**大数据竞赛平台——Kaggle 入门篇**

这篇文章适合那些刚接触Kaggle、想尽快熟悉Kaggle并且独立完成一个竞赛项目的网友，对于已经在Kaggle上参赛过的网友来说，大可不必耗费时间阅读本文。本文分为两部分介绍Kaggle，第一部分简单介绍Kaggle，第二部分将展示解决一个竞赛项目的全过程。如有错误，请指正！

**1、Kaggle简介**

Kaggle是一个数据分析的竞赛平台，网址：<https://www.kaggle.com/>

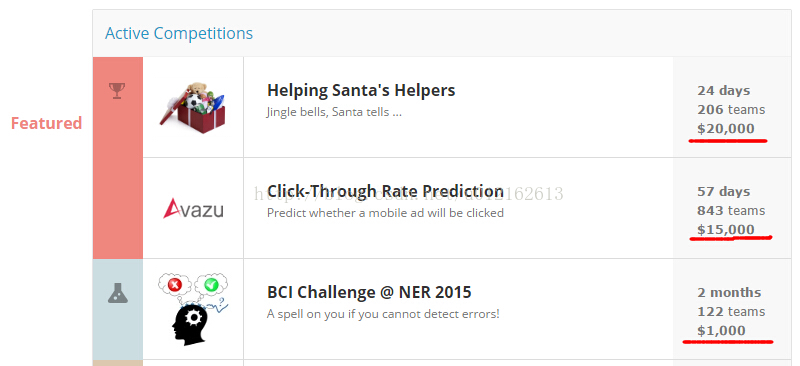
企业或者研究者可以将数据、问题描述、期望的指标发布到Kaggle上，以竞赛的形式向广大的数据科学家征集解决方

案，类似于[KDD-CUP](http://baike.baidu.com/link?url=NrfNIBhmEDicYb29WuBAAF7KuAC7prrp9oNmCbVi8tNmkgWmzt5agCGYwqZccByjyRPmj9lCBr1inZKAWmdPRs1WPk27P9LP2oATFySy0_BTX6dFTMOIyseNE6n3oboROM9o6SyDZ0lKKIs4Pvu-2L5N68-3Kt7By8vfh1y0YZxxLo2Z4qwXTGtYqGMpXtft3v-uuACcCQUNLsX50qJfX_oFv6w_K94cWRndUfJaqeW)（国际知识发现和数据挖掘竞赛）。Kaggle上的参赛者将数据下载下来，分析数据，然后运用机

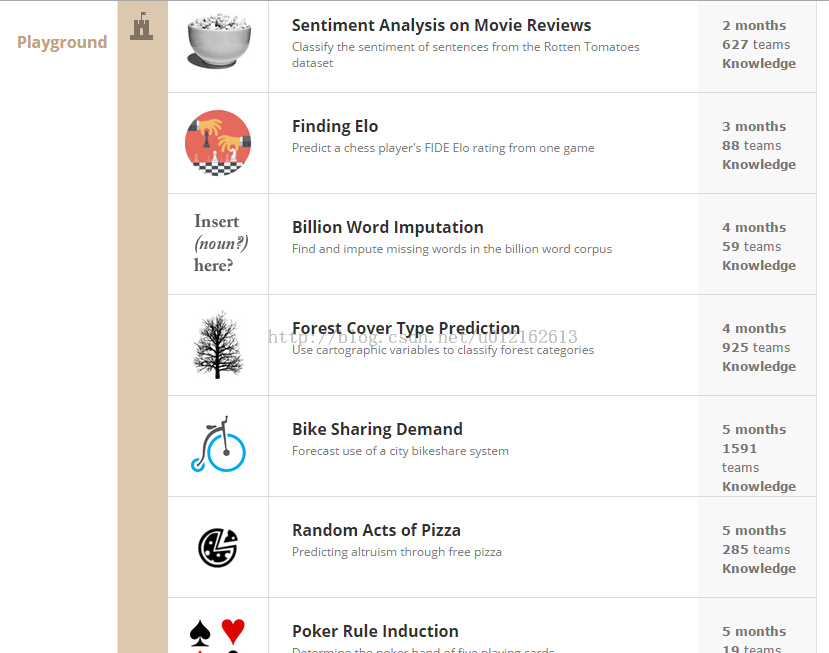
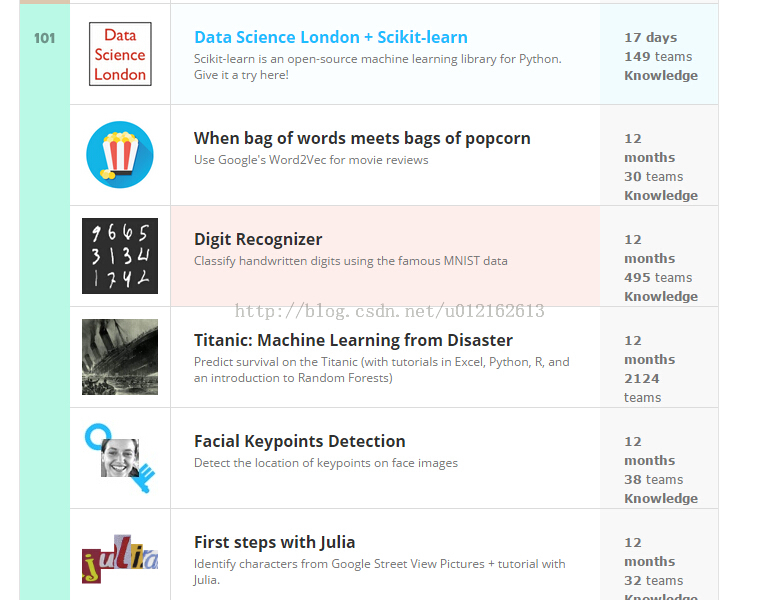
器学习、数据挖掘等知识，建立算法模型，解决问题得出结果，最后将结果提交，如果提交的结果符合指标要求并且在参赛者中排名第一，将获得比赛丰厚的奖金。更多内容可以参阅：[大数据众包平台](http://www.china-cloud.com/yunhudong/yunzhuanlan/zhuanlanrenwu/chen_/2013/0118/17375.html)

下面我以图文的形式介绍Kaggle：

**进入Kaggle网站：**

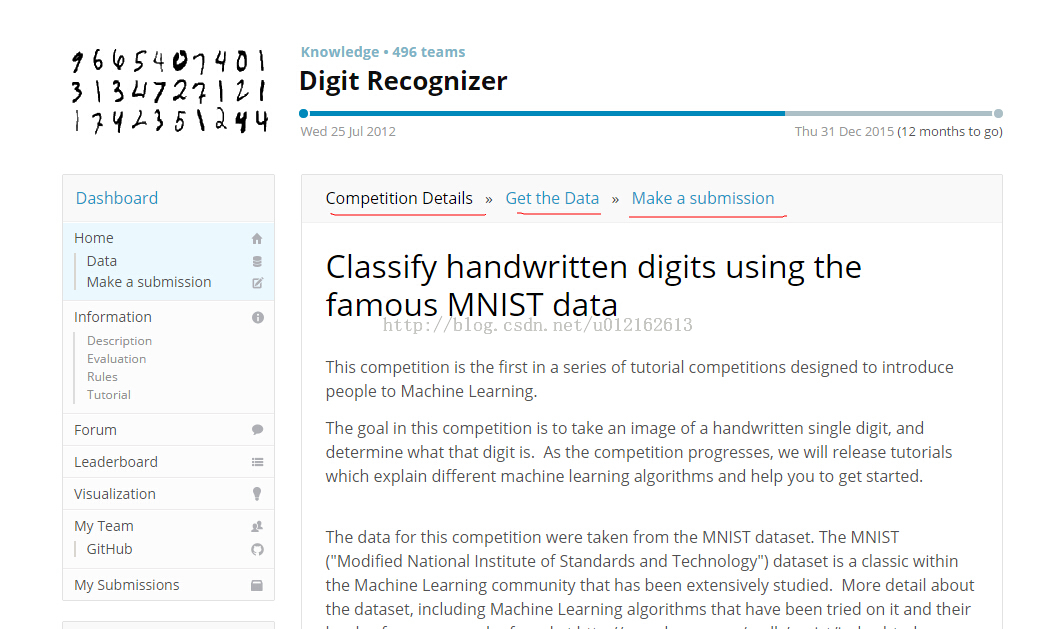


这是当前正在火热进行的有奖比赛，有冠军杯形状的是“Featured”，译为“号召”，召集数据科学高手去参赛。下面那个灰色的有试剂瓶形状的是“Research”，奖金少一点。这两个类别的比赛是有奖竞赛，难度自然不小，作为入门者，应该先做练习赛：

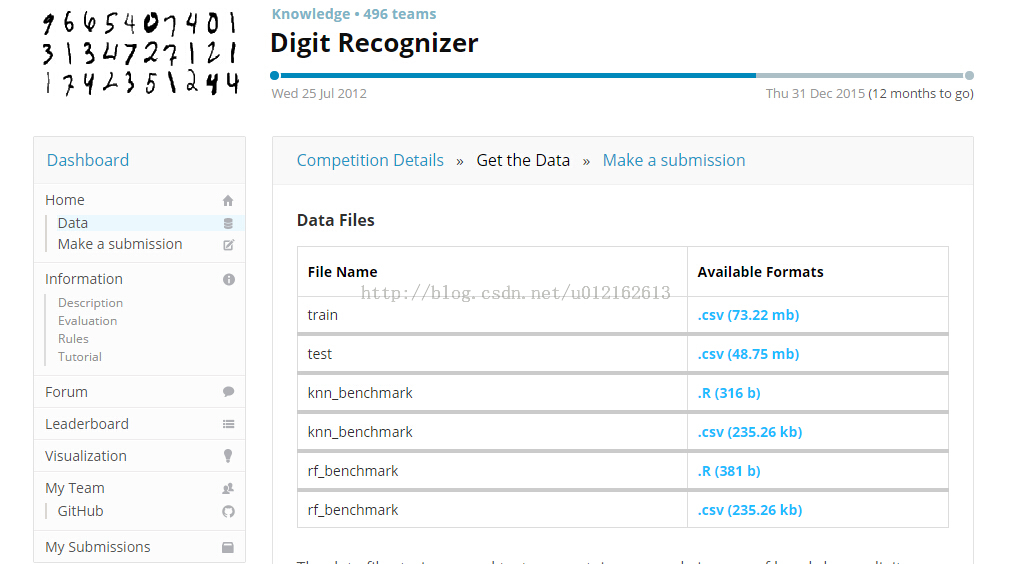


左图的比赛是“101”，右图的是“Playground”，都是练习赛，适合入门。入门Kaggle最好的方法就是独立完成101和playground这两个级别的竞赛项目。本文的第二部分将选101中的“Digit Recognition”作为讲解。

**点击进入赛题“Digit Recognition”**：

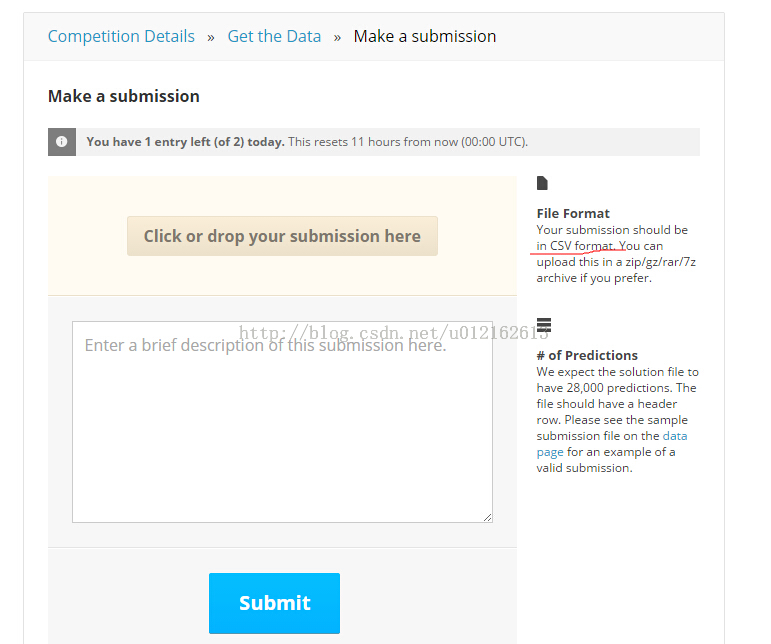


这是一个识别数字0～9的练习赛，**“Competition Details“**是这个比赛的描述，说明参赛者需要解决的问题。**”Get the Data“**是数据下载，参赛者用这些数据来训练自己的模型，得出结果，数据一般都是以csv格式给出：



其中，train.csv就是训练样本，test.csv就是测试样本，由于这个是训练赛，所以还提供了两种解决方案，knn\_benchmark.R和rf\_benchmark.R，前者是用R语。言写的knn算法程序，后者是用R语言写的随机森林算法程序，它们的结果分别是knn\_benchmark.csv和rf\_benchmark.csv。关于csv格式文件，我前一篇文章有详述：[【Python】csv模块的使用](http://blog.csdn.net/u012162613/article/details/41915859)。

得出结果后，接下来就是提交结果**”Make a submission“**：



要求提交的文件是csv格式的，假如你将结果保存在result.csv，那么点击”Click or drop submission here“，选中result.csv文件上传即可，系统将测试你提交的结果的准确率，然后排名。

另外，除了“Competition Details“、”Get the Data“、”Make a submission“，侧边栏的”Home“、”Information“、"Forum"等，也提供了关于竞赛的一些相关信息，包括排名、规则、辅导......

【以上是第一部分，暂且写这么多，有补充的以后再更】

**2、竞赛项目解题全过程**

**（1）知识准备**

首先，想解决上面的题目，还是需要一点ML算法的基础的，另外就是要会用编程语言和相应的第三方库来实现算法，常用的有：

Python以及对应的库numpy、scipy、scikit-learn（实现了ML的一些算法，可以直接用）、theano（DeepLearning的算法包）。

R语言、weka

如果用到深度学习的算法，cuda、caffe也可以用

总之，使用什么编程语言、什么平台、什么第三方库都无所谓，无论你用什么方法，Kaggle只需要你线上提交结果，线下你如何实现算法是没有限制的。

Ok，下面讲解题过程，以”Digit Recognition“为例，数字识别这个问题我之前写过两篇文章，分别用kNN算法和Logistic算法去实现，有完整的代码，有兴趣可以阅读：[kNN算法实现数字识别](http://blog.csdn.net/u012162613/article/details/41768407)、[Logistic回归实现数字识别](http://blog.csdn.net/u012162613/article/details/41844495)

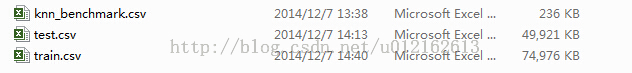
**（2）Digit Recognition解题过程**

下面我将采用kNN算法来解决Kaggle上的这道Digit Recognition训练题。上面提到，我之前用kNN算法实现过，这里我将直接copy之前的算法的核心代码，核心代码是关于kNN算法的主体实现，我不再赘述，我把重点放在**处理数据**上。

以下工程基于**Python、numpy**

* **获取数据**

从”Get the Data“下载以下三个csv文件：



* **分析train.csv数据**

train.csv是训练样本集，大小42001\*785，第一行是文字描述，所以实际的样本数据大小是42000\*785，其中第一列的每一个数字是它对应行的label，可以将第一列单独取出来，得到42000\*1的向量trainLabel，剩下的就是42000\*784的特征向量集trainData，**所以从train.csv可以获取两个矩阵trainLabel、trainData。**

下面给出代码，另外关于如何从csv文件中读取数据，参阅：[csv模块的使用](http://blog.csdn.net/u012162613/article/details/41915859)

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** loadTrainData():
2. l=[]
3. with open('train.csv') as file:
4. lines=csv.reader(file)
5. **for** line **in** lines:
6. l.append(line) #42001\*785
7. l.remove(l[0])
8. l=array(l)
9. label=l[:,0]
10. data=l[:,1:]
11. **return** nomalizing(toInt(data)),toInt(label)

