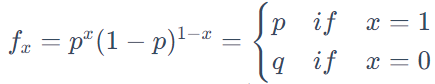
# 2.6.2 逻辑回归二分类到 SoftMax 回归多分类

SoftMax回归多分类是逻辑回归二分类的一种推广，逻辑回归通过最大似然估计更新参数（实际上通常使用梯度下降法）。最大似然估计是似然函数联合概率分布对应的最大值，而似然函数描述伯努利分布中概率（参数）的概率分布，伯努利分布为离散型概率分布。因此对于SoftMax的理解最好是从伯努利分布开始逐步的层层剥离。

### 2.6.2.1 伯努利分布（Bernouli distribution）

伯努利分布（Bernoulli distribution），又名两点分布或者0-1分布，是一个离散型概率分布（为纪念瑞士科学家Jakob I. Bernoulli而命名）。其实验对象只有两种可能结果，例如从不知比例只有黑色和白色球的罐子里取球（取出后需要放回后，再取球），白色为1，黑色为0。为白的概率记作，为黑的概率则为。其概率质量函数为：

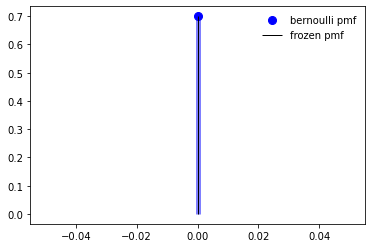
。

指定一种结果（例如为白球）的概率，可以使用scipy.stats库中的bernoulli进行相关计算，例如打印概率质量函数曲线，生成符合概率的随机数组等。如果定义，则生成的随机数组包含0和1两个元素，其比例围绕7:3上下浮动。

概率质量函数（probability mass function，pmf），是离散随机变量在各特定取值上的概率。概率质量函数和概率密度函数不同之处在于：概率质量函数是对离散随机变量定义的，本身代表该值的概率；概率密度函数是对连续随机变量定义的，本身不是概率，只是对连续随机变量的概率密度函数在某区间内进行积分后才是概率。

期望值：在概率论和统计学中，一个离散型随机变量的期望值（或数学期望，亦简称期望）是试验中每次可能结果乘以其结果概率的总和。期望值是该变量输出值的加权平均，可能与每一个结果都不相等，并不一定包含于其分布值域，也并不一定等于值域均值。

from scipy.stats import bernoulli  
import matplotlib.pyplot as plt  
import numpy as np  
  
fig, ax=plt.subplots(1, 1)  
  
p=0.3  
mean,var,skew,kurt=bernoulli.stats(p,moments='mvsk')  
# 打印概率质量函数/分布，pmf  
x=np.arange(bernoulli.ppf(0.01,p),bernoulli.ppf(0.99,p))  
ax.plot(x,bernoulli.pmf(x,p),'bo',ms=8,label='bernoulli pmf')  
ax.vlines(x,0,bernoulli.pmf(x,p),colors='b',lw=5,alpha=0.5)  
  
# Freeze the distribution and display the frozen  
rv=bernoulli(p)  
ax.vlines(x,0,rv.pmf(x),colors='k',linestyles='-',lw=1,label='frozen pmf')  
ax.legend(loc='best',frameon=False)  
plt.show()  
  
# 查看cdf和ppf的精度 Check accuracy of cdf and ppf。 cdf(k,p,loc=0),Cumulative distribution function;ppf(q, p, loc=0),Percent point function (inverse of cdf — percentiles).  
prob=bernoulli.cdf(x,p)  
print(".ppf(.cdf)={}".format(np.allclose(x,bernoulli.ppf(prob,p))))  
  
# 生成符合指定概率p的随机数组 Generate random numbers  
r=bernoulli.rvs(p, size=100)  
unique\_elements, counts\_elements=np.unique(r, return\_counts=True)  
print("生成符合伯努利分布的数组(p={})：\n{}\n包含元素为：{}，对应频数为：{}".format(p,r,unique\_elements, counts\_elements))



.ppf(.cdf)=True  
生成符合伯努利分布的数组(p=0.3)：  
[0 1 0 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 0  
 0 1 0 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 1  
 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 1]  
包含元素为：[0 1]，对应频数为：[61 39]

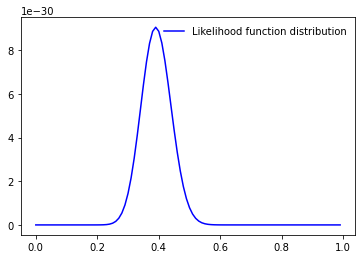
### 2.6.2.2 似然函数（Likelihood function）

在伯努利分布中，应用scipy.stats.bernoulli.rvs生成符合，包含100个要素的随机数组。这个过程可以用实际案例描述为黑罐子里取黑球和白球（两种球的数量比例满足0.3，和1-0.3=0.7）的过程（或者抛掷硬币的过程，如果是抛掷硬币，为1，即正面朝上的概率为0.3时，说明硬币分布不均匀，正面部分的质量更高，更容易让背面朝上），取得白球，即值为1时的概率为0.3；而取得黑球，即值为0时的概率为1-0.3=0.7。即，已知参数，罐子里白球和黑球的比例大约为3:7（球的总数为任意数），可以根据已知参数求得随机变量的输出结果的概率，例如随机抽取100次后，出现白球的概率是多少？（或黑球的概率是多少？），这个过程就是概率的描述；对于上述过程的近似反过程，如只知道指定数量的随机数组，即已知随机变量输出结果（随机抽取100次后，以数组表述黑白球的抽取结果），此时并不知道参数，而需要求得参数取值的概率分布（即估计参数的可能性，的取值范围在），这个过程就是似然的描述。概率函数和似然函数具有一样的模型，区别在于将谁看作变量，将谁看作参数。如果概率函数记作：，其中已知；则似然函数为：，其中已知，为一个包含值1和0（两种情况结果）的随机数组。

概率描述的过程，是应用概率质量函数/分布的公式，代入取得的白球概率，计算黑球（x=0）的概率为，可以推断出在随机取球100次之后，所取得白、黑球（事件）的概率为0.3和0.7。而似然描述的过程，是根据随机抽取100次后，用表述黑白球抽取结果的数组中的值（1，和0），即的值，逐一代入概率质量函数/分布的公式后求积。因为值为1或为0，因此会出现两种结果，当时，结果为；当时，结果为。如果随机抽取的球里，白球有29个，黑球有71个，则似然函数为 pp参数值(就是抽取黑白球的概率值)，将其代入似然函数，可以绘制抽取黑白球概率值的概率分布。

import sympy  
x\_, p\_=sympy.symbols('x p', positive=True)  
pmf=p\_\*\*x\_\*(1-p\_)\*\*(1-x\_) # 构建概率质量函数/分布的公式  
print("(离散型)概率质量函数，pmf={}".format(pmf))  
L=np.prod([pmf.subs(x\_, i) for i in r]) # 推导似然函数/分布公式。np.prod()用于计算所有元素的乘积  
print("似然函数,L={}".format(L))  
  
fig, ax=plt.subplots(1, 1)  
p\_x=np.arange(0,1,0.01) # 生成0-1连续的概率值  
L\_=sympy.lambdify(p\_,L,"numpy")  
ax.plot(p\_x,L\_(p\_x),'b-',ms=8,label='Likelihood function distribution')  
  
ax.legend(loc='best',frameon=False)  
plt.show()

(离散型)概率质量函数，pmf=p\*\*x\*(1 - p)\*\*(1 - x)  
似然函数,L=p\*\*39\*(1 - p)\*\*61



### 2.6.2.3 最大/极大似然估计（Maximum Likelihood Estimation，MLE）

给定一个概率分布，已知其概率密度函数（连续分布，Probability Density Function，pdf）或概率质量函数（离散分布，Probability Mass Function，pmf）为*fD*pnX1​,X2​,…,Xn f\_{D} L(θ∣x1​,x2​,…,xn​)=fθ​(x1​,x2​,…,xn​)Df0f0则为*X1,X2, …,Xn *  
*θ*θ

