## 项目报告

**机器学习课程项目：中文手写数字识别**

|  |  |  |
| --- | --- | --- |
| **队长:** | 钱克洋 | 201700800439 |
| **成员:** | 徐晏辰 | 201712345678 |
|  | 赵旭东 | 201912345678 |

**2020 年 12 月**

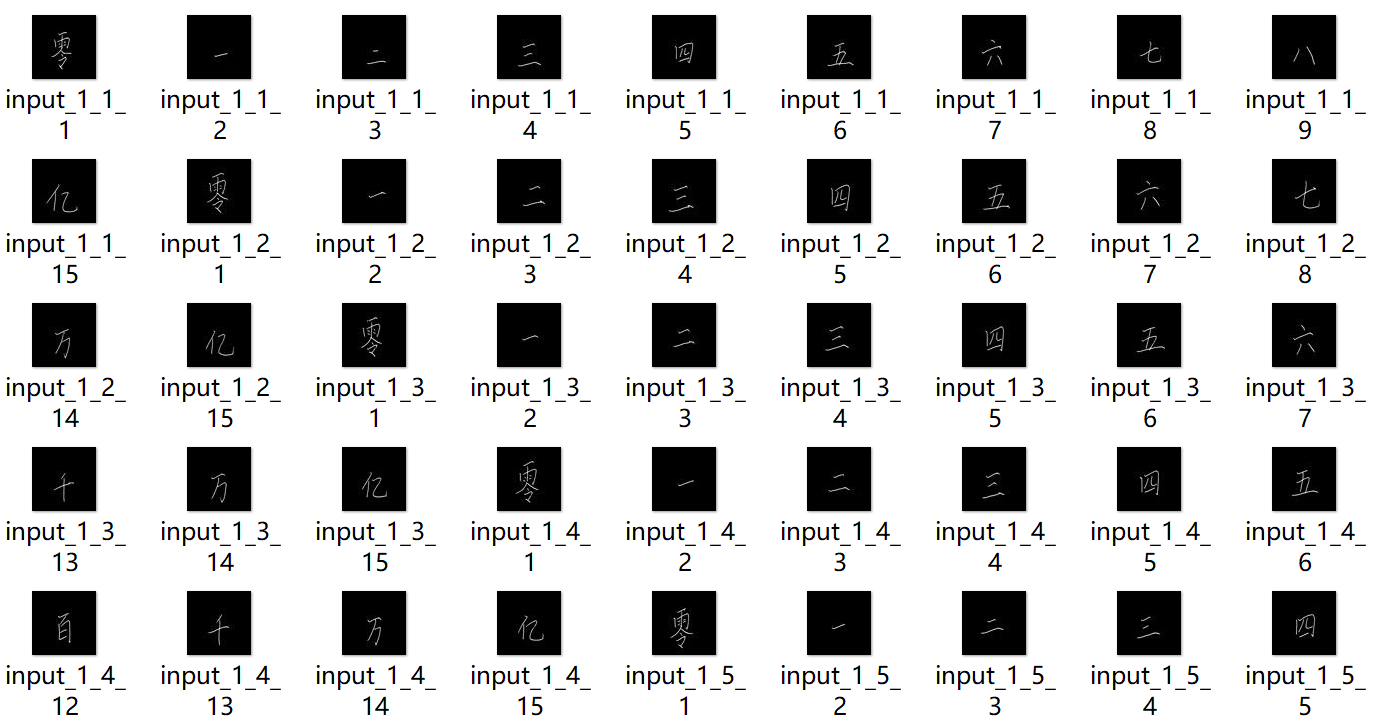
## 摘 要

我们将介绍通过特征降维和多分类正则化逻辑回归的方法实现中文手写数字识别的项目。我们使用pca和t-sne算法完成特征降维，并尝试使用尽可能小的数据集。我们对不同算法得到的、不同规模训练集分别建立3个正则化逻辑回归模型进行中文手写数字识别，以及尝试其他多种优化方法，并分层输出其识别率与混淆矩阵进行性能比较，从中获得某些经验与结论。

## 1 介绍



手写数字识别具有广泛的应用场景，其中以MNIST数据集最为经典，[MNIST](https://link.jianshu.com/?t=http://yann.lecun.com/exdb/mnist/)是一组经过预处理的手写数字图片数据集，它给很多机器学习的初学者提供了一个练手的机会，可以在真实的数据上用学到的算法来解决问题。由于很多的机器学习教程都以MNIST作为入门项目，因此它也被称作是机器学习领域的“hello world”。



但这次的项目为中文手写数字识别，在识别复杂度上高于阿拉伯数字的MNIST数据集，同时训练集中的图片也不像MNIST数据集中的图片是经过预处理的，很多图片数字并不完全在图片中心上，而且图片大小为64\*64，维度更高，在难度方面远大于MNIST数据集。这也对我们提出了更高的要求，我们需要对图片进行预处理。

我们使用的生产工具是JupyterLab与Jupyter Notebook。将Jupyter Notebook路径转到文件夹’ 房价预测\house-prices-advanced-regression-techniques’（即代码文件所在位置）下即可运行.ipynb代码文件。

### 1.1 主要问题

1.采用什么样的方法对图片进行预处理

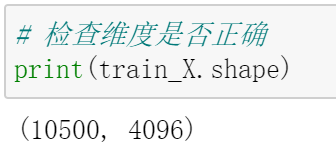
2.训练集与测试集的划分

3.模型的选择

4.后续其他优化的实现

### 1.2 解决方案

### 1.采用什么样的方法对图片进行预处理



图片维度过高64\*64=4096，势必会导致后面模型运行速度极慢，调参等操作会极不方便，于是我们考虑采用课上老师曾经教授过的pca和tsne方法对图片采取降维操作，并且到最后对两种方法的效果进行比较。

2.训练集与测试集的划分

在测试集和训练集的划分上，我们把训练集和测试集分为超大（10500,4500），大（4050,450）和小（1700，300）三种情况，来依次验证降维算法与模型的性能。