这里还有两个函数需要说明一下，toInt()函数，是将字符串转换为整数，因为从csv文件读取出来的，是字符串类型的，比如‘253’，而我们接下来运算需要的是整数类型的，因此要转换，int(‘253’)=253。toInt()函数如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** toInt(array):
2. array=mat(array)
3. m,n=shape(array)
4. newArray=zeros((m,n))
5. **for** i **in** xrange(m):
6. **for** j **in** xrange(n):
7. newArray[i,j]=int(array[i,j])
8. **return** newArray

nomalizing()函数做的工作是归一化，因为train.csv里面提供的表示图像的数据是0～255的，为了简化运算，我们可以将其转化为二值图像，因此将所有非0的数字，即1～255都归一化为1。nomalizing()函数如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** nomalizing(array):
2. m,n=shape(array)
3. **for** i **in** xrange(m):
4. **for** j **in** xrange(n):
5. **if** array[i,j]!=0:
6. array[i,j]=1
7. **return** array

* **分析test.csv数据**

test.csv里的数据大小是28001\*784，第一行是文字描述，因此实际的测试数据样本是28000\*784，与train.csv不同，没有label，28000\*784即28000个测试样本，我们要做的工作就是为这28000个测试样本找出正确的label。**所以从test.csv我们可以得到测试样本集testData**，代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** loadTestData():
2. l=[]
3. with open('test.csv') as file:
4. lines=csv.reader(file)
5. **for** line **in** lines:
6. l.append(line)
7. #28001\*784
8. l.remove(l[0])
9. data=array(l)
10. **return** nomalizing(toInt(data))

* **分析knn\_benchmark.csv**

前面已经提到，由于digit recognition是训练赛，所以这个文件是官方给出的参考结果，本来可以不理这个文件的，但是我下面为了对比自己的训练结果，所以也把knn\_benchmark.csv这个文件读取出来，这个文件里的数据是28001\*2，第一行是文字说明，可以去掉，第一列表示图片序号1～28000，第二列是图片对应的数字。从knn\_benchmark.csv可以得到28000\*1的测试结果矩阵testResult，代码：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** loadTestResult():
2. l=[]
3. with open('knn\_benchmark.csv') as file:
4. lines=csv.reader(file)
5. **for** line **in** lines:
6. l.append(line)
7. #28001\*2
8. l.remove(l[0])
9. label=array(l)
10. **return** toInt(label[:,1])

到这里，数据分析和处理已经完成，我们获得的矩阵有：trainData、trainLabel、testData、testResult

* **算法设计**

这里我们采用kNN算法来分类，核心代码：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** classify(inX, dataSet, labels, k):
2. inX=mat(inX)
3. dataSet=mat(dataSet)
4. labels=mat(labels)
5. dataSetSize = dataSet.shape[0]
6. diffMat = tile(inX, (dataSetSize,1)) - dataSet
7. sqDiffMat = array(diffMat)\*\*2
8. sqDistances = sqDiffMat.sum(axis=1)
9. distances = sqDistances\*\*0.5
10. sortedDistIndicies = distances.argsort()
11. classCount={}
12. **for** i **in** range(k):
13. voteIlabel = labels[0,sortedDistIndicies[i]]
14. classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
15. sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
16. **return** sortedClassCount[0][0]

关于这个函数，参考：[kNN算法实现数字识别](http://blog.csdn.net/u012162613/article/details/41768407)

简单说明一下，inX就是输入的单个样本，是一个特征向量。dataSet是训练样本，对应上面的trainData，labels对应trainLabel，k是knn算法选定的k，一般选择0～20之间的数字。这个函数将返回inX的label，即图片inX对应的数字。

对于测试集里28000个样本，调用28000次这个函数即可。

* **保存结果**

kaggle上要求提交的文件格式是csv，上面我们得到了28000个测试样本的label，必须将其保存成csv格式文件才可以提交，关于csv，参考：[【Python】csv模块的使用](http://blog.csdn.net/u012162613/article/details/41915859)。

代码:

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** saveResult(result):
2. with open('result.csv','wb') as myFile:
3. myWriter=csv.writer(myFile)
4. **for** i **in** result:
5. tmp=[]
6. tmp.append(i)
7. myWriter.writerow(tmp)

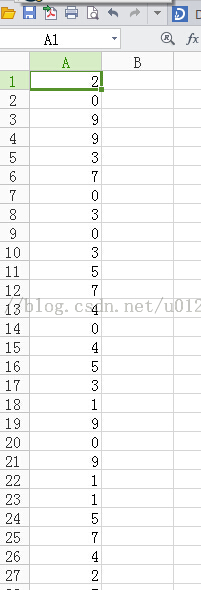
* **综合各函数**

上面各个函数已经做完了所有需要做的工作，现在需要写一个函数将它们组合起来解决digit recognition这个题目。我们写一个handwritingClassTest函数，运行这个函数，就可以得到训练结果result.csv。

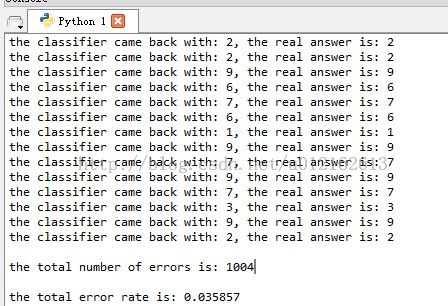
**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. **def** handwritingClassTest():
2. trainData,trainLabel=loadTrainData()
3. testData=loadTestData()
4. testLabel=loadTestResult()
5. m,n=shape(testData)
6. errorCount=0
7. resultList=[]
8. **for** i **in** range(m):
9. classifierResult = classify(testData[i], trainData, trainLabel, 5)
10. resultList.append(classifierResult)
11. **print** "the classifier came back with: %d, the real answer is: %d" % (classifierResult, testLabel[0,i])
12. **if** (classifierResult != testLabel[0,i]): errorCount += 1.0
13. **print** "\nthe total number of errors is: %d" % errorCount
14. **print** "\nthe total error rate is: %f" % (errorCount/float(m))
15. saveResult(resultList)

运行这个函数，可以得到result.csv文件：



2 0 9 9 3 7 0 3.......就是每个图片对应的数字。与参考结果knn\_benchmark.csv比较一下：



28000个样本中有1004个与kknn\_benchmark.csv中的不一样。错误率为3.5%，这个效果并不好，原因是我并未将所有训练样本都拿来训练，因为太花时间，我只取一半的训练样本来训练，即上面的结果对应的代码是：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41929171)

1. classifierResult = classify(testData[i], trainData[0:20000], trainLabel[0:20000], 5)

训练一半的样本，程序跑了将近70分钟（在个人PC上）。

* **提交结果**

将result.csv整理成kknn\_benchmark.csv那种格式，即加入第一行文字说明，加入第一列的图片序号，然后make a submission，结果准确率96.5%：



下载工程代码：[github地址](https://github.com/wepe/Kaggle-Solution/tree/master/Digit%20Recognizer)

# 利用python的theano库刷kaggle mnist排行榜

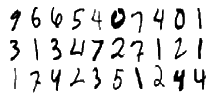
*斜体文字*==背景== theano 是一个python语言的库，实现了一些机器学习的方法，最大的特点是可以就像普通的python程序一样透明的使用GPU

theano的主页：<http://deeplearning.net/software/theano/index.html>

theano 同时也支持符号计算，并且和numpy相容，numpy是一个python的矩阵计算的库，可以让python具备matlab的计算能力，虽然没有matlab方便

numpy的主页：<http://www.numpy.org/>

MNIST是一个手写数字识别的公开数据集，我以为地球人都知道



mnist主页：<http://yann.lecun.com/exdb/mnist/>

其他大部分资源位于deeplearning向导的主页：

deeplearning.net向导: <http://deeplearning.net/tutorial/>

kaggle是一个供大家公开测试各种机器学习算法的平台，包括ICML和KDD cup一类的比赛都在上面进行，其中的入门测试集就是MNIST：

kaggle的MNIST主页：<http://www.kaggle.com/c/digit-recognizer>

目前发表的最好结果是卷积神经网络方法的0.23%错误率[[1]](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_note-mnist_homepage-1)，kaggle上被认可的最好结果是0.5%。看这个架势，mnist已经基本被大家解决了。不过本着实践出真知和学习threano用法的目的，我觉得用python的theano库对kaggle mnist刷个榜玩玩也不错。

|  |
| --- |
| 目录  [[隐藏](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn)]   * [1 数据转换与代码修改](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E6.95.B0.E6.8D.AE.E8.BD.AC.E6.8D.A2.E4.B8.8E.E4.BB.A3.E7.A0.81.E4.BF.AE.E6.94.B9)   + [1.1 输入数据修改](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E8.BE.93.E5.85.A5.E6.95.B0.E6.8D.AE.E4.BF.AE.E6.94.B9)   + [1.2 输出数据修改](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E8.BE.93.E5.87.BA.E6.95.B0.E6.8D.AE.E4.BF.AE.E6.94.B9)   + [1.3 平移数据](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E5.B9.B3.E7.A7.BB.E6.95.B0.E6.8D.AE) * [2 运行结果](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E8.BF.90.E8.A1.8C.E7.BB.93.E6.9E.9C) * [3 不明觉厉](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E4.B8.8D.E6.98.8E.E8.A7.89.E5.8E.89) * [4 改进](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E6.94.B9.E8.BF.9B) * [5 参考](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#.E5.8F.82.E8.80.83) |

## [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=1)]数据转换与代码修改

theano的代码位于:

<https://github.com/lisa-lab/DeepLearningTutorials>

我修改后的代码位于：

<https://github.com/chaosconst/DeepLearningTutorials>

### [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=2)]输入数据修改

原来是从cPickle导入：

*#############*

*# LOAD DATA #*

*#############*

*# Download the MNIST dataset if it is not present*

data\_dir, data\_file = os.path.split(dataset)

**if** (**not** os.path.isfile(dataset)) **and** data\_file == 'mnist.pkl.gz':

**import** urllib

origin = 'http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz'

**print** 'Downloading data from %s' % origin

urllib.urlretrieve(origin, dataset)

**print** '... loading data'

*# Load the dataset*

f = gzip.open(dataset, 'rb')

train\_set, valid\_set, test\_set = cPickle.load(f)

f.close()

更改为读取train.csv和test.csv，先初始化四个list。

**print** '... loading data'

train\_set=list();

valid\_set=list();

test\_set=list();

predict\_set=list();

valid\_set是用来在SGD迭代过程中，用来验证效果但不参与训练的数据集。每次只有确定在valid\_set上更有效，才继续进行目标函数的优化，这样可以防止过拟合。参见early-stopping[[2]](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_note-2)。

设定数据集的大小，如果是调试模式则减小数据集。

train\_set\_size = 36000;

valid\_set\_size = 5000;

test\_set\_size = 1000;

predict\_set\_size = 28000;

debug = "false";

**if** debug == "true":

train\_set\_size = 3600;

valid\_set\_size = 500;

test\_set\_size = 100;

predict\_set\_size = 2800;

MNIST共有7w条记录，其中6w是训练集，1w是测试集。theano的样例程序就是这么做的，但kaggle把7w的数据分成了两部分，train.csv一共42000行，test.csv一共28000行。实际可用来训练的数据只有42000行（由此估计最后的效果也会有相应的折扣）。theano把6w的训练集分为了5w的test\_set和1w的valid\_set，我在这里把42000行数据分为36000的train\_set、5000行的valid\_set和1000行的test\_set（训练时用不到）。

另外我建了一个predict\_set，用来保存准备提交给kaggle的数据。然后我进行了变量初始化并从文件读取数值，读取的时候把kaggle的int转化成了theano需要的float。

train\_set.append(numpy.ndarray(shape=(train\_set\_size,28\*28), dtype=theano.config.floatX));

train\_set.append(numpy.ndarray(shape=(train\_set\_size), dtype=int));

valid\_set.append(numpy.ndarray(shape=(valid\_set\_size,28\*28), dtype=theano.config.floatX));

valid\_set.append(numpy.ndarray(shape=(valid\_set\_size), dtype=int));

test\_set.append(numpy.ndarray(shape=(test\_set\_size,28\*28), dtype=theano.config.floatX));

test\_set.append(numpy.ndarray(shape=(test\_set\_size), dtype=int));

predict\_set.append(numpy.ndarray(shape=(predict\_set\_size,28\*28), dtype=theano.config.floatX));

predict\_set.append(numpy.ndarray(shape=(predict\_set\_size), dtype=int));

*#load data from kaggle test set*

**with** open('train.csv', 'rb') **as** csvfile:

datareader = csv.reader(csvfile, delimiter=',')

index=0;

**for** row **in** datareader:

**if** index<train\_set\_size :

train\_set[1][index] = string.atoi(row[0]);

**for** pixel\_index **in** xrange(1,28\*28+1) :

train\_set[0][index][pixel\_index-1] = string.atof(row[pixel\_index])/255;

**elif** index < train\_set\_size + valid\_set\_size :

valid\_set[1][index-train\_set\_size] = string.atoi(row[0]);

**for** pixel\_index **in** xrange(1,28\*28+1) :

valid\_set[0][index-train\_set\_size][pixel\_index-1] = string.atof(row[pixel\_index])/255;

**else** :

test\_set[1][index-train\_set\_size-valid\_set\_size] = string.atoi(row[0]);

**for** pixel\_index **in** xrange(1,28\*28+1) :

test\_set[0][index-train\_set\_size-valid\_set\_size][pixel\_index-1] = string.atof(row[pixel\_index])/255;

index+=1;

**if** index == train\_set\_size + valid\_set\_size + test\_set\_size :

**break**;

**print** '... loading predict dataset'

*#load data from kaggle test set*

**with** open('test.csv', 'rb') **as** csvfile:

datareader = csv.reader(csvfile, delimiter=',')

index=0;

**for** row **in** datareader:

**for** pixel\_index **in** xrange(0,28\*28) :

predict\_set[0][index][pixel\_index] = string.atof(row[pixel\_index])/255;

index+=1;

**if** index == predict\_set\_size:

**break**;

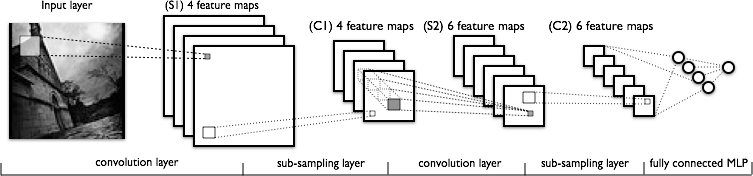
train\_set = tuple(train\_set);

valid\_set = tuple(valid\_set);

test\_set = tuple(test\_set);

predict\_set = tuple(predict\_set);

### [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=3)]输出数据修改

theano的convnet是由两个卷积层，一个hidden layer和一个logistic regression构成的，如图[[3]](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_note-3)： 

我们需要的是最后一层的输出，theano的样例程序在最后一层lr给了我们一个符号变量y\_pred，定义如下：

*# initialize with 0 the weights W as a matrix of shape (n\_in, n\_out)*

self.W = theano.shared(value=numpy.zeros((n\_in, n\_out),

dtype=theano.config.floatX),

name='W', borrow=True)

*# initialize the baises b as a vector of n\_out 0s*

self.b = theano.shared(value=numpy.zeros((n\_out,),

dtype=theano.config.floatX),

name='b', borrow=True)

*# compute vector of class-membership probabilities in symbolic form*

self.p\_y\_given\_x = T.nnet.softmax(T.dot(input, self.W) + self.b)

*# compute prediction as class whose probability is maximal in*

*# symbolic form*

self.y\_pred = T.argmax(self.p\_y\_given\_x, axis=1)

手册上说可以使用eval()对其进行实例化[[4]](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_note-4)：

predict\_results = layer3.y\_pred.eval({input:predict\_set\_x});

但是我这样做不行，只好用了很不理想的方案，原谅我

predict\_model = theano.function([index], layer3.predict(),

givens={

x: predict\_set\_x[index \* batch\_size: (index + 1) \* batch\_size]})