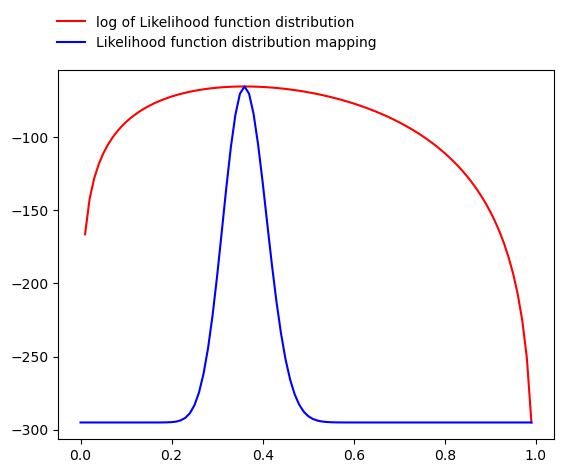
θ*的*最大似然估计。最大似然估计是样本的函数。

求最大似然估计就是对似然函数求导，并令其导函数为0时的解，即曲线变化趋于0的位置。其导函数为，计算相对复杂。而最大化一个似然函数同最大化它的自然对数是等价的，因此对似然函数取对数，其似然函数对数的导函数为，可见计算量得以大幅度减少；而由似然函数积的方式，转换为似然函数对数和的形式，也进一步减少了计算量。

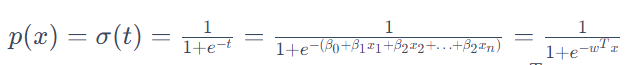
最后求得的最大似然估计值为29/100，与生成符合指定概率的随机数组中的值基本一致。

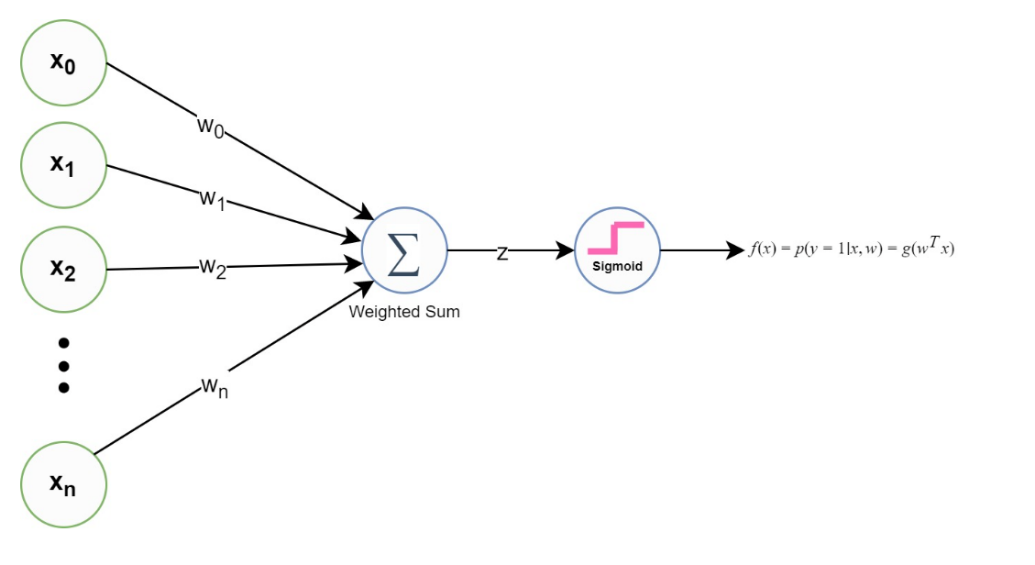
print("似然函数的导函数={}".format(sympy.diff(L,p\_)))  
L\_max\_=sympy.solve(sympy.diff(L,p\_),p\_)  
print("令似然函数的导数为0时，解得最大似然估计={}".format(L\_max\_))  
  
L\_log=sympy.expand\_log(sympy.log(L)) # sympy.expand\_log来简化数学表达式中的对数项  
print("似然函数的对数={}".format(L\_log))  
print("似然函数对数的导函数={}".format(sympy.diff(L\_log,p\_)))  
L\_log\_=sympy.lambdify(p\_,L\_log,"numpy")   
L\_max=sympy.solve(sympy.diff(L\_log,p\_),p\_) # sympy.solve令所有方程等于0，解方程或方程组。即求曲线最大值的位置  
print("令似然函数对数的导数为0时，解得最大似然估计={}".format(L\_max))  
  
fig, ax=plt.subplots(1, 1)  
L\_log\_values=L\_log\_(p\_x)  
ax.plot(p\_x,L\_log\_values,'r-',ms=8,label='log of Likelihood function distribution')  
  
def rescale\_linear(array, new\_min, new\_max):  
 """  
 function - 按指定区间缩放/映射数组/Rescale an arrary linearly.  
   
 Paras:  
 new\_min - 映射区间的最小值  
 new\_max - 映射区间的最大值  
 """  
 minimum, maximum=np.min(array), np.max(array)  
 m=(new\_max - new\_min) / (maximum - minimum)  
 b=new\_min - m \* minimum  
 return m \* array + b  
  
L\_log\_values=L\_log\_values[np.isfinite(L\_log\_values)]  
ax.plot(p\_x,rescale\_linear(L\_(p\_x),L\_log\_values.min(),L\_log\_values.max()),'b-',ms=8,label='Likelihood function distribution mapping')  
  
ax.legend(bbox\_to\_anchor=(0.32, 1), loc=8, borderaxespad=1, frameon=False)  
plt.show()

似然函数的导函数=-61\*p\*\*39\*(1 - p)\*\*60 + 39\*p\*\*38\*(1 - p)\*\*61  
令似然函数的导数为0时，解得最大似然估计=[39/100, 1]  
似然函数的对数=39\*log(p) + log((1 - p)\*\*61)  
似然函数对数的导函数=-61/(1 - p) + 39/p  
令似然函数对数的导数为0时，解得最大似然估计=[39/100]  
  
  
<lambdifygenerated-3>:2: RuntimeWarning: divide by zero encountered in log  
 return (39\*log(p) + log((1 - p)\*\*61))



### 2.6.2.4 逻辑回归(Logistic Regression, LR)[1]

逻辑回归模型为：,其中*σ*(*t*)，归一化到[0,1]之间。而因为为二元分类，类标只有两类值1和0，特征值经过加权和（参数全部初始化为1）喂入sigmoid函数后所得到的值位于[0,1]区间，则该值与类标的差值和就为逻辑回归的损失函数，通过随机梯度下降法更新初始化的参数（并未使用最大似然估计，可能无法解析求解），最终训练所得的参数应该使得逻辑回归模型的预测结果趋近于类标1或0。逻辑回归模型是线性模型和sigmoid函数的组合，这与解释“反向传播”部分所构建的隐含层和输出层的网络结构是一样的，可以互相印证理解。

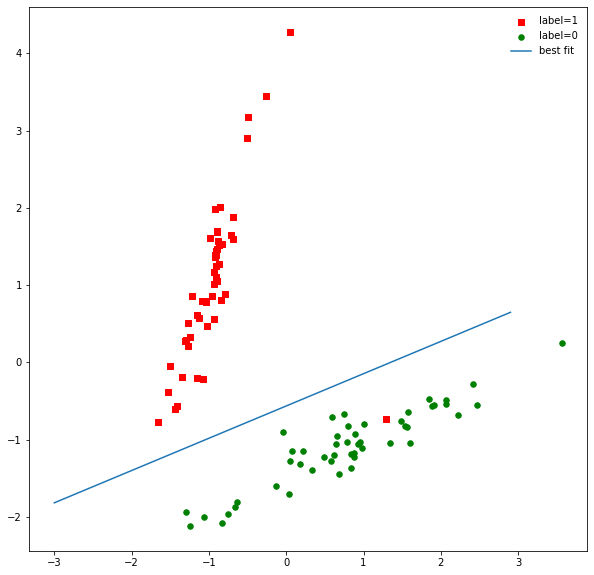


随机梯度下降stocGradAscent1方法的定义直接迁移*Machine learning in action*[2]书中的代码，作者解释了优化的随机梯度下降，主要改进包括：1，每次迭代时，调整更新步长alpha值（即是学习率lr），学习率会越来越小，从而缓解系数的高频波动。同时为了避免迭代不断减小至0，约束学习率大于一个稍微大点的常数项，对应代码为alpha = 4/(1.0+j+i)+0.0001，其中，i和j为迭代次数，随迭代的增加降低alpha值。同时，改变样本的优化顺序，即是随机选择样本来更新回归系数。这样可以减少周期性的波动。对应的代码为randIndex = int(np.random.uniform(0,len(dataIndex)))。

