《机器学习基础》实验报告

一、实验目的

掌握线性模型、对率回归算法原理。

二、实验项目内容

1. 理解对率回归算法原理。
2. 编程实现对数几率回归算法。
3. 将算法应用于西瓜数据集 3.0、鸢尾花数据集分类问题。

三、实验过程或算法（源程序）

1. 对数几率回归模型

本实验使用的 Logistics 回归模型虽然名为回归，但实际上是一种二分类模型，由条件概率分布 P(y|**x**)表示。其中随机变量 x 为样本属性的向量，向量的每个元素描述一个样本的对应属性，随机变量 y 取值为 0 或 1，描述其分类标签。P(y=0|**x**)表示该样本为负类的概率，P(y=1|**x**)表示正类的概率。

对于每一个输入的样本属性向量 **x**，对其做一个高维到一维的映射 x->z：

$$z=\mathbf{w}*\mathbf{x}+b$$

其中 w 是转换矩阵，参数值需要训练得到；z 称为预测值，是一个实数。

然而二分类任务的输出标签为 y={0,1}，因此需要将 z 映射成 0/1 值。最理想的 z->y 映射为“单位跃阶函数”，但考虑到其不具有连续、光滑、可微等优良数学特性，采用**对数几率函数**替代：

$$y=\frac{1}{1+e^{-z}}$$

该函数在 z=0 处变化很快，z>0 和 z<0 处趋于 0/1，基本满足二分类任务的要求，并且具有连续、光滑、可微等优良的数学特性。

于是变换可得：

$$\ln\frac{y}{1-y}=w*x+b$$

令 P(y=1|x)=y，视为正类的概率；P(y=1|x)=1-y，视为负类的概率。其中 y/(1-y)为事件发生与不发生的概率的比值，称为几率，因此该模型称为对数几率。解微分方程可得：

$$P(y=1\,|x)=\frac{e^{w\cdot x+b}}{1+e^{w\cdot x+b}}$$

$$P(y=0\,|x)=\frac{1}{1+e^{w\cdot x+b}}$$

综上所述，输出 y=1 的对数几率是关于输入 x 的线性函数表示，即对数几率回归模型。

2. 模型参数学习

上述模型中，还有一个关键的参数没有确定，那就是 w，西瓜书上给

出的方法是使用牛顿法/梯度下降法最大化对数似然函数来确定 w。由于梯度下降法求解的是最小值，因此这里将对数似然函数取负数，再采用**梯度下降法最小化对数似然损失函数**（也叫**交叉熵损失函数**）来求解 w。其实更常见的损失函数是最小二乘损失函数，但由于其非凸非凹的函数特性，不便于做梯度下降，因此本实验不采用。

为了使得推导过程和求解更为方便，考虑将 w^T*x+b 统一成乘积形式，即令 $\mathbf{w=(b,w)^T}$，$\mathbf{x=(1,x)^T}$ 即可。于是可得 $z=w^T*x$，下面都采用该乘积形式，梯度下降时优化的参数 w 和 b 全部合并为 w。

根据交叉熵损失函数定义和上面的简化，可得到损失函数 J(w)如下：

$$J(w)=-\frac{1}{m}\sum_{1}^{m}[yi*\log(h_w(xi))+(1-yi)*\log(1-h_w(xi))]$$

$$h_w(x)=g(w^T*x)$$

$$g(z)=\frac{1}{1+e^{-z}}$$

其中 m 为样本数量，交叉熵损失函数是样本集损失的均值。

根据梯度下降法，不断迭代求解 w，迭代如下：

$$w^{k+1}=w^k-\alpha*\frac{\partial J(w)}{\partial w}$$

$$\frac{\partial J(w)}{\partial w}=\frac{1}{m}\sum_{1}^{m}(h_w(xi)-yi)*xi$$

可以设置迭代次数，迭代次数到达设置值后，得到训练结果 w，回代可进行预测分类。

对于 2 维属性的样本，还可以利用训练好的 w 在平面图上绘出分界线：由于参数合并，事实上 b=w[0]，分界标准为：w[0]+w[1]*x+w[2]*y 是否大于 0，若是则为正类，否则为反类。因此 w[0]+w[1]*x+w[2]*y=0 即为二维平面上的分类边界。

3. 使用对率回归算法实现多分类

上面两个部分介绍了**对数几率回归模型**及其参数求解，但该模型只适用于西瓜数据集，用于解决**二分类问题**。鸢尾花数据集为多分类问题，如果仍想用对数几率回归模型，可以使用 **OvO** 或者 **OvR** 将数据拆分，然后推广到多类。

对于 **OvO** 拆分策略，需要将样本集**按标签类别进行分类**，然后将 k 个类别**两两组合**，训练出 k(k-1)/2 个二分类器。将测试集样本传入这 k(k-1)/2 个二分类器进行预测，得到 k(k-1)/2 个分类预测标签，选取**出现次数最多的分类预测标签**作为最终的分类标签。

对于 **OvR** 拆分策略，需要将样本集**按标签类别进行分类**，每次选取 **1 个类作为正类**，**其余** k-1 个类别作为**负类**，训练出 k 个二分类器。将测试集样本传入这 k 个二分类器进行预测，得到 k 个分类预测标签。因为预测结果的**负类**中包含了不止一个分类，因此需要将其**剔除**。因此选取**出现次数最多的正分类预测标签**作为最终的分类标签。

