Assignment 1, Problem 5

DigitalEd **Jan 27**

Hey all,
Is there a way to check for duplicates without doing a nested loop for each dimension?
My numpy isn't that good so I may be missing the obvious (I discovered bincount today!).

Is this the numpy way to do it?

foreach image1 in training: // basically do training[1], then training[2], etc
foreach image2 in test:
if((image1==image2).all()):
++dupCnt;

Pardon my mixed language pseudo code.

----- Ed

---

baran_jana **Jan 28**

You need a nested loop but you can reduce the number of comparisons by
1) comparing only data for the same label
2) hashing into more buckets (labels)

I used the builtin python hash function with modulo and got these results:

train set size 200000 test size 18724 validation size 10000

train / test
label 0 buckets 18232 1840
label 1 buckets 17943 1844
label 2 buckets 18074 1839
label 3 buckets 17902 1835
label 4 buckets 18172 1835
label 5 buckets 18142 1843
label 6 buckets 18277 1841
label 7 buckets 17942 1838
label 8 buckets 16021 1587
label 9 buckets 18099 1842
comparisons 3898 duplicates 2396
1 loops, best of 1: 1.85 s per loop

train / validation
label 0 buckets 18232 968
label 1 buckets 17943 1007
label 2 buckets 18074 996
label 3 buckets 17902 985

label 4 buckets 18172 1005
label 5 buckets 18142 968
label 6 buckets 18277 1008
label 7 buckets 17942 984
label 8 buckets 16021 971
label 9 buckets 18099 970
comparisons 1081 duplicates 244
1 loops, best of 1: 1.39 s per loop

test / validation
label 0 buckets 1773 945
label 1 buckets 1765 991
label 2 buckets 1763 962
label 3 buckets 1748 955
label 4 buckets 1765 978
label 5 buckets 1764 946
label 6 buckets 1761 989
label 7 buckets 1755 964
label 8 buckets 1525 951
label 9 buckets 1760 949
comparisons 963 duplicates 23
1 loops, best of 1: 303 ms per loop

---

miel_shayne **Jan 28**

Careful. Comparing only samples with the same label could miss a lot of duplicates that have different labels. I got 2444 train/test duplicates and 1040 train/valid duplicates.

In order to make mine fast, I only compared samples whose data summed to the same number.

---

beader_chen **Jan 28**

Hi DigitalEd:
you can try numpy.corrcoef which can produce a correlation coefficient matrix for 2 dataset directly. Then you can simply count how many values in the upper triangle part of this corr matrix are above a given threshold which means 'duplicate'. Because numpy.corrcoef will use a very efficient BLAS to deal with the computation, so it will be much faster then a nested loop. Be aware of memory issue if the corr matrix is too big, you may consider to divide the train_set by rows into small batches and do the overlap count respectively, then sums up the result together.

---

jkarimi91 **Jan 30**

can you elaborate on what you mean by "hashing into more buckets"?

---

jose_a_magana **Jan 30**

In the case of overlapping between images with different labels, how should we solve the conflict?
This shows the transformation/rescaling we have done is maybe not appropriate.

Can you show examples of images where this has happened?

These cases are noise to the system. We have two objects that are equal but that have different labels.

---

jose_a_magana **Jan 30**

How have you done the hash of each image?
ndarray is unhashable by default
Also I am surprised of the speed of your code for the size of the data

---

jkarimi91 **Jan 30**

has anyne been able to accomplish this task on the order of minutes? I have 200,000 train and 10,000 validation samples; just checking for duplicates between these two sets will take ~13 hours

---

eythian **Jan 30**

I did this for finding duplicates, it seems to work and runs in 10 or 20 seconds:

```
train_dataset.flags.writeable=False
test_dataset.flags.writeable=False
dup_table={}
for idx,img in enumerate(train_dataset):
    h = hash(img.data)
    if h in dup_table and (train_dataset[dup_table[h]].data == img.data):
       print 'Duplicate image: %d matches %d' % (idx, dup_table[h])
    dup_table[h] = idx
for idx,img in enumerate(test_dataset):
    h = hash(img.data)
    if h in dup_table and (train_dataset[dup_table[h]].data == img.data):
        print 'Test image %d is in the training set' % idx
```

---

xsh6528 **Jan 30**

I use the following code to hash

train_dataset_hashed_set = set([hashlib.sha1(image_array).hexdigest() for image_array in train_dataset])

---

wangweimin777 **Jan 31**

Another way is to use bloom filter **https://en.wikipedia.org/wiki/Bloom_filter**, which is fast and efficient. It has a 100% recall rate but False positive are possible. Since we are not required to get the accurate overlapping rate, an estimate will be good enough and accuracy can be controlled to a certain level.