3.模型的选择

我们选择多分类正则化逻辑回归作为中文手写数字识别的模型算法。其中原因包括：

①关于中文手写数字识别的可用文档资料不多，但在MNIST手写数字识别数据集上，较大样本下多分类逻辑回归算法的准确率超过90%。

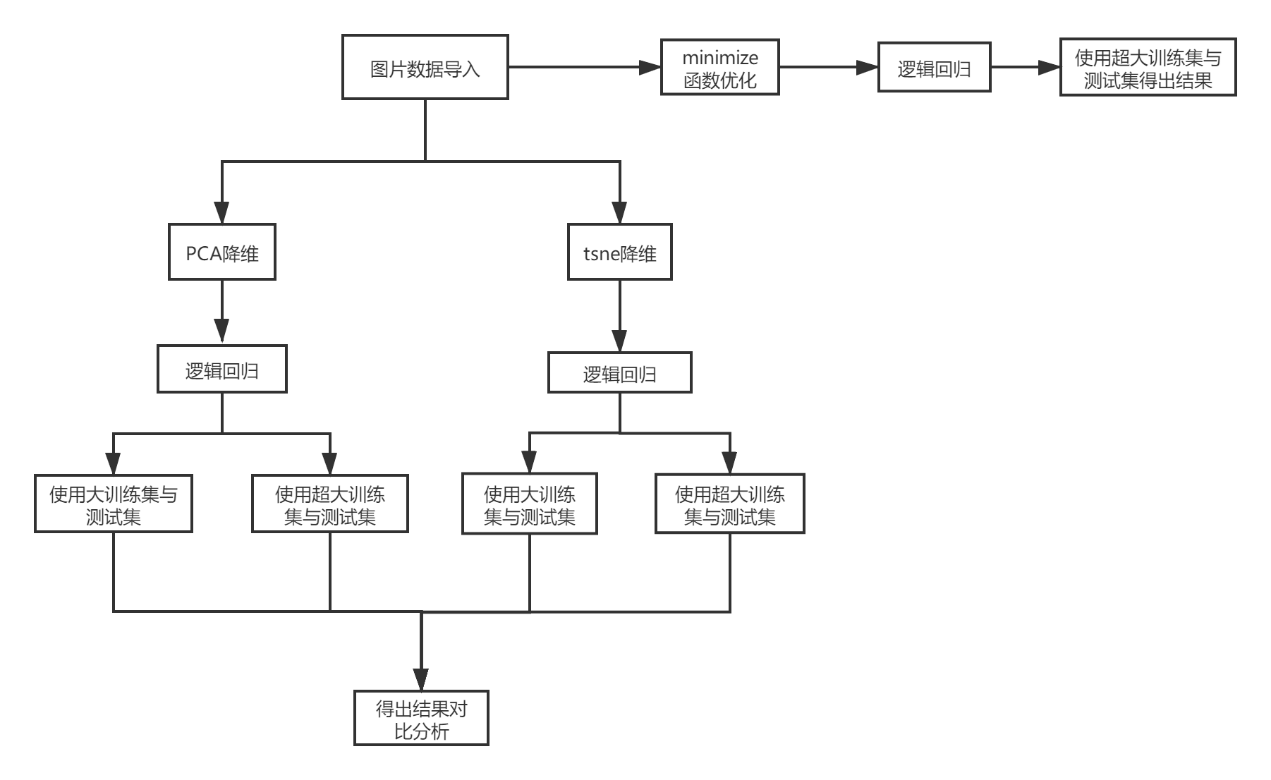
②我们不是很追求性能，希望通过比较弱的方法使不同特征提取算法的结果以及数据集规模对结果的影响更显著。逻辑回归算法实现简单，调参较方便。

4.后续其他优化的实现

尝试采用scipy.optimize中的minimize方法[3]对图片预处理。

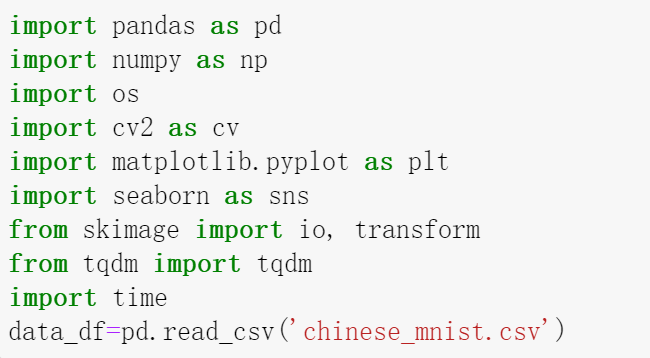
## 2 方法

我们先导入图片数据后，分别采用pca,tsne,minimize方法对图片进行预处理后，分割不同大小的训练集与测试集，采用逻辑回归的模型对测试集进行预测，最后输出不同数字的识别率与混淆矩阵，最后对不同的方法进行性能分析。