当训练求得参数（权重值）后，通过逻辑回归模型预测的结果是位于0到1之间的浮点数，以0.5为界，大于0.5返回类标为1，否则类标为0作为分类的输出。重新生成分类数据集，用于测试逻辑回归模型精度。

'''  
Created on Oct 27, 2010  
Logistic Regression Working Module /https://github.com/pbharrin/machinelearninginaction/blob/master/Ch05/logRegres.py  
@author: Peter  
  
updated on Fri Jan 8 17:42:57 2021 @author: Richie Bao  
'''  
class LogisticRegression:  
 '''  
 class - 自定义逻辑回归（二元分类）  
 '''  
 def \_\_init\_\_(self):  
 pass   
   
 def generate\_dataset\_linear(self,slop,intercept,num=100,multiple=10,magnitude=50):   
 '''  
 function - 根据一元一次函数构建二维离散型分类数据集，类标为1和0  
 '''  
 import numpy as np  
  
 X\_1=np.random.random(num)-0.5  
 X\_1\*=multiple  
 X\_2=slop\*X\_1+intercept  
 mag\_random\_val=np.random.random(num)-0.5  
 mag\_random\_val\*=magnitude  
 mask=mag\_random\_val>=0  
 y=mask\*1  
 X\_2+=mag\_random\_val  
 X=np.stack((np.ones(len(X\_1)),X\_1,X\_2),axis=1)  
 return X,y  
   
 def make\_classification\_dataset(self,n\_features=2,n\_classes=2,n\_samples=100,n\_informative=2,n\_redundant=0,n\_clusters\_per\_class=1):   
 '''  
 function - 使用Sklearn提供的make\_classification方法，直接构建离散型分类数据集  
   
 Paras:  
 参数查看sklearn库make\_classification方法  
 '''   
 from sklearn.datasets import make\_classification  
  
 X,y=make\_classification(n\_samples=n\_samples,n\_features=n\_features,n\_classes=n\_classes, n\_redundant=n\_redundant, n\_informative=n\_informative,n\_clusters\_per\_class=n\_clusters\_per\_class)  
 X=np.append(np.ones(len(X)).reshape(-1,1),X,axis=1)  
 return X,y  
   
 def plot(self,X,y,weights,figsize=(10,10)):   
 '''  
 function - 绘制逻辑回归结果  
 '''  
 import matplotlib.pyplot as plt  
  
 fig=plt.figure(figsize=figsize)  
 ax=fig.add\_subplot(111)  
 ax.scatter(X[y==1][:,1],X[y==1][:,2],s=30,c='red',marker='s',label='label=1')  
 ax.scatter(X[y==0][:,1],X[y==0][:,2],s=30,c='green',label='label=0')  
   
 x=np.arange(-3.0,3.0,0.1)  
 y=(-weights[0]-weights[1]\*x)/weights[2]  
 ax.plot(x, y,label='best fit')  
   
 ax.legend(loc='best',frameon=False)  
 plt.show()   
   
 def sigmoid(self,x):  
 '''  
 function - 逻辑回归函数\_sigmoid函数  
 '''  
 return 1.0/(1+np.exp(-x))  
   
 def stocGradAscent1(self,dataMatrix, classLabels, numIter=150):   
 '''  
 function - 随机梯度下降（优化）  
 '''  
 from tqdm import tqdm  
  
 m,n = np.shape(dataMatrix)  
 weights = np.ones(n) # initialize to all ones  
 for j in tqdm(range(numIter)):  
 dataIndex = list(range(m))  
 for i in range(m):  
 alpha = 4/(1.0+j+i)+0.0001 # apha decreases with iteration, does not   
 randIndex = int(np.random.uniform(0,len(dataIndex)))# go to 0 because of the constant  
 h = self.sigmoid(sum(dataMatrix[randIndex]\*weights))  
 error = classLabels[randIndex] - h  
 weights = weights + alpha \* error \* dataMatrix[randIndex]  
 del(dataIndex[randIndex])  
 return weights   
   
 def classifyVector(self,inX, weights):  
 '''  
 function - 代入梯度下降法训练的权重值于逻辑回归函数（sigmoid），预测样本特征对应的类标  
 '''  
 prob = self.sigmoid(sum(inX\*weights))  
 if prob > 0.5: return 1.0  
 else: return 0.0  
   
 def test\_accuracy(self,X\_,y\_,weights):   
 '''  
 function - 测试训练模型  
 '''  
 from tqdm import tqdm  
  
 m,n=np.shape(X\_)  
 matches\_num=0  
 for i in tqdm(range(m)):  
 pred=self.classifyVector(X\_[i,:],weights)   
 if pred==np.bool(y\_[i]):  
 matches\_num+=1  
 accuracy=float(matches\_num)/m  
 print("测试数据集精度：{}".format(accuracy))  
 return accuracy  
   
  
LR=LogisticRegression()  
X,y=LR.make\_classification\_dataset(n\_features=2,n\_classes=2,n\_samples=100)  
weights=LR.stocGradAscent1(X,y,numIter=150)  
  
X\_,y\_=LR.generate\_dataset\_linear(slop=5,intercept=5,num=100,multiple=10,magnitude=50) # 建立测试数据集，LR.make\_classification\_dataset和LR.generate\_dataset\_linear均可以生成测试数据集  
accuracy=LR.test\_accuracy(X\_,y\_,weights)  
LR.plot(X,y,weights)

100%|██████████| 150/150 [00:00<00:00, 848.40it/s]  
 0%| | 0/100 [00:00<?, ?it/s]C:\Users\richi\AppData\Local\Temp/ipykernel\_19124/1082695133.py:101: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool\_` here.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations  
 if pred==np.bool(y\_[i]):  
100%|██████████| 100/100 [00:00<00:00, 132773.16it/s]  
  
  
测试数据集精度：0.75



* sklearn库LogisticRegression方法实现

LogisticRegression方法与上述自定义方法的逻辑回归模型训练的结果保持一致，精度均为0.85。通过不同多次运行（获取不同的训练数据集和测试数据集)，各自结果保持一致。混淆矩阵分类精度计算结果显示，总共100个样本，真实值为0的类标计55个，误判为类标1的为14个；真实值为1的类标计45个，误判为类标1的为1个。精确度、召回率及调和平均值的结果由classification\_report方法计算。

from sklearn.linear\_model import LogisticRegression   
from sklearn.metrics import confusion\_matrix,classification\_report  
import matplotlib.pyplot as plt  
  
logreg=LogisticRegression ()  
logreg.fit(X,y)  
predictions=logreg.predict(X\_)  
CM=confusion\_matrix(y\_,predictions,labels=logreg.classes\_)  
print("confusion\_matrix:\n{}".format(CM))  
plt.matshow(CM)  
plt.title('confusion matrix')  
plt.ylabel('True lable')  
plt.xlabel('Predicted label')  
plt.colorbar()  
plt.show()  
  
print(classification\_report(y\_,predictions))

confusion\_matrix:  
[[39 17]  
 [ 6 38]]

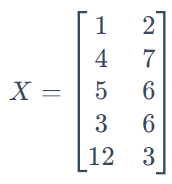


precision recall f1-score support  
  
 0 0.87 0.70 0.77 56  
 1 0.69 0.86 0.77 44  
  
 accuracy 0.77 100  
 macro avg 0.78 0.78 0.77 100  
weighted avg 0.79 0.77 0.77 100