There are also well implemented python packages for Bloom filter to use

---

shrimp_323646 **Jan 31**

I believe **@eythian** 's solution is the most natural one, but you can make it much faster using sets:

```
import time

def check_overlaps(images1, images2):
    images1.flags.writeable=False
    images2.flags.writeable=False
    start = time.clock()
    hash1 = set([hash(image1.data) for image1 in images1])
    hash2 = set([hash(image2.data) for image2 in images2])
    all_overlaps = set.intersection(hash1, hash2)
    return all_overlaps, time.clock()-start

r, execTime = check_overlaps(train_dataset, test_dataset)
print "# overlaps between training and test sets:", len(r), "execution time:", e
r, execTime = check_overlaps(train_dataset, valid_dataset)
print "# overlaps between training and validation sets:", len(r), "execution tim
r, execTime = check_overlaps(valid_dataset, test_dataset)
print "# overlaps between validation and test sets:", len(r), "execution time:",
```

The result:

```
# overlaps between training and test sets: 2080 execution time: 0.979457 s
# overlaps between training and validation sets: 973 execution time: 0.955054 s
# overlaps between validation and test sets: 116 execution time: 0.134191 s
```

Note that this method ignores the duplicates inside the 3 datasets.

Sivakumaran **Feb 1**

Nice solution. I have a few doubts.
1. With a dataset of 200k images, the collision probability is high (for 32 bit digests). Will a higher bit digest function improve the solution?

1. This will only measure the overlap. To create a sanitized dataset, we will *have* to do a comparison and remove the duplicates, isnt it?

DigitalEd **Feb 1**

I've found the easiest way to speed things up (~12-13 hours down to ~90 minutes) is to split the data into pieces and run concurrently. A classic divide and conquer approach using Pools.

```
from multiprocessing import Pool, freeze_support

def ProcessChunk(a,b):
    dupCnt =0
    for x in range(0,len(a)):
```

```
            for y in range(0,len(b)):
                if((a[x]==b[y]).all()):
                    dupCnt = dupCnt+1
        return dupCnt

    if __name__ == '__main__':
        freeze_support() # for Windows and Python-isms
        pool = Pool()

        # where a is the training data and b is the test data
        # I have 8 CPUs so I broke a into 8 chunks but simplified it here to reduce
        result1 = pool.apply_async(ProcessChunk,[a[0:5000],b])
        result2 = pool.apply_async(ProcessChunk,[a[5001:10000],b])

        ans1 = result1.get()
        ans2 = result2.get()

        print ans1 + ans2
```

---

shrimp_323646 **Feb 1**

What you say is true, the code I wrote works under the no collisions assumption, which can be discussed. There are two different images that may receive the same hash, so this may be an issue to count the number of overlaps but not to create a sanitized version of the dataset. Once you know the intersection between your two sets, you can decide to reassign the images accordingly, you might move images that did not really require it, but in the end there will be no overlap.

If you really want to do all the comparisons, then the bucket method is probably the best. I've combined it with spark to get fast results. To initialize pyspark:

```
import os
import sys

SPARK_HOME = "/home/mat/spark-1.5.2-bin-hadoop2.6/" #replace with the path to yo
os.environ['SPARK_HOME'] = SPARK_HOME
sys.path.insert(0, os.path.join(SPARK_HOME, "python"))
sys.path.insert(0, os.path.join(SPARK_HOME, 'python/lib/py4j-0.8.2.1-src.zip'))

os.environ['PYSPARK_SUBMIT_ARGS'] = "pyspark-shell"

from pyspark import SparkContext
from pyspark import SparkConf

conf = SparkConf().setAppName("Check-overlaps")\
                  .setMaster("local[4]") #I'll just run in local on 4 CPUs
sc = SparkContext(conf=conf)
```

Then some code to create the buckets and calling spark for the comparisons might look like:

```
def check_overlaps_spark_nohash(images1, images2, numBuckets=120):
    #filling buckets:
    start = time.time()
    buckets1 = defaultdict(list)
    buckets2 = defaultdict(list)
    all_overlaps = []
    for image1 in images1:
        bucketId = int(np.sum(image1)*1234)%numBuckets
        buckets1[bucketId].append(image1)
    for image2 in images2:
        bucketId = int(np.sum(image2)*1234)%numBuckets
        buckets2[bucketId].append(image2)
    #comparing images in same bucket
    for bucketId in buckets1:
        images1_rdd = sc.parallelize(buckets1[bucketId])
        images2_rdd = sc.parallelize(buckets2[bucketId])
        overlaps = images1_rdd.cartesian(images2_rdd)\
                              .map(lambda x: 1 if np.array_equal(x[0], x[1]) els
                              .collect()
        all_overlaps.append(sum(overlaps))
    return sum(all_overlaps), time.time()-start

r, execTime = check_overlaps_spark_nohash(train_dataset, test_dataset)
print "# overlaps between training and test sets:", r, "execution time:", execTi
r, execTime = check_overlaps_spark_nohash(train_dataset, valid_dataset)
print "# overlaps between training and validation sets:", r, "execution time:",
r, execTime = check_overlaps_spark_nohash(valid_dataset, test_dataset)
print "# overlaps between validation and test sets:", r, "execution time:", exec
```

