## 0.\*.CSV和\*.code的含义

\*.CSV中-3表示没有做，1选A，2选B

\*.code第一行的数字是一个随机种子，不用管。

\*.CSV中的第1题对应\*.Code中第0题。

## 1.生成data

numpy读写文件:

http://blog.csdn.net/sherrylml/article/details/51494052

<http://blog.csdn.net/qq_24330285/article/details/51576912>

\*.csv生成两个np数组，one person correspond：

①[提交时间(h)，所用时间(s)，来自IP(IP地址)，来自IP(省会)，性别，年龄]

②[题目1序号，……，题目5序号]

\*.code生成一个np数组，一个题目序号对应：

①[A选项概率 A选项人数 0 0 1 0 0 2 0 B选项概率 B选项人数 1 0 1 0 0 0 0]

2.生成data

组合\*.csv②和\*.code①生成data文件，data每一行为：

A选项概率 A选项人数 0 0 1 0 0 2 0 B选项概率 B选项人数 1 0 1 0 0 0 0 选A/B

## 2.用data生成lmdb

LMDB\_PATH是存LMDB的路径

N是数据的行数

DATA\_H\*DATA\_W等于特征数目

14行说的是要用另一个数组赋值给X和Y初始化他们，另一个数组是从data中读进来的，eg:

#read file  
file\_object=open(FILE\_TEST\_Free)  
try:  
    lines0=file\_object.readlines()  
finally:  
    file\_object.close()  
  
print len(lines0)  
#process space and \n  
for i in range(0,len(lines0)):  
    lines0[i]=lines0[i].split()

#numpy.random.shuffle(lines0)  
  
FEATURE  = numpy.zeros((NUM\_TEST\_Free,NUM\_FEATURE),dtype=numpy.int64)  
TAG = numpy.zeros( NUM\_TEST\_Free           ,dtype=numpy.int64)  
  
for r in range(0,NUM\_TEST\_Free):  
    for c in range(0,NUM\_FEATURE):  
        FEATURE[r][c] = lines0[r][c]  
    TAG[r] = lines0[r][NUM\_FEATURE]

#write lmdb  
env = lmdb.open(LMDB\_TEST\_Free,map\_size=LMDB\_Free\_MAX\_SIZE)  
with env.begin(write=True) as txn:  
  for i in range(0,NUM\_TEST\_Free):  
    datum = caffe.proto.caffe\_pb2.Datum()  
    datum.channels = X\_TEST\_Free.shape[1]  
    datum.height = X\_TEST\_Free.shape[2]  
    datum.width = X\_TEST\_Free.shape[3]  
    #datum.data = X\_TEST[i].tobytes()  
    datum.data = X\_TEST\_Free[i].tostring()  
    datum.label = int(Y\_TEST\_Free[i])  
    str\_id = '{:08}'.format(i)  
    txn.put(str\_id.encode('ascii'),datum.SerializeToString())

19行map\_size不够大时会报溢出

30行报错换成datum.data = X[i].tobytes()

1. **import** numpy as np
2. **import** lmdb
3. **import** caffe
5. LMDB\_PATH=”./mylmdb”
6. N = 1000
7. DATA\_H=1
8. DATA\_W=16
10. # Let's pretend this is interesting data
11. X = np.zeros((N, 1, DATA\_H, DATA\_W), dtype=np.uint8)
12. y = np.zeros(N, dtype=np.int64)
14. # We need to prepare the database for the size. We'll set it 10 times
15. # greater than what we theoretically need. There is little drawback to
16. # setting this too big. If you still run into problem after raising
17. # this, you might want to try saving fewer entries in a single
18. # transaction.
19. map\_size =  100000
21. env = lmdb.open(LMDB\_PATH, map\_size=map\_size)
23. with env.begin(write=True) as txn:
24. # txn is a Transaction object
25. **for** i **in** range(N):
26. datum = caffe.proto.caffe\_pb2.Datum()
27. datum.channels = X.shape[1]
28. datum.height = X.shape[2]
29. datum.width = X.shape[3]
30. datum.data = X[i].tobytes()  # or .tostring() if numpy < 1.9
31. datum.label = int(y[i])
32. str\_id = '{:08}'.format(i)
34. # The encode is only essential in Python 3
35. txn.put(str\_id.encode('ascii'), datum.SerializeToString())

