，并不是编码的理论性能越好，分类的性能就越好，因为机器学习问题设计到很多因素。

1.4 请描述软间隔支持向量机的原始模型及其对偶模型，并指出满足哪些条件的样本是支持向量

在线性可分的情况想找到“最大间隔”的超平面 $y_i(w^T x_i + b) \geqslant 1, i = 1, 2, ..., m$，这就是硬间隔的支持向量机的基本型。然而，在现实任务中往往很难找到合适的核函数使得训练样本在特征空间中线性可分，为了解决这个问题，允许支持向量机的一些样本上出错，具体说来是允许某些样本不满足上述的约束条件，称之为“软间隔”。从而引入松弛变量 $\xi_i \geqslant 0$，上述式子改写为：

$$y_i(w^T x_i + b) \geqslant 1 - \xi_i\,, \quad \min\sum_{i=1}^{n}\xi_i$$

这样优化目标为：

$$\min_{w,b,\xi_i}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$s.t. \quad y_i(w^T x + i + b) \geqslant 1 - \xi_i$$

$$\xi_i \geqslant 0\,,\ i = 1, 2, ..., m$$

其中 C 为大于 0 的常数。这是一个二次规划问题，于是通过拉格朗日乘子法得到拉格朗日函数：

$$L(w, b, \alpha, \xi, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i(1 - \xi_i - y_i(w^T x_i + b)) - \sum_{i=1}^{m}\mu_i\xi_i$$

其中 $\alpha_i \geqslant 0, \mu_i \geqslant 0$ 是拉格朗日乘子。求解该函数于是得到相应的对偶问题：

$$\max_{\alpha}\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$s.t. \quad \sum_{i=1}^{m} a_i y_i = 0$$

$$0 \leqslant \alpha_i \leqslant C \quad i = 1, 2, ..., m$$

在软间隔支持向量机中，满足 $\alpha_j > 0$ 的样本均称为支持向量，因为若 $\alpha_j > 0$，则必有 $y_i f(x_i) = 1 - \xi_i$。

1.5 请简述支持向量机的原理

支持向量机 (Support Vector Machine) 是一种基于统计学习理论的学习机，其实现的思想是，通过某种事先选择的非线性映射将输入向量 x 映射到一个高维特征空间 z，在这个空间中构造最优分类超平面，从而使正例和反例样本之间的分离界限达到最大。从概念上说，支持向量是那些离决策面最近的数据点，它们决定了最优分类超平面的位置。因此，支持向量机的任务是给定训练集 $T = \{(x_1, y_1), ..., (x_n, y_n)\}, y_i \in \{+1, -1\}$，估计出最大间隔分类超平面，即其损失函数为正确分类样本的函数间隔最大。

2 计算机编程

2.1 实验任务

假定 x_1 中的 200 个点属于第一类，假定 x_2 中的 200 个点属于第二类，假定 x_3 中的 200 个点属于第三类，假定 x_4 中的 200 个点属于第四类，假定 x_5 中的 200 个点属于第五类，数据集如图 1。请