The result:

```
# overlaps between training and test sets: 538336 execution time: 128.359083176
# overlaps between training and validation sets: 250461 execution time: 80.41745
# overlaps between validation and test sets: 30109 execution time: 17.7448010445
```

Keep in mind that all the comparisons are done e.g. the number of overlaps between [1,3,4,1] and [1,1,2,1] will be 6, not 1.

---

dixon1e **Feb 1**

That is insanely fast. By comparison my original code was taking hours, this is clever and easy to understand. Thank you.

---

rvisual01 **Feb 1**

Thank you for sharing, had never heard of Spark until reading this.

---

stmax82 **Feb 2**

2 seconds with numpy

```
import time
import hashlib

t1 = time.time()

train_hashes = [hashlib.sha1(x).digest() for x in train_dataset]
valid_hashes = [hashlib.sha1(x).digest() for x in valid_dataset]
test_hashes  = [hashlib.sha1(x).digest() for x in test_dataset]

valid_in_train = np.in1d(valid_hashes, train_hashes)
test_in_train  = np.in1d(test_hashes,  train_hashes)
test_in_valid  = np.in1d(test_hashes,  valid_hashes)

valid_keep = ~valid_in_train
test_keep  = ~(test_in_train | test_in_valid)

valid_dataset_clean = valid_dataset[valid_keep]
valid_labels_clean  = valid_labels [valid_keep]

test_dataset_clean = test_dataset[test_keep]
test_labels_clean  = test_labels [test_keep]

t2 = time.time()

print("Time: %0.2fs" % (t2 - t1))
print("valid -> train overlap: %d samples" % valid_in_train.sum())
print("test  -> train overlap: %d samples" % test_in_train.sum())
print("test  -> valid overlap: %d samples" % test_in_valid.sum())
```

Assignment 1 - problem 6 - About the scores from LogisticRegression

Assignment1, Problem5 : Lot of similar images in test and train set

---

Sivakumaran **Feb 2**

@stmax82 , lol, this discussion keeps getting better, elegant and more interesting. Very nice solution 