其中predict函数为：

**def** predict(self):

**return** T.mul(self.y\_pred,1);

我对技术不敬畏，对不起各位了。

这样我们就得到可以操作的数组，写入输出文件：

predict\_res\_array = [predict\_model(i) **for** i **in** xrange(n\_predict\_batches)]

**print** predict\_res\_array;

f = open("predict\_res","w+");

**for** y\_pred\_item\_array **in** predict\_res\_array:

**for** y\_pred\_item **in** y\_pred\_item\_array:

f.write(str(y\_pred\_item)+'**\n**');

f.close();

### [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=4)]平移数据

以上可以差不多达到1.0%的误差，和理论值0.5%还有差距，我觉得可能是因为数据不够多，所以我对输入输出数据进行了平移预处理。 输入数据平移：

*#!/bin/bash*

**awk** -F , '{

for (shiftx=-1;shiftx<=1;shiftx++) {

for (shifty=-1;shifty<=1;shifty++) {

printf $1",";

for (y=0;y<28;y++) {

for (x=1;x<=28;x++) {

x\_shift = x + shiftx;

y\_shift = y + shifty;

if ((x\_shift<1) || (x\_shift>28) || (y\_shift<0) || (y\_shift>=28)) {

printf "0,";

} else {

i=y\_shift\*28+x\_shift+1;

printf $i",";

}

}

}

printf"\n"

}}

//g' **|** **sed** 's/,$//g'

输出的时候让平移后的9个位置进行投票，boost了一把

*#!/bin/bash*

**awk** '{

dist[$0]++;

if (NR%9==0) {

best=1;

for (x in dist) {

if (dist[x]>dist[best]) best=x

}

printf best"\n"

delete dist;

}

}'

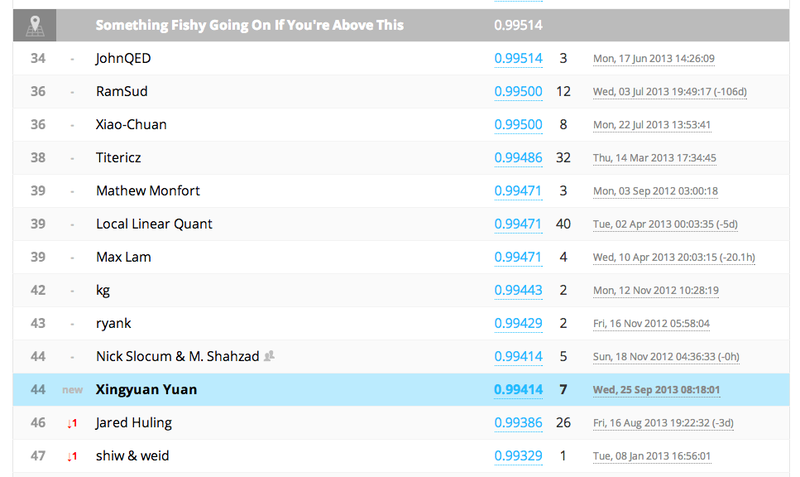
ok，万事俱备，刷榜吧！

## [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=5)]运行结果

[kaggle传送门](http://www.kaggle.com/c/digit-recognizer/leaderboard)

valid\_set\_error=0.90 test\_set\_error=0.68

[](http://wiki.swarma.net/index.php/File:Kaggle_mnist_rank1.png)

[](http://wiki.swarma.net/index.php/File:Kaggle_mnist_rank2.png)

刷到前10，我感觉可以了，再往上刷10名就要被怀疑作弊了。

## [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=6)]不明觉厉

simple cell到complex cell是怎么实现的？

1. 拿着某一斜率的filter去扫一遍全局的图像
2. 把图像分割成nxn份,做pooling（可以是max pooling）

两个二维向量卷积的意思就是扫一遍，类似于你在暗处拿着一个手电筒把一篇文章看一遍。扫的每一帧的具体操作就是相乘（找相似的特征，仅仅是相乘就可以了）。卷积不是目的，扫一遍算相似度才是。

当做polling的时候，时空信息就消失了，本来是28x28维的空间，如果4x4方块做pooling，就只剩下7x7的位置信息了。取而代之的，是feature域的信息。典型的“时空样本变换”，不过这个是98年就做出来的，实在是很赞。

学习方案是构造一个损失函数，然后用SGD求解，因为有很多层，所以损失函数的梯度计算超级复杂，参数也很多很多，不过theano有一个库，可以自动计算梯度。先进行符号计算，然后Sample一些输入数据算梯度。

嗯，大概就是这个样子吧。

## [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=7)]改进

* 肖达说：“Hinton组的cuda-convnet GPU卷积库确实快，实现同样结构的卷积神经网做MNIST手写体分类，比theano的GPU卷积快5倍多。另一个发现，用sgd优化时，max kernel norm constraint比weight decay好用”。
* cuda-convnet, <https://code.google.com/p/cuda-convnet/>
* weight decay不知道theano用了没有

## [[编辑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&action=edit&section=8)]参考

1. [↑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_ref-mnist_homepage_1-0) <http://yann.lecun.com/exdb/mnist/>, mnist homepage
2. [↑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_ref-2) <http://deeplearning.net/tutorial/gettingstarted.html#early-stopping>
3. [↑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_ref-3) <http://deeplearning.net/tutorial/lenet.html>
4. [↑](http://wiki.swarma.net/index.php?title=%E5%88%A9%E7%94%A8python%E7%9A%84theano%E5%BA%93%E5%88%B7kaggle_mnist%E6%8E%92%E8%A1%8C%E6%A6%9C&variant=zh-cn#cite_ref-4) <http://deeplearning.net/software/theano/tutorial/adding.html#adding-two-scalars>

* pylearn2的convnet，<http://nbviewer.ipython.org/urls/raw.github.com/lisa-lab/pylearn2/master/pylearn2/scripts/tutorials/convolutional_network.ipynb>
* theano是肖达告诉我的，GPU也是借用肖达的工作站的，非常感谢！
* [深度学习读书会](http://wiki.swarma.net/index.php/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0%E8%AF%BB%E4%B9%A6%E4%BC%9A)
* [lwta-theano](http://wiki.swarma.net/index.php/Lwta-theano)

# [Kaggle实战(一)](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/)

By [nanjunxiao](http://nanjunxiao.github.io/about)

 发表于 2015-07-30

**文章目录**

1. [1. 工欲善其事，必先利其器](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#工欲善其事，必先利其器)
2. [2. 机器学习流程](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#机器学习流程)
3. [3. 比赛实战](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#比赛实战)
   1. [3.1. 1. Bike Sharing Demand](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#1-_Bike_Sharing_Demand)
   2. [3.2. 2. Bag of Words Meets Bags of Popcorn](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#2-_Bag_of_Words_Meets_Bags_of_Popcorn)
   3. [3.3. 3. Titanic: Machine Learning from Disaster](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#3-_Titanic:_Machine_Learning_from_Disaster)
   4. [3.4. 4. San Francisco Crime Classification](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#4-_San_Francisco_Crime_Classification)
   5. [3.5. 5. Caterpillar Tube Pricing](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#5-_Caterpillar_Tube_Pricing)
4. [4. Join Kaggle](http://nanjunxiao.github.io/2015/07/30/Kaggle%E5%AE%9E%E6%88%98%E4%B8%80/#Join_Kaggle)

第一次接触kaggle比赛，是在听完台大林轩田老师的机器学习基石和技法课程之后。都说实践出真知，为了系统的巩固下机器学习实战技能，成为一名合格的数据挖掘工程师，我踏入了kaggle大门。

参加比赛很耗费时间和精力，由于本人已经工作，只能利用业余时间有选择的参加，希望能从中学到东西。我选择的比赛都是有监督的学习（当然用到了非监督方法，比如Bag of Words Meets Bags of Popcorn就用到了Collapsed Gibbs LDA），概括起来就是分类/排序/回归。下面就跟大家分享下近两个月我参加的比赛，欢迎一起探讨。

## 工欲善其事，必先利其器

首先介绍下我经常使用的机器学习工具:

1. **scikit-learn**. 涵盖了基本能想到的各种机器学习算法，由于本人python党，我把它当作matlab和R的替代品。
2. **xgboost**. 华盛顿大学机器学习大牛陈天奇出品的GBRT框架，果然是刷比赛利器。
3. **liblinear/libsvm**. 台大林智仁团队的佳作，工业界很多也在用。
4. **pandas**. 处理数据python包，DataFrame那叫一个好用。

## 机器学习流程

拿到一个比赛，我的一般套路是：

1. **读懂比赛介绍，明确是哪类问题**:分类/排序/回归。
2. **数据特征处理**。这个是最耗时也是最重要的，正所谓“**数据和特征决定了效果上限，模型和算法决定了逼近这个上限的程度**”。其实这点我还有很大欠缺，汗！
3. **Cross validation数据集切分**。数据集很大完全可以hold out一份作为测试集（不是待提交结果的测试集，此处是用来CV的），数据集偏小就需要K-fold或者Leave-one-out了，如果训练集有时序关系，还要注意测试集选取最后时间片的。这点我自我批评，有时为了省事，直接就提交结果做CV了。咳咳，这有点像imagenet比赛作弊了，只是我没用小号增加提交次数。
4. **常用算法/默认参数跑结果作为baseline**。这个需要一些经验和直觉，一般来说Tree Based的Random Forest和GBRT效果都不会太烂，如果特征维度很大很稀疏这时就需要试试线性SVM和LR了。
5. **接下来就是调参了**，这个我也没用太多经验，一般就是GridSearchCV或者RandomizedSearchCV。有人推荐Hyperopt库，接下来调研下。
6. **迭代**。为了取得比较好的结果，下面就是2/3/4/5不断迭代了。
7. **Blending**.上面说的都是单模型，最后让你结果更general/low variance，提升一个档次的就是结果ensemble了（不是指gbrt/rf的ensemble，是多种模型的融合）。这里我一般就是简单的多种模型结果的averaging（weighted）or voting，这里推荐一篇ensemble selection paper([http://www.cs.cornell.edu/~alexn/papers/shotgun.icml04.revised.rev2.pdf)。](http://www.cs.cornell.edu/~alexn/papers/shotgun.icml04.revised.rev2.pdf)%E3%80%82)

## 比赛实战

### 1. Bike Sharing Demand

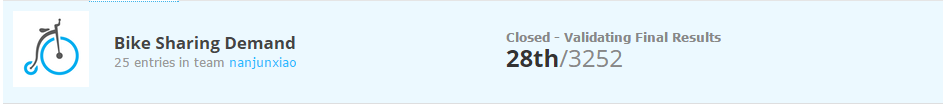
Forecast use of a city bikeshare system. <https://www.kaggle.com/c/bike-sharing-demand>

这是一个回归问题，最后预测租车数量。这里需要注意一点，最后总数实际等于casual+registered。原始共10个特征，包括datetime特征，season/holiday等类别特征，temp/atemp等数值特征，没有特征缺失值。评价指标为RMSLE，其实就是RMSE原来的p和a加1取ln。

当时正在研究GBRT，所以使用了xgboost。由于使用RMSLE，xgboost自带的loss是square loss，eval\_metric是RMSE，这时两种选择1.修改xgboost代码，派生新的优化objective，求新objective的gradient（一阶导）/hessian（二阶导），派生新的eval\_metric；2.训练数据的y做ln(y+1)转化，最后预测时再做exp(y^)-1就转回来了。当然2简单了，我也是这么实施的。

关于数据特征处理，datetime转成y/m/d/h/dayofweek，y/m等类别特征由于有连续性，这里没有做one-hot编码。经过cv最后cut掉了日/season。

Xgboost参数其实没有怎么去调，shrinkage=0.1，tree\_num=1000，depth=6，其他默认。

效果三次提升拐点分别是：1.RMSE转换为RMLSE(square loss转为square log loss)，说明预测值的范围很大，log转化后bound更tight了；2.cut了日/season特征；3.转换为对casual和registered的分别回归问题，在加和。最后RMLSE结果为**0.36512**，public LB最好为30位，最终private LB为**28**，还好说明没有overfit。  
[](http://nanjunxiao.github.io/img/Bike%20Sharing%20Demand.png)

### 2. Bag of Words Meets Bags of Popcorn

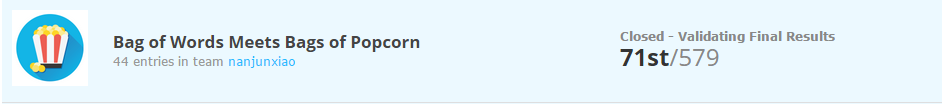
Use Google’s Word2Vec for movie reviews. <https://www.kaggle.com/c/word2vec-nlp-tutorial>

这是一个文本情感二分类问题。25000的labeled训练样本，只有一个raw text 特征”review“。评价指标为AUC，所以这里提交结果**需要用概率**，我开始就掉坑里了，结果一直上不来。

比赛里有教程如何使用word2vec进行二分类，可以作为入门学习材料。我没有使用word embeddinng，直接采用BOW及ngram作为特征训练，效果还凑合，后面其实可以融合embedding特征试试。对于raw text我采用TfidfVectorizer(stop\_words=’english’, ngram\_range=(1,3), sublinear\_tf=True, min\_df=2)，并采用卡方检验进行特征选择，经过CV，最终确定特征数为200000。

单模型我选取了GBRT/NB/LR/linear SVC。GBRT一般对于维度较大比较稀疏效果不是很好，但对于该数据表现不是很差。NB采用MultinomialNB效果也没有想象的那么惊艳。几个模型按效果排序为linear SVC(0.95601)>LR(0.94823)>GBRT(0.94173)>NB(0.93693)，看来线性SVM在文本上还是很强悍的。

后续我又采用LDA生成主题特征，本来抱着很大期望，现实还是那么骨感，采用上述单模型AUC最好也只有0.93024。既然单独使用主题特征没有提高，那和BOW融合呢？果然work了!后面试验证实特征融合还是linear SVC效果最好，LDA主题定为500，而且不去除停用词效果更好，AUC为**0.95998**。

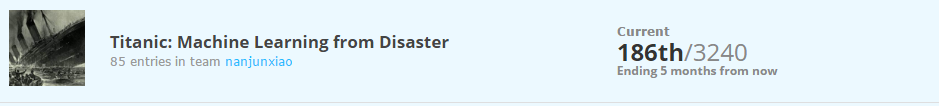
既然没有时间搞单模型了，还有最后一招，多模型融合。这里有一个原则就是模型尽量多样，不一定要求指标最好。最终我选取5组不是很差的多模型结果进行average stacking，AUC为**0.96115**，**63**位。最终private LB跌倒了**71st**，应该融合word enbedding试试，没时间细搞了。  
[](http://nanjunxiao.github.io/img/Bag%20of%20Words%20Meets%20Bags%20of%20Popcorn.png)

### 3. Titanic: Machine Learning from Disaster

Predict survival on the Titanic. <https://www.kaggle.com/c/titanic>

硬二分类问题，不需要预测概率，给出0/1即可，评价指标为accuracy。说句题外话，网上貌似有遇难者名单，LB上好几个score 1.0的。有坊间说，score超过90%就怀疑作弊了，不知真假，不过top300绝大多数都集中在0.808-0.818。这个题目我后面没有太多的改进想法了，求指导啊~

数据包括数值和类别特征，并存在缺失值。类别特征这里我做了one-hot-encode，缺失值是采用均值/中位数/众数需要根据数据来定，我的做法是根据pandas打印出列数据分布来定。

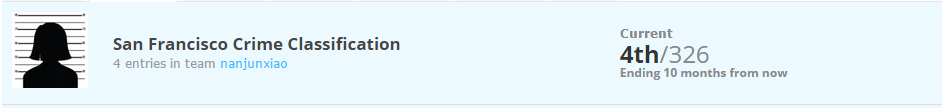
模型我采用了DT/RF/GBDT/SVC，由于xgboost输出是概率，需要指定阈值确定0/1，可能我指定不恰当，效果不好**0.78847**。效果最好的是RF，**0.81340**。这里经过筛选我使用的特征包括’Pclass’,’Gender’, ‘Cabin’,’Ticket’,’Embarked’,’Title’进行onehot编码，’Age’,’SibSp’,’Parch’,’Fare’,’class\_age’,’Family’ 归一化。我也尝试进行构建一些新特征和特征组合，比如title分割为Mr/Mrs/Miss/Master四类或者split提取第一个词，添加fare\_per\_person等，pipeline中也加入feature selection，但是效果都没有提高，求指导~  
[](http://nanjunxiao.github.io/img/Titanic.png)