上述两种拆分策略的预测性能取决于具体数据分布，但多数情况下相

差不大。前者训练时间短，存储开销和测试时间长；后者训练时间长，存储开销和测试时间小。本实验在做鸢尾花数据集分类时将采用 OvO 拆分方法。

4. 实验过程

（以下分析步骤只展示主要功能代码，完整代码见附录）

·对数几率回归模型 LogisticModel.py

（1）设计 LogisticModel 类时，定义其属性如下：

```
class LogisticModel(object):
    def __init__(self,X,y,epoch_times=1000,visible=False,alpha=0.001):
        '''args:
            X(m*n):np数组的样本集,包含m个样本,每个样本包含n个属性
            y:标签集
            m/n:样本数/属性数
            w:权重向量
            X_new:扩展后的样本集
            epoch_times:梯度下降法中的迭代次数
            visible:训练过程是否可视化
            alpha:梯度下降法中的学习率
        '''
        self.X=X
        self.y=y
        self.m=self.X.shape[0]
        self.n=self.X.shape[1]
        self.w=self.Init_wgt()
        self.X_new=self.Add_bias()
        self.visible=visible
        self.alpha=alpha
        self.epoch_times=epoch_times
```

（2）定义 LogisticModel 类的辅助函数，用于扩展输入样本 X、初始化权重矩阵：

```
def Add_bias(self):            #为样本集X增加一列并返回新数组(m*(n+1))
    X_new=np.ones((self.m,self.n+1))
    X_new[:,1:]=self.X         #x=(1,x)
    return X_new
def Init_wgt(self):            #初始化w(n+1,1):-1~1上均匀分布的随机数
    w=np.random.uniform(-1.0,1.0,size=self.n+1)
    w.reshape(self.n+1,1)      #一定要合并大小
    return w
```

这两个函数需要最先定义，用于**初始化成员属性**。

（3）定义 LogisticModel 类的 Sigmoid 函数，并计算其梯度，具体数学推导可查看`模型概述`：

```
def Sigmoid(self,z):             #激活函数
    return expit(z)              #expit(z)=1/(1+np.exp(-z))
def Sigmoid_gradient(self,z):    #激活函数的梯度
    sg=self.Sigmoid(z)
    return sg*(1-sg)             #对应位置相乘即可
```

（4）定义 LogisticModel 类的损失函数，用于记录训练过程中损失函数的下降过程；以及该损失函数的梯度：

```python
def Cost(self,w,x):              #损失函数J(w),实数值
    cst=0.0
    z=np.matmul(x,w)
    for i in range(self.m):
        hw=self.Sigmoid(z[i])
        cst+=self.y[i]*np.log(hw)+(1-self.y[i])*np.log(1-hw)
    cst/=(-self.m)
    return cst
```

```python
def Gradient(self,w,x):          #J(w)对w的导数,与w同维的向量
    cnt=[0]*(self.n+1)
    z=np.matmul(x,w)
    for i in range(self.m):
        hw=self.Sigmoid(z[i])
        cnt+=(hw-self.y[i])*x[i]
    cnt/=self.m
    cnt=np.array(cnt)
    cnt.reshape(self.n+1,1)
    return cnt
```

（5）定义 LogisticModel 类的梯度下降过程，通过最小化损失函数来求解 w：

```python
def GD(self):                     #梯度下降求解w,并记录损失函数的下降过程
    cst=[self.Cost(self.w,self.X_new)]
    for i in range(self.epoch_times):
        if self.visible:          #训练过程可视化
            sys.stderr.write("\rEpoch times: %d/%d "%(i+1,self.epoch_times)+\
                "█ "*(i//(self.epoch_times//20)))
            sys.stderr.flush()

        delta=self.Gradient(self.w,self.X_new)
        self.w-=self.alpha*delta
        cst.append(self.Cost(self.w,self.X_new))
    print()
    return cst
```

由于迭代次数过大时训练时间较长，为了实验者**掌握模型训练进度**，加入了**打印进度条**，将训练过程可视化。

```
Epoch times: 34177/100000 ██████
```

```
Epoch times: 63831/100000 ████████████
```

·预测西瓜数据集 watermelon.py

（1）对于西瓜数据集，先读取 csv 中数据：

```python
#读取西瓜数据集
f=open(r'ML\实验1\data\watermelon_3a.csv',encoding='UTF-8')
reader=csv.reader(f)
rows=[row for row in reader]          #rows[i]=[编号,密度,含糖率,好瓜]
X=[]                                  #样本集
y=[]                                  #标签值
for i in range(1,len(rows)):
    X.append([float(rows[i][1]),float(rows[i][2])])
    y.append(int(rows[i][3]))
X=np.array(X)                         #将list转变成numpy数组
y=np.array(y)
```

（2）实例化模型并进行训练：

```python
#实例化LogisticModel对象并进行训练
ep=1000                               #训练次数,可更改
lm=LM(X,y,ep)
cst=lm.GD()
w=lm.w
```

其中 ep 为训练迭代次数，是一个超参数，其取值会影响模型的准确性：一开始将其设为 1000，准确率只有 53%；将其设为 1000000 时准确率可达 76%。