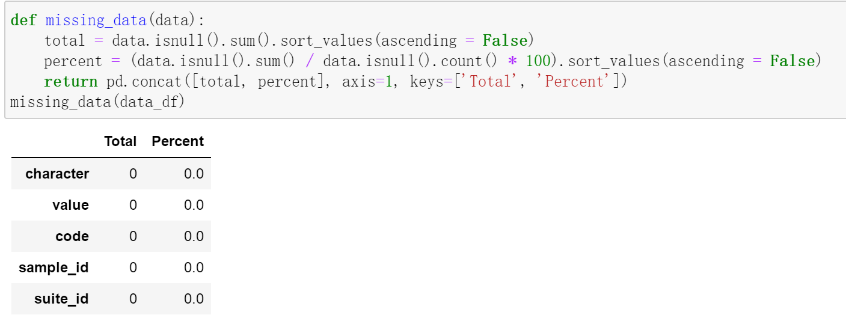


### 2.1 数据导入

导入库与csv文件

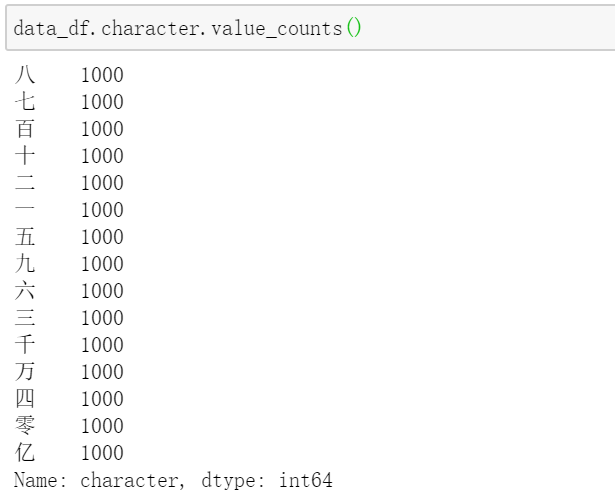


检查数据是否有缺漏，由图可以没有缺漏。



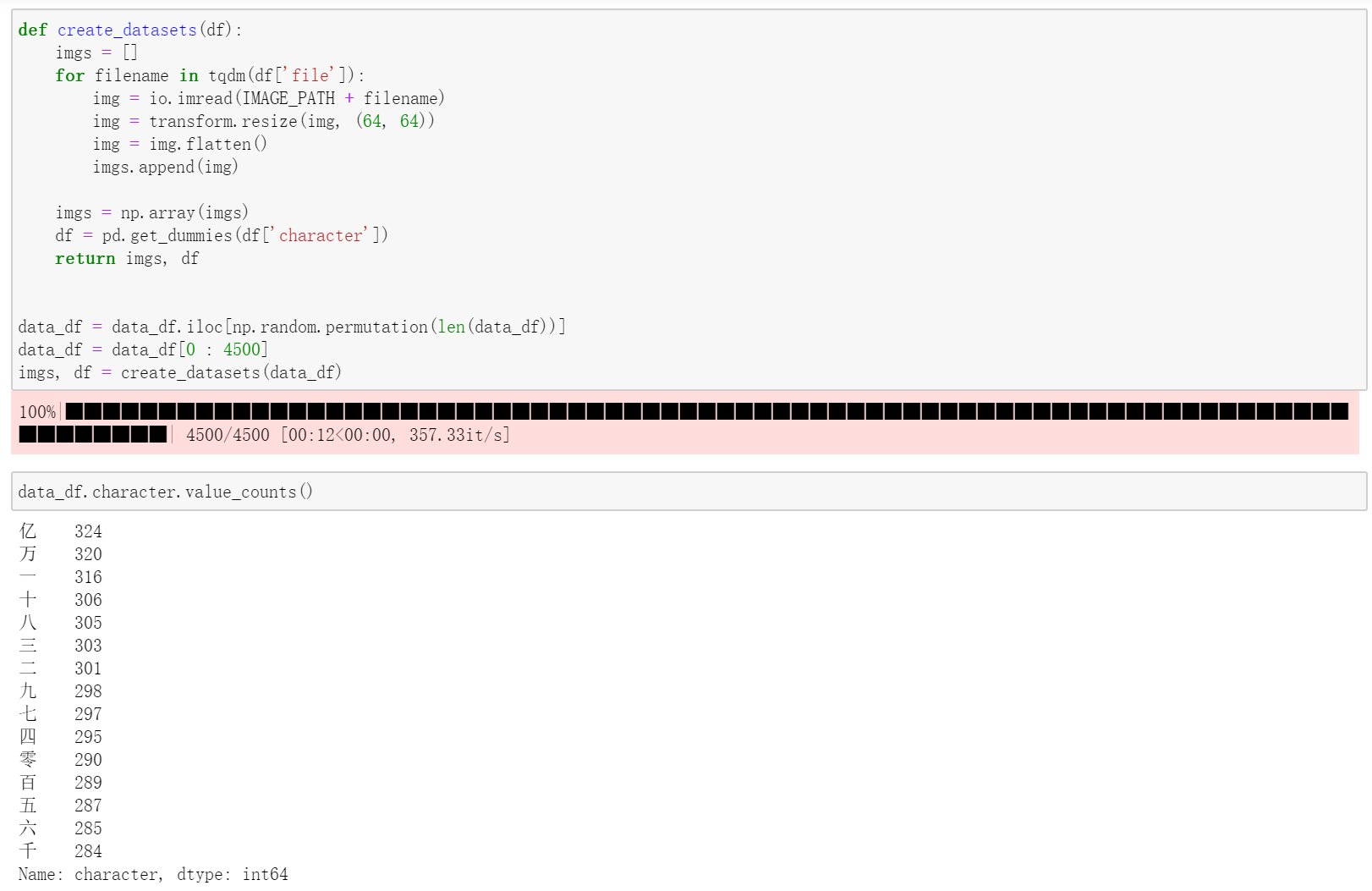
导入图片数据并与csv文件相对应，获得相应图片的对应的中文汉字。

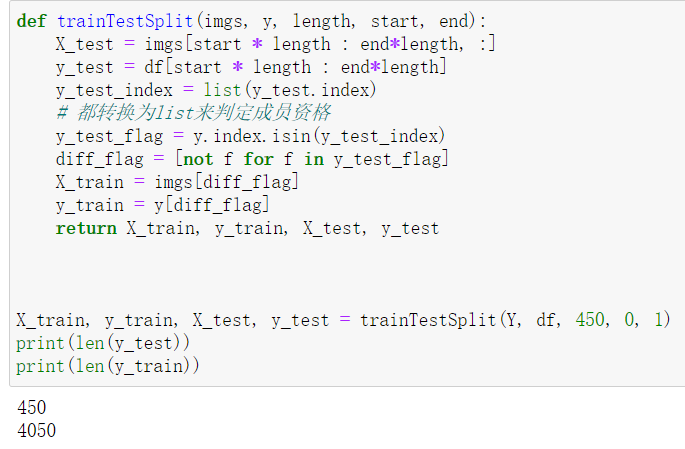


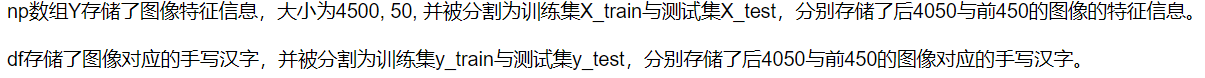


### 2.2 数据集划分

因为tsne执行时间过长，我们随机选取了4500个图片全集，为了便于与tsne算法比较，pca算法也采用了同样的4500个图片全集。



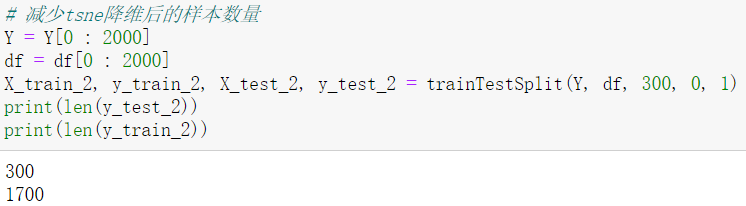




考虑到t-sne降维后得到的特征比pca可以少得多，我们准备了更小的t-sne降维后数据集。

其中训练集包含1700个样本（以下称“小训练集”），测试集包含300个样本。之前划分的大训练集的规模是小训练集的约2.4倍。

除此以外，我们还准备了未降维的全特征数据集，训练集包含10500个样本（以下称“超大训练集”），测试集包含4500个样本，特征数为4096。



### 2.3 PCA降维

先采用pca降维保留95%的特征



可见pca降维后维度大大降低为637维



### 2.4 tsne算法进一步降维

def cal\_pairwise\_dist(x):