### 4. San Francisco Crime Classification

Predict the category of crimes that occurred in the city by the bay. <https://www.kaggle.com/c/sf-crime>

这是一个多分类问题，一般三种处理方法：**one vs all, one vs one, softmax**，信息损失逐渐递减。87809条训练数据，数据包括datetime/类别/数值特征，没有缺失值，label共39种。评价指标为logloss，这里要说下和AUC的区别，**AUC更强调相对排序**。

我抽取后特征包括year,m,d,h,m,dow,district,address,x,y，模型选择softmax objective的LR和xgboost。这两个模型对特征挑食，有不同的偏好，LR喜好0/1类别或者locale到0-1的数值特征，而xgboost更喜好原始的数值特征，而且对缺失值也能很好的处理。所以对于LR就是2个归一化的数值特征和8个待one-hot编码的特征，对于xgboost是8个原始数值特征（包括year/m/d等，具有连续性）和2个待one-hot编码的特征。

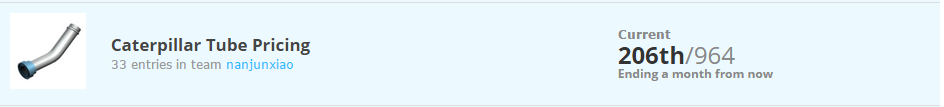
LR效果要略好于xgboost效果，logloss分别为**2.28728/2.28869**，最好位置为**3rd**，目前跌到**4th**，后面找时间再搞一搞。  
[](http://nanjunxiao.github.io/img/San%20Francisco%20Crime%20Classification.png)

### 5. Caterpillar Tube Pricing

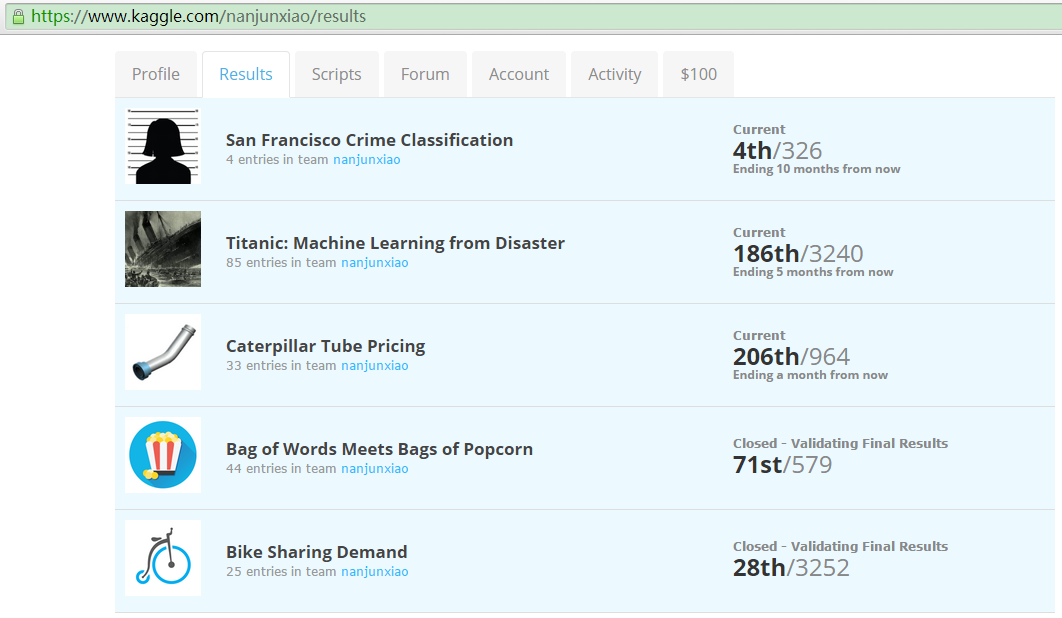
Model quoted prices for industrial tube assemblies. <https://www.kaggle.com/c/caterpillar-tube-pricing>

这也是一个回归问题，预测tube报价。30213条训练样本，特征分散在N个文件中，需要你left join起来。评价指标为RMLSE，哈哈，是不是很熟悉？对，跟bike sharing的一样，所以怎么转换知道了吧？看，做多了套路trick也就了然了。感觉这个需要领域知识，但其实有些特征我是不知道含义的，anyway，先merge所有特征不加domain特征直接搞起。

这是我见过小样本里特征处理最麻烦的(后面的CTR大数据处理更耗时)，它特征分散在多个文件中，还好我们有神器pandas，直接**left join**搞定。这里有些trick需要注意，比如comp\_\*文件要用append不能join，这样正好是一个全集，否则就会多个weight特征了。特征存在缺失值，这里我全部采用0值，不知是否恰当？

模型我主要试了RF和xgboost，RF tree\_num=1000，其他默认值，RMLSE=0.255201，主要精力放在了xgboost上，调了几次参数(depth=58,col\_sample=0.75,sample=0.85,shrinkage=0.01,tree\_num=2000)，最好**RMLSE=0.231220**，最好位置**120th**，目前跌倒**206th**了，看来需要好好搞搞特征了！  
[](http://nanjunxiao.github.io/img/Caterpillar%20Tube%20Pricing.png)

## Join Kaggle

你看我搞的还凑合吧，so，快来加入kaggle吧，求组队:-)  
[](http://nanjunxiao.github.io/img/mykaggle.png)

# [Kaggle实战(二)](http://nanjunxiao.github.io/2015/07/31/Kaggle%E5%AE%9E%E6%88%98%E4%BA%8C/)

By [nanjunxiao](http://nanjunxiao.github.io/about)

 发表于 2015-07-31

**文章目录**

1. [1. 比赛实战](http://nanjunxiao.github.io/2015/07/31/Kaggle%E5%AE%9E%E6%88%98%E4%BA%8C/#比赛实战)
   1. [1.1. 6. Display Advertising Challenge](http://nanjunxiao.github.io/2015/07/31/Kaggle%E5%AE%9E%E6%88%98%E4%BA%8C/#6-_Display_Advertising_Challenge)
   2. [1.2. 7. Avito Context Ad Clicks](http://nanjunxiao.github.io/2015/07/31/Kaggle%E5%AE%9E%E6%88%98%E4%BA%8C/#7-_Avito_Context_Ad_Clicks)
2. [2. 总结与感悟](http://nanjunxiao.github.io/2015/07/31/Kaggle%E5%AE%9E%E6%88%98%E4%BA%8C/#总结与感悟)

上一篇都是针对小数据集的，入门不建议从大数据集开始，可以不用考虑机器内存，不用out-of-core的online learning，不用考虑分布式，可以专注模型本身。

接下来我做了两个广告CTR预估相关的比赛，不过比赛当时都已经closed了，还好，我们还可以提交结果看看close时能排到的位置。

## 比赛实战

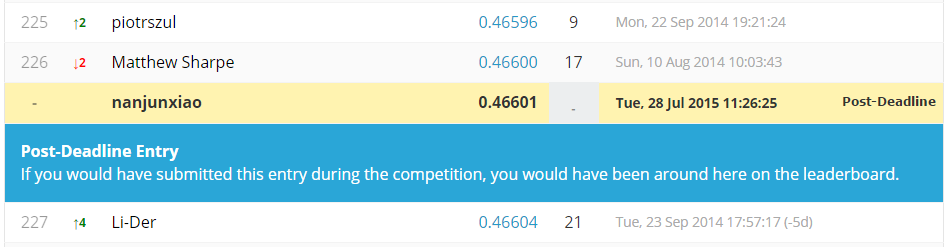
### 6. Display Advertising Challenge

Predict click-through rates on display ads. <https://www.kaggle.com/c/criteo-display-ad-challenge>

这是一个广告CTR预估的比赛，由知名广告公司Criteo赞助举办。数据包括4千万训练样本，500万测试样本，特征包括13个数值特征，26个类别特征，评价指标为logloss。

CTR工业界做法一般都是LR，只是特征会各种组合/transform，可以到上亿维。这里我也首选LR，特征缺失值我用的众数，对于26个类别特征采用one-hot编码，数值特征我用pandas画出来发现不符合正态分布，有很大偏移，就没有scale到[0,1]，采用的是根据五分位点（min,25%,中位数,75%,max）切分为6个区间(负值/过大值分别分到了1和6区间作为异常值处理)，然后一并one-hot编码，最终特征100万左右，训练文件20+G。

强调下可能遇到的坑：1.one-hot最好自己实现，除非你机器内存足够大(需全load到numpy，而且非sparse);2.LR最好用SGD或者mini-batch，而且out-of-core模式(<http://scikit-learn.org/stable/auto_examples/applications/plot_out_of_core_classification.html#example-applications-plot-out-of-core-classification-py>), 除非还是你的内存足够大;3.Think twice before code.由于数据量大，中间出错重跑的话时间成品比较高。

我发现sklearn的LR和liblinear的LR有着截然不同的表现，sklearn的L2正则化结果好于L1，liblinear的L1好于L2，我理解是他们优化方法不同导致的。最终结果liblinear的LR的L1最优，logloss=**0.46601**，LB为**227th/718**，这也正符合lasso产生sparse的直觉。  
[](http://nanjunxiao.github.io/img/Display%20Advertising%20Challenge.png)

我也单独尝试了xgboost，logloss=0.46946，可能还是和GBRT对高维度sparse特征效果不好有关。Facebook有一篇论文把GBRT输出作为transformed feature喂给下游的线性分类器，取得了不错的效果，可以参考下。（Practical Lessons from Predicting Clicks on Ads at Facebook）

我只是简单试验了LR作为baseline，后面其实还有很多搞法，可以参考forum获胜者给出的solution，比如：1. Vowpal Wabbit工具不用区分类别和数值特征；2.libFFM工具做特征交叉组合；3.feature hash trick；4.每个特征的评价点击率作为新特征加入；5.多模型ensemble等。

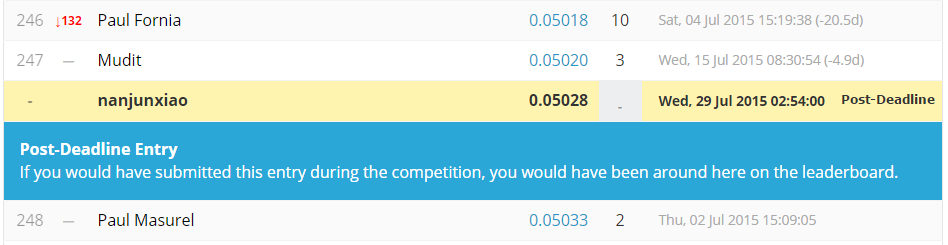
### 7. Avito Context Ad Clicks

Predict if context ads will earn a user’s click. <https://www.kaggle.com/c/avito-context-ad-clicks>

跟上一个CTR比赛不同的是，这个数据没有脱敏，特征有明确含义，userinfo/adinfo/searchinfo等特征需要和searchstream文件 join起来构成完整的训练/测试样本。数据包含392356948条训练样本，15961515条测试样本，特征基本都是id类别特征和query/title等raw text特征。评价指标还是logloss。

由于数据量太大，跑一组结果太过耗时，根据比赛6的参考，目前我只选择liblinear lasso LR做了一组结果。最终目标是预测contextual ad，为了减小数据量，\*searchstream都过滤了非contextual的，visitstream和phonerequeststream及params目前我都没有使用，但其实都是很有价值的特征（比如query和title各种similarity），后面可以尝试。

对于这种大数据，在小内存机器上sklearn和pandas处理起来已经非常吃力了，这时就需要自己定制实现left join和one-hot-encoder了，采用按行这种out-of-core方式，不过真心是慢啊。类似比赛6，price数值特征还是三分位映射成了类别特征和其他类别特征一起one-hot，最终特征大概600万左右，当然要用sparse矩阵存储了，train文件大小40G。

Libliear貌似不支持mini-batch,为了省事没办法只好找一台大内存服务器专门跑lasso LR了。由于上面过滤了不少有价值信息，也没有类似libFM或libFFM做特征交叉组合，效果不好，logloss只有0.05028，LB排名248th/414。  
[](http://nanjunxiao.github.io/img/Avito%20Context%20Ad%20Clicks.png)

对于该比赛需要好好调研下大牛们的做法，看看相关paper了，自己瞎搞跑一遍太耗时间了，加油吧~

## 总结与感悟

通过参加kaggle提高了自己的机器学习实战能力，对问题和数据有了一些感觉，大致了解了各模型的适用场景。当然还有很多需要提高，比如特征组合/transform/hash trick，模型ensemble方法等，实现的scalable(比如采用pipeline)。

Ps:一定要挑选几个适合自己的高效工具包，并对其中2-3个看过源码，最好能做到定制优化。希望大家都加入到kaggle，欢迎一起探讨提高~

# [2015阿里天池大数据竞赛-Solution](http://blog.csdn.net/u012162613/article/details/45034813)

**竞赛介绍：**[链接](http://tianchi.aliyun.com/competition/introduction.htm?spm=5176.100066.333.2.umhl4N&raceId=1)

这篇文章记录2015阿里天池大数据竞赛中，我的一些代码，由于目前还在比赛中，仅分享一个naive solution，基于规则，代码主页在我的github上：[链接](https://github.com/wepe/AliTianChi)，下面是代码说明。有兴趣的请看代码注释，恕不详述。

## 本repo目录说明

* **data** 存放数据
* **preprocess** 数据预处理
* **rule** 根据规则生成提交文件
* **model** 训练机器学习模型（暂时不分享）

## 代码使用说明

* fork本repo，非Github用户请点右下角的Downlown ZIP
* 解压后，将tianchi\_mobile\_recommend\_train\_user.csv以及tianchi\_mobile\_recommend\_train\_item.csv放入/data/目录下
* 仅需两个步骤即可获得一份提交文件，**F1可达到7.6%**
  + 第一步，进入/preprocess/目录，运行data\_preprocess.py
  + 第二步，进入/rule/目录，运行gen\_submission\_by\_rule.py
* 完成上面两个步骤后，在/rule/目录下会生成一份tianchi\_mobile\_recommendation\_predict.csv文件，提交它。

## 补充说明

* **纯Python**，无任何依赖项。
* 关于代码实现的功能，在每份代码文件中均有注释，代码可能写得比较乱，也可能有bug，欢迎issues。
* 如果你想获得更高的F1值，修改gen\_submission\_by\_rule.py这份文件，加入一些规则，**F1可以达到9%以上**。
* **建议在Linux下运行**；在我的PC上（8核），上面两个步骤总共花了不到20分钟。
* 请在规则的基础上，做特征工程，训练模型，这才是参赛目的。
* 进入第二赛季后，请删了这些代码，不适合处理大数据。

# Kaggle入门——使用scikit-learn解决DigitRecognition问题

## 1、scikit-learn简介

scikit-learn是一个基于NumPy、SciPy、Matplotlib的开源机器学习工具包，采用Python语言编写，主要涵盖分类、

回归和聚类等算法，例如knn、SVM、逻辑回归、朴素贝叶斯、随机森林、k-means等等诸多算法，官网上代码和文档

都非常不错，对于机器学习开发者来说，是一个使用方便而强大的工具，节省不少开发时间。

scikit-learn官网指南：<http://scikit-learn.org/stable/user_guide.html>

上一篇文章[《大数据竞赛平台—Kaggle入门》](http://blog.csdn.net/u012162613/article/details/41929171)  我分两部分内容介绍了Kaggle，在第二部分中，我记录了解决Kaggle上的竞赛项目DigitRecognition的整个过程，当时我是用自己写的kNN算法，尽管自己写歌kNN算法并不会花很多时间，但是当我们想尝试更多、更复杂的算法，如果每个算法都自己实现的话，会很浪费时间，这时候scikit-learn就发挥作用了，我们可以直接调用scikit-learn的算法包。当然，对于初学者来说，最好还是在理解了算法的基础上，来调用这些算法包，如果有时间，自己完整地实现一个算法相信会让你对算法掌握地更深入。