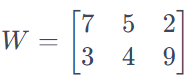
### 2.6.2.5 SoftMax回归（函数、归一化指数函数）

#### 1）自定义SoftMax回归多分类

* 定义线性函数——linear\_weights(self,X)

用于SoftMax函数输入的对象，是由线性函数计算求取的结果。例如假设有两个特征，每个特征下包含5个样本，；

y=[0 ​1 ​2​]

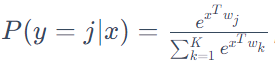
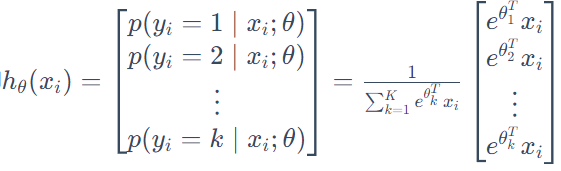
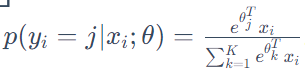
，为2行（特征数）3列（类标数）。将每一个样本（包含两个特征值，例如第一个样本特征值为1，2），分别对应权重矩阵每一列的两个权值（例如第1列，7，3）计算加权和，结果为，以此类推，对于第一个样本特征值其余对应的权重值计算加权和结果为：13，20。其它样本同上。这个计算的过程就是数组的点积计算，可以由np.dot方法实现。由此根据每个类标对应一个初始化的权值，计算了每个样本对应3个类标的输出值，用于SoftMax函数的输入。

X=np.array([[1,2],[4,7],[5,6],[3,6],[12,3]])  
print("X=",X)  
W=np.array([[7,5,2],[3,4,9]])  
print("y=",W)  
z=np.dot(X,W)  
print("z=",z)

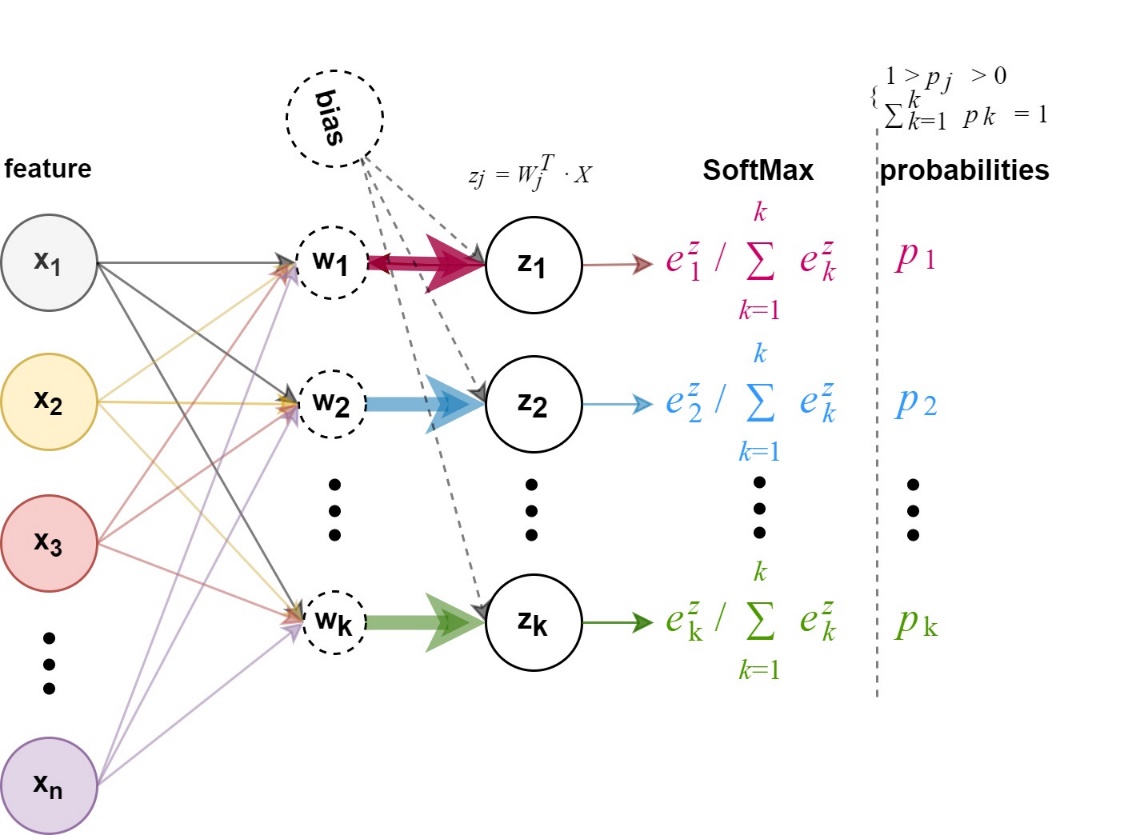
X= [[ 1 2]  
 [ 4 7]  
 [ 5 6]  
 [ 3 6]  
 [12 3]]  
y= [[7 5 2]  
 [3 4 9]]  
z= [[13 13 20]  
 [49 48 71]  
 [53 49 64]  
 [39 39 60]  
 [93 72 51]]

* 定义SoftMax函数——softmax(self,z)[3,4]

SoftMax回归/函数，或称为归一化指数函数，是逻辑回归/函数（Logistic Regression）的一种推广。将一个含任意实数的K维向量z“压缩”到另一个K维实向量中，使得每一个元素的范围都在(0,1)之间，并且所有元素的和为1。该函数的形式通常为：*for j= 1,…, ,K*

*xj*   
*{(*x*1​,*y*1​),(*x*2​,*y*2​),…,(*xn*​,*yn*​)}*yi*∈* {1,2, ,k}x\_{i}  θ*1​,*θ*2​,…,*θk*​∈*θ *x*ij。

不同的作者可能对同一问题的公式表述有所差异，例如不同的变量符号，或者表述规则，例如上述两种阐述中，模型的权值分别被表述为和，亦或者以矩阵的方式表述的方程组等。不管公式的书写方式如何，其核心的算法始终是保持一致。



* 定义损失函数/交叉熵（cross entropy）损失函数——loss(self,y\_one\_hot,y\_probs)

SoftMax运算符将输出变换为一个合法的类标预测分布（联合预测概率，预测概率分布），真实标签通过独热编码转换后的形式也是一种类别分布表达（0值为概率为0，1值为概率为1），二者形状保持一致。评估预测概率和真实标签的分布，只需要估计对应正确类别的预测概率。这样衡量两个概率分布差异的测量函数，可以使用交叉熵（cross entropy），公式为：**  中对应类标（对应正确类别）的预测概率。

根据交叉熵的定义，SoftMax回归的损失函数（代价函数）计算公式为：，其中为样本数，代表类标数，是示性函数，1{值为真的表达式}=1，1{值为假的表达式}=0。SoftMax回归的输出值的形状为(m,k)，每一样本（总共m个样本）对应有k（类标数）个概率值，每一类标对应的概率值为（联合预测概率）。概率值位于[0,1]之间，当取对数后，对应值转换到[-inf,0]之间。概率越大的值，即越趋近于1，当取对数后其值越趋近于0。示性函数对应数据集类标列的独热编码(One-Hot Encoding)。例如类标有4类[0,1,2,3]，假设一个样本对应的类标为3，经过独热编码后，表示为[0. 0. 0. 1.]，可理解为对应类标位置的值取1，而没有对应的位置均为0。而该样本经过SoftMax计算后结果为[0.31132239 0.2897376 0.19737688 0.20156313]，即该样本对应各个类标的概率值。对取对数后，结果为[-1.16692628 -1.23877959 -1.62264029 -1.60165265]，概率越大的值，例如0.31132239，其对数结果越趋近于0，对应-1.16692628。将取对数后的值与该样本类标的独热编码相乘，结果为[-0. -0. -0. -1.60165265]，即仅保留该样本对应类标位置概率值的对数，用以表述概率值的误差（如果概率值为1，即100%的正确，其对数为0，即误差为0；如果概率值为0，即100%的错误，其对数为负无穷，即误差无限大）。计算所有样本的误差值后求和平均，即为SoftMax损失函数的的结果，表示预测误差的大小。