'''计算pairwise 距离, x是matrix

(a-b)^2 = a^w + b^2 - 2\*a\*b

'''

sum\_x = np.sum(np.square(x), 1)

dist = np.add(np.add(-2 \* np.dot(x, x.T), sum\_x).T, sum\_x)

return dist

def cal\_perplexity(dist, idx = 0, beta = 1.0):

'''计算perplexity, D是距离向量，

idx指dist中自己与自己距离的位置，beta是高斯分布参数

这里的perp仅计算了熵，方便计算

'''

prob = np.exp(-dist \* beta)

# 设置自身prob为0

prob[idx] = 0

sum\_prob = np.sum(prob)

perp = np.log(sum\_prob + 1e-12) + beta \* np.sum(dist \* prob) / (sum\_prob + 1e-12)

prob /= (sum\_prob + 1e-12)

return perp, prob

def seach\_prob(x, tol = 1e-5, perplexity = 30.0):

'''二分搜索寻找beta,并计算pairwise的prob

'''

# 初始化参数

print("Computing pairwise distances...")

(n, d) = x.shape

dist = cal\_pairwise\_dist(x)

pair\_prob = np.zeros((n, n))

beta = np.ones((n, 1))

# 取log，方便后续计算

base\_perp = np.log(perplexity + 1e-12)

for i in range(n):

if i % 500 == 0:

print("Computing pair\_prob for point %s of %s ..." %(i,n))

betamin = -np.inf

betamax = np.inf

perp, this\_prob = cal\_perplexity(dist[i], i, beta[i])

# 二分搜索,寻找最佳sigma下的prob

perp\_diff = perp - base\_perp

tries = 0

while np.abs(perp\_diff) > tol and tries < 50:

if perp\_diff > 0:

betamin = beta[i].copy()

if betamax == np.inf or betamax == -np.inf:

beta[i] = beta[i] \* 2

else:

beta[i] = (beta[i] + betamax) / 2

else:

betamax = beta[i].copy()

if betamin == np.inf or betamin == -np.inf:

beta[i] = beta[i] / 2

else:

beta[i] = (beta[i] + betamin) / 2

# 更新perb,prob值

perp, this\_prob = cal\_perplexity(dist[i], i, beta[i])

perp\_diff = perp - base\_perp

tries = tries + 1

# 记录prob值

pair\_prob[i,] = this\_prob

print("Mean value of sigma: ", np.mean(np.sqrt(1 / beta)))

return pair\_prob

def tsne(x, no\_dims = 15, perplexity = 30.0, max\_iter = 300):

'''t-sne算法

先使用PCA降维到initial\_dims维，再用t-sne降维到 no\_dims维.

'''

# Check inputs

if isinstance(no\_dims, float):

print("Error: array x should have type float.")

return -1

if round(no\_dims) != no\_dims:

print("Error: number of dimensions should be an integer.")

return -1

# 初始化参数和变量

(n, d) = x.shape

initial\_momentum = 0.5

final\_momentum = 0.8

# eta 学习速率

eta = 6000

min\_gain = 0.01

y = np.random.randn(n, no\_dims)

dy = np.zeros((n, no\_dims))

iy = np.zeros((n, no\_dims))

gains = np.ones((n, no\_dims))

# 对称化

P = seach\_prob(x, 1e-5, perplexity)

P = P + np.transpose(P)

P = P / np.sum(P)

# early exaggeration

P = P \* 4

P = np.maximum(P, 1e-12)

# Run iterations

for iter in range(max\_iter):

# Compute pairwise affinities

sum\_y = np.sum(np.square(y), 1)

num = 1 / (1 + np.add(np.add(-2 \* np.dot(y, y.T), sum\_y).T, sum\_y))

num[range(n), range(n)] = 0

Q = num / np.sum(num)

Q = np.maximum(Q, 1e-12)

# Compute gradient

PQ = P - Q

for i in range(n):

dy[i,:] = np.sum(np.tile(PQ[:,i] \* num[:,i], (no\_dims, 1)).T \* (y[i,:] - y), 0)

# Perform the update

if iter < 20:

momentum = initial\_momentum

else:

momentum = final\_momentum

gains = (gains + 0.2) \* ((dy > 0) != (iy > 0)) + (gains \* 0.8) \* ((dy > 0) == (iy > 0))

gains[gains < min\_gain] = min\_gain

iy = momentum \* iy - eta \* (gains \* dy)

y = y + iy

y = y - np.tile(np.mean(y, 0), (n, 1))

# Compute current value of cost function

if (iter + 1) % 100 == 0:

if iter > 100:

C = np.sum(P \* np.log(P / Q))

else:

C = np.sum( P / 4 \* np.log( P / 4 / Q))

print("Iteration ", (iter + 1), ": error is ", C)

# Stop lying about P-values

if iter == 100:

P = P / 4

return y

start = time.perf\_counter()

Y = tsne(Y\_pca, 100, 50)