（3）将训练好的模型测试准确率，并将训练中损失函数下降的过程量化。为了更好的比对预测结果，将预测值 z，预测标签值 y 和真实标签值 $y_{真}$绘制成折线图，并将预测分类与真实分类绘制成散点图：

```python
#计算准确率
pred_z=[]
pred_l=[]
acc=0
for x in lm.X_new:
    z=np.matmul(x,w)
    pred_z.append(z)
    if z>=0:
        pred_l.append(1)
    else:
        pred_l.append(0)
    if(pred_l[len(pred_l)-1]==y[len(pred_l)-1]):
        acc+=1
acc/=len(pred_l)
print('Accuracy of classification: %.2f%%'%(acc*100))
```

```python
#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()
```

```python
#绘制分类结果2(真实分类&预测分类)
fig0=plt.figure(figsize=(10,5))
ax=plt.subplot(121)                      #真实分类
ax.scatter(X[:8,0],X[:8,1])              #正类
ax.scatter(X[8:,0],X[8:,1])              #负类
ax.set_xlabel('density')
ax.set_ylabel('Sugar content')
ax.set_title("Real label")
ax=plt.subplot(122)                      #预测分类
ax.scatter(pred1[:,0],pred1[:,1])
ax.scatter(pred0[:,0],pred0[:,1])
ax.set_xlabel('density')
ax.set_ylabel('Sugar content')
ax.set_title("Prediction label")
a=np.linspace(0,1,1000)
b=-(w[1]*a+w[0])/w[2]
ax.plot(a,b)
plt.show()

#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()
```

·预测鸢尾花数据集 Iris.py

（1）对于鸢尾花数据集，可以直接从 sklearn 中获取：

```python
#读取鸢尾花数据集
iris = datasets.load_iris()              #sklearn中自带的鸢尾花数据集的字典,包含了许多信息
data=iris['data']                        #数据集属性值的np数组
label=iris['target']                     #数据集标签值的np数组
feature=iris['feature_names']            #数据集属性名列表
target=iris['target_names']
target=list(target)                      #标签名列表
```

其中获取到的 iris 实际上是一个字典，其中包含了鸢尾花数据集、标签名、属性名、数据类型、样本数量等信息，通过键值可提取想要的数据。

（2）由于西瓜数据集样本数太少，因此没有对其划分训练集与测试集。而鸢尾花数据集样本较多，因此将其按 7:3 划分为训练集和测试集：

```python
#将数据集按7:3分割成训练集和测试集
data_train=[]                            #训练集属性
data_test=[]                             #训练集标签
label_train=[]                           #测试集属性
label_test=[]                            #测试集标签
for i in range(len(data)):
    if(i%10<7):
        data_train.append(data[i])
        label_train.append(label[i])
    else:
        data_test.append(data[i])
        label_test.append(label[i])
```

（3）前面介绍过，由于鸢尾花问题是一个多分类问题，需要采用 OvO 进行拆分，即按标签类别进行分类后两两组合再进行训练：

```
#从训练集中划分不同类别的样本
k=len(target)                        #类别数
data_train_target=[[] for i in range(k)]     #按类别划分的训练样本属性
for i in range(len(data_train)):
    data_train_target[label_train[i]].append(data_train[i])

#采用OvO拆分策略推广对数几率回归模型
Pred_l=[]                            #存放k(k-1)/2个模型对测试集的预测结果
ep=100000
for i in range(k):
    for j in range(k):
        if(i==j):
            continue
        #将i作为正类,j作为负类
        Xtmp=data_train_target[i].copy()
        Xtmp.extend(data_train_target[j])
        ytmp=[1]*len(data_train_target[i])
        ytmp2=[0]*len(data_train_target[j])
        ytmp.extend(ytmp2)
        Xtmp=np.array(Xtmp)
        ytmp=np.array(ytmp)
        lm=LM(Xtmp,ytmp,ep)
        cst=lm.GD()
        w=lm.w
        Pred_l.append(Cal(data_test,w,i,j))
        print("Finished training on the basis of Trainset %d and %d."%(i,j))
pred_l=[]                            #存放测试集的最终预测结果
```

值得注意的是，两两组合时选取前者为正类，后者为负类，才能够适用于 LM 模型。否则将其原始类别带入模型计算会因为一些标签超出 0/1 带来问题。

（4）对每个测试样本，选取分类结果最多标签值作为其类别，并计算其准确率：

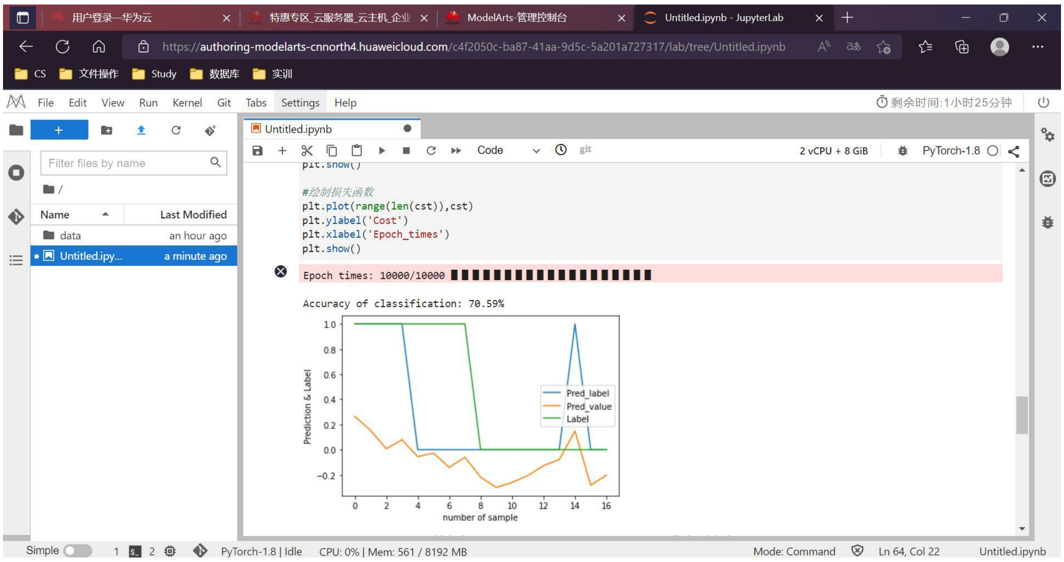
```
for i in range(len(Pred_l[0])):      #遍历列
    l_lst={}                         #暂时存放一列中不同标签的出现次数
    for j in range(len(Pred_l)):     #遍历一列中的每个元素
        l_lst[Pred_l[j][i]]=l_lst.get(Pred_l[j][i],0)+1
    k,v=max(l_lst.items(),key=lambda x:x[1])
    pred_l.append(k)
print(pred_l)

#计算准确率
acc=0
for i in range(len(pred_l)):
    if(pred_l[i]==label_test[i]):
        acc+=1
acc/=len(label_test)
print('Accuracy of classification: %.2f%%'%(acc*100))
```