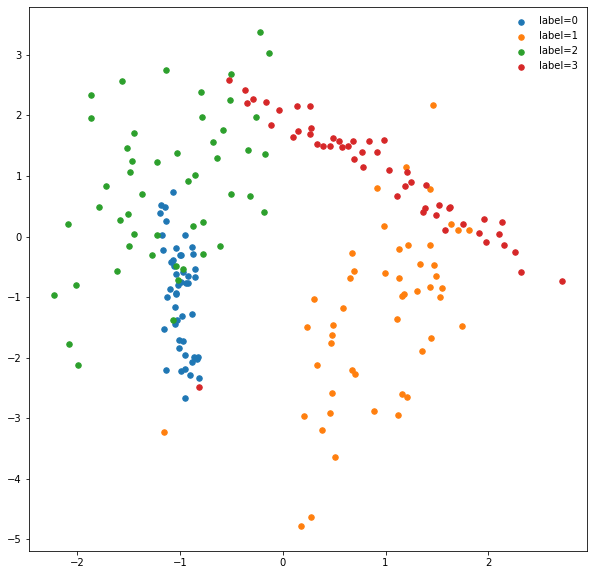
示性函数（特征函数，Characteristic function）可以代表不同的概念，最通常且多数统称为指示函数，方程式为:，其中在集合中，任意子集内一点含值1，于集合内一点含值0。

* 定义梯度下降法，权值更新——gradient(self,X,y\_one\_hot,y\_probs,lr,lambda\_)

利用梯度下降法最小化损失函数，求解的梯度，同时因为如果训练数据不多，容易出现过拟合现象，而增加了一个正则项（具体解释见正则化部分），求解梯度公式为（未给推断过程）：，对应的代码为grad\_loss\_L=-(1.0/self.m\_samples)\*np.dot((y\_one\_hot-y\_probs).T,X)+lambda\_\*self.weights。

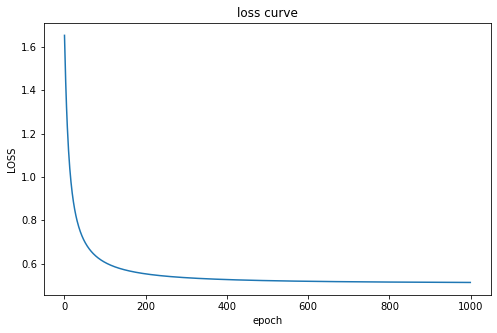
def make\_classification\_dataset\_(n\_features=2,n\_classes=2,n\_samples=100,n\_informative=2,n\_redundant=0,n\_clusters\_per\_class=1,figsize=(10,10),plot=True):  
 '''  
 function - 使用Sklearn提供的make\_classification方法，直接构建离散型分类数据集：并打印查看  
  
 Paras:  
 参数查看sklearn库make\_classification方法  
 '''   
 from sklearn.datasets import make\_classification  
 import matplotlib.pyplot as plt  
 import numpy as np  
  
 X,y=make\_classification(n\_samples=n\_samples,n\_features=n\_features,n\_classes=n\_classes, n\_redundant=n\_redundant, n\_informative=n\_informative,n\_clusters\_per\_class=n\_clusters\_per\_class)  
 print("类标：",np.unique(y))  
   
 if plot:  
 fig=plt.figure(figsize=figsize)  
 ax=fig.add\_subplot(111)  
 for label in np.unique(y):  
 ax.scatter(X[y==label][:,0],X[y==label][:,1],s=30,label='label={}'.format(label))  
  
 ax.legend(loc='best',frameon=False)  
 plt.show()   
  
 return X,y  
  
X,y=make\_classification\_dataset\_(n\_features=2, n\_redundant=0, n\_informative=2,n\_clusters\_per\_class=1, n\_classes=4,n\_samples=200)

类标： [0 1 2 3]



class softmax\_multiClass\_classification\_UDF:  
 '''  
 class - 自定义softmax回归多分类  
 '''  
 def \_\_init\_\_(self,m\_samples,n\_features,n\_classes):  
 self.m\_samples,self.n\_features=m\_samples,n\_features  
 self.n\_classes=n\_classes  
 self.weights=np.random.rand(self.n\_classes,self.n\_features)  
 self.loss\_overall=list()  
   
   
 def softmax(self,Z):  
 '''  
 function - 定义softmax函数  
 '''  
 exp\_sum=np.sum(np.exp(Z),axis=1,keepdims=True)  
 return np.exp(Z)/exp\_sum  
   
 def linear\_weights(self,X):  
 '''  
 function - 定义线性函数  
 '''  
 return np.dot(X,self.weights.T)  
   
 def loss(self,y\_one\_hot,y\_probs):  
 '''  
 function - 定义损失函数  
 '''  
 return -(1.0/self.m\_samples)\*np.sum(y\_one\_hot\*np.log(y\_probs))  
   
 def one\_hot(self,y):  
 '''  
 function - numpy实现one-hot-code  
 '''  
 return np.squeeze(np.eye(self.n\_classes)[y.reshape(-1)])  
   
 def gradient(self,X,y\_one\_hot,y\_probs,lr,lambda\_):  
 '''  
 function - 定义梯度下降法，权值更新  
 '''  
 # 求解梯度  
 grad\_loss\_L=-(1.0/self.m\_samples)\*np.dot((y\_one\_hot-y\_probs).T,X)+lambda\_\*self.weights  
 # 更新权值   
 grad\_loss\_L[:,0]=grad\_loss\_L[:,0]-lambda\_\*self.weights[:,0]  
 self.weights=self.weights-lr\*grad\_loss\_L   
   
 def predict\_test(self,X\_,y\_):  
 '''  
 function - 预测  
 '''  
 y\_probs\_=self.softmax(self.linear\_weights(X\_))  
 y\_pred=np.argmax(y\_probs\_,axis=1).reshape((-1,1))  
 accuracy=np.sum(y\_pred==y\_.reshape((-1,1)))/len(X\_)  
 print("accuracy:%.5f"%accuracy)  
   
 return y\_pred  
   
 def loss\_curve(self,figsize=(8,5)):   
 '''  
 function - 更新打印损失曲线  
 '''  
 import matplotlib.pyplot as plt  
  
 fig=plt.figure(figsize=figsize)  
 plt.plot(np.arange(len(self.loss\_overall)),self.loss\_overall)  
 plt.title("loss curve")  
 plt.xlabel('epoch')  
 plt.ylabel('LOSS')  
 plt.show()  
   
 def train(self,X,y,epochs,lr=0.1,lambda\_=0.01):   
 '''  
 function - 训练模型  
 '''   
 from tqdm import tqdm  
  
 for epoch in tqdm(range(epochs)):  
 # 执行加权和与softmax函数  
 y\_probs=self.softmax(self.linear\_weights(X))  
 # 计算损失函数  
 y\_one\_hot=self.one\_hot(y.reshape((-1,1)))  
 self.loss\_overall.append(self.loss(y\_one\_hot,y\_probs))  
 # 求解梯度与权值更新  
 self.gradient(X,y\_one\_hot,y\_probs,lr,lambda\_)   
   
   
import numpy as np   
m\_samples,n\_features,n\_classes=X.shape[0],X.shape[1],len(np.unique(y))  
sm\_classi=softmax\_multiClass\_classification\_UDF(m\_samples,n\_features,n\_classes)   
epochs=1000  
sm\_classi.train(X,y,epochs,lr=0.1,lambda\_=0.01,)  
sm\_classi.loss\_curve()

100%|██████████| 1000/1000 [00:00<00:00, 13637.49it/s]



X\_,y\_=make\_classification\_dataset\_(n\_features=2, n\_redundant=0, n\_informative=2,n\_clusters\_per\_class=1, n\_classes=4,n\_samples=100,plot=False)  
y\_pred=sm\_classi.predict\_test(X\_,y\_)