OK，话休絮烦，下面进入第二部分。

## 2、使用scikit-learn解决DigitRecognition

我发现自己很喜欢用DigitRecognition这个问题来练习分类算法，因为足够简单。如果你还不知道DigitRecognition问题是什么，请先简单了解一下：[Kaggle DigitRecognition](https://www.kaggle.com/c/digit-recognizer) ，在我上一篇文章中也有描述：[《大数据竞赛平台—Kaggle入门》](http://blog.csdn.net/u012162613/article/details/41929171) 。下面我使用scikit-learn中的算法包kNN（k近邻）、SVM（支持向量机）、NB（朴素贝叶斯）来解决这个问题，解决问题的关键步骤有两个：1、处理数据。2、调用算法。

### （1）处理数据

这一部分与上一篇文章[《大数据竞赛平台—Kaggle入门》](http://blog.csdn.net/u012162613/article/details/41929171) 中第二部分的数据处理是一样的，本文不打算重复，下面只简单地罗列各个函数及其功能，在本文最后部分也有详细的代码。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. **def** loadTrainData():
2. #这个函数从train.csv文件中获取训练样本:trainData、trainLabel
3. **def** loadTestData():
4. #这个函数从test.csv文件中获取测试样本:testData
5. **def** toInt(array):
6. **def** nomalizing(array):
7. #这两个函数在loadTrainData()和loadTestData()中被调用
8. #toInt()将字符串数组转化为整数，nomalizing()归一化整数
9. **def** loadTestResult():
10. #这个函数加载测试样本的参考label，是为了后面的比较
11. **def** saveResult(result,csvName):
12. #这个函数将result保存为csv文件，以csvName命名

“处理数据”部分，我们从train.csv、test.csv文件中获取了训练样本的feature、训练样本的label、测试样本的feature，在程序中我们用trainData、trainLabel、testData表示。

### （2）调用scikit-learn中的算法

kNN算法

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. #调用scikit的knn算法包
2. **from** sklearn.neighbors **import** KNeighborsClassifier
3. **def** knnClassify(trainData,trainLabel,testData):
4. knnClf=KNeighborsClassifier()#default:k = 5,defined by yourself:KNeighborsClassifier(n\_neighbors=10)
5. knnClf.fit(trainData,ravel(trainLabel))
6. testLabel=knnClf.predict(testData)
7. saveResult(testLabel,'sklearn\_knn\_Result.csv')
8. **return** testLabel

kNN算法包可以自己设定参数k，默认k=5，上面的comments有说明。

更加详细的使用，推荐上官网查看：<http://scikit-learn.org/stable/modules/neighbors.html>

SVM算法

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. #调用scikit的SVM算法包
2. **from** sklearn **import** svm
3. **def** svcClassify(trainData,trainLabel,testData):
4. svcClf=svm.SVC(C=5.0) #default:C=1.0,kernel = 'rbf'. you can try kernel:‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
5. svcClf.fit(trainData,ravel(trainLabel))
6. testLabel=svcClf.predict(testData)
7. saveResult(testLabel,'sklearn\_SVC\_C=5.0\_Result.csv')
8. **return** testLabel

SVC()的参数有很多，核函数默认为'rbf'（径向基函数）,C默认为1.0

更加详细的使用，推荐上官网查看：<http://scikit-learn.org/stable/modules/svm.html>

朴素贝叶斯算法

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. #调用scikit的朴素贝叶斯算法包,GaussianNB和MultinomialNB
2. **from** sklearn.naive\_bayes **import** GaussianNB      #nb for 高斯分布的数据
3. **def** GaussianNBClassify(trainData,trainLabel,testData):
4. nbClf=GaussianNB()
5. nbClf.fit(trainData,ravel(trainLabel))
6. testLabel=nbClf.predict(testData)
7. saveResult(testLabel,'sklearn\_GaussianNB\_Result.csv')
8. **return** testLabel
10. **from** sklearn.naive\_bayes **import** MultinomialNB   #nb for 多项式分布的数据
11. **def** MultinomialNBClassify(trainData,trainLabel,testData):
12. nbClf=MultinomialNB(alpha=0.1)      #default alpha=1.0,Setting alpha = 1 is called Laplace smoothing, while alpha < 1 is called Lidstone smoothing.
13. nbClf.fit(trainData,ravel(trainLabel))
14. testLabel=nbClf.predict(testData)
15. saveResult(testLabel,'sklearn\_MultinomialNB\_alpha=0.1\_Result.csv')
16. **return** testLabel

上面我尝试了两种朴素贝叶斯算法:高斯分布的和多项式分布的。多项式分布的函数有参数alpha可以自设定。

更加详细的使用，推荐上官网查看：<http://scikit-learn.org/stable/modules/naive_bayes.html>

**使用方法总结：**

第一步：首先确定使用哪种分类器，这一步可以设置各种参数，比如:

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. svcClf=svm.SVC(C=5.0)

第二步：接这个分类器要使用哪些训练数据？调用fit方法，比如:

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. svcClf.fit(trainData,ravel(trainLabel))

**fit(X,y)说明：**

X:  对应trainData

array-like, shape = [n\_samples, n\_features]，X是训练样本的特征向量集，n\_samples行n\_features列，即每个训练样本占一行，每个训练样本有多少特征就有多少列。

y:  对应trainLabel

array-like, shape = [n\_samples]，y必须是一个行向量，这也是上面为什么使用numpy.ravel()函数的原因。

第三步：使用分类器预测测试样本，比如：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. testLabel=svcClf.predict(testData)

调用predict方法。

第四步：保存结果，这一步是取决于我们解决问题的要求，因为本文以DigitRecognition为例，所以有：

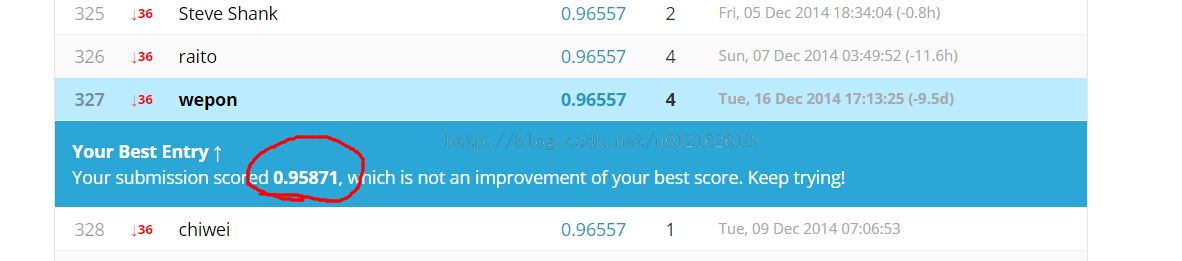
**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. saveResult(testLabel,'sklearn\_SVC\_C=5.0\_Result.csv')

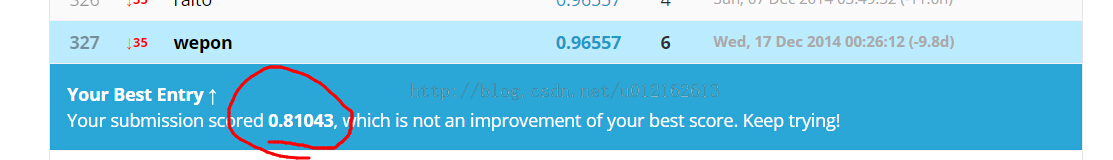
### （3）make a submission

上面基本就是整个开发过程了，下面看一下各个算法的效果，在Kaggle上make a submission

knn算法的效果，准确率95.871%



朴素贝叶斯，alpha=1.0，准确率81.043%



SVM，linear核，准确率93.943%

## 3、工程文件

CSDN下载：[Kaggle 入门-使用scikit-learn解决DigitRecoginition](http://download.csdn.net/download/u012162613/8268443)

Github：<https://github.com/wepe/Kaggle-Solution>

贴一下代码：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41978235)

1. #!/usr/bin/python
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Tue Dec 16 21:59:00 2014
6. @author: wepon
8. @blog:http://blog.csdn.net/u012162613
9. """
11. **from** numpy **import** \*
12. **import** csv
14. **def** toInt(array):
15. array=mat(array)
16. m,n=shape(array)
17. newArray=zeros((m,n))
18. **for** i **in** xrange(m):
19. **for** j **in** xrange(n):
20. newArray[i,j]=int(array[i,j])
21. **return** newArray
23. **def** nomalizing(array):
24. m,n=shape(array)
25. **for** i **in** xrange(m):
26. **for** j **in** xrange(n):
27. **if** array[i,j]!=0:
28. array[i,j]=1
29. **return** array
31. **def** loadTrainData():
32. l=[]
33. with open('train.csv') as file:
34. lines=csv.reader(file)
35. **for** line **in** lines:
36. l.append(line) #42001\*785
37. l.remove(l[0])
38. l=array(l)
39. label=l[:,0]
40. data=l[:,1:]
41. **return** nomalizing(toInt(data)),toInt(label)  #label 1\*42000  data 42000\*784
42. #return trainData,trainLabel
44. **def** loadTestData():
45. l=[]
46. with open('test.csv') as file:
47. lines=csv.reader(file)
48. **for** line **in** lines:
49. l.append(line)#28001\*784
50. l.remove(l[0])
51. data=array(l)
52. **return** nomalizing(toInt(data))  #  data 28000\*784
53. #return testData
55. **def** loadTestResult():
56. l=[]
57. with open('knn\_benchmark.csv') as file:
58. lines=csv.reader(file)
59. **for** line **in** lines:
60. l.append(line)#28001\*2
61. l.remove(l[0])
62. label=array(l)
63. **return** toInt(label[:,1])  #  label 28000\*1
65. #result是结果列表
66. #csvName是存放结果的csv文件名
67. **def** saveResult(result,csvName):
68. with open(csvName,'wb') as myFile:
69. myWriter=csv.writer(myFile)
70. **for** i **in** result:
71. tmp=[]
72. tmp.append(i)
73. myWriter.writerow(tmp)

76. #调用scikit的knn算法包
77. **from** sklearn.neighbors **import** KNeighborsClassifier
78. **def** knnClassify(trainData,trainLabel,testData):
79. knnClf=KNeighborsClassifier()#default:k = 5,defined by yourself:KNeighborsClassifier(n\_neighbors=10)
80. knnClf.fit(trainData,ravel(trainLabel))
81. testLabel=knnClf.predict(testData)
82. saveResult(testLabel,'sklearn\_knn\_Result.csv')
83. **return** testLabel
85. #调用scikit的SVM算法包
86. **from** sklearn **import** svm
87. **def** svcClassify(trainData,trainLabel,testData):
88. svcClf=svm.SVC(C=5.0) #default:C=1.0,kernel = 'rbf'. you can try kernel:‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
89. svcClf.fit(trainData,ravel(trainLabel))
90. testLabel=svcClf.predict(testData)
91. saveResult(testLabel,'sklearn\_SVC\_C=5.0\_Result.csv')
92. **return** testLabel
94. #调用scikit的朴素贝叶斯算法包,GaussianNB和MultinomialNB
95. **from** sklearn.naive\_bayes **import** GaussianNB      #nb for 高斯分布的数据
96. **def** GaussianNBClassify(trainData,trainLabel,testData):
97. nbClf=GaussianNB()
98. nbClf.fit(trainData,ravel(trainLabel))
99. testLabel=nbClf.predict(testData)
100. saveResult(testLabel,'sklearn\_GaussianNB\_Result.csv')
101. **return** testLabel
103. **from** sklearn.naive\_bayes **import** MultinomialNB   #nb for 多项式分布的数据
104. **def** MultinomialNBClassify(trainData,trainLabel,testData):
105. nbClf=MultinomialNB(alpha=0.1)      #default alpha=1.0,Setting alpha = 1 is called Laplace smoothing, while alpha < 1 is called Lidstone smoothing.
106. nbClf.fit(trainData,ravel(trainLabel))
107. testLabel=nbClf.predict(testData)
108. saveResult(testLabel,'sklearn\_MultinomialNB\_alpha=0.1\_Result.csv')
109. **return** testLabel

112. **def** digitRecognition():
113. trainData,trainLabel=loadTrainData()
114. testData=loadTestData()
115. #使用不同算法
116. result1=knnClassify(trainData,trainLabel,testData)
117. result2=svcClassify(trainData,trainLabel,testData)
118. result3=GaussianNBClassify(trainData,trainLabel,testData)
119. result4=MultinomialNBClassify(trainData,trainLabel,testData)
121. #将结果与跟给定的knn\_benchmark对比,以result1为例
122. resultGiven=loadTestResult()
123. m,n=shape(testData)
124. different=0      #result1中与benchmark不同的label个数，初始化为0
125. **for** i **in** xrange(m):
126. **if** result1[i]!=resultGiven[0,i]:
127. different+=1

# [Kaggle比赛-Otto Group Product Classification-打败一半参赛队伍的简单解法](http://blog.csdn.net/u012162613/article/details/44601523)

分类： [Machine Learning](http://blog.csdn.net/u012162613/article/category/2760005)2015-03-24 23:02 969人阅读 [评论](http://blog.csdn.net/u012162613/article/details/44601523#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/u012162613/article/details/44601523#report)

[机器学习](http://www.csdn.net/tag/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0)[kaggle比赛](http://www.csdn.net/tag/kaggle%e6%af%94%e8%b5%9b)[随机森林](http://www.csdn.net/tag/%e9%9a%8f%e6%9c%ba%e6%a3%ae%e6%9e%97)

目录[(?)[+]](http://blog.csdn.net/u012162613/article/details/44601523)

## 简介

[Otto Group Product Classification Challenge](https://www.kaggle.com/c/otto-group-product-classification-challenge)是Kaggle上目前正在进行的一个比赛，目前已1000+队伍参赛，由Otto公司赞助1W美刀，数据也是来自于该公司的产品，提供了train.csv、test.csv、samplesubmission.csv三份数据。train.csv里包含了6万多个样本，每个样本有一个id，93个特征值feat\_1~feat\_93,以及类别target，一共9种类别：class\_1~class\_9。test.csv里有14万多测试样本，只有id以及93个特征，参赛者的任务是对这些样本分类。

## 提交结果的格式

官网上给出的格式：

id,Class\_1,Class\_2,Class\_3,Class\_4,Class\_5,Class\_6,Class\_7,Class\_8,Class\_9

1,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0

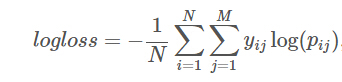
2,0.0,0.2,0.3,0.3,0.0,0.0,0.1,0.1,0.0

* 1
* 2
* 3

也就是说，可以不用给出确切的类别，给出属于各个类别的概率也行。这一点很重要，我一开始没注意看，浪费了许多精力瞎折腾。所以但凡比赛，官网给出的信息最好都仔细看一遍。

## 评分标准

评分的公式见：[这里](https://www.kaggle.com/c/otto-group-product-classification-challenge/details/evaluation)



我稍微解释一下这条公式，i表示样本，j表示类别。Pij代表第i个样本属于类别j的概率，如果第i个样本真的属于类别j，则yij等于1，否则为0。

假如你将所有的测试样本都正确分类，所有pij都是1，那每个log(pij)都是0，最终的logloss也是0。

假如第1个样本本来是属于1类别的，但是你给它的类别概率pij=0.1，那logloss就会累加上log(0.1)这一项。我们知道这一项是负数，而且pij越小，负得越多，如果pij=0，将是无穷。这会导致这种情况：你分错了一个，logloss就是无穷。这当然不合理，为了避免这一情况，官方是这样处理的：   
http://i.imgur.com/LDdL0CP.jpg

也就是说最小不会小于10^-15。

## 开始解题

废话终于说完，进入主题。下面讲一个很naive的solution，基本上不用动脑，只要会简单地使用sklearn、numpy，就能打败目前一半的参赛队伍。

代码放在我的github上[wepe/Kaggle-Solution](https://github.com/wepe/Kaggle-Solution/tree/master/Otto%20Group%20Product%20Classification%20Challenge)，分几部分来讲：