由于鸢尾花数据集有 4 种属性，因此无法通过散点坐标将其汇入一张

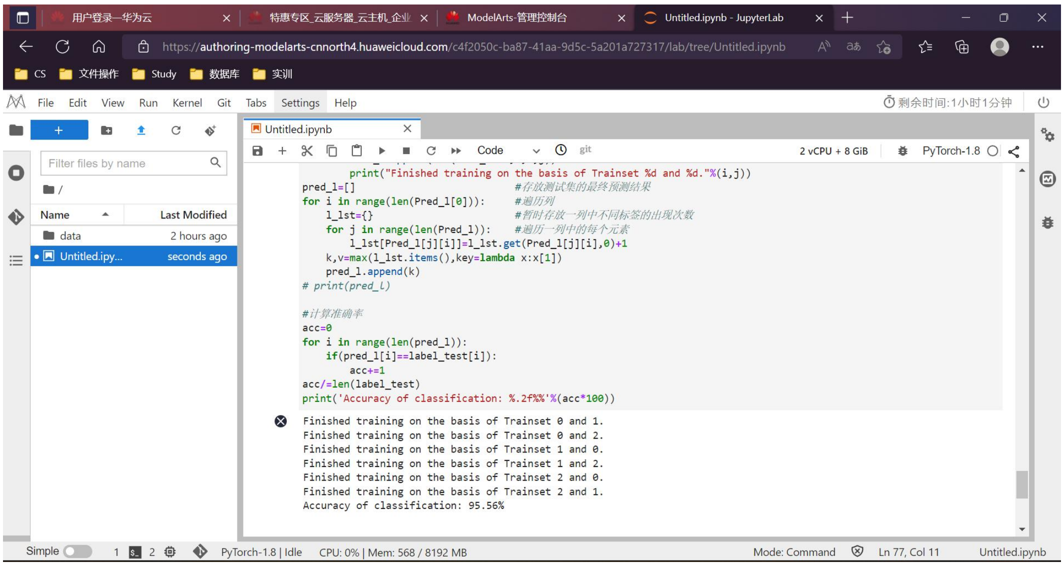
图（现实空间目前没有发现 4 维图像）。此处略去其结果的可视化过程。

• 开发平台

华为云 ModelArts JupyterLab：



（此图为西瓜数据集迭代 10000 次的训练结果，但随机初始化 w 矩阵会对结果造成一定的不稳定性）



（此图为鸢尾花数据集迭代 3000 次的结果）

四、实验结果及分析

实验结果

• 西瓜数据集

对于**西瓜数据集**，迭代次数 ep 很大程度上决定了模型的精度：

ep=1000 时，准确率只有 41.36%；

ep=10000 时，准确率为 52.94%；

ep=100000 时，准确率达到 64.71%；

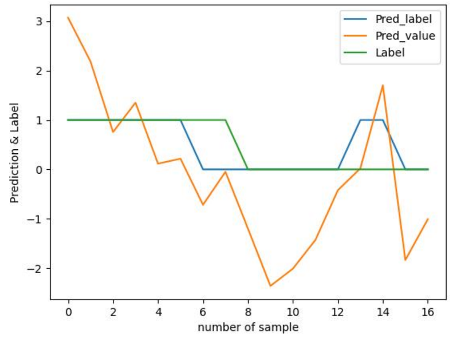
ep=1000000 时，准确率提高到 76.47%；

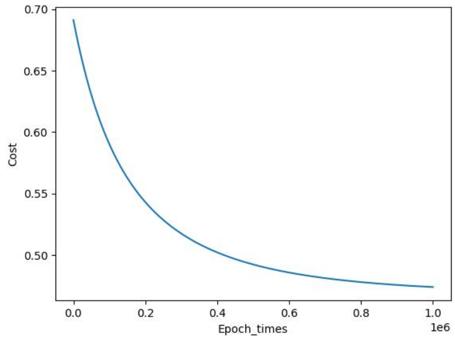
ep=10000000 时，准确率提高到 **88.24%**；（就是很费时间，花了半个多小时…）

