Deep Learning

Assignment 1

The objective of this assignment is to learn about simple data curation practices, and familiarize you with some of the data we'll be reusing later.

This notebook uses the notMNIST (http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html) dataset to be used with python experiments. This dataset is designed to look like the classic MNIST (http://yann.lecun.com/exdb/mnist/) dataset, while looking a little more like real data: it's a harder task, and the data is a lot less 'clean' than MNIST.

In [2]:
```
# These are all the modules we'll be using later. Make sure you can import them
# before proceeding further.
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
import os
import sys
import tarfile
from IPython.display import display, Image
from scipy import ndimage
from sklearn.linear_model import LogisticRegression
from six.moves.urllib.request import urlretrieve
from six.moves import cPickle as pickle


# Config the matplotlib backend as plotting inline in IPython
%matplotlib inline
```

First, we'll download the dataset to our local machine. The data consists of characters rendered in a variety of fonts on a 28x28 image. The labels are limited to 'A' through 'J' (10 classes). The training set has about 500k and the testset 19000 labelled examples. Given these sizes, it should be possible to train models quickly on any machine.

In [3]:
```
url = 'http://commondatastorage.googleapis.com/books1000/'
last_percent_reported = None
data_root = '.' # Change me to store data elsewhere

def download_progress_hook(count, blockSize, totalSize):
  """A hook to report the progress of a download. This is mostly intended for users with
  slow internet connections. Reports every 5% change in download progress.
  """
  global last_percent_reported
  percent = int(count * blockSize * 100 / totalSize)

  if last_percent_reported != percent:
    if percent % 5 == 0:
      sys.stdout.write("%s%%" % percent)
      sys.stdout.flush()
    else:
      sys.stdout.write(".")
      sys.stdout.flush()

    last_percent_reported = percent

def maybe_download(filename, expected_bytes, force=False):
  """Download a file if not present, and make sure it's the right size."""
  dest_filename = os.path.join(data_root, filename)
  if force or not os.path.exists(dest_filename):
    print('Attempting to download:', filename)
    filename, _ = urlretrieve(url + filename, dest_filename, reporthook=download_progress_hook)
    print('\nDownload Complete!')
  statinfo = os.stat(dest_filename)
  if statinfo.st_size == expected_bytes:
    print('Found and verified', dest_filename)
  else:
    raise Exception(
      'Failed to verify ' + dest_filename + '. Can you get to it with a browser?')
  return dest_filename

train_filename = maybe_download('notMNIST_large.tar.gz', 247336696)
test_filename = maybe_download('notMNIST_small.tar.gz', 8458043)
```

```
Attempting to download: notMNIST_large.tar.gz
0%....5%....10%....15%....20%....25%....30%....35%....40%....45%....50%....55%....60%....65%....70%....75%....80%....
85%....90%....95%....100%
Download Complete!
Found and verified ./notMNIST_large.tar.gz
Attempting to download: notMNIST_small.tar.gz
0%....5%....10%....15%....20%....25%....30%....35%....40%....45%....50%....55%....60%....65%....70%....75%....80%....
85%....90%....95%....100%
Download Complete!
Found and verified ./notMNIST_small.tar.gz
```

Extract the dataset from the compressed .tar.gz file. This should give you a set of directories, labelled A through J.

In [10]:
```
num_classes = 10
np.random.seed(133)

def maybe_extract(filename, force=False):
  root = os.path.splitext(os.path.splitext(filename)[0])[0]  # remove .tar.gz
  if os.path.isdir(root) and not force:
    # You may override by setting force=True.
    print('%s already present - Skipping extraction of %s.' % (root, filename))
  else:
    print('Extracting data for %s. This may take a while. Please wait.' % root)
    tar = tarfile.open(filename)
    sys.stdout.flush()
    tar.extractall(data_root)
    tar.close()
  data_folders = [
    os.path.join(root, d) for d in sorted(os.listdir(root))
    if os.path.isdir(os.path.join(root, d))]
  if len(data_folders) != num_classes:
    raise Exception(
      'Expected %d folders, one per class. Found %d instead.' % (
        num_classes, len(data_folders)))
  print(data_folders)
  return data_folders

train_folders = maybe_extract(train_filename)
test_folders = maybe_extract(test_filename)
```

```
./notMNIST_large already present - Skipping extraction of ./notMNIST_large.tar.gz.
['./notMNIST_large/A', './notMNIST_large/B', './notMNIST_large/C', './notMNIST_large/D', './notMNIST_large/E', './not
MNIST_large/F', './notMNIST_large/G', './notMNIST_large/H', './notMNIST_large/I', './notMNIST_large/J']
./notMNIST_small already present - Skipping extraction of ./notMNIST_small.tar.gz.
['./notMNIST_small/A', './notMNIST_small/B', './notMNIST_small/C', './notMNIST_small/D', './notMNIST_small/E', './not
MNIST_small/F', './notMNIST_small/G', './notMNIST_small/H', './notMNIST_small/I', './notMNIST_small/J']
```

