**逻辑斯谛回归实验**

曹可想 2016100104008

1. **逻辑斯谛回归**

在处理分类问题，线性回归因为不能叫一个定性的响应变量自然的转化为两水平以上的定量变量来建立线性回归模型，所以不适合用来处理分类问题。在处理简单二分类问题时常用的方法是逻辑斯谛回归。

在逻辑斯谛回归中，用0和1表示两个类别，需要训练的模型如下：

模型的代价函数（Loss Function）如下：

最小化这个函数意味着当时，要使尽可能大；当时，要使尽可能大，即尽可能小。

的梯度如下：

最后使用梯度下降法就可以得到可接受的。

1. **Python代码及实验结果**

在写代码的过程中最难的一步是对MNIST数据库二进制文件的读取，之后的部分都比较简单。

当，迭代次数时，预测准确率为：0.99669031。

Main函数

|  |
| --- |
| **import** numpy **as** np  **from** binary\_read **import** **\***  **from** sklearn**.**model\_selection **import** train\_test\_split  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  # 读取数据  images **=** Read**.**read\_img**()**  labels **=** Read**.**read\_lab**()**  test\_images **=** Read**.**read\_test\_image**()**  test\_labels **=** Read**.**read\_test\_label**()**  data\_cls **=** Standard**(**images**,** labels**)**  data **=** data\_cls**.**standard**()**  test\_data\_cls **=** Standard**(**test\_images**,** test\_labels**)**  test\_data **=** test\_data\_cls**.**standard**()**  #calculate  cal **=** Calculate**(**data**)**  theta **=** cal**.**cal\_theta**()**  # accurate rate  cls\_accurate **=** Accurate**(**test\_data**,** theta**)**  accurate **=** cls\_accurate**.**cal\_acc**()**  **print(**accurate**)** |