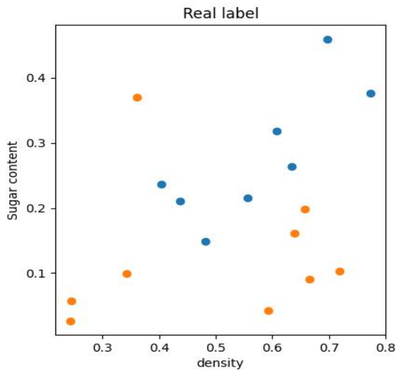
以迭代次数为 10^6 和 10^7 为例，终端输出模型准确率，预测值、预测标签和真实标签，损失函数：

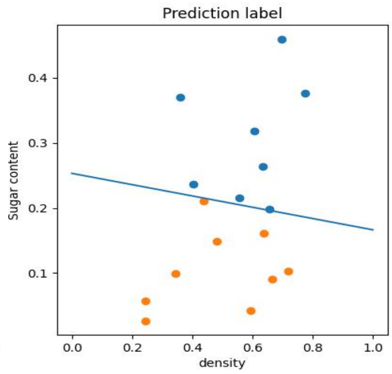
ep=10^6：

```
Epoch times: 1000000/1000000 ||||||||||||||||||
Accuracy of classification: 76.47%
```



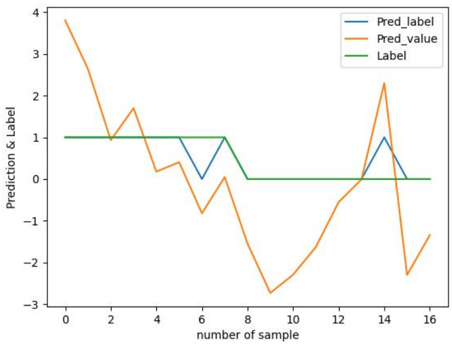


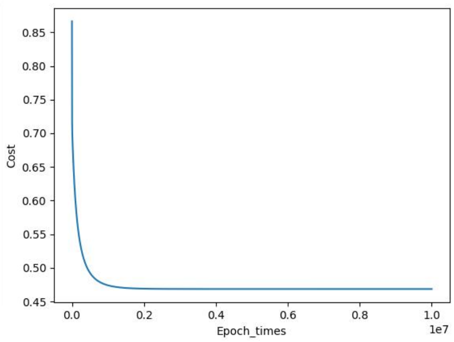


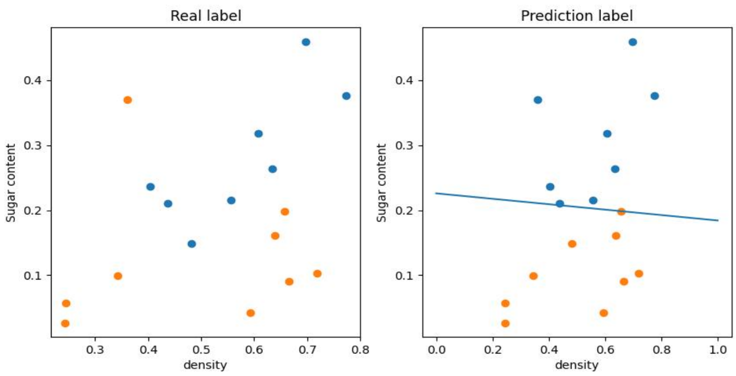


ep=10^7：

```
Epoch times: 10000000/10000000 ||||||||||||||||||
Accuracy of classification: 88.24%
```







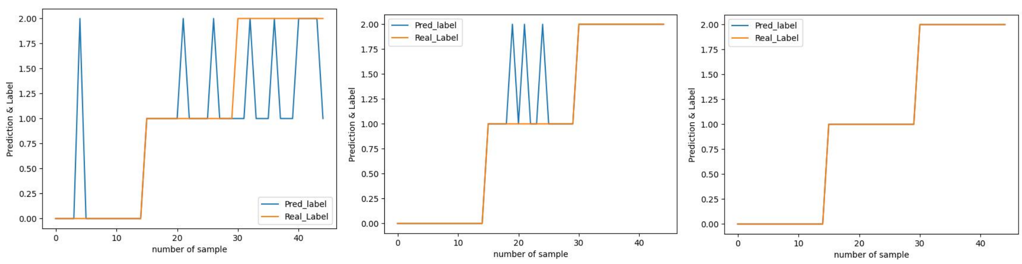
• 鸢尾花数据集

对于**鸢尾花数据集**，迭代次数 ep 也可以决定模型的精度：

ep=1000 时，准确率只有 73.33%；

ep=3000 时，准确率提高到 95.56%；

ep=5000 时，准确率就高达 **100.00%**；



如上图从左到右为 3 种 ep 取值下的分类结果，可以看到，当迭代次数达到 5000 时，模型就可以实现完美准确的分类。该取值下的终端输出结果截图如下：

```
Finished training on the basis of Trainset 0 and 1.
Finished training on the basis of Trainset 0 and 2.
Finished training on the basis of Trainset 1 and 0.
Finished training on the basis of Trainset 1 and 2.
Finished training on the basis of Trainset 2 and 0.
Finished training on the basis of Trainset 2 and 1.
Accuracy of classification: 100.00%
```

前几行是 OvO 拆分过程的模型训练，最后一行为模型的最终准确率。

调试过程：

1. **问题**：初始化 w 时用随机数生成，计算 x*w 时维数不匹配；

解决方法：初始化 w 后需要 reshape 维数：

```
w=np.random.uniform(-1.0,1.0,size=self.n+1)
w.reshape(self.n+1,1)          #一定要合并大小
```