Problem 1

Let's take a peek at some of the data to make sure it looks sensible. Each exemplar should be an image of a character A through J rendered in a different font. Display a sample of the images that we just downloaded. Hint: you can use the package IPython.display.

In [9]:
```
from IPython.display import Image
Image(filename='notMNIST_small/A/QnJ1c2g0NTUgQlQudHRm.png')
Image(filename='notMNIST_small/A/SVRDR2FyYW1vbmRTdGQtQmQub3Rm.png')
```

Out[9]: 

Now let's load the data in a more manageable format. Since, depending on your computer setup you might not be able to fit it all in memory, we'll load each class into a separate dataset, store them on disk and curate them independently. Later we'll merge them into a single dataset of manageable size.

We'll convert the entire dataset into a 3D array (image index, x, y) of floating point values, normalized to have approximately zero mean and standard deviation ~0.5 to make training easier down the road.

A few images might not be readable, we'll just skip them.

In [8]:
```
image_size = 28  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

def load_letter(folder, min_num_images):
  """Load the data for a single letter label."""
  image_files = os.listdir(folder)
  dataset = np.ndarray(shape=(len(image_files), image_size, image_size),
                       dtype=np.float32)
  print(folder)
  num_images = 0
  for image in image_files:
    image_file = os.path.join(folder, image)
    try:
```

```
      image_data = (ndimage.imread(image_file).astype(float) -
                    pixel_depth / 2) / pixel_depth
      if image_data.shape != (image_size, image_size):
        raise Exception('Unexpected image shape: %s' % str(image_data.shape))
      dataset[num_images, :, :] = image_data
      num_images = num_images + 1
    except IOError as e:
      print('Could not read:', image_file, ':', e, '- it\'s ok, skipping.')

  dataset = dataset[0:num_images, :, :]
  if num_images < min_num_images:
    raise Exception('Many fewer images than expected: %d < %d' %
                    (num_images, min_num_images))

  print('Full dataset tensor:', dataset.shape)
  print('Mean:', np.mean(dataset))
  print('Standard deviation:', np.std(dataset))
  return dataset

def maybe_pickle(data_folders, min_num_images_per_class, force=False):
  dataset_names = []
  for folder in data_folders:
    set_filename = folder + '.pickle'
    dataset_names.append(set_filename)
    if os.path.exists(set_filename) and not force:
      # You may override by setting force=True.
      print('%s already present - Skipping pickling.' % set_filename)
    else:
      print('Pickling %s.' % set_filename)
      dataset = load_letter(folder, min_num_images_per_class)
      try:
        with open(set_filename, 'wb') as f:
          pickle.dump(dataset, f, pickle.HIGHEST_PROTOCOL)
      except Exception as e:
        print('Unable to save data to', set_filename, ':', e)

  return dataset_names

train_datasets = maybe_pickle(train_folders, 45000)
test_datasets = maybe_pickle(test_folders, 1800)
```

```
Pickling ./notMNIST_large/A.pickle.
./notMNIST_large/A
Could not read: ./notMNIST_large/A/Um9tYW5hIEJvbGQucGZi.png : cannot identify image file - it's ok, skipping.
Could not read: ./notMNIST_large/A/SG90IE11c3RhcmQgQlROIFBvc3Rlci50dGY=.png : cannot identify image file - it's ok, s
kipping.
Could not read: ./notMNIST_large/A/RnJlaWdodERpc3BCb29rSXRhbGljLnR0Zg==.png : cannot identify image file - it's ok, s