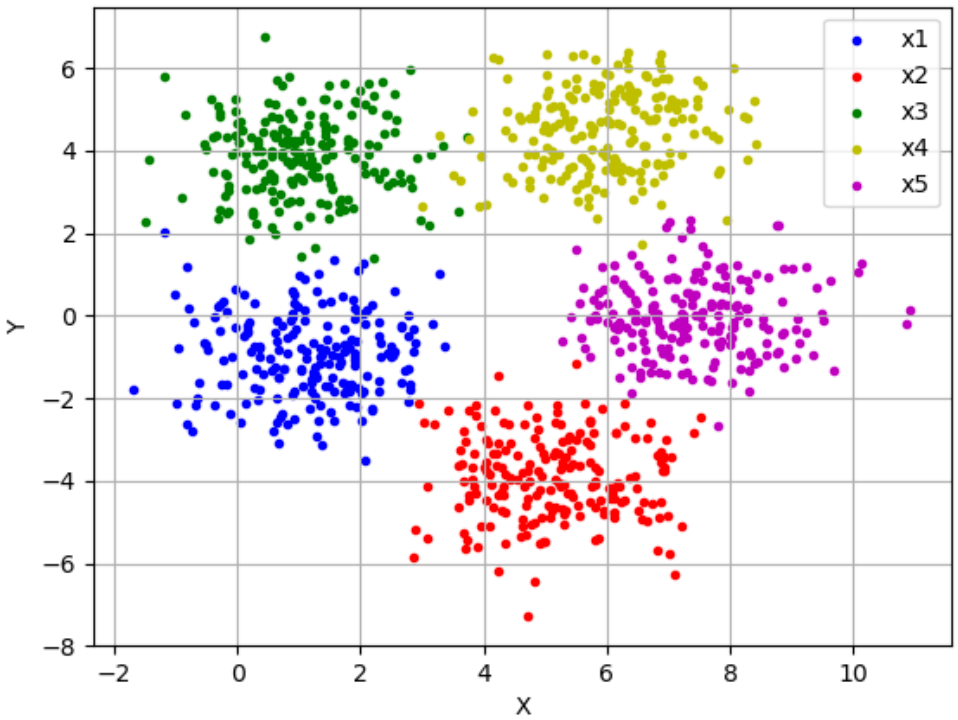

图 1: dataset

编写程序实现：

1、 采用正则化 softmax 回归方法来实现分类；

2、 采用 SVM 对其分类

要求：

(1) 每类随机采用 60% 的数据做训练样本，采用剩余的 40% 样本作测试。请给出正确率随着正则化参数而改变的曲线或者表格；

(2) 采用交叉验证来选择正则化参数

2.2 实验报告

完成第一个实验，关键实验过程：

(1) 根据作业给定的参数生成数据集，部分代码见图 2。

```
def _init_dataset_parm():
    parm={}
    parm['mu']=[]
    parm['mu'].append(np.array([1,-1]))
    parm['mu'].append(np.array([5,-4]))
    parm['mu'].append(np.array([1,4]))
    parm['mu'].append(np.array([6,4.5]))
    parm['mu'].append(np.array([7.5,0.0]))
    parm['Sigma']=np.array([[1,0],[0,1]])
    parm['labels_num']=5
    parm['total']=200
    return parm

#(5,200,3)
def _init_dataset():
    parm=_init_dataset_parm()
    dataset=[]
    for i in range(parm['labels_num']):
        dataset.append(pd.DataFrame(np.random.multivariate_normal(parm['mu'][i],parm['Sigma'],parm['total']),columns=list('xy')))
        dataset[i]['label']=i
    return dataset,parm['labels_num']
```

图 2: 生成数据集

(2) 根据正则化 softmax 回归原理实现模型，图 3。

(3) 实现小批量训练与测试，图 4。

(4) 交叉验证确定超参数，具体是：首先将数据分为五份，一份做测试集，其余做训练集进行训练，五份数据分别做一次测试集，确定一个超参数。要确定的超参数有：训练轮次、学习率、小批量大小、正则化项。因此，这一步总共有 20 个模型，以选择学习率为例，图 5、6。

(5) 选定最佳超参数后，将数据重新打乱将 60% 做训练集，40% 做测试集，训练模型。重复 20 次，计算准确率的平均值与方差，图 7。

实验结果：准确率随学习率、训练轮次变化而变化，见图 8，准确率随小批量大小、正则化项变化而变化，见图 9。

最终训练的模型准确率平均值为 0.9645，方差为 3.5978×10^{-5}，图 10。

```
import numpy as np
#softmax计算模型
def softmax_model(features,w,b):
    '''
    w_b=nd.concat(w,b.reshape((-1,num_labels)),dim=0)
    x_1=nd.concat(x_nd,nd.ones((len(x_nd),1),ctx=mx.gpu()),dim=1)
    x=nd.dot(x_1,w_b)
    '''
    x=nd.dot(features,w)+b
    x_exp=x.exp()
    partition=x_exp.sum(axis=1,keepdims=True)
    return x_exp/partition
#正则化
def regularization(w,b,reg):
    w_b=nd.concat(w,b.reshape((-1,len(b))),dim=0).asnumpy()
    W_F=float(reg*np.linalg.norm(w_b))
    return reg*W_F
#损失函数
def cross_entropy(y_hat,y,reg=0):
    return -nd.pick(y_hat.log(),y)+reg
```