Function函数

|  |
| --- |
| **import** numpy **as** np  **import** struct  **class** **Read:**  @staticmethod  **def** read\_img**():**  **with** open**(**'train-images-idx3-ubyte'**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  offset **=** 0 # 读取位置偏移数，以byte为单位  fmt\_header **=** '>IIII'  **(**magic**,** imageNum**,** rows**,** cols**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,** offset**)**  # '>IIII'是说使用大端法读取4个unsinged int32，一个unsigned int32是4 bytes，在二进制文件中占8个十六进制数。  image\_size **=** rows**\***cols  offset **+=** struct**.**calcsize**(**fmt\_header**)**  fmt\_image **=** '>'**+**str**(**image\_size**)+**'B' # '>784B'的意思就是用大端法读取784个unsigned byte  images **=** np**.**empty**((**imageNum**,** image\_size**))** # 二维数组每行表示一张图片  **for** i **in** range**(**imageNum**):**  img **=** struct**.**unpack\_from**(**fmt\_image**,** buf**,** offset**)**  offset **+=** image\_size # 也可以用 offset += struct.calcsize(fmt\_header)  images**[**i**]** **=** np**.**array**(**img**).**reshape**(**1**,** **-**1**)**  **return** images  @staticmethod  **def** read\_lab**():**  **with** open**(**'train-labels-idx1-ubyte'**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  offset **=** 0 # 读取位置偏移数，以byte为单位  fmt\_header **=** '>II'  **(**magic**,** labelNum**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,** offset**)**  offset **+=** struct**.**calcsize**(**fmt\_header**)**  fmt\_label **=** '>'**+**str**(**labelNum**)+**'B'  labels **=** struct**.**unpack\_from**(**fmt\_label**,** buf**,** offset**)**  labels **=** np**.**array**(**labels**).**reshape**([**labelNum**,** 1**])**  **return** labels  @staticmethod  **def** read\_test\_image**():**  **with** open**(**'t10k-images-idx3-ubyte'**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  offset **=** 0  fmt\_header **=** '>IIII'  **(**magic**,** imageNum**,** rows**,** cols**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,** offset**)**  offset **+=** struct**.**calcsize**(**fmt\_header**)**  image\_size **=** rows**\***cols  test\_images **=** np**.**empty**([**imageNum**,** image\_size**])**  fmt\_image **=** '>'**+**str**(**image\_size**)+**'B'  **for** i **in** range**(**imageNum**):**  test\_images**[**i**]** **=** np**.**array**(**struct**.**unpack\_from**(**fmt\_image**,** buf**,** offset**)).**reshape**([**1**,-**1**])**  offset **+=** struct**.**calcsize**(**fmt\_image**)**  **return** test\_images  @staticmethod  **def** read\_test\_label**():**  **with** open**(**"t10k-labels-idx1-ubyte"**,** 'rb'**)** **as** f**:**  buf **=** f**.**read**()**  fmt\_header **=** '>II'  offset **=** 0  **(**magic**,** labelNum**)** **=** struct**.**unpack\_from**(**fmt\_header**,** buf**,**offset**)**  offset **+=** struct**.**calcsize**(**fmt\_header**)**  fmt\_label **=** '>' **+** str**(**labelNum**)** **+** 'B'  test\_label **=** np**.**array**(**struct**.**unpack\_from**(**fmt\_label**,** buf**,** offset**)).**reshape**([**labelNum**,** **-**1**])**  **return** test\_label  **class** **Standard:**  **def** \_\_init\_\_**(**self**,** images**,** labels**):**  self**.**images **=** images  self**.**labels **=** labels  **def** standard**(**self**):**  comb **=** np**.**c\_**[**self**.**images**,** self**.**labels**]**  index **=** np**.**where**((**comb**[:,** **-**1**]** **==** 0**)** **|** **(**comb**[:,** **-**1**]** **==** 1**))**  images **=** self**.**images**[**index**]**  **(**rows**,** cols**)** **=** images**.**shape  images\_mean **=** np**.**mean**(**images**,** axis**=**1**).**reshape**([**rows**,** 1**])**  images\_std **=** np**.**std**(**images**,** axis**=**1**).**reshape**([**rows**,** 1**])**  images\_mean **=** np**.**repeat**(**images\_mean**,** cols**,** axis**=**1**)**  images\_std **=** np**.**repeat**(**images\_std**+**0.1**,** cols**,** axis**=**1**)**  images **=** **(**images**-**images\_mean**)/**images\_std  data **=** np**.**c\_**[**np**.**ones**([**rows**,** 1**]),** images**,** self**.**labels**[**index**]]**  **return** data  **class** **Calculate:**  **def** \_\_init\_\_**(**self**,** data**):**  self**.**data **=** data  **def** cal\_theta**(**self**):**  X **=** self**.**data**[:,** 0**:-**1**]**  **(**rows**,** cols**)** **=** X**.**shape  Y **=** self**.**data**[:,** **-**1**].**reshape**([**rows**,** 1**])**  theta **=** np**.**repeat**(**0**,** cols**,** axis**=**0**).**reshape**([**cols**,** 1**])**  n **=** 60  alpha **=** 0.000001  **for** i **in** range**(**n**):**  segma **=** 1**/(**1**+**np**.**exp**(-**X**.**dot**(**theta**)))**  theta **=** theta **-** X**.**T**.**dot**(**segma**-**Y**)\***alpha**/**rows  **return** theta  **class** **Accurate:**  **def** \_\_init\_\_**(**self**,** test\_data**,** theta**):**  self**.**data **=** test\_data  self**.**theta **=** theta  **def** cal\_acc**(**self**):**  X **=** self**.**data**[:,** 0**:-**1**]**  Y **=** self**.**data**[:,** **-**1**].**reshape**([**X**.**shape**[**0**],** 1**])**  segma **=** 1**/(**1**+**np**.**exp**(-**X**.**dot**(**self**.**theta**)))**  pre\_Y **=** **(**segma **>=** 0.5**)**  accurate **=** sum**(**Y **==** pre\_Y**)/**len**(**Y**)**  **return** accurate |