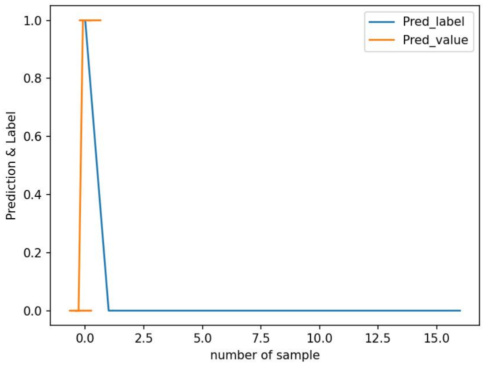
2. **问题**：计算 x*w 结果的维数为 n*m，而不是 n*1

```
[[-0.66497617  0.217422   -1.59000641]
 [-0.66497617  0.65226599 -3.18001282]
 [-0.66497617  1.08710999 -4.77001923]
 [-0.66497617  1.52195399 -6.36002564]
 [-0.66497617  1.95679798 -7.95003205]]
```

解决方法：计算矩阵相乘需要用 **np.matmul()**而不是*：

```
[-2.03756058 -3.192723   -4.34788541 -5.50304783 -6.65821024]
```

3. **问题**：绘图时绘制出的不是函数：



解决方法：坐标轴中绘制 3 条线时不能将其与 x 一起放入 plt.plot()，否则会被编译成两两对应的横纵坐标

```
# plt.plot(range(len(pred_l)),pred_l,pred_z,y)
plt.plot(range(len(pred_l)),pred_l,pred_z)
plt.plot(range(len(pred_l)),y)
```

4. **问题**：处理鸢尾花数据集时将其按 7:3 分割为训练集和测试集，原本打算将前 70%作为训练集，但后 30%全是第 3 类，造成样本严重不平衡：

```
data_train=data[:105]              #训练集属性
data_test=data[105:]               #训练集标签
label_train=label[:105]            #测试集属性
label_test=label[105:]             #测试集标签
```

解决方法：将其按每 10 个中 7 个放入训练集 3 个放入测试集即可：

```
#将数据集按7:3分割成训练集和测试集
data_train=[]                      #训练集属性
data_test=[]                       #训练集标签
label_train=[]                     #测试集属性
label_test=[]                      #测试集标签
for i in range(len(data)):
    if(i%10<7):
        data_train.append(data[i])
        label_train.append(label[i])
    else:
        data_test.append(data[i])
        label_test.append(label[i])
```

5. **问题**：在 OvO 拆分类别时需要复制 data 中数组，直接赋值会使得后续操作改变 data；

解决方法：采用 copy()复制数组即可：

```
Xtmp=data_train_target[i].copy()
```

6. **问题**：OvO 拆分后训练集用于 LogisticModel 模型训练出现数学错误；
解决方法：不能使用其各自标签，而是采用 0/1，否则 LogisticModel 模型计算损失函数时会出现未定义的数学问题：

```
#将i作为正类,j作为负类
Xtmp=data_train_target[i].copy()
Xtmp.extend(data_train_target[j])
ytmp=[1]*len(data_train_target[i])
ytmp2=[0]*len(data_train_target[j])
ytmp.extend(ytmp2)
```

7. **问题**：迭代次数过大时，模型准确率降低；
解决方法：迭代次数过大，训练过拟合，因此需要在保证准确率的前提下选择合适的迭代次数。

五、模型总结与思考

对数几率回归模型，虽然名字是回归，但是实际上它是处理分类问题的算法。将样本特征 x 与预测标签 y 构造成 $\ln\frac{y}{1-y}=w*x+b$ 的形式，通过梯度下降法拟合参数矩阵 w，最终实现 x->y 的映射。当 y>0.5 时则预测为正类，否则为负类。

通过实验发现，模型超参数、数据集样本数、数据集样本分布都会影响模型的训练效果和准确率：

• **模型超参数**：本实验中的模型超参数主要是训练次数和学习率。由于本实验中学习率设置的较小，增加了训练时间，但对准确性影响不大。因此此处主要分析**训练次数**对模型准确性：当训练次数较少时，模型训练参数 w 未收敛，因此准确率较低；随着训练次数增多，w 逐渐收敛，准确率不断提高；当训练次数过多，一旦超过测试集拐点，可能会出现过拟合现象（如问题 7）。

• **随机初始化矩阵**：由于模型在初始化 w 时采用随机数，因此在同一超参数下模型的训练准确率会上下波动。经实验发现，同一超参数下模型随机初始化参数矩阵可能使准确率上下波动 10%。

• **数据集**：对比一下同一模型在两个数据集上的训练效果：由于**西瓜数据集样本数量太少**，分割训练集与测试集后几乎无法完成训练和测试，因此将数据集同时作为训练集与测试集。但即便用训练集测试数据，因为数据太少的缘故，即使训练 10000000 次，**准确率较低**也难以超过 90%。反观鸢尾花数据集，样本数较多，将其按 7:3 划分训练集和测试集，迭代 5000 次即可得到 100%的准确率。

值得一提的是，由于西瓜数据集只有 17 个样本，不仅训练效果不好，在随机初始矩阵的影响下，训练结果也相当**不稳定**。即使较多迭代次数下同一模型也可能得到相差较大的准确性，鸢尾花数据集就基本没有该不稳定性。