kipping.
Full dataset tensor: (52909, 28, 28)
Mean: -0.128498763375
Standard deviation: 0.425739630876
Pickling ./notMNIST_large/B.pickle.
./notMNIST_large/B
Could not read: ./notMNIST_large/B/TmlraXNFRi1TZW1pQm9sZEl0YWxpYy5vdGY=.png : cannot identify image file - it's ok, s
kipping.
Full dataset tensor: (52911, 28, 28)
Mean: -0.00756217816938
Standard deviation: 0.417319423128
Pickling ./notMNIST_large/C.pickle.
./notMNIST_large/C
Full dataset tensor: (52912, 28, 28)
Mean: -0.142316210531
Standard deviation: 0.42150325882
Pickling ./notMNIST_large/D.pickle.
./notMNIST_large/D
Could not read: ./notMNIST_large/D/VHJhbnNpdCBCb2xkLnR0Zg==.png : cannot identify image file - it's ok, skipping.
Full dataset tensor: (52911, 28, 28)
Mean: -0.057452651285
Standard deviation: 0.434108444886
Pickling ./notMNIST_large/E.pickle.
./notMNIST_large/E
Full dataset tensor: (52912, 28, 28)
Mean: -0.0701376211677
Standard deviation: 0.428995075459
Pickling ./notMNIST_large/F.pickle.
./notMNIST_large/F
Full dataset tensor: (52912, 28, 28)
Mean: -0.125927367659
Standard deviation: 0.429678336457
Pickling ./notMNIST_large/G.pickle.
./notMNIST_large/G
Full dataset tensor: (52912, 28, 28)
```

```
Mean: -0.0947839534684
Standard deviation: 0.421885765538
Pickling ./notMNIST_large/H.pickle.
./notMNIST_large/H
Full dataset tensor: (52912, 28, 28)
Mean: -0.0687865920909
Standard deviation: 0.430602397597
Pickling ./notMNIST_large/I.pickle.
./notMNIST_large/I
Full dataset tensor: (52912, 28, 28)
Mean: 0.0307373563653
Standard deviation: 0.44968647946
Pickling ./notMNIST_large/J.pickle.
./notMNIST_large/J
Full dataset tensor: (52911, 28, 28)
Mean: -0.153450547878
Standard deviation: 0.397230166265
Pickling ./notMNIST_small/A.pickle.
./notMNIST_small/A
Could not read: ./notMNIST_small/A/RGVtb2NyYXRpY2FCb2xkT2xkc3R5bGUgQm9sZC50dGY=.png : cannot identify image file - i
t's ok, skipping.
Full dataset tensor: (1872, 28, 28)
Mean: -0.132588651196
Standard deviation: 0.445976787004
Pickling ./notMNIST_small/B.pickle.
./notMNIST_small/B
Full dataset tensor: (1873, 28, 28)
Mean: 0.00535609548942
Standard deviation: 0.45708097892
Pickling ./notMNIST_small/C.pickle.
./notMNIST_small/C
Full dataset tensor: (1873, 28, 28)
Mean: -0.141488899809
Standard deviation: 0.441032042215
Pickling ./notMNIST_small/D.pickle.
./notMNIST_small/D
Full dataset tensor: (1873, 28, 28)
Mean: -0.049209324555
Standard deviation: 0.460473684846
Pickling ./notMNIST_small/E.pickle.
./notMNIST_small/E
Full dataset tensor: (1873, 28, 28)
```

```
Mean: -0.0598965320662
Standard deviation: 0.456132730513
Pickling ./notMNIST_small/F.pickle.
./notMNIST_small/F
Could not read: ./notMNIST_small/F/Q3Jvc3NvdmVyIEJvbGRPYmxpcXVlLnR0Zg==.png : cannot identify image file - it's ok, s
kipping.
Full dataset tensor: (1872, 28, 28)
Mean: -0.118149849709
Standard deviation: 0.451102384706
Pickling ./notMNIST_small/G.pickle.
./notMNIST_small/G
Full dataset tensor: (1872, 28, 28)
Mean: -0.0925147221098
Standard deviation: 0.448486794208
Pickling ./notMNIST_small/H.pickle.
./notMNIST_small/H
Full dataset tensor: (1872, 28, 28)
Mean: -0.058672809318
Standard deviation: 0.457384289343
Pickling ./notMNIST_small/I.pickle.
./notMNIST_small/I
Full dataset tensor: (1872, 28, 28)
Mean: 0.0526477142339
Standard deviation: 0.472704660845
Pickling ./notMNIST_small/J.pickle.
./notMNIST_small/J
Full dataset tensor: (1872, 28, 28)
Mean: -0.151672725255
Standard deviation: 0.449549033005
```

