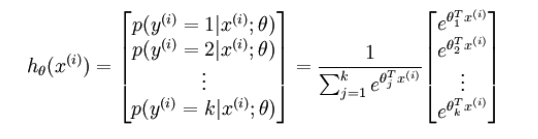
**Softmax回归实验**

曹可想 2016100104008

1. **理论原理**

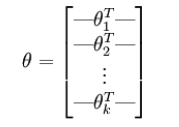
逻辑斯谛回归在处理二分类问题时很有效，但遇到多分类问题时不再起作用,softmax回归使用一种处理多分类问题的有效算法。

对于给定的测试输入,我们想用假设函数针对每一个类别j估算出概率值。也就是说，我们想估计的每一种分类结果出现的概率。因此，我们的假设函数将要输出一个维的向量（向量元素的和为1）来表示这个估计的概率值。具体地说，我们的假设函数形式如下：

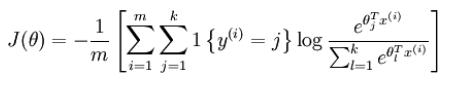


其中是模型的参数。这一项对概率分布进行归一化，使得所有概率之和为1。

为了方便起见我们同样使用符号来表示全部的模型参数。在实现Softmax回归时，将用一个的矩阵来表示会很方便，该矩阵是将按行罗列起来得到的,如下所示：

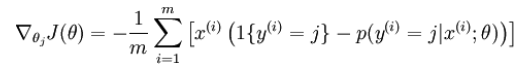


softmax回归的损失函数（Loss Function）为：



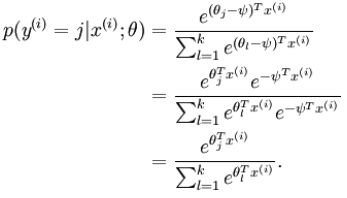
（其中是示性函数,其取值规律为：1{条件为真}=1，1{条件为假}=0）

经过求导，我们得到梯度公式如下：



有了上面的偏导数公式以后，我们就可以将它代入到梯度下降法等算法中，来求得合适的。

Softmax 回归有一个不寻常的特点：它有一个“冗余”的参数集。为了便于阐述这一特点，假设我们从参数向量中减去了向量，这时，每一个都变成了。此时假设函数变成了以下的式子：



换句话说，从中减去完全不影响假设函数的预测结果！这表明前面的softmax 回归模型中存在冗余的参数。更正式一点来说，Softmax 模型被过度参数化了。对于任意一个用于拟合数据的假设函数，可以求出多组参数值，这些参数得到的是完全相同的假设函数。

1. **Python代码及结果**

Softmax代码实现过程中，对梯度函数的矩阵表示很困难，在经过对公式的长时间研究后才发现它的矩阵表示。

在，循环次数时得到模型的准确率为0.921.

Main函数

|  |
| --- |
| **import** numpy **as** np  **import** class\_fun **as** cf  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  # 读取MNIST数据集  file **=** 'train-images-idx3-ubyte'  images **=** cf**.**read\_images**(**file**)**  test\_file **=** 't10k-images-idx3-ubyte'  test\_images **=** cf**.**read\_images**(**test\_file**)**  file\_ **=** 'train-labels-idx1-ubyte'  labels **=** cf**.**read\_labels**(**file\_**)**  test\_file\_ **=** 't10k-labels-idx1-ubyte'  test\_labels **=** cf**.**read\_labels**(**test\_file\_**)**  # standardize操作，并在第一列加1  images **=** cf**.**standard**(**images**)**  test\_images **=** cf**.**standard**(**test\_images**)**  theta **=** cf**.**calculate**(**images**,** labels**)**  accu **=** cf**.**accurate**(**theta**,** test\_images**,** test\_labels**)**  **print(**accu**)** |

Function函数

|  |
| --- |
| **import** struct  **import** numpy **as** np  **from** scipy **import** sparse  **def** read\_images**(**file**):**  **with** open**(**file**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  fmt\_header **=** '>IIII'  offset **=** 0  **(**magic**,** imgNum**,** rows**,** cols**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,** offset**)**  img\_size **=** rows**\***cols  fmt\_img **=** '>'**+**str**(**img\_size**)+**'B'  offset **+=** struct**.**calcsize**(**fmt\_header**)**  images **=** np**.**empty**([**imgNum**,** img\_size**])**  **for** i **in** range**(**imgNum**):**  img **=** struct**.**unpack\_from**(**fmt\_img**,** buf**,** offset**)**  offset **+=** img\_size  images**[**i**]** **=** np**.**array**(**img**).**reshape**(**1**,** **-**1**)**  **return** images  **def** read\_labels**(**file**):**  **with** open**(**file**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  fmt\_header **=** '>II'  offset **=** 0  **(**magic**,** labNum**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,** offset**)**  fmt\_label **=** '>'**+**str**(**labNum**)+**'B'  offset **+=** struct**.**calcsize**(**fmt\_header**)**  labels **=** np**.**array**(**struct**.**unpack\_from**(**fmt\_label**,** buf**,** offset**)).**reshape**([**labNum**,** **-**1**])**  **return** labels  **def** standard**(**images**):**  **(**rows**,** cols**)** **=** images**.**shape  mean **=** np**.**mean**(**images**,** axis**=**1**).**reshape**([**rows**,** **-**1**])**  mean **=** np**.**repeat**(**mean**,** cols**,** axis**=**1**)**  std **=** np**.**std**(**images**,** axis**=**1**).**reshape**([**rows**,** **-**1**])+**0.1  std **=** np**.**repeat**(**std**,** cols**,** axis**=**1**)**  images **=** **(**images**-**mean**)/**std  images **=** np**.**c\_**[**np**.**ones**([**rows**,** 1**]),** images**]**  **return** images  **def** calculate**(**X**,** Y**):**  K **=** 10 # 分类个数  **(**rows**,** cols**)** **=** X**.**shape # 60000, 785  Y **=** sparse**.**coo\_matrix**((**np**.**ones**(**rows**),** **(**np**.**arange**(**0**,** rows**),** Y**.**reshape**(**rows**,))),** shape**=(**rows**,** K**))**  theta **=** np**.**zeros**([**K**,** cols**])**  cycle **=** 300  alpha **=** 0.18  **for** i **in** range**(**cycle**):**  numerator **=** np**.**exp**(**X**.**dot**(**theta**.**T**))**  **(**a**,** b**)** **=** numerator**.**shape  denominator **=** np**.**repeat**(**np**.**sum**(**numerator**,** axis**=**1**).**reshape**([**a**,** 1**]),** b**,** axis**=**1**)**  P **=** numerator**/**denominator  J **=** **-**1**/**rows **\*** **(**Y**-**P**).**T**.**dot**(**X**)**  theta **=** theta **-** alpha**\***J  **return** theta  **def** accurate**(**theta**,** images**,** labels**):**  numerator **=** np**.**exp**(**images**.**dot**(**theta**.**T**))**  **(**a**,** b**)** **=** numerator**.**shape  denominator **=** np**.**repeat**(**np**.**sum**(**numerator**,** axis**=**1**).**reshape**([**a**,** 1**]),** b**,** axis**=**1**)**  P **=** numerator **/** denominator  Y **=** np**.**argmax**(**P**,** axis**=**1**).**reshape**([**P**.**shape**[**0**],** 1**])**  accu **=** sum**(**Y **==** labels**)/**len**(**labels**)**  **return** accu |