其实对数几率回归及模型还有一定的改进空间，如加入正则化项防止过拟合、softmax 推广 logistic 模型等。

实验中还有两个时常困扰我的问题，一个是 numpy 数组和 list 不同使

用方法以及混淆，时常在绘图时出现类型错误。由于 numpy 数组在数据统计和绘图中功能强大，list 更容易操作，因此实验中在绘图前一律采用 list，只有当需要传入 plot()时再转换成 numpy 数组。另一个是矩阵乘法，一开始没有注意，直接使用 a*b 的方式，后来出现维数不匹配的问题，调试发现 a*b 表示矩阵按位相乘，应该使用 np.matmul()作矩阵乘法。

通过本次实验，我加深了对对数几率回归模型原理的理解，在实践中收获了许多理论知识学习时难以发现的细节问题。

六、代码附录

LogisticModel.py

```
'''对数几率回归模型
    z=w*x+b:将样本属性映射为预测值
    y=g(z):将标签值映射到分类标签(也不完全是 0/1,Sigmoid 函数取值只是接近 0 或 1)
    交叉熵作为损失函数,使用梯度下降法求解 w
'''
import sys
import numpy as np
from scipy.special import expit

class LogisticModel(object):
    def __init__(self,X,y,epoch_times=1000,visible=False,alpha=0.001):
        '''args:
            X(m*n):np 数组的样本集,包含 m 个样本,每个样本包含 n 个属性
            y:标签集
            m/n:样本数/属性数
            w:权重向量
            X_new:扩展后的样本集
            epoch_times:梯度下降法中的迭代次数
            visible:训练过程是否可视化
            alpha:梯度下降法中的学习率
        '''
        self.X=X
        self.y=y
        self.m=self.X.shape[0]
        self.n=self.X.shape[1]
        self.w=self.Init_wgt()
        self.X_new=self.Add_bias()
        self.visible=visible
        self.alpha=alpha
        self.epoch_times=epoch_times

    def Add_bias(self):                #为样本集 X 增加一列并返回新数组(m*(n+1))
        X_new=np.ones((self.m,self.n+1))
```

```
        X_new[:,1:]=self.X            #x=(1,x)
        return X_new
    def Init_wgt(self):               #初始化 w(n+1,1):-1~1 上均匀分布的随机数
        w=np.random.uniform(-1.0,1.0,size=self.n+1)
        w.reshape(self.n+1,1)         #一定要合并大小
        return w

    def Sigmoid(self,z):              #激活函数
        return expit(z)               #expit(z)=1/(1+np.exp(-z))
    def Sigmoid_gradient(self,z):     #激活函数的梯度
        sg=self.Sigmoid(z)
        return sg*(1-sg)              #对应位置相乘即可

    def Cost(self,w,x):               #损失函数 J(w),实数值
        cst=0.0
        z=np.matmul(x,w)
        for i in range(self.m):
            hw=self.Sigmoid(z[i])
            cst+=self.y[i]*np.log(hw)+(1-self.y[i])*np.log(1-hw)
        cst/=(-self.m)
        return cst
    def Gradient(self,w,x):           #J(w)对 w 的导数,与 w 同维的向量
        cnt=[0]*(self.n+1)
        z=np.matmul(x,w)
        for i in range(self.m):
            hw=self.Sigmoid(z[i])
            cnt+=(hw-self.y[i])*x[i]
        cnt/=self.m
        cnt=np.array(cnt)
        cnt.reshape(self.n+1,1)
        return cnt

    def GD(self):                     #梯度下降求解 w,并记录损失函数的下降过程
        cst=[self.Cost(self.w,self.X_new)]
        for i in range(self.epoch_times):
            if self.visible:          #训练过程可视化
                sys.stderr.write("\rEpoch times: %d/%d "%(i+1,self.epoch_times)+\
                    "█ "*(i//(self.epoch_times//20)))
                sys.stderr.flush()
            delta=self.Gradient(self.w,self.X_new)
            self.w-=self.alpha*delta
            cst.append(self.Cost(self.w,self.X_new))
        return cst
```

```
# arr=np.array([[1,2],[3,4],[5,6],[7,8],[9,10]])
# y=np.array([0.1,0.2,0.3,0.4,0.5])
# lm=LogisticModel(arr,y,1000)
# cst=lm.GD()
# print()
# print(cst[950:])
```