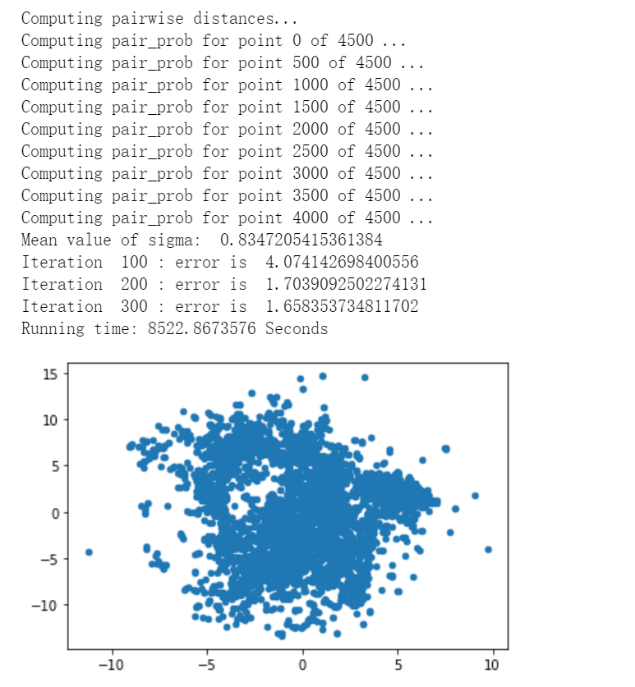
end = time.perf\_counter()

print('Running time: %s Seconds'%(end-start))

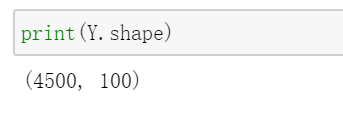
plt.scatter(Y[:,0], Y[:,1], 20)

plt.show()