## 数据预处理

严格来说，我并没有做多少数据预处理的工作。只是一些加载数据的函数，分别是   
loadTrainSet()和loadTestSet(),加载训练数据集和测试数据集。代码如下，里面顺便做了归一化以及零均值(这里不做standardize差别也不大)。

#load train set

def loadTrainSet():

traindata = []

trainlabel = []

table = {"Class\_1":1,"Class\_2":2,"Class\_3":3,"Class\_4":4,"Class\_5":5,"Class\_6":6,"Class\_7":7,"Class\_8":8,"Class\_9":9}

with open("train.csv") as f:

rows = csv.reader(f)

rows.next()

for row in rows:

l = []

for i in range(1,94):

l.append(int(row[i]))

traindata.append(l)

trainlabel.append(table.get(row[-1]))

f.close()

traindata = np.array(traindata,dtype="float")

trainlabel = np.array(trainlabel,dtype="int")

#Standardize(zero-mean,nomalization)

mean = traindata.mean(axis=0)

std = traindata.std(axis=0)

traindata = (traindata - mean)/std

#shuffle the data

randomIndex = [i for i in xrange(len(trainlabel))]

random.shuffle(randomIndex)

traindata = traindata[randomIndex]

trainlabel = trainlabel[randomIndex]

return traindata,trainlabel

#load test set

def loadTestSet():

testdata = []

with open("test.csv") as f:

rows = csv.reader(f)

rows.next()

for row in rows:

l = []

for i in range(1,94):

l.append(int(row[i]))

testdata.append(l)

f.close()

testdata = np.array(testdata,dtype="float")

#Standardize(zero-mean,nomalization)

mean = testdata.mean(axis=0)

std = testdata.std(axis=0)

testdata = (testdata - mean)/std

return testdata

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48

代码挺长的，但其实都是些简单的读取和处理工作。（如果不想写这么繁琐的代码，可以试试数据分析包pandas，几行代码完成这些工作）。

## 模型评估

模型评估一般都是从训练数据集中分出一部分来做validation，一般我们是用validation accuracy来评估的（如上所述），但是这个比赛既然官方给出了评估公式，我们就根据这个公式写个评估函数：

#Evaluation function

#Refer to:https://www.kaggle.com/c/otto-group-product-classification-challenge/details/evaluation

def evaluation(label,pred\_label):

num = len(label)

logloss = 0.0

for i in range(num):

p = max(min(pred\_label[i][label[i]-1],1-10\*\*(-15)),10\*\*(-15))

logloss += np.log(p)

logloss = -1\*logloss/num

return logloss

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

**以上三个函数loadTrainSet()、loadTestSet()、evaluation()我都放在代码文件preprocess.py中，此外在该文件中还定义了一个saveResult()函数，根据test set的预测类别test\_label生成一个用于提交结果的submission.csv文件。**

## 分类算法

直接调用sklearn里面的随机森林，参数调节：随便调了几次决策树的数目，可以使logloss达到0.5左右，这个得分目前已经打败了一半的参赛队伍。

def rf(train\_data,train\_label,val\_data,val\_label,test\_data,name="RandomForest\_submission.csv"):

print "Start training Random forest..."

rfClf = RandomForestClassifier(n\_estimators=400,n\_jobs=-1)

rfClf.fit(train\_data,train\_label)

#evaluate on validation set

val\_pred\_label = rfClf.predict\_proba(val\_data)

logloss = preprocess.evaluation(val\_label,val\_pred\_label)

print "logloss of validation set:",logloss

print "Start classify test set..."

test\_label = rfClf.predict\_proba(test\_data)

preprocess.saveResult(test\_label,filename = name)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

这个函数的参数val\_data就是验证集数据，从train.csv里面取出6000个构成，大约占1/10的数据。

这部分的代码放在文件RandomForest.py中。

有兴趣的去我[github获取代码](https://github.com/wepe/Kaggle-Solution/tree/master/Otto%20Group%20Product%20Classification%20Challenge)试试吧。Have Fun！

# （1）kNN算法\_手写识别实例——基于Python和NumPy函数库

### 1、kNN算法简介

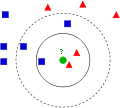
kNN算法，即K最近邻(k-NearestNeighbor)分类算法，是最简单的机器学习算法之一，算法思想很简单：从训练样本集中选择k个与测试样本“距离”最近的样本，这k个样本中出现频率最高的类别即作为测试样本的类别。下面的简介选自wiki百科：<http://zh.wikipedia.org/wiki/%E6%9C%80%E8%BF%91%E9%84%B0%E5%B1%85%E6%B3%95>

## 方法

* 目标：分类未知类别案例。
* 输入：待分类未知类别案例项目。已知类别案例集合D ，其中包含 j个已知类别的案例。
* 输出：项目可能的类别。

### 步骤

如下图  
我们考虑样本为二维的情况下，利用knn方法进行二分类的问题。图中三角形和方形是已知类别的样本点，这里我们假设三角形为正类，方形为负类。图中圆形点是未知类别的数据，我们要利用这些已知类别的样本对它进行分类。

* [](http://zh.wikipedia.org/wiki/File:KnnClassification.svg)

k近邻算法例子示意图

**分类过程如下：**  
1 首先我们事先定下k值（就是指k近邻方法的k的大小，代表对于一个待分类的数据点，我们要寻找几个它的邻居）。这边为了说明问题，我们取两个k值，分别为3和5；  
2 根据事先确定的距离度量公式（如：欧氏距离），得出待分类数据点和所有已知类别的样本点中，距离最近的k个样本。  
3 统计这k个样本点中，各个类别的数量。如上图，如果我们选定k值为3，则正类样本（三角形）有2个，负类样本（方形）有1个，那么我们就把这个圆形数据点定为正类；而如果我们选择k值为5，则正类样本（三角形）有2个，负类样本（方形）有3个，那么我们这个数据点定为负类。即，根据k个样本中，数量最多的样本是什么类别，我们就把这个数据点定为什么类别。

**补充：**

**优缺点：**

（1）优点：

算法简单，易于实现，不需要参数估计，不需要事先训练。

（2）缺点：

属于懒惰算法，“平时不好好学习，考试时才临阵磨枪”，意思是kNN不用事先训练，而是在输入待分类样本时才开始运行，这一特点导致kNN计算量特别大，而且训练样本必须存储在本地，内存开销也特别大。

**K的取值：**

参数k的取值一般通常不大于20。——《机器学习实战》

## 2、手写识别实例

kNN算法主要被应用于文本分类、相似推荐。本文将描述一个分类的例子，是《机器学习实战》一书中的实例，使用python语言以及数值计算库NumPy。下面首先简单介绍本实例编程开发过程中所用到的python、numpy中的函数。

### 2.1 python、numpy函数

NumPy库总包含两种基本的数据类型：矩阵和数组，矩阵的使用类似Matlab，本实例用得多的是数组array。

shape()

shape是numpy函数库中的方法，用于查看矩阵或者数组的维素

>>>shape(array) 若矩阵有m行n列，则返回(m,n)

>>>array.shape[0] 返回矩阵的行数m，参数为1的话返回列数n

tile()

tile是numpy函数库中的方法，用法如下:

>>>tile(A,(m,n))  将数组A作为元素构造出m行n列的数组

sum()

sum()是numpy函数库中的方法

>>>array.sum(axis=1)按行累加，axis=0为按列累加

argsort()

argsort()是numpy中的方法，得到矩阵中每个元素的排序序号

>>>A=array.argsort()  A[0]表示排序后 排在第一个的那个数在原来数组中的下标

dict.get(key,x)

python中字典的方法，get(key,x)从字典中获取key对应的value，字典中没有key的话返回0

sorted()

python中的方法

min()、max()

numpy中有min()、max()方法，用法如下

>>>array.min(0)  返回一个数组，数组中每个数都是它所在列的所有数的最小值

>>>array.min(1)  返回一个数组，数组中每个数都是它所在行的所有数的最小值

listdir('str')

python的operator中的方法

>>>strlist=listdir('str')  读取目录str下的所有文件名，返回一个字符串列表

split()

python中的方法，切片函数

>>>string.split('str')以字符str为分隔符切片，返回list

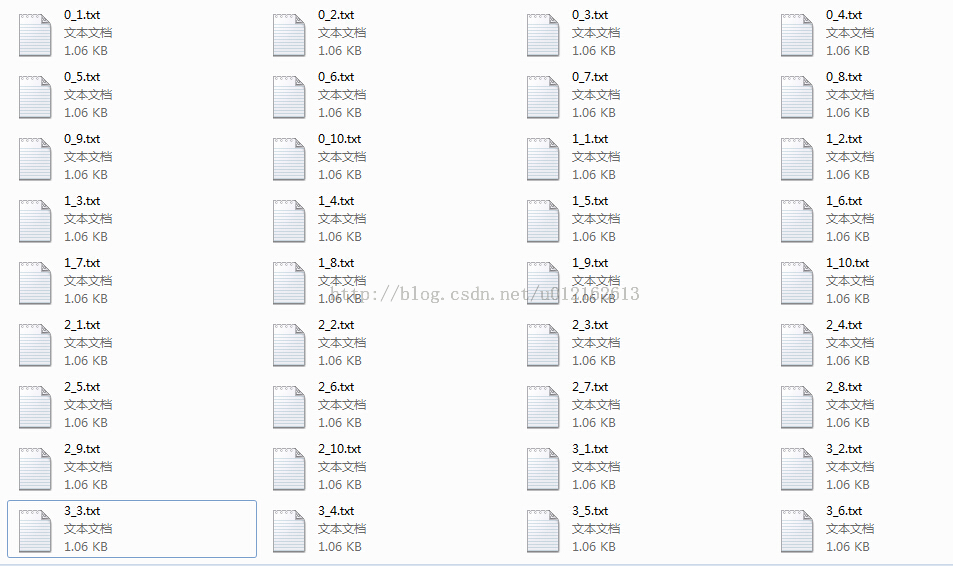
关于更多的numpy中的函数，可以查阅官网：<http://docs.scipy.org/doc/>

### 2.2  编程实现“手写识别”

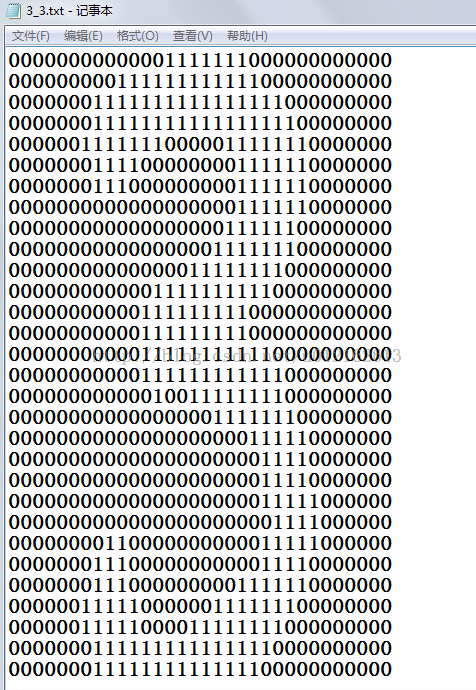
手写识别的概念：是指将在手写设备上书写时产生的轨迹信息转化为具体字码。

手写识别系统是个很大的项目，识别汉字、英语、数字、其他字符。本文只是个小demo，重点不在手写识别而在于理解kNN，因此只识别0～9单个数字。

输入格式：每个手写数字已经事先处理成32\*32的二进制文本，存储为txt文件。0～9每个数字都有10个训练样本，5个测试样本。训练样本集如下图：



**打开3\_3.txt这个文件看看：**



上面的背景介绍完了，现在编程实现，大概分为**三个步骤**：

（1）将每个图片（即txt文本，以下提到图片都指txt文本）转化为一个向量，即32\*32的数组转化为1\*1024的数组，这个1\*1024的数组用机器学习的术语来说就是特征向量。

（2）训练样本中有10\*10个图片，可以合并成一个100\*1024的矩阵，每一行对应一个图片。（这是为了方便计算，很多机器学习算法在计算的时候采用矩阵运算，可以简化代码，有时还可以减少计算复杂度）。

（3）测试样本中有10\*5个图片，我们要让程序自动判断每个图片所表示的数字。同样的，对于测试图片，将其转化为1\*1024的向量，然后计算它与训练样本中各个图片的“距离”（这里两个向量的距离采用欧式距离），然后对距离排序，选出较小的前k个，因为这k个样本来自训练集，是已知其代表的数字的，所以被测试图片所代表的数字就可以确定为这k个中出现次数最多的那个数字。

**第一步：转化为1\*1024的特征向量。程序中的filename是文件名，比如3\_3.txt**

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41768407)

1. <span style="font-family:SimSun;font-size:18px;">#样本是32\*32的二值图片，将其处理成1\*1024的特征向量
2. **def** img2vector(filename):
3. returnVect = zeros((1,1024))
4. fr = open(filename)
5. **for** i **in** range(32):
6. lineStr = fr.readline()
7. **for** j **in** range(32):
8. returnVect[0,32\*i+j] = int(lineStr[j])
9. **return** returnVect</span>

**第二步、第三步：将训练集图片合并成100\*1024的大矩阵，同时逐一对测试集中的样本分类**

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41768407)

1. <span style="font-family:SimSun;font-size:18px;">**def** handwritingClassTest():
2. #加载训练集到大矩阵trainingMat
3. hwLabels = []
4. trainingFileList = listdir('trainingDigits')           #os模块中的listdir('str')可以读取目录str下的所有文件名，返回一个字符串列表
5. m = len(trainingFileList)
6. trainingMat = zeros((m,1024))
7. **for** i **in** range(m):
8. fileNameStr = trainingFileList[i]                  #训练样本的命名格式：1\_120.txt
9. fileStr = fileNameStr.split('.')[0]                #string.split('str')以字符str为分隔符切片，返回list，这里去list[0],得到类似1\_120这样的
10. classNumStr = int(fileStr.split('\_')[0])           #以\_切片，得到1，即类别
11. hwLabels.append(classNumStr)
12. trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
14. #逐一读取测试图片，同时将其分类
15. testFileList = listdir('testDigits')
16. errorCount = 0.0
17. mTest = len(testFileList)
18. **for** i **in** range(mTest):
19. fileNameStr = testFileList[i]
20. fileStr = fileNameStr.split('.')[0]
21. classNumStr = int(fileStr.split('\_')[0])
22. vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
23. classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
24. **print** "the classifier came back with: %d, the real answer is: %d" % (classifierResult, classNumStr)
25. **if** (classifierResult != classNumStr): errorCount += 1.0
26. **print** "\nthe total number of errors is: %d" % errorCount
27. **print** "\nthe total error rate is: %f" % (errorCount/float(mTest))</span>

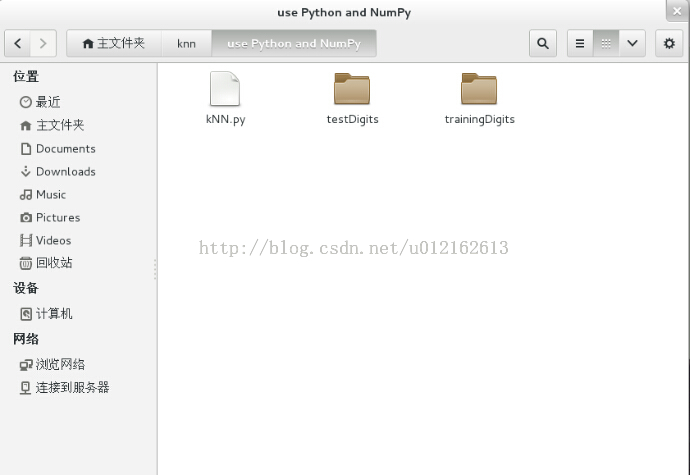
**这里面的函数classify()为分类主体函数，计算欧式距离，并最终返回测试图片类别：**

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41768407)