图 3: softmax 模型、正则化项、损失函数

```
#   训练
    for epoch in range(epochs_num):
        loss_sum=0
        for x_train,y_train in train_iter:
            with autograd.record():

                y_hat=softmax_model(x_train,w,b)
                regularization_strength=regularization(w,b,reg)
                loss=loss_fuction(y_hat,y_train,regularization_strength)

            loss.backward()
            sgd([w,b],lr,batch_size)
            loss_sum+=nd.mean(loss).asscalar()
#   测试

        acc=0
        for x_test,y_test in test_iter:
            pred=softmax_model(x_test,w,b)
            temp=nd.argmax(pred,axis=1)==y_test.T
            correct=temp.asnumpy().mean()
#             print(temp.shape)
            acc+=correct

#         print('Epoch {}'.format(epoch))
        train_loss_results.append(loss_sum/len(train_iter))
        test_acc.append(acc/len(test_iter))
```

图 4: 实现训练与测试

```python
for i in range(1,6):
    print(i)
    lr.append(i*0.01)
    epochs_num.append(i*50)
    batch_size.append((2**i)*16)
    reg.append(random.randint(20*(i-1)+1,20*i))
```

图 5: 待选超参数

```python
print('init done. training lr...')
lr_acc=[]
for i in range(k_fold):
    print(i)
    x_train,y_train,x_test,y_test=_block_dataset(x_data=x_data,y_data=y_data,index=i,k_fold=k_fold)
    test_acc=softmax_mxnet(features_train=x_train,labels_train=y_train,features_test=x_test,labels_test=y_tes
    batch_size=batch_size[1],lr=lr[i],reg=reg[1],w=w,b=b,save_path=save_path)
    lr_acc.append(test_acc)
print('train lr done.')
best_lr=best_parm(lr,lr_acc)
print('the best lr: {}'.format(best_lr))
print('--------------------')
```

图 6: 交叉验证选择超参数 lr

```python
test_acc=[]
for i in range(20):
    print('training {}...'.format(i))
    x_train,y_train,x_test,y_test=_final_dataset(x_data=x_data,y_data=
    temp_acc=softmax_mxnet(features_train=x_train,labels_train=y_train
    batch_size=best_batch_size,lr=best_lr,reg=best_reg,save_path=save_
    test_acc.append(temp_acc)
    print('done.')
    print('--------------------')
acc_mean=np.mean(test_acc)
acc_var=np.var(test_acc)
print('Acc: mean = {} \tvar = {} '.format(acc_mean,acc_var))
softmax_tf(x_train,y_train,x_test,y_test)
```