**运行过程及结果可视化显示：**

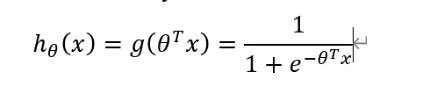


最终降为100维

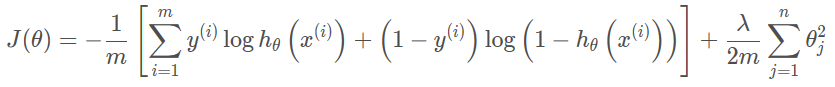


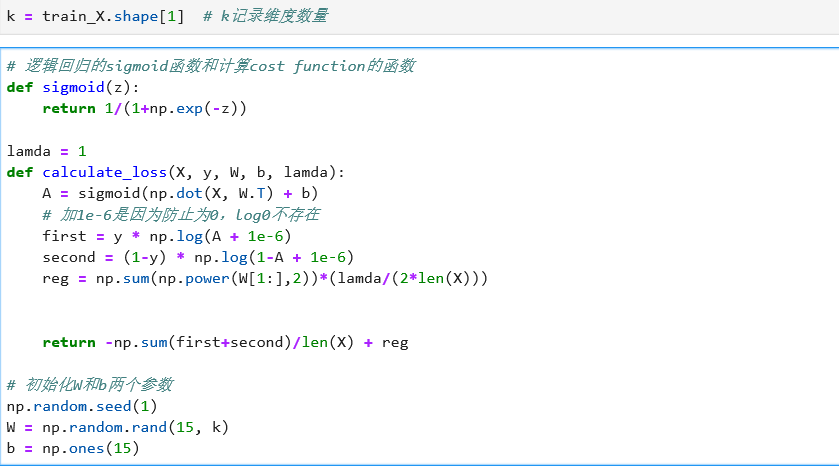
### 2.5逻辑回归模型建立

Sigmoid函数：



正则化代价函数：



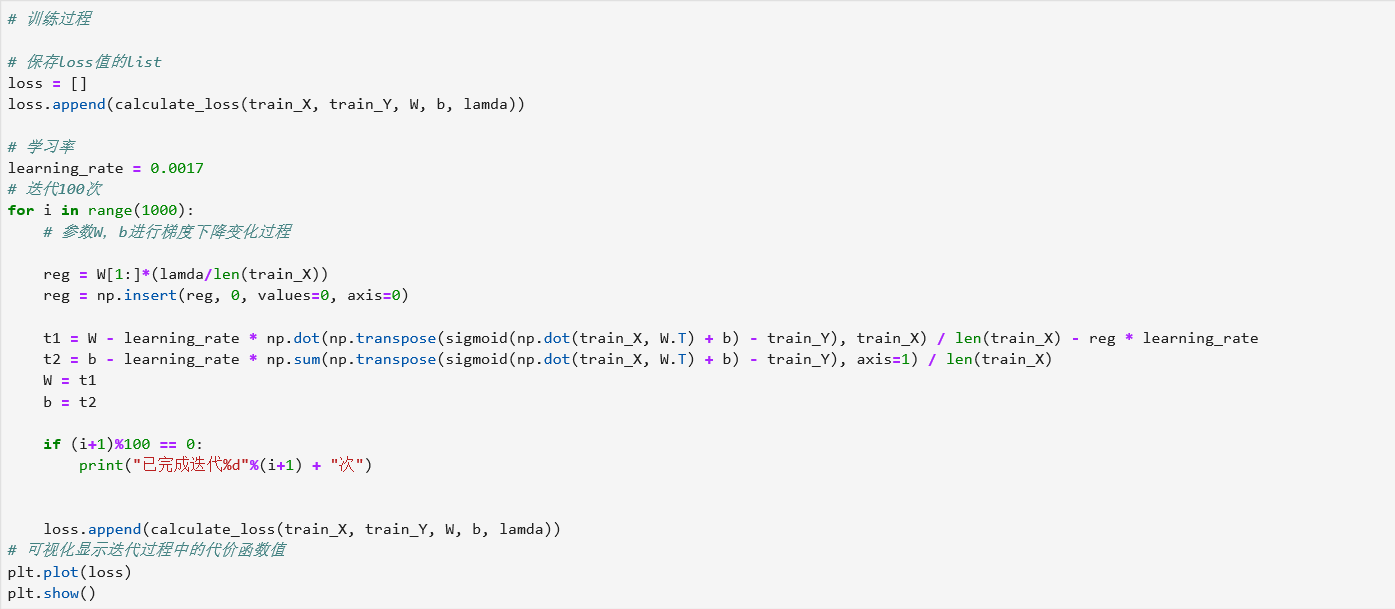


K是由上面降维处理数据所得，k为637维。

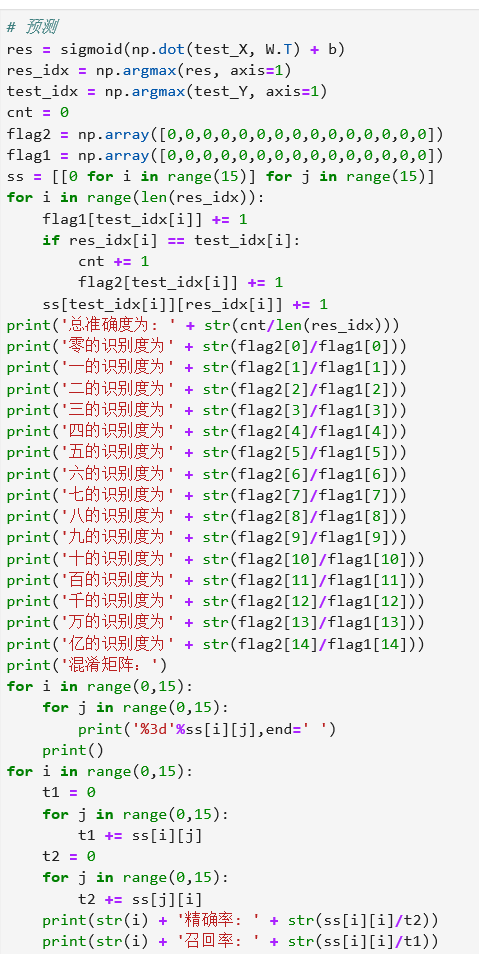
我们采用15个分类器进行手写数字识别。其中，w是逻辑回归的系数，b为常数。

(逻辑回归y=w.t\*x+b)

训练过程其主要为梯度下降算法，调参得到最优方案

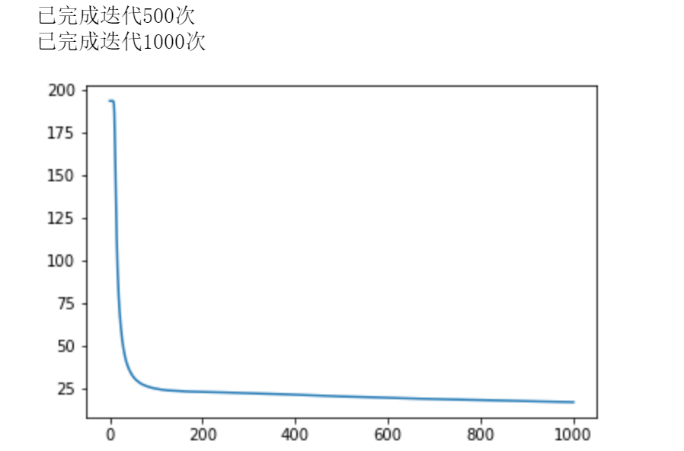


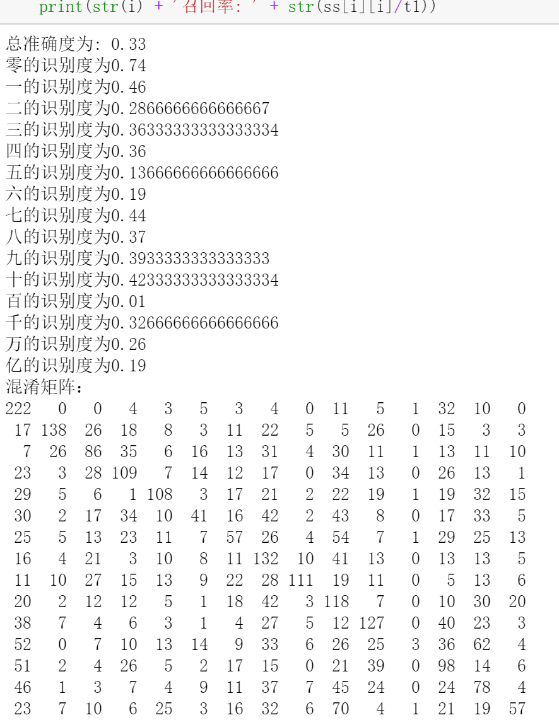
以下是通过上面的训练过程得到的最优方案进行对测试集的预测，并得到这个方案的准确率，以及每个数字的识别率。为更好的分析结果，我们输出了混淆矩阵，分析每个数据的情况，得到每个数字的精确率和召回率，由此我们可以进一步做出判断，来得到相应结论。