1. <span style="font-family:SimSun;font-size:18px;">#分类主体程序，计算欧式距离，选择距离最小的k个，返回k个中出现频率最高的类别
2. #inX是所要测试的向量
3. #dataSet是训练样本集，一行对应一个样本。dataSet对应的标签向量为labels
4. #k是所选的最近邻数目
5. **def** classify0(inX, dataSet, labels, k):
6. dataSetSize = dataSet.shape[0]                       #shape[0]得出dataSet的行数，即样本个数
7. diffMat = tile(inX, (dataSetSize,1)) - dataSet       #tile(A,(m,n))将数组A作为元素构造m行n列的数组
8. sqDiffMat = diffMat\*\*2
9. sqDistances = sqDiffMat.sum(axis=1)                  #array.sum(axis=1)按行累加，axis=0为按列累加
10. distances = sqDistances\*\*0.5
11. sortedDistIndicies = distances.argsort()             #array.argsort()，得到每个元素的排序序号
12. classCount={}                                        #sortedDistIndicies[0]表示排序后排在第一个的那个数在原来数组中的下标
13. **for** i **in** range(k):
14. voteIlabel = labels[sortedDistIndicies[i]]
15. classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1 #get(key,x)从字典中获取key对应的value，没有key的话返回0
16. sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True) #sorted()函数，按照第二个元素即value的次序逆向（reverse=True）排序
17. **return** sortedClassCount[0][0]</span>

## 3、工程文件

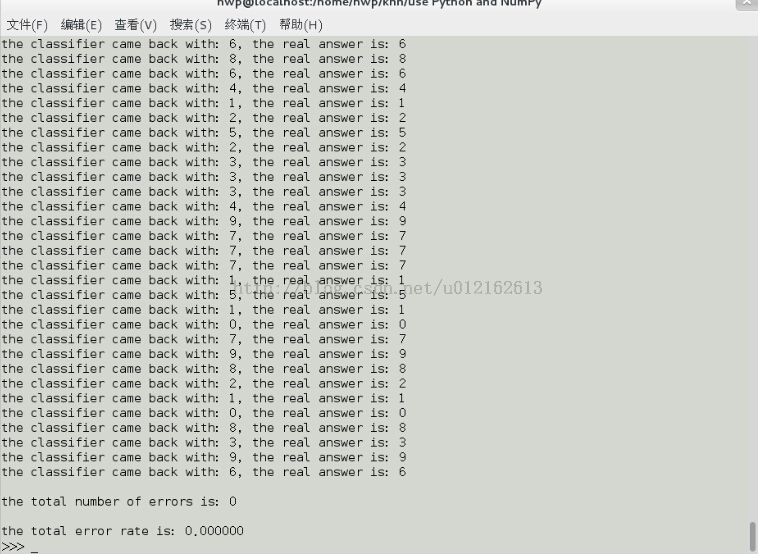
整个工程文件包括源代码、训练集、测试集，可到github获取：[github地址](https://github.com/wepe/MachineLearning-Demo/tree/master/kNN/use%20Python%20and%20NumPy)



进入use Python and NumPy目录，打开python开发环境，import kNN模块，调用手写识别函数：



因为我用的训练集和测试集都比较小，所以凑巧没有识别错误的情况：



# （2）logistic回归\_\_基于Python和Numpy函数库

## 1、算法简介

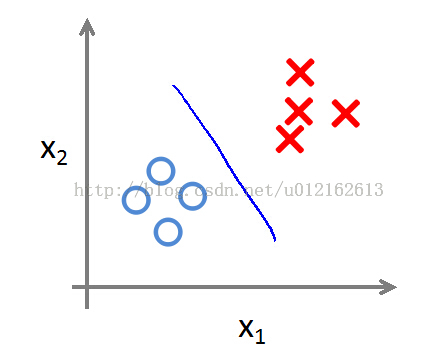
本文的重点放在算法的工程实现上，关于算法的原理不具体展开，logistic回归算法很简单，可以看看Andrew Ng的视频：<https://class.coursera.org/ml-007>，也可以看看一些写得比较好的博文：[洞庭之子的博文](http://blog.csdn.net/dongtingzhizi/article/details/15962797)。下面我只列出一些个人认为重要的点。

**回归的概念**：假设有一些数据点，我们用一条直线对这些点进行拟合，这个拟合过程就称作回归。

logistic回归算法之所以称作“logistic”，是因为它运用了**logistic函数**，即[sigmoid函数](http://science.scileaf.com/library/214)。

logistic回归算法一般用于**二分类问题**（当然也可以多类别，后面会讲）。

**logistic回归的算法思想：**

****

用上面的图来分析，每个O或X代表一个特征向量，这里是二维的，可以写成x=(x1,x2)。

用logistic回归进行分类的**主要思想**就是根据现有数据集，对分类边界建立回归公式，拿上面这个图来说，就是根据

这些OOXX，找出那条直线的公式：Θ0\*x0+Θ1\*x1+Θ2\*x2=Θ0+Θ1\*x1+Θ2\*x2=0  （x0=1）。

因为上图是二维的，所以参数Θ=(Θ0，Θ1, Θ2)，分类边界就由这个(Θ0，Θ1, Θ2)确定，对于更高维的情况也

是一样的，所以无论二维三维更高维，分类边界可以统一表示成**f(x)=ΘT\*x**  （ΘT表示Θ的转置）。

对于分类边界上的点，代入分类边界函数就得到f(x)=0,同样地，对于分类边界之上的点，代入得到f(x)>0，对于分

类边界之下的点，代入得到f(x)<0。这样就可以依据f(x)大于0或者小于0来分类了。

logistic回归的最后一步就是将f(x)作为输入，代入Sigmoid函数，当f(x)>0时，sigmoid函数的输出就大0.5，且随

着f(x)趋于正无穷，sigmoid函数的输出趋于1。当f(x)<0时，sigmoid函数的输出就小于0.5，且随着f(x)趋于负无

穷，sigmoid函数的输出趋于0。

所以我们要寻找出**最佳参数Θ**，使得对于1类别的点x，f(x)趋于正无穷，对于0类别的点x，f(x)趋于负无穷（实际编

程中不可能正/负无穷，只要足够大/小即可）。

总结一下思绪，**logistic回归的任务就是要找到最佳的拟合参数**。下图的g(z)即sigmoid函数，跟我上面讲的一样，

将f(x)=**ΘT\*x**作为g(z)的输入。

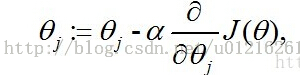


以上就是logistic回归的思想，重点在于怎么根据训练数据求得最佳拟合参数Θ？这可以用**最优化算法**来求解，比如

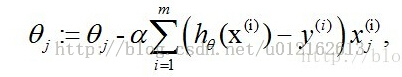
常用的**梯度上升算法**，关于梯度上升算法这里也不展开，同样可以参考上面推荐的博文。

所谓的梯度，就是函数变化最快的方向，我们一开始先将参数Θ设为全1，然后在算法迭代的每一步里计算梯度，沿

着梯度的方向移动，以此来改变参数Θ，直到Θ的拟合效果达到要求值或者迭代步数达到设定值。Θ的更新公式：



alpha是步长，一系列推导后：



这个公式也是下面写代码所用到的。

后话：理解logistic回归之后可以发现，其实它的**本质是线性回归**，得到**ΘT\*x**的过程跟线性回归是一样的，只不过

后面又将**ΘT\*x**作为logistic函数的输入，然后再判断类别。

## 2、工程实例

logistic回归一般用于二分类问题，比如判断一封邮件是否为垃圾邮件，判断照片中的人是男是女，预测一场比赛输还

是赢......当然也可以用于多分类问题，比如k类别，就进行k次logistic回归。

我的前一篇文章：[kNN算法\_\_手写识别](http://blog.csdn.net/u012162613/article/details/41768407#t2) 讲到用kNN算法识别数字0～9，这是个十类别问题，如果要用logistic回归，

得做10次logistic回归，第一次将0作为一个类别，1～9作为另外一个类别，这样就可以识别出0或非0。同样地可以将

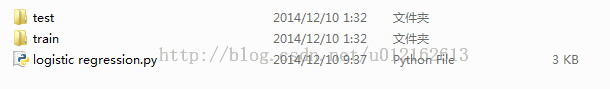
1作为一个类别，0、2～9作为一个类别，这样就可以识别出1或非1........

本文的实例同样是识别数字，但为了简化，我只选出0和1的样本，这是个二分类问题。下面开始介绍实现过程：

### （1）工程文件说明

在我的工程文件目录下，有训练样本集train和测试样本集test，源代码文件命名为logistic regression.py

训练样本集train和测试样本集test里面只有0和1样本：



logistic regression.py实现的功能：从train里面读取训练数据，然后用梯度上升算法训练出参数Θ，接着用参数Θ来预

测test里面的测试样本，同时计算错误率。

### （2）源代码解释

* **loadData(direction)函数**

实现的功能就是从文件夹里面读取所有训练样本，每个样本（比如0\_175.txt）里有32\*32的数据，程序将32\*32的数据

整理成1\*1024的向量，这样从每个txt文件可以得到一个1\*1024的特征向量X，而其类别可以从文件名“0\_175.txt”里截

取0出来。因此，从train文件夹我们可以获得一个训练矩阵m\*1024和一个类别向量m\*1，m是样本个数。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. **def** loadData(direction):
2. trainfileList=listdir(direction)
3. m=len(trainfileList)
4. dataArray= zeros((m,1024))
5. labelArray= zeros((m,1))
6. **for** i **in** range(m):
7. returnArray=zeros((1,1024))  #每个txt文件形成的特征向量
8. filename=trainfileList[i]
9. fr=open('%s/%s' %(direction,filename))
10. **for** j **in** range(32):
11. lineStr=fr.readline()
12. **for** k **in** range(32):
13. returnArray[0,32\*j+k]=int(lineStr[k])
14. dataArray[i,:]=returnArray   #存储特征向量
16. filename0=filename.split('.')[0]
17. label=filename0.split('\_')[0]
18. labelArray[i]=int(label)     #存储类别
19. **return** dataArray,labelArray

代码里面用到python os模块里的listdir()，用于从文件夹里读取所有文件，返回的是列表。

python里的open()函数用于打开文件，之后用readline()一行行读取

* **sigmoid(inX)函数**

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. **def** sigmoid(inX):
2. **return** 1.0/(1+exp(-inX))

* **gradAscent(dataArray,labelArray,alpha,maxCycles)函数**

用梯度下降法计算得到回归系数，alpha是步长，maxCycles是迭代步数。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. **def** gradAscent(dataArray,labelArray,alpha,maxCycles):
2. dataMat=mat(dataArray)    #size:m\*n
3. labelMat=mat(labelArray)      #size:m\*1
4. m,n=shape(dataMat)
5. weigh=ones((n,1))
6. **for** i **in** range(maxCycles):
7. h=sigmoid(dataMat\*weigh)
8. error=labelMat-h    #size:m\*1
9. weigh=weigh+alpha\*dataMat.transpose()\*error
10. **return** weigh

用到numpy里面的mat，矩阵类型。shape()用于获取矩阵的大小。

这个函数返回参数向量Θ，即权重weigh

* **classfy(testdir,weigh)函数**

分类函数，根据参数weigh对测试样本进行预测，同时计算错误率

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. **def** classfy(testdir,weigh):
2. dataArray,labelArray=loadData(testdir)
3. dataMat=mat(dataArray)
4. labelMat=mat(labelArray)
5. h=sigmoid(dataMat\*weigh)  #size:m\*1
6. m=len(h)
7. error=0.0
8. **for** i **in** range(m):
9. **if** int(h[i])>0.5:
10. **print** int(labelMat[i]),'is classfied as: 1'
11. **if** int(labelMat[i])!=1:
12. error+=1
13. **print** 'error'
14. **else**:
15. **print** int(labelMat[i]),'is classfied as: 0'
16. **if** int(labelMat[i])!=0:
17. error+=1
18. **print** 'error'
19. **print** 'error rate is:','%.4f' %(error/m)

* **digitRecognition(trainDir,testDir,alpha=0.07,maxCycles=10)函数**

整合上面的所有函数，调用这个函数进行数字识别，alpha和maxCycles有默认形参，这个可以根据实际情况更改。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. **def** digitRecognition(trainDir,testDir,alpha=0.07,maxCycles=10):
2. data,label=loadData(trainDir)
3. weigh=gradAscent(data,label,alpha,maxCycles)
4. classfy(testDir,weigh)

用loadData函数从train里面读取训练数据，接着根据这些数据，用gradAscent函数得出参数weigh，最后就可以用拟

合参数weigh来分类了。

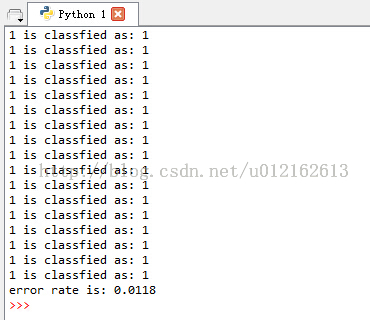
## 3、试验结果

工程文件可以到这里下载：[github地址](https://github.com/wepe/MachineLearning-Demo/tree/master/logistic%20regression)

运行logistic regression.py，采用默认形参：alpha=0.07,maxCycles=10，看下效果，错误率0.0118

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. >>> digitRecognition('train','test')



改变形参，alpah=0.01，maxCycles=50，看下效果，错误率0.0471

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/41844495)

1. >>> digitRecognition('train','test',0.01,50)



这两个参数可以根据实际情况调整

# scikit-learn中PCA的使用方法

@author：wepon

@blog：<http://blog.csdn.net/u012162613/article/details/42192293>

在前一篇文章 [主成分分析(PCA)](http://blog.csdn.net/u012162613/article/details/42177327) 中，我基于python和numpy实现了PCA算法，主要是为了加深对算法的理解，算法的实现很粗糙，实际应用中我们一般调用成熟的包，本文就结束scikit-learn中PCA使用的方法和需要注意的细节，参考：[sklearn.decomposition.PCA](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

## 1、函数原型及参数说明

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. sklearn.decomposition.PCA(n\_components=None, copy=True, whiten=False)

参数说明：

n\_components:

意义：PCA算法中所要保留的主成分个数n，也即保留下来的特征个数n

类型：int 或者 string，缺省时默认为None，所有成分被保留。

          赋值为int，比如n\_components=1，将把原始数据降到一个维度。

          赋值为string，比如n\_components='mle'，将自动选取特征个数n，使得满足所要求的方差百分比。

copy:

类型：bool，True或者False，缺省时默认为True。

意义：表示是否在运行算法时，将原始训练数据复制一份。若为True，则运行PCA算法后，原始训练数据的值不            会有任何改变，因为是在原始数据的副本上进行运算；若为False，则运行PCA算法后，原始训练数据的              值会改，因为是在原始数据上进行降维计算。

whiten:

类型：bool，缺省时默认为False

意义：白化，使得每个特征具有相同的方差。关于“白化”，可参考：[Ufldl教程](http://deeplearning.stanford.edu/wiki/index.php/%E7%99%BD%E5%8C%96)

## 2、PCA对象的属性

components\_ ：返回具有最大方差的成分。

explained\_variance\_ratio\_：返回 所保留的n个成分各自的方差百分比。

n\_components\_：返回所保留的成分个数n。

mean\_：

noise\_variance\_：

## 3、PCA对象的方法

* fit(X,y=None)

fit()可以说是scikit-learn中通用的方法，每个需要训练的算法都会有fit()方法，它其实就是算法中的“训练”这一步骤。因为PCA是无监督学习算法，此处y自然等于None。

fit(X)，表示用数据X来训练PCA模型。

函数返回值：调用fit方法的对象本身。比如pca.fit(X)，表示用X对pca这个对象进行训练。

* fit\_transform(X)

用X来训练PCA模型，同时返回降维后的数据。

newX=pca.fit\_transform(X)，newX就是降维后的数据。

* inverse\_transform()

将降维后的数据转换成原始数据，X=pca.inverse\_transform(newX)

* transform(X)