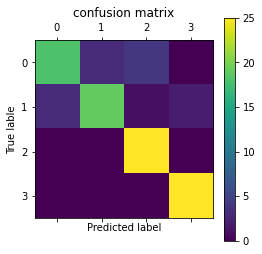
类标： [0 1 2 3]  
accuracy:0.87000

#### 2）sklearn机器学习库实现SoftMax回归多分类

SoftMax回归多分类sklearn库提供的方法同上述的二分类，应用LogisticRegression实现，只是需要配置multi\_class参数（default=’auto’，因此也无需配置，会自动识别）；参数solver需要配置为求解多分类的优化项。其测试数据集的计算结果同上述自定义类的精度接近。

from sklearn.linear\_model import LogisticRegression   
from sklearn.metrics import confusion\_matrix,classification\_report  
import matplotlib.pyplot as plt  
  
logreg\_multiClassi=LogisticRegression (solver='lbfgs',multi\_class="multinomial")  
logreg\_multiClassi.fit(X,y)  
predictions=logreg\_multiClassi.predict(X\_)  
CM=confusion\_matrix(y\_,predictions,labels=logreg\_multiClassi.classes\_)  
print("confusion\_matrix:\n{}".format(CM))  
plt.matshow(CM)  
plt.title('confusion matrix')  
plt.ylabel('True lable')  
plt.xlabel('Predicted label')  
plt.colorbar()  
plt.show()  
  
print(classification\_report(y\_,predictions))

confusion\_matrix:  
[[18 3 4 0]  
 [ 3 19 1 2]  
 [ 0 0 25 0]  
 [ 0 0 0 25]]



precision recall f1-score support  
  
 0 0.86 0.72 0.78 25  
 1 0.86 0.76 0.81 25  
 2 0.83 1.00 0.91 25  
 3 0.93 1.00 0.96 25  
  
 accuracy 0.87 100  
 macro avg 0.87 0.87 0.87 100  
weighted avg 0.87 0.87 0.87 100

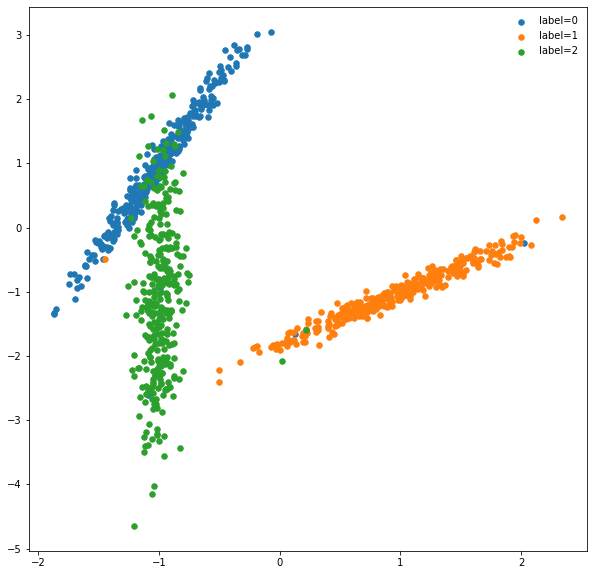
#### 3）PyTorch深度学习库实现SoftMax回归多分类——自定义

应用torch.utils.data库中的TensorDataset可以很方便的建立张量数据集；应用random\_split切分数据集为训练、验证和测试数据集；应用DataLoader将数据集转换为可迭代对象，用于模型训练的数据加载。并可以定义在每轮迭代中随机均匀采样多个样本组成一个小批量（指定batch\_size参数值），使用小批量计算梯度。也可配置参数num\_workers，使用多进程加速数据读取。如果num\_workers=0则为不使用额外的进程来加速读取数据。

PyTorch自定义SoftMax回归的算法同上述阐述，算法保持一致，只是在代码的书写过程中，需要根据PyTorch的语法规则做出调整，尤其PyTorch张量运算自动求梯度的方式，大幅度减轻了代码编写的复杂程度。

X,y=make\_classification\_dataset\_(n\_features=2, n\_redundant=0, n\_informative=2,n\_clusters\_per\_class=1, n\_classes=3,n\_samples=1000)  
import torch.utils.data as data\_utils  
import torch  
# 01-建立数据集  
data\_iter=data\_utils.TensorDataset(torch.from\_numpy(X).double(),torch.from\_numpy(y).long())  
# 02-切分数据集  
train\_set, val\_set=data\_utils.random\_split(data\_iter,[800,200])  
# 03-配置每批大小，batch size  
train\_data\_loader=data\_utils.DataLoader(train\_set,batch\_size=50,shuffle=True,num\_workers=2)  
val\_data\_loader=data\_utils.DataLoader(val\_set,batch\_size=50,shuffle=True,num\_workers=2)

类标： [0 1 2]



class softmax\_multiClass\_classification\_UDF\_pytorch:  
 '''  
 class - 自定义softmax回归多分类\_PyTorch版  
 '''  
 def \_\_init\_\_(self,num\_inputs,num\_outputs):  
 print("num\_inputs/feature={},num\_outputs/label={}".format(num\_inputs,num\_outputs))  
 self.num\_inputs=num\_inputs  
 self.num\_outputs=num\_outputs  
 self.W=torch.tensor(np.random.normal(0, 0.01, (self.num\_inputs, self.num\_outputs)), dtype=torch.double,requires\_grad=True) # 需要对权值求梯度   
 self.b=torch.zeros(num\_outputs,dtype=torch.float,requires\_grad=True) # 需要对偏置求梯度  
 self.params=[self.W,self.b]  
   
 def SoftMax(self,Z):  
 '''  
 function - 定义SoftMax函数  
 '''  
 Z\_exp=Z.exp()  
 exp\_sum=Z\_exp.sum(dim=1,keepdim=True)  
 return Z\_exp/exp\_sum  
   
 def net(self,X):  
 '''  
 function - 定义模型，含线性模型输入，SoftMax回归输出  
 '''  
 return self.SoftMax(torch.mm(X.view((-1,self.num\_inputs)),self.W)+self.b)  
   
 def cross\_entropy(self,y\_pred,y):  
 '''  
 function - 定义交叉熵损失函数  
 '''  
 return -torch.log(y\_pred.gather(1,y.view(-1,1))) # torch.gather，收集输入的特定维度指定位置的数值。即提取出对应正确类别的预测概率  
   
 def accuracy(self,y\_pred,y):  
 '''  
 function - 定义分类准确率，即正确预测数量与总预测数量之比  
 '''  
 return (y\_pred.argmax(dim=1)==y).float().mean().item()  
   
 def evaluate\_accuracy(self,data\_iter):  
 '''  
 funtion - 平均模型net在数据集data\_iter上的准确率  
 '''  
 accu\_sum,n=0.0,0  
 for X,y in data\_iter:  
 accu\_sum+=(self.net(X).argmax(dim=1)==y).float().sum().item()  
 n+=y.shape[0]  
 return accu\_sum/n  
   
 def sgd(self,lr):  
 '''  
 funtion - 梯度下降  
 '''  
 for param in self.params:  
 param.data-=lr\*param.grad   
   
 def train(self,train\_iter,epochs,lr,test\_iter=None):   
 '''  
 function - 训练模型  
 '''   
 from tqdm.auto import tqdm  
  
 for epoch in tqdm(range(epochs)):   
 train\_l\_sum, train\_acc\_sum, n = 0.0, 0.0, 0  
 for X,y in train\_iter:  
 # 01-线性模型输入，SoftMax回归输出  
 y\_pred=self.net(X)  
 # 02-计算损失函数  
 l=self.cross\_entropy(y\_pred,y).sum()  
 # 03-参数梯度清零  
 if self.params is not None and self.params[0].grad is not None:  
 for param in self.params:  
 param.grad.data.zero\_()  
 # 04-计算给定张量的梯度和，此处为损失函数的反向传播  
 l.backward()  
 # 05-求梯度  
 self.sgd(lr)  
 # 06-每批误差和  
 train\_l\_sum += l.item()  
 # 07-每批正确率  
 train\_acc\_sum += (y\_pred.argmax(dim=1) == y).sum().item()  
 n += y.shape[0]  
 if test\_iter is not None:  
 test\_acc=self.evaluate\_accuracy(test\_iter)  
 if epoch%100==0:  
 print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f'% (epoch + 1, train\_l\_sum / n, train\_acc\_sum / n, test\_acc))   
   
  
import torch  
import torchvision  
import numpy as np  
  
num\_inputs,num\_outputs=X.shape[1],len(np.unique(y))  
sm\_classi\_pytorch=softmax\_multiClass\_classification\_UDF\_pytorch(num\_inputs,num\_outputs)  
epochs,lr=1000,0.1  
sm\_classi\_pytorch.train(train\_data\_loader, epochs,lr,val\_data\_loader)