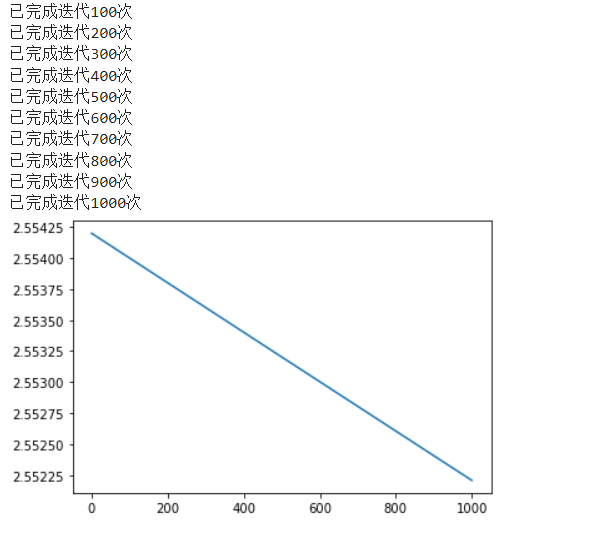
## 3 实验性能

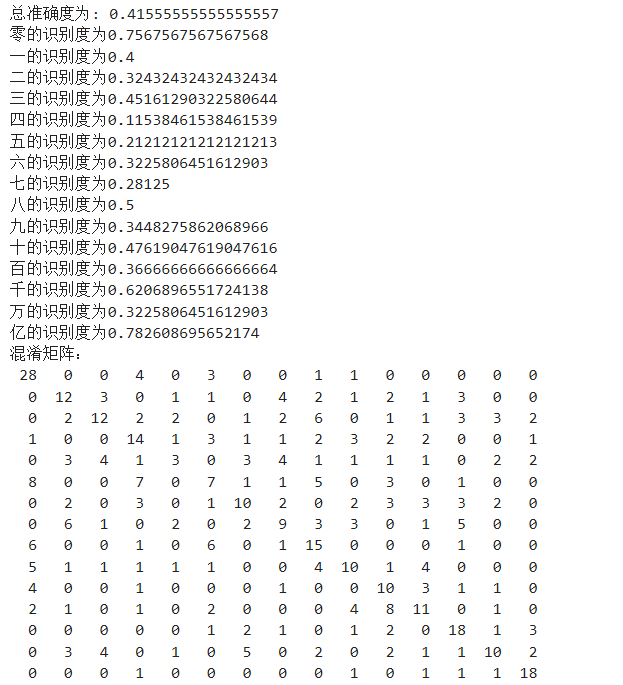
1. **未使用降维和优化得出的超大训练集的模型的迭代过程和预测结果**





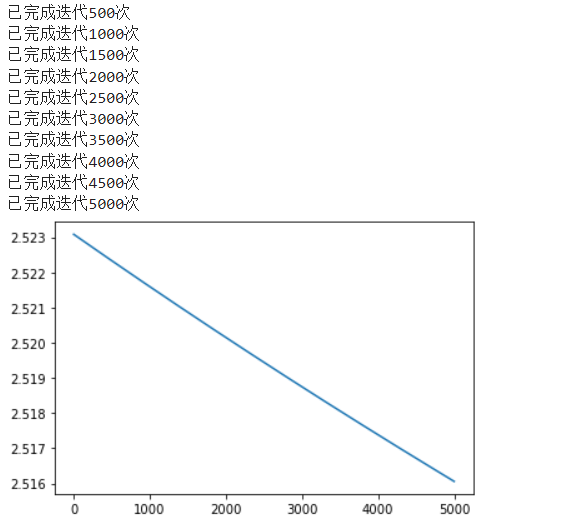
1. **使用仅pca降维得出的大训练集的模型的迭代过程和预测结果**

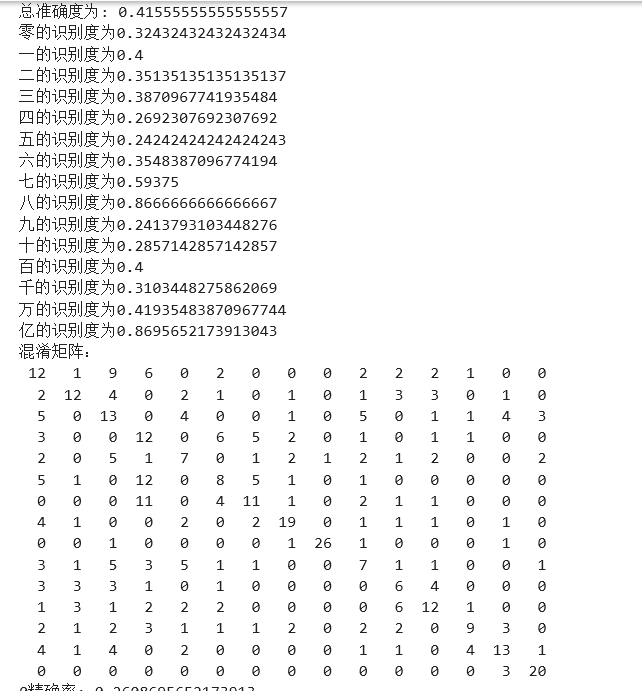




可见与使用未降维的超大训练集的模型相比，总准确度提升8%左右

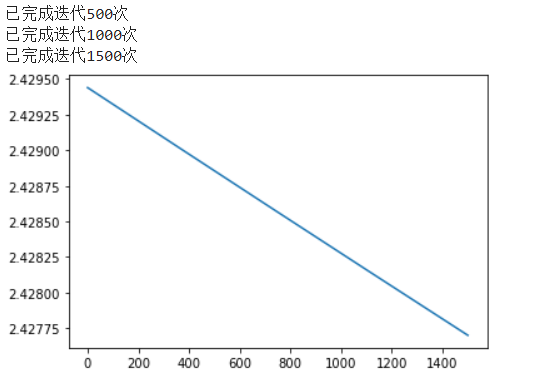
1. **用t-sne进一步降维得出的大训练集的模型的结果**

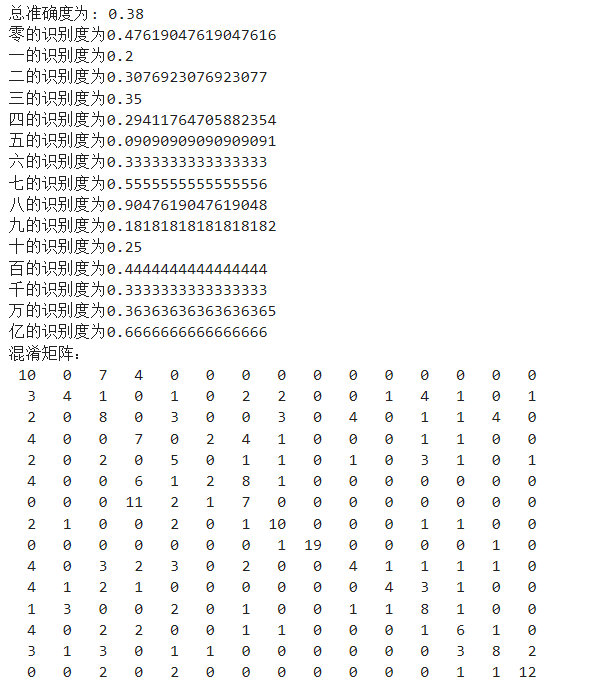
****

****

与前一模型结果在总准确度上相等，在分层识别度上不同。这一“奇迹”一定程度上是因为比较粗糙的调参工作。

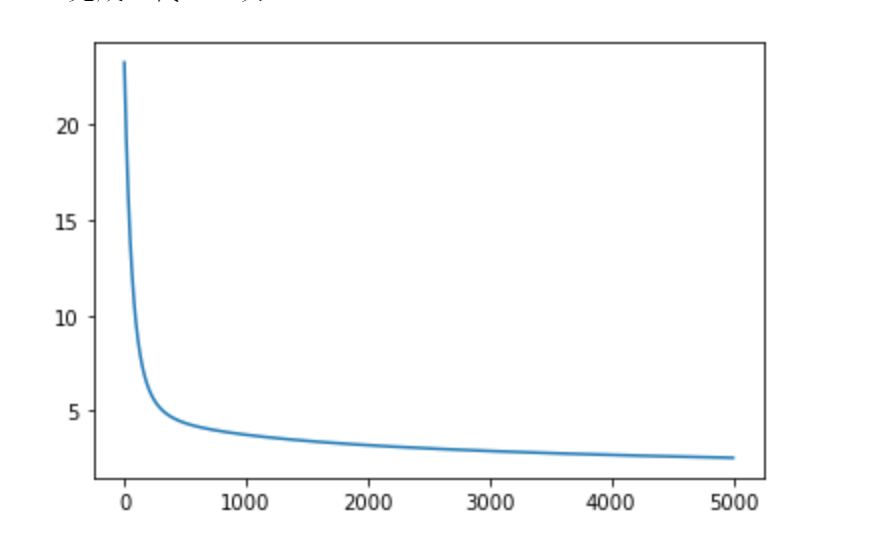
1. **用t-sne进一步降维得出的小训练集的模型的结果**

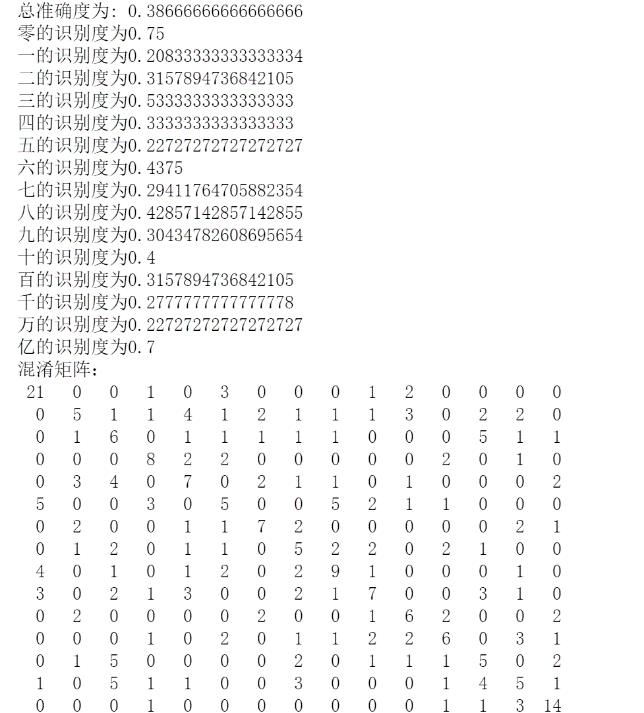




总准确度上低了3%。

1. **用pca进一步降维得出的小训练集的模型的结果**

****

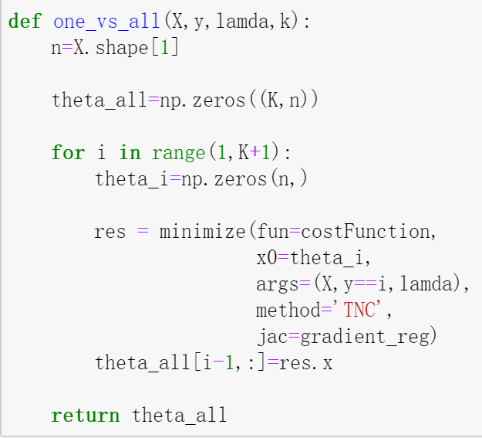


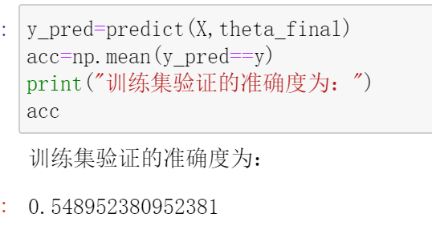
总准确度上低了2.5%。

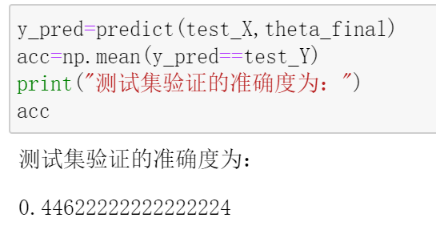
1. **其他优化方法尝试**

通过查阅资料，我们还尝试使用了scipy库中的minimize的’TNC’方法优化，该方法在阿拉伯数字手写数字识别上效果较好（94%），但在中文手写数字识别上效果大幅降低。

lamda=600000时，使用超大训练集的结果如下







### 4 讨论(结论)

一．多分类正则化逻辑回归在中文手写数字识别训练集上的效果可能远不如MNIST手写数字识别训练集，这点超出预期。但也部分和我们使用的数据集规模小、调参粗糙有关。