watermelon.py

```
'''西瓜数据集 3.0
    仅对其中的"密度"和"含糖率"属性做对数几率回归分类
    呈现训练过程,绘制分类结果,评估模型准确性
'''
import csv
import numpy as np
import matplotlib.pyplot as plt
from LogisticModel import LogisticModel as LM

#读取西瓜数据集
f=open(r'ML\实验 1\data\watermelon_3a.csv',encoding='UTF-8')
reader=csv.reader(f)
rows=[row for row in reader]          #rows[i]=[编号,密度,含糖率,好瓜]
X=[]                                   #样本集
y=[]                                   #标签值
for i in range(1,len(rows)):
    X.append([float(rows[i][1]),float(rows[i][2])])
    y.append(int(rows[i][3]))
X=np.array(X)                          #将 list 转变成 numpy 数组
y=np.array(y)

#实例化 LogisticModel 对象并进行训练
ep=100000                              #训练次数,可更改
lm=LM(X,y,ep,True)
cst=lm.GD()
w=lm.w

#计算准确率
pred_z=[]
pred_l=[]
acc=0
for x in lm.X_new:
    z=np.matmul(x,w)
    pred_z.append(z)
    if z>=0:
        pred_l.append(1)
```

```
    else:
        pred_l.append(0)
    if(pred_l[len(pred_l)-1]==y[len(pred_l)-1]):
        acc+=1
acc/=len(pred_l)
print('\nAccuracy of classification: %.2f%%'%(acc*100))

pred1=[]                                #预测正类的样本点集合
pred0=[]                                #预测负类的样本点集合
for i in range(len(pred_l)):
    if(pred_l[i]==1):
        pred1.append([X[i][0],X[i][1]])
    else:
        pred0.append([X[i][0],X[i][1]])
pred1=np.array(pred1)
pred0=np.array(pred0)

#绘制分类结果 1(预测值+预测标签+真实标签)
plt.plot(range(len(pred_l)),pred_l,pred_z)
plt.plot(range(len(pred_l)),y)
plt.ylabel('Prediction & Label')
plt.xlabel('number of sample')
plt.legend(['Pred_label','Pred_value','Label'])
plt.show()

#绘制分类结果 2(真实分类&预测分类)
fig0=plt.figure(figsize=(10,5))
ax=plt.subplot(121)                      #真实分类
ax.scatter(X[:8,0],X[:8,1])              #正类
ax.scatter(X[8:,0],X[8:,1])              #负类
ax.set_xlabel('density')
ax.set_ylabel('Sugar content')
ax.set_title("Real label")
ax=plt.subplot(122)                      #预测分类
ax.scatter(pred1[:,0],pred1[:,1])
ax.scatter(pred0[:,0],pred0[:,1])
ax.set_xlabel('density')
ax.set_ylabel('Sugar content')
ax.set_title("Prediction label")
a=np.linspace(0,1,1000)
b=-(w[1]*a+w[0])/w[2]
ax.plot(a,b)
plt.show()
```

```
#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()
```

Iris.py

```
'''鸢尾花数据集
    150 个样本,3 个类别('setosa', 'versicolor', 'virginica'),
    4 个属性('sepal length/cm','sepal width/cm','petal length/cm','petal width/cm'),
    使用 OvO 拆分类别推广对数几率回归模型
    呈现训练过程,绘制分类结果,评估模型准确性
'''
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from LogisticModel import LogisticModel as LM

def Cal(X,w,c1,c0):                    #计算 X*w,预测 X_new 对应分类
    '''args:
        X:测试集
        w:参数
        c1:正类标签
        c0:负类标签
    '''
    pred=[]
    for x in X:
        x_new=list(x.copy())
        x_new.insert(0,1)
        z=np.matmul(x_new,w)
        if z>=0:
            pred.append(c1)
        else:
            pred.append(c0)
    return pred

#读取鸢尾花数据集
iris = datasets.load_iris()          #sklearn 中自带的鸢尾花数据集的字典,包含了许多信
息
data=iris['data']                    #数据集属性值的 np 数组
label=iris['target']                 #数据集标签值的 np 数组
feature=iris['feature_names']        #数据集属性名列表
target=iris['target_names']
target=list(target)                  #标签名列表
```

```
#将数据集按 7:3 分割成训练集和测试集
data_train=[]                        #训练集属性
data_test=[]                         #训练集标签
label_train=[]                       #测试集属性
label_test=[]                        #测试集标签
for i in range(len(data)):
    if(i%10<7):
        data_train.append(data[i])
        label_train.append(label[i])
    else:
        data_test.append(data[i])
        label_test.append(label[i])

#从训练集中划分不同类别的样本
k=len(target)                        #类别数
data_train_target=[[] for i in range(k)]     #按类别划分的训练样本属性
for i in range(len(data_train)):
    data_train_target[label_train[i]].append(data_train[i])

#采用 OvO 拆分策略推广对数几率回归模型
Pred_l=[]                            #存放 k(k-1)/2 个模型对测试集的预测结果
ep=100000
for i in range(k):
    for j in range(k):
        if(i==j):
            continue
        #将 i 作为正类,j 作为负类
        Xtmp=data_train_target[i].copy()
        Xtmp.extend(data_train_target[j])
        ytmp=[1]*len(data_train_target[i])
        ytmp2=[0]*len(data_train_target[j])
        ytmp.extend(ytmp2)
        Xtmp=np.array(Xtmp)
        ytmp=np.array(ytmp)
        lm=LM(Xtmp,ytmp,ep)
        cst=lm.GD()
        w=lm.w
        Pred_l.append(Cal(data_test,w,i,j))
        print("Finished training on the basis of Trainset %d and %d."%(i,j))
pred_l=[]                            #存放测试集的最终预测结果
for i in range(len(Pred_l[0])):      #遍历列
    l_lst={}                         #暂时存放一列中不同标签的出现次数
    for j in range(len(Pred_l)):     #遍历一列中的每个元素
```

```
        l_lst[Pred_l[j][i]]=l_lst.get(Pred_l[j][i],0)+1
    k,v=max(l_lst.items(),key=lambda x:x[1])
    pred_l.append(k)
print(pred_l)

#计算准确率
acc=0
for i in range(len(pred_l)):
    if(pred_l[i]==label_test[i]):
        acc+=1
acc/=len(label_test)
print('Accuracy of classification: %.2f%%'%(acc*100))
```