---

day **Feb 2**

I tried a couple of the approaches mentioned here and compared. Here is the code:

```
import time
def fast_overlaps_num_set_and_hash(images1, images2):
    images1.flags.writeable=False
```

```
        images2.flags.writeable=False
        hash1 = set([hash(image1.data) for image1 in images1])
        hash2 = set([hash(image2.data) for image2 in images2])
        all_overlaps = set.intersection(hash1, hash2)
        return len(all_overlaps)
    def find_dups_and_overlaps(images1, images2):
        images1.flags.writeable=False
        images2.flags.writeable=False
        dup_table={}
        duplicates1 = []
        for idx,img in enumerate(images1):
            h = hash(img.data)
            if h in dup_table and (images1[dup_table[h]].data == img.data):
                duplicates1.append((idx, dup_table[h]))
                #print 'Duplicate image: %d matches %d' % (idx, dup_table[h])
            dup_table[h] = idx
        overlaps = []
        for idx,img in enumerate(images2):
            h = hash(img.data)
            if h in dup_table and (images1[dup_table[h]].data == img.data):
                overlaps.append((dup_table[h], idx))
                #print 'Test image %d is in the training set' % idx
        return duplicates1, overlaps
    def num_overlaps_with_diff_labels(overlap_indices, labels1, labels2):
        count = 0
        for olap in overlap_indices:
            if labels1[olap[0]] != labels2[olap[1]]:
                count += 1
        return count
    def faster_overlaps_hashlib_and_numpy():
        import hashlib
        train_hashes = [hashlib.sha1(x).digest() for x in train_dataset]
    valid_hashes = [hashlib.sha1(x).digest() for x in valid_dataset]
    test_hashes  = [hashlib.sha1(x).digest() for x in test_dataset]

    valid_in_train = np.in1d(valid_hashes, train_hashes)
    test_in_train  = np.in1d(test_hashes,  train_hashes)
    test_in_valid  = np.in1d(test_hashes,  valid_hashes)

    valid_keep = ~valid_in_train
    test_keep  = ~(test_in_train | test_in_valid)

    valid_dataset_clean = valid_dataset[valid_keep]
    valid_labels_clean  = valid_labels [valid_keep]

    test_dataset_clean = test_dataset[test_keep]
    test_labels_clean  = test_labels [test_keep]

    print("valid -> train overlap: %d samples" % valid_in_train.sum())
    print("test  -> train overlap: %d samples" % test_in_train.sum())
    print("test  -> valid overlap: %d samples" % test_in_valid.sum())
    print '\nMethod 1: hash and check equality'
```

```
t1 = time.time()
train_dups, train_valid_overlaps = find_dups_and_overlaps(train_dataset, valid_
test_dups, test_train_overlaps = find_dups_and_overlaps(test_dataset, train_data
valid_dups, valid_test_overlaps = find_dups_and_overlaps(valid_dataset, test_dat
print 'train dups: %s, test_dups: %s, valid_dups: %s' % (len(train_dups), len(te
print 'train/valid overlaps: %s, of which %s have different labels' % \
    (len(train_valid_overlaps), num_overlaps_with_diff_labels(train_valid_overla
print 'test/train overlaps: %s, of which %s have different labels' % \
    (len(test_train_overlaps), num_overlaps_with_diff_labels(test_train_overlaps
print 'valid/test overlaps: %s, of which %s have different labels' % \
    (len(valid_test_overlaps), num_overlaps_with_diff_labels(valid_test_overlaps
t2 = time.time()
print("Time: %0.2fs" % (t2 - t1))
print '\nMethod 2: hash and set'
t1 = time.time()
print 'fast train/validation overlaps: %s ' % fast_overlaps_num_set_and_hash(tra
print 'fast train/test overlaps: %s' % fast_overlaps_num_set_and_hash(train_data
print 'fast test/validation overlaps: %s' % fast_overlaps_num_set_and_hash(test_
t2 = time.time()
print("Time: %0.2fs" % (t2 - t1))
print '\nMethod 3: hashlib and numpy'
t1 = time.time()
faster_overlaps_hashlib_and_numpy()
t2 = time.time()
print("Time: %0.2fs" % (t2 - t1))
```

The results are below:

```
Method 1: hash and check equality
train dups: 12458, test_dups: 208, valid_dups: 170
train/valid overlaps: 1173, of which 12 have different labels
test/train overlaps: 3457, of which 35 have different labels
valid/test overlaps: 196, of which 139 have different labels
Time: 2.27s
Method 2: hash and set
fast train/validation overlaps: 1063
fast train/test overlaps: 1163
fast test/validation overlaps: 58
Time: 2.21s
Method 3: haslib and numpy
valid -> train overlap: 1173 samples
test  -> train overlap: 1324 samples
test  -> valid overlap: 196 samples
Time: 1.81s
```

I guess for a quick manual check, I prefer the simpler first method. Although it isn't the fastest, it is the one that lends itself most readily to checking if the labels are the same (because it saves indices that are the same) and it seems to have caught significantly more overlaps between test and train (unless there is an error in the code and it is reporting too many).

Assigment 1 - Problem5

meener777 **Feb 2**

Nice solutions here.

I notice that none of the posted solutions handle the case of "close" images. Per the question:
What about near duplicates between datasets? (images that are almost identical)

To do this I would think one would have to compute a metric (norm of b-a should be okay) but would have to do so for every single pair, using a threshold to determine what "too close" was. I havent let my code for this finish running yet but it seems like it will take a Long Time (tm).

shrimp_323646 **Feb 3**

To find close images, hashing the images wont help (or maybe doing some locally sensitive hashing (LSH), someone should investigate ...). One solution could be to use the bucket approach and compare the matrices with the `numpy.allclose` function, but the comparison will be local (pixel level), as you said, it might be better to use some global distance metric between the images like the norm of the difference of two images:

```
def d(image1, image2):
    '''Euclidean distance without the sqrt'''
    return np.sum(np.power(image1-image2, 2))
```

If you use the bucket idea with a distance metric like this it shouldn't take too long to compute.

vinayakjaiswal01 **May 26**

Most of the solutions are using hash to find duplicates
But there is a question to find near duplicates

I have used the **ssim** number calculated using function in **scikit-image** package