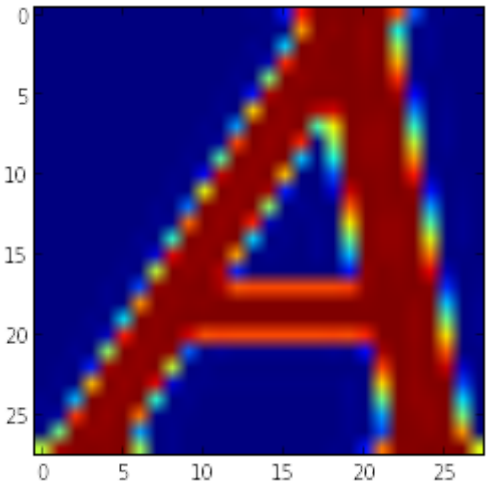
Problem 2

Let's verify that the data still looks good. Displaying a sample of the labels and images from the ndarray. Hint: you can use matplotlib.pyplot.

In [21]:
```
# filename='notMNIST_small/A/QnJ1c2g0NTUgQlQudHRm.png'
print(train_datasets)
print(test_datasets)
dataset = load_letter(folder='notMNIST_small/A', min_num_images=1800)
# print(dataset[2])
plt.imshow(dataset[2])
```

```
['./notMNIST_large/A.pickle', './notMNIST_large/B.pickle', './notMNIST_large/C.pickle', './notMNIST_large/D.pickle',
 './notMNIST_large/E.pickle', './notMNIST_large/F.pickle', './notMNIST_large/G.pickle', './notMNIST_large/H.pickle',
 './notMNIST_large/I.pickle', './notMNIST_large/J.pickle']
['./notMNIST_small/A.pickle', './notMNIST_small/B.pickle', './notMNIST_small/C.pickle', './notMNIST_small/D.pickle',
 './notMNIST_small/E.pickle', './notMNIST_small/F.pickle', './notMNIST_small/G.pickle', './notMNIST_small/H.pickle',
 './notMNIST_small/I.pickle', './notMNIST_small/J.pickle']
notMNIST_small/A
Could not read: notMNIST_small/A/RGVtb2NyYXRpY2FCb2xkT2xkc3R5bGUgQm9sZC50dGY=.png : cannot identify image file - it's
ok, skipping.
Full dataset tensor: (1872, 28, 28)
Mean: -0.132588651196
Standard deviation: 0.445976787004
```

Out[21]: <matplotlib.image.AxesImage at 0xd18a1d0>



Merge and prune the training data as needed. Depending on your computer setup, you might not be able to fit it all in memory, and you can tune `train_size` as needed. The labels will be stored into a separate array of integers 0 through 9.

Also create a validation dataset for hyperparameter tuning.

In [23]:
```
def make_arrays(nb_rows, img_size):
  if nb_rows:
    dataset = np.ndarray((nb_rows, img_size, img_size), dtype=np.float32)
    labels = np.ndarray(nb_rows, dtype=np.int32)
  else:
    dataset, labels = None, None
  return dataset, labels

def merge_datasets(pickle_files, train_size, valid_size=0):
  num_classes = len(pickle_files)
  valid_dataset, valid_labels = make_arrays(valid_size, image_size)
  train_dataset, train_labels = make_arrays(train_size, image_size)
  vsize_per_class = valid_size // num_classes
  tsize_per_class = train_size // num_classes

  start_v, start_t = 0, 0
  end_v, end_t = vsize_per_class, tsize_per_class
  end_l = vsize_per_class+tsize_per_class
  for label, pickle_file in enumerate(pickle_files):
    try:
      with open(pickle_file, 'rb') as f:
        letter_set = pickle.load(f)
        # let's shuffle the letters to have random validation and training set
        np.random.shuffle(letter_set)
        if valid_dataset is not None:
          valid_letter = letter_set[:vsize_per_class, :, :]
          valid_dataset[start_v:end_v, :, :] = valid_letter
          valid_labels[start_v:end_v] = label
          start_v += vsize_per_class
          end_v += vsize_per_class

        train_letter = letter_set[vsize_per_class:end_l, :, :]
        train_dataset[start_t:end_t, :, :] = train_letter
        train_labels[start_t:end_t] = label
```

```
        start_t += tsize_per_class
        end_t += tsize_per_class
    except Exception as e:
      print('Unable to process data from', pickle_file, ':', e)
      raise

  return valid_dataset, valid_labels, train_dataset, train_labels


train_size = 200000
valid_size = 10000
test_size = 10000

valid_dataset, valid_labels, train_dataset, train_labels = merge_datasets(
  train_datasets, train_size, valid_size)
_, _, test_dataset, test_labels = merge_datasets(test_datasets, test_size)

print('Training:', train_dataset.shape, train_labels.shape)
print('Validation:', valid_dataset.shape, valid_labels.shape)
print('Testing:', test_dataset.shape, test_labels.shape)
```

```
Training: (200000, 28, 28) (200000,)
Validation: (10000, 28, 28) (10000,)
Testing: (10000, 28, 28) (10000,)
```

Problem 3

Another check: we expect the data to be balanced across classes. Verify that.