图 7: 计算模型准确率均值与方差

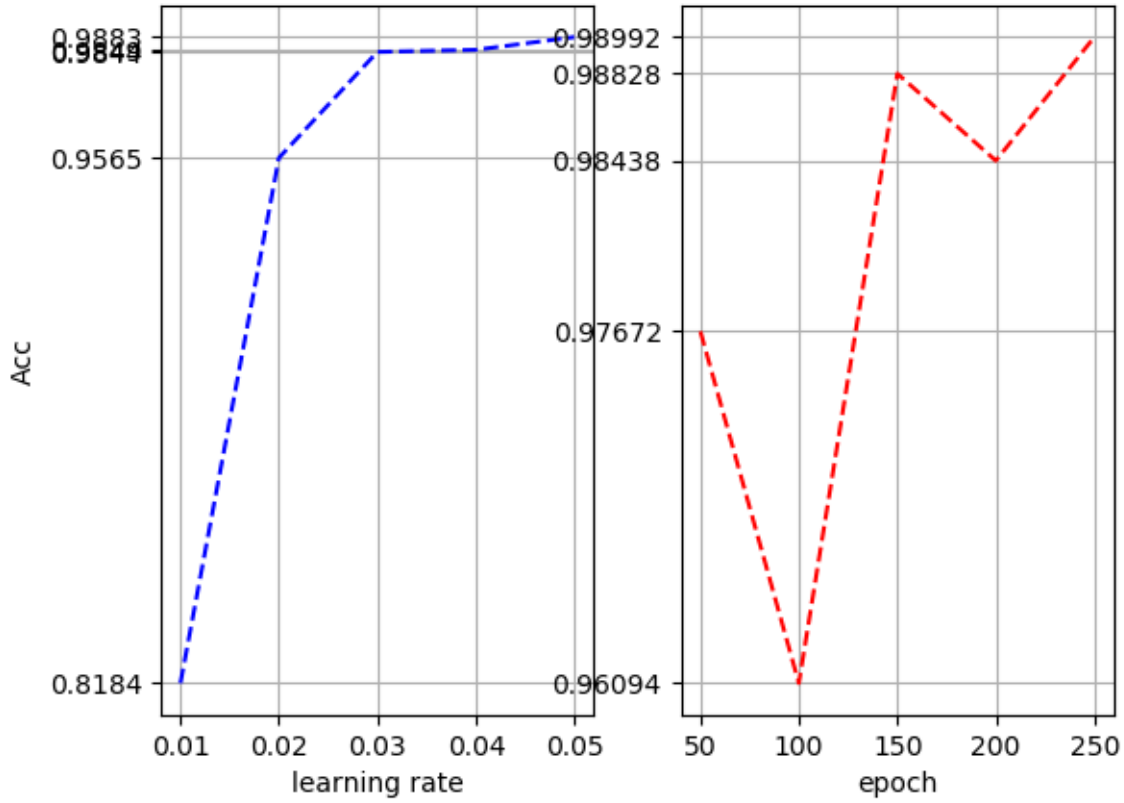

图 8: 准确率随学习率、总迭代次数变化

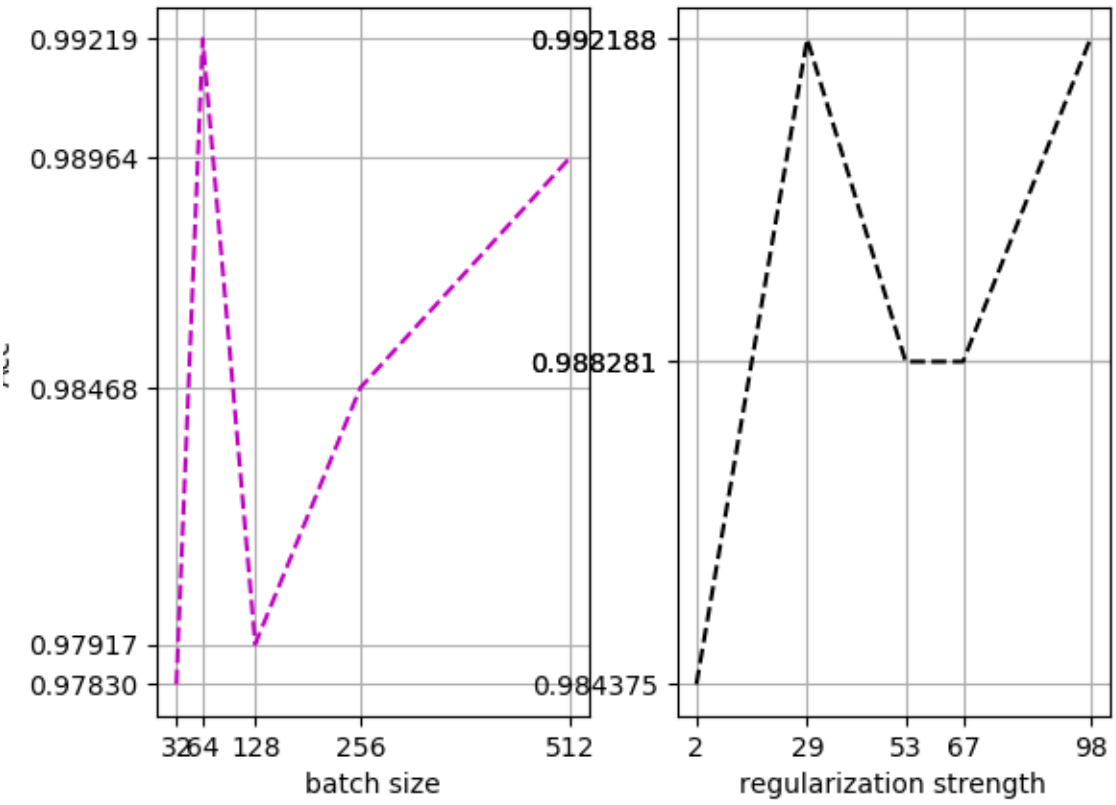

图 9: 准确率随小批量、正则化项变化

```
43          print('------------------')
44      acc_mean=np.mean(test_acc)
45      acc_var=np.var(test_acc)
46      print('Acc: mean = {} \tvar = {} '.format(acc_mean,acc_var)
47  #     softmax_tf(x_train,y_train,x_test,y_test)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

------------------
training 17...
done.
------------------
training 18...
done.
------------------
training 19...
done.
------------------
Acc: mean = 0.9647267857142857  var = 3.597802455357177e-05
```

图 10: 最终模型准确率的均值与方差