Here is my soln

```
import time

from skimage.measure import compare_ssim as ssim

def get_similarity(dataset,image):
    return [round(ssim(data,image),4) for data in dataset]    #round is used to

start=time.time()

#first we take two boundary images
white_image=0.5*np.ones((28,28),dtype=np.float32)
black_image=-0.5*np.ones((28,28),dtype=np.float32)

#find similarity with the white image
train_similar_white=get_similarity(train_dataset,white_image)
test_similar_white=get_similarity(test_dataset,white_image)
valid_similar_white=get_similarity(valid_dataset,white_image)

#find similarity with black image
train_similar_black=get_similarity(train_dataset,black_image)
test_similar_black=get_similarity(test_dataset,black_image)
valid_similar_black=get_similarity(valid_dataset,black_image)

#zip the values
train_similarity=set(zip(train_similar_white,train_similar_black))
test_similarity=set(zip(test_similar_white,test_similar_black))
valid_similarity=set(zip(valid_similar_white,valid_similar_black))

print("No. of overlaps between training and test sets : ",len(train_similarity.
print("No. of overlaps between training and validation sets : ",len(train_simila
print("No. of overlaps between validation and test sets : ",len(valid_similarity

print("Execution time : %fs"%(time.time()-start))
def overlap_count(dataset1,dataset2):
    dataset1.flags.writeable=False
    dataset2.flags.writeable=False
    hash1=set((hash(image.data) for image in dataset1))
    hash2=set((hash(image.data) for image in dataset2))
    diff=hash1.intersection(hash2)
    return len(diff)
```

```
start=time.time()

print("No. of overlaps between training and test sets : ",overlap_count(train_da
print("No. of overlaps between training and validation sets : ",overlap_count(t
print("No. of overlaps between validation and test sets : ",overlap_count(valid_

print("Execution time : %fs"%(time.time()-start))
```

Output:

```
No. of overlaps between training and test sets :  6662
No. of overlaps between training and validation sets :  6694
No. of overlaps between validation and test sets :  2649
Execution time : 151.795583s
No. of overlaps between training and test sets :  1174
No. of overlaps between training and validation sets :  1003
No. of overlaps between validation and test sets :  72
Execution time : 2.757114s
```

The second part uses the normal hash as specified by **@shrimp_323646**

Used 0.5 and -0.5 because those were the min and max values in the datasets
Also, using 4 to round was just a way

I know this is not a perfect soln
So suggestions are welcome

---

fr_andres **May 28**

thank you very much!!!!

Zion93 **May 31**

It took about 4 min because you just compared the subdata.

---

nishant_agrawal **Jun 15**
Forum Mentor

A non-sophisticated to find overlap, albeit time consuming:

```
import time
start_time = time.time()

train_dataset_flat=train_dataset.reshape(200000,784)
valid_dataset_flat=valid_dataset.reshape(65536,784)
test_dataset_flat=test_dataset.reshape(18000,784)
```

```
trainset = set([tuple(x) for x in train_dataset_flat])
validset = set([tuple(x) for x in valid_dataset_flat])
testset = set([tuple(x) for x in test_dataset_flat])

train_valid_count=np.array([x for x in trainset & validset]).shape[0]
train_test_count=np.array([x for x in trainset & testset]).shape[0]
print("Time taken to count deferences in time:\n- %s seconds ---" % (time.time(

print("Similars in Train(",train_dataset_flat.shape[0],") and Validation(" , va
print("Similars in Train (",train_dataset_flat.shape[0],") and Test (",test_data
```

LoveMeow **Jun 17**

I am getting 0 duplicates between all datasets, is this normal? I assumed it is since we take the validation and training data from the same data but one after the other, and the test_set from another file? But it seems everybody here has duplicates

```
import hashlib
def checkDuplicates(data1, data2):
data1.flags.writeable = False
data2.flags.writeable = False
duplicates = 0
table = {}
for i in range(0,len(data1),1):
h = hashlib.md5(data1[i,:,:].data)
table[h] = i
for id,image in enumerate(data2):
h = hashlib.md5(image.data)
if (h in table) and (data1[table[h],:,:].data == image.data):
duplicates += 1
print (duplicates)
return duplicates
```

that is code, any help is appreciated!

---

nishant_agrawal **Jun 17**
Forum Mentor

Am afraid, it is not correct; unless you consider only a small amount of data.

---

alexandre.salome **Jun 19**

Correct code for a hash is:

```
h = hashlib.md5()
h.update("a string") # does not work with arrays
```

```
digest = h.hexdigest()
```

As commented above, you cannot pass an array, so what I did is that I pickle'd the array:

```
# ...
h.update(pickle.dumps(my_array))
```

Hope it helps!

Nathanhere **Jun 27**

This is an awesome solution!

POPULAR NANODEGREE PROGRAMS +

STUDENT RESOURCES +

UDACITY +

INQUIRIES +