4.solver.prototxt,train\_net.sh,train\_val.prototxt

运行：

cd ~/caffe

./path0/path1/ train\_net.sh (path0,path1用路径信息替换)

test\_iter\*batch\_size等于测试数据的数目

Eg:

train\_net.sh  
#!/usr/bin/env sh  
set -e  
  
LOG=./TjDetection/Log/my.log  
  
./build/tools/caffe train \  
     --solver=./TjDetection/net/prototxt/solver.prototxt --gpu=0 2>&1 | tee $LOG

solver.prototx

net: "/home/gongyue/caffe/TjDetection/net/prototxt/train\_val.prototxt"  
test\_iter: 1  
test\_interval: 10  
base\_lr: 0.001  
lr\_policy: "step"  
gamma: 0.95  
stepsize: 1000  
display: 20  
max\_iter: 6000  
momentum: 0.9  
weight\_decay: 0.0005  
snapshot: 100  
snapshot\_prefix: "/home/gongyue/caffe/TjDetection/net/snapshot/"  
solver\_mode: GPU  
train\_val.prototxt  
name: "GYNET"  
layer {  
  name: "DATA"  
  type: "Data"  
  top: "data"  
  top: "label"  
  include {  
    phase: TRAIN  
  }  
  data\_param {  
    source: "TjDetection/LMDB/normal/train"  
    batch\_size: 1000  
    backend: LMDB  
  }  
}  
layer {  
  name: "DATA"  
  type: "Data"  
  top: "data"  
  top: "label"  
  include {  
    phase: TEST  
  }  
  data\_param {  
    source: "TjDetection/LMDB/normal/test1in"  
    batch\_size: 1  
    backend: LMDB  
  }  
}  
layer {  
  name: "IP1"  
  type: "InnerProduct"  
  bottom: "data"  
  top: "ip1"  
  param {  
    lr\_mult: 1  
    decay\_mult: 1  
  }  
  param {  
    lr\_mult: 2  
    decay\_mult: 0  
  }  
  inner\_product\_param {  
    num\_output: 48  
    weight\_filler {  
      type: "xavier"  
    }  
    bias\_filler {  
      type: "constant"  
    }  
  }  
}  
layer {  
  name: "IP2"  
  type: "InnerProduct"  
  bottom: "ip1"  
  top: "ip2"  
  param {  
    lr\_mult: 1  
    decay\_mult: 1  
  }  
  param {  
    lr\_mult: 2  
    decay\_mult: 0  
  }  
  inner\_product\_param {  
    num\_output: 48  
    weight\_filler {  
      type: "xavier"  
    }  
    bias\_filler {  
      type: "constant"  
    }  
  }  
}  
layer {  
  name: "IP3"  
  type: "InnerProduct"  
  bottom: "ip2"  
  top: "ip3"  
  param {  
    lr\_mult: 1  
    decay\_mult: 1  
  }  
  param {  
    lr\_mult: 2  
    decay\_mult: 0  
  }  
  inner\_product\_param {  
    num\_output: 2  
    weight\_filler {  
      type: "xavier"  
    }  
    bias\_filler {  
      type: "constant"  
    }  
  }  
}  
layer {  
  name: "LOSS"  
  type: "SoftmaxWithLoss"  
  bottom: "ip3"  
  bottom: "label"  
  top: "loss"  
}  
layer {  
  name: "ACCURACY"  
  type: "Accuracy"  
  bottom: "ip3"  
  bottom: "label"  
  top: "accuracy"  
  include {  
    phase: TEST  
  }  
}