C:\Users\richi\anaconda3\envs\USDA\_database\lib\site-packages\torchvision\io\image.py:11: UserWarning: Failed to load image Python extension: Could not find module 'C:\Users\richi\anaconda3\envs\USDA\_database\Lib\site-packages\torchvision\image.pyd' (or one of its dependencies). Try using the full path with constructor syntax.  
 warn(f"Failed to load image Python extension: {e}")  
  
  
num\_inputs/feature=2,num\_outputs/label=3  
  
  
  
 0%| | 0/1000 [00:00<?, ?it/s]  
  
  
epoch 1, loss 0.4431, train acc 0.800, test acc 0.850  
epoch 101, loss 0.2677, train acc 0.927, test acc 0.945  
epoch 201, loss 0.2896, train acc 0.921, test acc 0.920  
epoch 301, loss 0.2863, train acc 0.925, test acc 0.900  
epoch 401, loss 0.2622, train acc 0.939, test acc 0.910  
epoch 501, loss 0.2723, train acc 0.938, test acc 0.920  
epoch 601, loss 0.2951, train acc 0.938, test acc 0.925  
epoch 701, loss 0.2566, train acc 0.941, test acc 0.925  
epoch 801, loss 0.2672, train acc 0.936, test acc 0.910  
epoch 901, loss 0.2847, train acc 0.944, test acc 0.940

#### 4）PyTorch深度学习库实现SoftMax回归多分类——调用已有算法

注意上述分开定义SoftMax回归和交叉熵损失函数可能会造成数值不稳定，而PyTorch提供的torch.nn.CrossEntropyLoss方法，结合了nn.LogSoftmax()和nn.NLLLoss()于单独的一个类中。因此，在定义net网络时，仅含有一个线性模型，而自定义的flattenLayer是用于转换特征输入向量的形状，也置于net网络中。

此时定义的SoftMax回归多分类方法并未以类的形式出现，而是为单独的函数，可以将flattenLayer、evaluate\_accuracy、sgd\_v1及train\_v1函数置于util\_?.py工具文件中，方便日后直接调用。

import torch  
from torch import nn  
import numpy as np  
from collections import OrderedDict  
  
# 将每批次样本X的形状转换为(batch\_size,-1)  
class flattenLayer(nn.Module):  
 def \_\_init\_\_(self):  
 super(flattenLayer,self).\_\_init\_\_()  
 def forward(self,x):  
 return x.view(x.shape[0],-1)  
  
# 定义模型  
net=nn.Sequential(  
 OrderedDict([  
 ('flatten',flattenLayer()),  
 ('linear',nn.Linear(num\_inputs,num\_outputs))   
 ])  
 )  
  
# 初始化模型的权重参数  
nn.init.normal\_(net.linear.weight,mean=0,std=0.01)  
nn.init.constant\_(net.linear.bias,val=0)  
  
# SoftMax,交叉熵损失函数CrossEntropyLossntropyLoss()  
loss = nn.CrossEntropyLoss()   
  
# 优化算法  
optimizer=torch.optim.SGD(net.parameters(),lr=0.1)  
  
def evaluate\_accuracy(data\_iter, net):  
 '''  
 funtion - 平均模型net在数据集data\_iter上的准确率  
 '''  
 accu\_sum,n=0.0,0  
 for X,y in data\_iter:  
 accu\_sum+=(net(X).argmax(dim=1)==y).float().sum().item()  
 n+=y.shape[0]  
 return accu\_sum/n  
  
def sgd\_v1(params,lr):  
 '''  
 funtion - 梯度下降，v1版  
 '''  
 for param in params:  
 param.data-=lr\*param.grad   
  
def train\_v1(net,train\_iter, test\_iter, loss, num\_epochs,params=None, lr=None, optimizer=None,interval\_print=100):   
 '''  
 function - 训练模型，v1版  
   
 Paras:  
 net - 构建的模型结构  
 train\_iter - 可迭代训练数据集  
 test\_iter - 可迭代测试数据集  
 loss - 损失函数  
 num\_epochs - 训练迭代次数  
 params=None - 初始化模型参数，以列表形式表述，例如[W,b]  
 lr=None, - 学习率  
 optimizer=None - 优化函数  
 interval\_print=100 - 打印反馈信息间隔周期  
 '''  
 from tqdm.auto import tqdm  
  
 for epoch in tqdm(range(num\_epochs)):  
 train\_l\_sum, train\_acc\_sum, n=0.0, 0.0, 0  
 for X, y in train\_iter:  
 y\_pred=net(X)  
 l=loss(y\_pred,y).sum()  
   
 # 梯度清零  
 if optimizer is not None:  
 optimizer.zero\_grad()  
 elif params is not None and params[0].grad is not None:  
 for param in params:  
 param.grad.data.zero\_()  
   
 l.backward()  
 if optimizer is None:  
 sgd\_v1(params,lr) # 应用自定义的梯度下降法  
 else:  
 optimizer.step() # 应用torch.optim.SGD库的梯度下降法  
   
 train\_l\_sum += l.item()  
 train\_acc\_sum += (y\_pred.argmax(dim=1) == y).sum().item()  
 n += y.shape[0]  
 if test\_iter is not None:   
 test\_acc = evaluate\_accuracy(test\_iter, net)  
 if epoch%interval\_print==0:  
 print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f'%(epoch + 1, train\_l\_sum / n, train\_acc\_sum / n, test\_acc))   
  
X,y=make\_classification\_dataset\_(n\_features=2, n\_redundant=0, n\_informative=2,n\_clusters\_per\_class=1, n\_classes=3,n\_samples=1000,plot=False)  
import torch.utils.data as data\_utils  
# 01-建立数据集  
data\_iter=data\_utils.TensorDataset(torch.from\_numpy(X).float(),torch.from\_numpy(y).long())  
# 02-切分数据集  
train\_set, val\_set=data\_utils.random\_split(data\_iter,[800,200])  
# 03-配置每批大小，batch size  
train\_data\_loader=data\_utils.DataLoader(train\_set,batch\_size=50,shuffle=True,num\_workers=2)  
val\_data\_loader=data\_utils.DataLoader(val\_set,batch\_size=50,shuffle=True,num\_workers=2)   
   
num\_epochs=500  
train\_v1(net=net,train\_iter=train\_data\_loader, test\_iter=val\_data\_loader, loss=loss, num\_epochs=num\_epochs,params=None, lr=None, optimizer=optimizer)

类标： [0 1 2]  
  
  
  
 0%| | 0/500 [00:00<?, ?it/s]  
  
  
epoch 1, loss 0.0166, train acc 0.814, test acc 0.875  
epoch 101, loss 0.0051, train acc 0.912, test acc 0.915  
epoch 201, loss 0.0050, train acc 0.914, test acc 0.915  
epoch 301, loss 0.0050, train acc 0.916, test acc 0.915  
epoch 401, loss 0.0050, train acc 0.917, test acc 0.920

#### 5）PyTorch\_SoftMax回归多分类，用于图像数据集Fashion-MNIST

* 图像数据集[Fashion-MNIST](https://www.kaggle.com/zalando-research/fashionmnist)①

Fashion-MNIST包括60,000个例子的训练集和10,000个例子的测试集。每个示例都是一个灰度图像，总共784像素，与来自10个类的标签相关联，可用于替代原始[MNIST手写数字数据集](http://yann.lecun.com/exdb/mnist/)②，对机器学习算法进行基准测试。 图像的像素都有一个与之相关联的像素值，表示该像素的明度和暗度，数字越大表示越暗。像素值为0到255的整数。训练和测试数据集有785列，第一列由标签组成，表示服装的种类。其余的列包含关联图像的像素值。如果要定位一幅图像一个像素的位置，可以应用定位，其中是0到27之间的整数，像素()位于一个矩阵的第i行和第j列。