将数据X转换成降维后的数据。当模型训练好后，对于新输入的数据，都可以用transform方法来降维。

此外，还有get\_covariance()、get\_precision()、get\_params(deep=True)、score(X, y=None)等方法，以后用到再补充吧。

## 4、example

以一组二维的数据data为例，data如下，一共12个样本（x,y），其实就是分布在直线y=x上的点，并且聚集在x=1、2、3、4上，各3个。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> data
2. array([[ 1.  ,  1.  ],
3. [ 0.9 ,  0.95],
4. [ 1.01,  1.03],
5. [ 2.  ,  2.  ],
6. [ 2.03,  2.06],
7. [ 1.98,  1.89],
8. [ 3.  ,  3.  ],
9. [ 3.03,  3.05],
10. [ 2.89,  3.1 ],
11. [ 4.  ,  4.  ],
12. [ 4.06,  4.02],
13. [ 3.97,  4.01]])

data这组数据，有两个特征，因为两个特征是近似相等的，所以用一个特征就能表示了，即可以降到一维。下面就来看看怎么用sklearn中的PCA算法包。

（1）n\_components设置为1，copy默认为True，可以看到原始数据data并未改变，newData是一维的，并且明显地将原始数据分成了四类。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> **from** sklearn.decomposition **import** PCA
2. >>> pca=PCA(n\_components=1)
3. >>> newData=pca.fit\_transform(data)
4. >>> newData
5. array([[-2.12015916],
6. [-2.22617682],
7. [-2.09185561],
8. [-0.70594692],
9. [-0.64227841],
10. [-0.79795758],
11. [ 0.70826533],
12. [ 0.76485312],
13. [ 0.70139695],
14. [ 2.12247757],
15. [ 2.17900746],
16. [ 2.10837406]])
17. >>> data
18. array([[ 1.  ,  1.  ],
19. [ 0.9 ,  0.95],
20. [ 1.01,  1.03],
21. [ 2.  ,  2.  ],
22. [ 2.03,  2.06],
23. [ 1.98,  1.89],
24. [ 3.  ,  3.  ],
25. [ 3.03,  3.05],
26. [ 2.89,  3.1 ],
27. [ 4.  ,  4.  ],
28. [ 4.06,  4.02],
29. [ 3.97,  4.01]])

（2）将copy设置为False，原始数据data将发生改变。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> pca=PCA(n\_components=1,copy=False)
2. >>> newData=pca.fit\_transform(data)
3. >>> data
4. array([[-1.48916667, -1.50916667],
5. [-1.58916667, -1.55916667],
6. [-1.47916667, -1.47916667],
7. [-0.48916667, -0.50916667],
8. [-0.45916667, -0.44916667],
9. [-0.50916667, -0.61916667],
10. [ 0.51083333,  0.49083333],
11. [ 0.54083333,  0.54083333],
12. [ 0.40083333,  0.59083333],
13. [ 1.51083333,  1.49083333],
14. [ 1.57083333,  1.51083333],
15. [ 1.48083333,  1.50083333]])

（3）n\_components设置为'mle'，看看效果，自动降到了1维。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> pca=PCA(n\_components='mle')
2. >>> newData=pca.fit\_transform(data)
3. >>> newData
4. array([[-2.12015916],
5. [-2.22617682],
6. [-2.09185561],
7. [-0.70594692],
8. [-0.64227841],
9. [-0.79795758],
10. [ 0.70826533],
11. [ 0.76485312],
12. [ 0.70139695],
13. [ 2.12247757],
14. [ 2.17900746],
15. [ 2.10837406]])

（4）对象的属性值

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> pca.n\_components
2. 1
3. >>> pca.explained\_variance\_ratio\_
4. array([ 0.99910873])
5. >>> pca.explained\_variance\_
6. array([ 2.55427003])
7. >>> pca.get\_params
8. <bound method PCA.get\_params of PCA(copy=True, n\_components=1, whiten=False)>

我们所训练的pca对象的n\_components值为1，即保留1个特征，该特征的方差为2.55427003，占所有特征的方差百分比为0.99910873，意味着几乎保留了所有的信息。get\_params返回各个参数的值。

（5）对象的方法

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> newA=pca.transform(A)

对新的数据A，用已训练好的pca模型进行降维。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42192293)

1. >>> pca.set\_params(copy=False)
2. PCA(copy=False, n\_components=1, whiten=False)

设置参数。

# 主成分分析(PCA)——基于python+numpy

## 1、PCA算法介绍

主成分分析（Principal Components Analysis），简称PCA，是一种数据降维技术，用于数据预处理。一般我们获取的原始数据维度都很高，比如1000个特征，在这1000个特征中可能包含了很多无用的信息或者噪声，真正有用的特征才100个，那么我们可以运用PCA算法将1000个特征降到100个特征。这样不仅可以去除无用的噪声，还能减少很大的计算量。

PCA算法是如何实现的？

简单来说，就是将数据从原始的空间中转换到新的特征空间中，例如原始的空间是三维的(x,y,z)，x、y、z分别是原始空间的三个基，我们可以通过某种方法，用新的坐标系(a,b,c)来表示原始的数据，那么a、b、c就是新的基，它们组成新的特征空间。在新的特征空间中，可能所有的数据在c上的投影都接近于0，即可以忽略，那么我们就可以直接用(a,b)来表示数据，这样数据就从三维的(x,y,z)降到了二维的(a,b)。

问题是如何求新的基(a,b,c)?

一般步骤是这样的：先对原始数据零均值化，然后求协方差矩阵，接着对协方差矩阵求特征向量和特征值，这些特征向量组成了新的特征空间。具体的细节，推荐Andrew Ng的网页教程：[Ufldl 主成分分析](http://deeplearning.stanford.edu/wiki/index.php/%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90) ，写得很详细。

## 2、PCA算法实现

语言：Python

函数库：Numpy

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. >>> **import** numpy as np

根据上面提到的一般步骤来实现PCA算法

### （1）零均值化

假如原始数据集为矩阵dataMat，dataMat中每一行代表一个样本，每一列代表同一个特征。零均值化就是求每一列的平均值，然后该列上的所有数都减去这个均值。也就是说，这里零均值化是对每一个特征而言的，零均值化都，每个特征的均值变成0。实现代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. **def** zeroMean(dataMat):
2. meanVal=np.mean(dataMat,axis=0)     #按列求均值，即求各个特征的均值
3. newData=dataMat-meanVal
4. **return** newData,meanVal

函数中用numpy中的mean方法来求均值，axis=0表示按列求均值。

该函数返回两个变量，newData是零均值化后的数据，meanVal是每个特征的均值，是给后面重构数据用的。

### （2）求协方差矩阵

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. newData,meanVal=zeroMean(dataMat)
2. covMat=np.cov(newData,rowvar=0)

numpy中的cov函数用于求协方差矩阵，参数rowvar很重要！若rowvar=0，说明传入的数据一行代表一个样本，若非0，说明传入的数据一列代表一个样本。因为newData每一行代表一个样本，所以将rowvar设置为0。

covMat即所求的协方差矩阵。

### （3）求特征值、特征矩阵

调用numpy中的线性代数模块linalg中的eig函数，可以直接由covMat求得特征值和特征向量：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. eigVals,eigVects=np.linalg.eig(np.mat(covMat))

eigVals存放特征值，行向量。

eigVects存放特征向量，每一列带别一个特征向量。

特征值和特征向量是一一对应的

### （4）保留主要的成分[即保留值比较大的前n个特征]

第三步得到了特征值向量eigVals，假设里面有m个特征值，我们可以对其排序，排在前面的n个特征值所对应的特征向量就是我们要保留的，它们组成了新的特征空间的一组基n\_eigVect。将零均值化后的数据乘以n\_eigVect就可以得到降维后的数据。代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. eigValIndice=np.argsort(eigVals)            #对特征值从小到大排序
2. n\_eigValIndice=eigValIndice[-1:-(n+1):-1]   #最大的n个特征值的下标
3. n\_eigVect=eigVects[:,n\_eigValIndice]        #最大的n个特征值对应的特征向量
4. lowDDataMat=newData\*n\_eigVect               #低维特征空间的数据
5. reconMat=(lowDDataMat\*n\_eigVect.T)+meanVal  #重构数据
6. **return** lowDDataMat,reconMat

代码中有几点要说明一下，首先argsort对特征值是从小到大排序的，那么最大的n个特征值就排在后面，所以eigValIndice[-1:-(n+1):-1]就取出这个n个特征值对应的下标。【python里面，list[a:b:c]代表从下标a开始到b，步长为c。】

reconMat是重构的数据，乘以n\_eigVect的转置矩阵，再加上均值meanVal。

OK，这四步下来就可以从高维的数据dataMat得到低维的数据lowDDataMat，另外，程序也返回了重构数据reconMat，有些时候reconMat课便于数据分析。

贴一下总的代码：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. #零均值化
2. **def** zeroMean(dataMat):
3. meanVal=np.mean(dataMat,axis=0)     #按列求均值，即求各个特征的均值
4. newData=dataMat-meanVal
5. **return** newData,meanVal
7. **def** pca(dataMat,n):
8. newData,meanVal=zeroMean(dataMat)
9. covMat=np.cov(newData,rowvar=0)    #求协方差矩阵,return ndarray；若rowvar非0，一列代表一个样本，为0，一行代表一个样本
11. eigVals,eigVects=np.linalg.eig(np.mat(covMat))#求特征值和特征向量,特征向量是按列放的，即一列代表一个特征向量
12. eigValIndice=np.argsort(eigVals)            #对特征值从小到大排序
13. n\_eigValIndice=eigValIndice[-1:-(n+1):-1]   #最大的n个特征值的下标
14. n\_eigVect=eigVects[:,n\_eigValIndice]        #最大的n个特征值对应的特征向量
15. lowDDataMat=newData\*n\_eigVect               #低维特征空间的数据
16. reconMat=(lowDDataMat\*n\_eigVect.T)+meanVal  #重构数据
17. **return** lowDDataMat,reconMat

## 3、选择主成分个数

文章写到这里还没有完，应用PCA的时候，对于一个1000维的数据，我们怎么知道要降到几维的数据才是合理的？即n要取多少，才能保留最多信息同时去除最多的噪声？一般，我们是通过方差百分比来确定n的，这一点在Ufldl教程中说得很清楚，并且有一条简单的公式，下面是该公式的截图：



根据这条公式，可以写个函数，函数传入的参数是百分比percentage和特征值向量，然后根据percentage确定n，代码如下：

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. **def** percentage2n(eigVals,percentage):
2. sortArray=np.sort(eigVals)   #升序
3. sortArray=sortArray[-1::-1]  #逆转，即降序
4. arraySum=sum(sortArray)
5. tmpSum=0
6. num=0
7. **for** i **in** sortArray:
8. tmpSum+=i
9. num+=1
10. **if** tmpSum>=arraySum\*percentage:
11. **return** num

那么pca函数也可以重写成百分比版本，默认百分比99%。

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42177327)

1. **def** pca(dataMat,percentage=0.99):
2. newData,meanVal=zeroMean(dataMat)
3. covMat=np.cov(newData,rowvar=0)    #求协方差矩阵,return ndarray；若rowvar非0，一列代表一个样本，为0，一行代表一个样本
4. eigVals,eigVects=np.linalg.eig(np.mat(covMat))#求特征值和特征向量,特征向量是按列放的，即一列代表一个特征向量
5. n=percentage2n(eigVals,percentage)                 #要达到percent的方差百分比，需要前n个特征向量
6. eigValIndice=np.argsort(eigVals)            #对特征值从小到大排序
7. n\_eigValIndice=eigValIndice[-1:-(n+1):-1]   #最大的n个特征值的下标
8. n\_eigVect=eigVects[:,n\_eigValIndice]        #最大的n个特征值对应的特征向量
9. lowDDataMat=newData\*n\_eigVect               #低维特征空间的数据
10. reconMat=(lowDDataMat\*n\_eigVect.T)+meanVal  #重构数据
11. **return** lowDDataMat,reconMat

# 【简化数据】奇异值分解(SVD)

@author：wepon

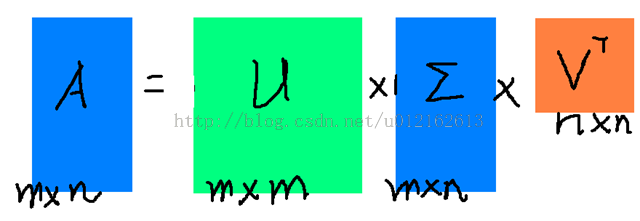
@blog：http://blog.csdn.net/u012162613/article/details/42214205

## 1、简介

奇异值分解（singular Value Decomposition），简称SVD，线性代数中矩阵分解的方法。假如有一个矩阵A，对它进行奇异值分解，可以得到三个矩阵：

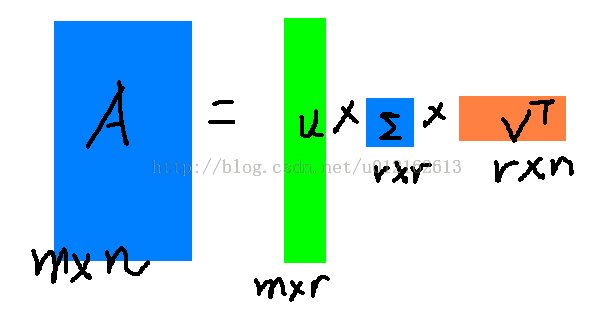
http://img.blog.csdn.net/20141228153754275?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdTAxMjE2MjYxMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

这三个矩阵的大小：



矩阵sigma(即上图U和V中间的矩阵)除了对角元素不为0，其他元素都为0，并且对角元素是从大到小排列的，前面的元素比较大，后面的很多元素接近0。这些对角元素就是奇异值。

sigma中有n个奇异值，但是由于排在后面的很多接近0，所以我们可以仅保留比较大的r个奇异值：



实际应用中，我们仅需保留着三个比较小的矩阵，就能表示A，不仅节省存储量，在计算的时候更是减少了计算量。SVD在信息检索（隐性语义索引）、图像压缩、推荐系统、金融等领域都有应用。

在上一篇文章[主成分分析](http://blog.csdn.net/u012162613/article/details/42177327)中，我是通过特征值分解的方法来实现PCA的，除了特征值分解，还可以用奇异值分解来实现PCA。特征值和奇异值二者之间是有关系的：上面我们由矩阵A获得了奇异值sigma(i)，假如方阵A\*A'的特征值为lamda(i)，则：sigma(i)^2=lamda(i)。可以发现，求特征值必须要求矩阵是方阵，而求奇异值对任意矩阵都可以，因此PCA的实现其实用SVD的更多，在scikit-learn中，PCA算法其实也是通过SVD来实现的。

## 2、在python中使用SVD

numpy中的linalg已经实现了SVD，可以直接调用

**[python]** [view plaincopy](http://blog.csdn.net/u012162613/article/details/42214205)

1. >>> A=mat([[1,2,3],[4,5,6]])
2. >>> **from** numpy **import** linalg as la
3. >>> U,sigma,VT=la.svd(A)
4. >>> U
5. matrix([[-0.3863177 , -0.92236578],
6. [-0.92236578,  0.3863177 ]])
7. >>> sigma
8. array([ 9.508032  ,  0.77286964])
9. >>> VT
10. matrix([[-0.42866713, -0.56630692, -0.7039467 ],
11. [ 0.80596391,  0.11238241, -0.58119908],
12. [ 0.40824829, -0.81649658,  0.40824829]])

有一点需要注意，sigma本来应该跟A矩阵的大小2\*3一样，但linalg.svd()只返回了一个行向量的sigma，并且只有2个奇异值（本来应该有3个），这是因为第三个奇异值为0，舍弃掉了。之所以这样做，是因为当A是非常大的矩阵时，只返回奇异值可以节省很大的存储空间。当然，如果我们要重构A，就必须先将sigma转化为矩阵。

推荐文章，也是本文参考的一篇文章：[机器学习中的数学(5)-强大的矩阵奇异值分解(SVD)及其应用](http://www.cnblogs.com/LeftNotEasy/archive/2011/01/19/svd-and-applications.html)