二．稍大数据集下t-sne降维速度比pca慢得多而且增长快。对于手写中文数字的数据集，pca降维4500个样本的数据集仅需约4分钟，t-sne（除pca初步降维的耗时外）降维1500个样本也需近十分钟，4500个样本需1~2小时视电脑性能与参数不等，6000个样本需数小时。

三．在样本规模不够大时，特征数量过多会导致显著过拟合，降维后使用更小训练集的模型性能也会更好。

四．t-sne降维后模型性能似乎没有发生改变，说明了其降维本领的高超；同时，由于特征数量显著减少，在本次实验中仅需40%的样本规模就在逻辑回归模型中跑出相近的性能。但需要进一步对比实验确认。

五．除速度外，t-sne降维最大的问题可能在于它有太多的重要超参需要调整，同时缺少确认降维结果性能的办法，给调参带来困难，而按照网上的说法， t-sne对超参的要求严苛，设置不当会导致结果很差。相较之下，pca不仅易于实现，能明确知道其保留的特征比例，而且是自动化的。这可能是t-sne算法的实际应用很少的另一重要原因。

六．主要的遗憾包括：我们没有进行更精细的降维和模型调参或进行更多的性能比较；我们没有实现效果更好的模型，如神经网络+softmax分类器；我们曾尝试直接使用t-sne算法聚类，但没有成功；等等。

最后，要感谢金老师的耐心指导，在一节节下课后耐心回答我们的问题，让我们了解要做些什么。也要感谢助教们的帮助，包括无论何时都会耐心回答问题的魏心玮学姐、提供了我们建议的希浩宇学长等。

### 5 参考文献

1. *chinese mnist exploratory data analysis,kaggle*
2. *chinese mnist classification tensorflow,kaggle*
3. [*https://www.bilibili.com/video/BV1mt411p7kG*](https://www.bilibili.com/video/BV1mt411p7kG)*吴恩达机器学习系列课后作业-03多分类、神经网络前向传播*

[4]老师提供的t-sne代码

[5]t-sne笔记

http://www.datakit.cn/blog/2017/02/05/t\_sne\_full.html#5-%E4%BB%A3%E7%A0%81