图像标签（类标）为：

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle boot

服务于PyTorch深度学习框架，主要用来构建计算机视觉模型的[torchvision](https://pytorch.org/vision/stable/index.html)③包中，torchvision.datasets可以用来加载常用的数据集；torchvision.models包含常用的模型结果；torchvision.transforms用于图片的变换；torchvision.utils含有用的工具等。

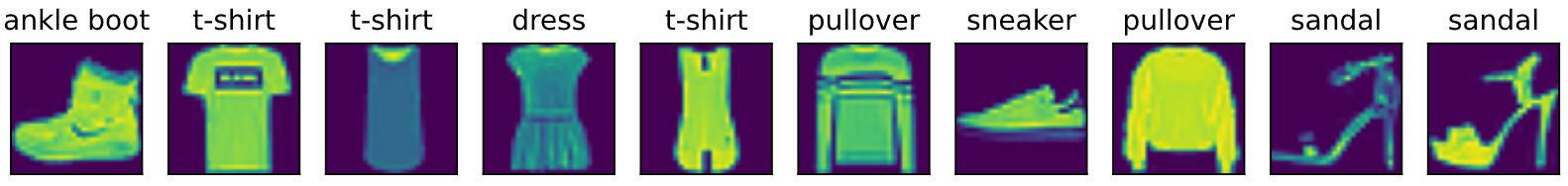
.ToTensor()将所有图像数据shape（h/高,w/宽,c/通道数），像素值位于[0，255]的PIL图片（或者数据类型为np.unit8的NumPy数组），转换为tensor张量shape(c,h,w)，数据类型为torch.float32，像素值位于[0.0,1.0]。

import torchvision  
import torchvision.transforms as transforms  
mnist\_train=torchvision.datasets.FashionMNIST(root='./dataset/', train=True, download=True, transform=transforms.ToTensor())   
mnist\_test=torchvision.datasets.FashionMNIST(root='./dataset/', train=False, download=True, transform=transforms.ToTensor())

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./dataset/FashionMNIST\raw\train-images-idx3-ubyte.gz  
  
  
  
 0%| | 0/26421880 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\raw\train-images-idx3-ubyte.gz to ./dataset/FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\raw\train-labels-idx1-ubyte.gz  
  
  
  
 0%| | 0/29515 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\raw\train-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./dataset/FashionMNIST\raw\t10k-images-idx3-ubyte.gz  
  
  
  
 0%| | 0/4422102 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\raw\t10k-images-idx3-ubyte.gz to ./dataset/FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\raw\t10k-labels-idx1-ubyte.gz  
  
  
  
 0%| | 0/5148 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\raw\t10k-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\raw

print("mnist-数据类型：{}\nmnist\_train大小={},mnist\_test大小={}".format(type(mnist\_train),len(mnist\_train),len(mnist\_test)))  
feature, label=mnist\_train[0]  
  
def fashionMNIST\_label\_num2text(labels\_int):  
 '''  
 function - 将Fashion-MNIST数据集，整型类标转换为名称  
 '''  
 labels\_text=['t-shirt', 'trouser', 'pullover', 'dress', 'coat','sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']  
 return [labels\_text[int(i)] for i in labels\_int]  
  
print("feature.shape={},lable={}，{}".format(feature.shape,label,fashionMNIST\_label\_num2text([label])))  
  
def fashionMNIST\_show(imgs,labels,figsize=(12, 12)):  
 '''  
 function - 打印显示Fashion-MNIST数据集图像  
 '''   
 import matplotlib.pyplot as plt  
 from IPython import display  
 import matplotlib\_inline.backend\_inline  
  
 matplotlib\_inline.backend\_inline.set\_matplotlib\_formats('svg') # 以svg格式打印显示  
 \_, axs=plt.subplots(1, len(imgs), figsize=figsize)  
 for ax,img,label in zip(axs,imgs,labels):  
 ax.imshow(img.view((28,28)).numpy())  
 ax.set\_title(label)  
 ax.axes.get\_xaxis().set\_visible(False)  
 ax.axes.get\_yaxis().set\_visible(False)  
 plt.show()  
imgs\_len=10   
X=[mnist\_train[i][0] for i in range(imgs\_len)]  
y=[mnist\_train[i][1] for i in range(imgs\_len)]  
fashionMNIST\_show(imgs=X,labels=fashionMNIST\_label\_num2text(y),figsize=(12, 12))  
  
# 读取小批量  
import torch.utils.data as data\_utils  
  
batch\_size=256  
num\_workers=4  
train\_iter=data\_utils.DataLoader(mnist\_train, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)  
test\_iter=data\_utils.DataLoader(mnist\_test, batch\_size=batch\_size, shuffle=False, num\_workers=num\_workers)

mnist-数据类型：<class 'torchvision.datasets.mnist.FashionMNIST'>  
mnist\_train大小=60000,mnist\_test大小=10000  
feature.shape=torch.Size([1, 28, 28]),lable=9，['ankle boot']



将上述常用的自定义函数放置于util\_A模块中，方便调用，例如flattenLayer类。外部仅定义了net模型，loss损失函数和optimizer优化函数。

import util\_A  
from torch import nn  
from collections import OrderedDict  
import torch  
  
# 定义模型  
num\_inputs,num\_outputs=28\*28,10  
net=nn.Sequential(  
 OrderedDict([  
 ('flatten',util\_A.flattenLayer()),  
 ('linear',nn.Linear(num\_inputs,num\_outputs))   
 ])  
 )  
  
# SoftMax,交叉熵损失函数CrossEntropyLossntropyLoss()  
loss=nn.CrossEntropyLoss()   
  
# 优化算法  
optimizer=torch.optim.SGD(net.parameters(),lr=0.1)  
  
num\_epochs=500  
util\_A.train\_v1(net,train\_iter, test\_iter, loss, num\_epochs,params=None, lr=None, optimizer=optimizer,interval\_print=100)

0%| | 0/500 [00:00<?, ?it/s]  
  
  
epoch 1, loss 0.0031, train acc 0.746, test acc 0.783  
epoch 101, loss 0.0015, train acc 0.869, test acc 0.843  
epoch 201, loss 0.0014, train acc 0.871, test acc 0.844  
epoch 301, loss 0.0014, train acc 0.874, test acc 0.840  
epoch 401, loss 0.0014, train acc 0.875, test acc 0.841

注释（Notes）：

① 图像数据集 Fashion-MNIST，（<https://www.kaggle.com/zalando-research/fashionmnist>）。

② MNIST 手写数字数据集，（<http://yann.lecun.com/exdb/mnist>）。

③ torchvision，包含流行的数据集、模型架构和计算机视觉的常见图像变换，<https://pytorch.org/vision/stable/index.html>）。

参考文献（References）:

[1] Cavin Hackeling. Mastering Machine Learning with scikit-learn[M].Packt Publishing Ltd.July 2017.Second published. 中文版为：Cavin Hackeling.张浩然译.scikit-learning 机器学习[M].人民邮电出版社.2019.2

[2] Peter Harrington.Machine learning in action[M].NY:Manning Publications; 1st edition (November 24, 2020).

[3] Softmax Activation (NOTES ON STATISTICS, PROBABILITY and MATHEMATICS), http://rinterested.github.io/statistics/softmax.html; https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/.

[4] Aston Zhang, Zack C. Lipton, Mu Li,etc.Dive into Deep Learning[M]. <https://d2l.ai/>；中文版-阿斯顿.张,李沐,扎卡里.C. 立顿,亚历山大.J. 斯莫拉.动手深度学习[M].人民邮电出版社,北京,2019-06-01. <https://zh.d2l.ai/>