In [25]:
```
from collections import Counter
print(Counter(train_labels))
print(Counter(valid_labels))
print(Counter(test_labels))
```

```
Counter({0: 20000, 1: 20000, 2: 20000, 3: 20000, 4: 20000, 5: 20000, 6: 20000, 7: 20000, 8: 20000, 9: 20000})
Counter({0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000, 5: 1000, 6: 1000, 7: 1000, 8: 1000, 9: 1000})
Counter({0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000, 5: 1000, 6: 1000, 7: 1000, 8: 1000, 9: 1000})
```

Next, we'll randomize the data. It's important to have the labels well shuffled for the training and test distributions to match.

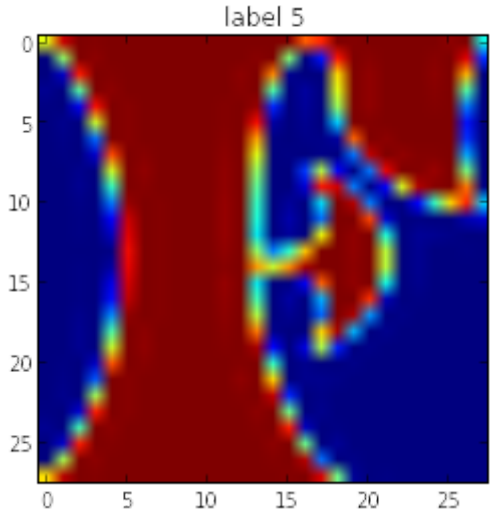
In [0]:
```
def randomize(dataset, labels):
  permutation = np.random.permutation(labels.shape[0])
  shuffled_dataset = dataset[permutation,:,:]
  shuffled_labels = labels[permutation]
  return shuffled_dataset, shuffled_labels
train_dataset, train_labels = randomize(train_dataset, train_labels)
test_dataset, test_labels = randomize(test_dataset, test_labels)
valid_dataset, valid_labels = randomize(valid_dataset, valid_labels)
```

Problem 4

Convince yourself that the data is still good after shuffling!

In [37]:
```
def display_random(dataset, labels):
    index = np.random.randint(0, labels.shape[0])
    plt.imshow(dataset[index])
    plt.title("label " + str(labels[index]))

display_random(train_dataset, train_labels)
display_random(valid_dataset, valid_labels)
display_random(test_dataset, test_labels)
```



Finally, let's save the data for later reuse:

In [35]:
```
pickle_file = os.path.join(data_root, 'notMNIST.pickle')

try:
  f = open(pickle_file, 'wb')
  save = {
    'train_dataset': train_dataset,
    'train_labels': train_labels,
    'valid_dataset': valid_dataset,
    'valid_labels': valid_labels,
    'test_dataset': test_dataset,
    'test_labels': test_labels,
    }
  pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
  f.close()
except Exception as e:
  print('Unable to save data to', pickle_file, ':', e)
  raise
```

In [0]:
```
statinfo = os.stat(pickle_file)
print('Compressed pickle size:', statinfo.st_size)
```

```
Compressed pickle size: 718193801
```

Problem 5

By construction, this dataset might contain a lot of overlapping samples, including training data that's also contained in the validation and test set! Overlap between training and test can skew the results if you expect to use your model in an environment where there is never an overlap, but are actually ok if you expect to see training samples recur when you use it. Measure how much overlap there is between training, validation and test samples.

Optional questions:

- What about near duplicates between datasets? (images that are almost identical)
- Create a sanitized validation and test set, and compare your accuracy on those in subsequent assignments.

In [41]:
```
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]

print("train with overlaps: %s, without overlaps: %s" % (len(train_dataset), len(train_hashes)))
print("valid with overlaps: %s, without overlaps: %s" % (len(valid_dataset), len(valid_hashes)))
print("test with overlaps: %s, without overlaps: %s" % (len(test_dataset), len(test_hashes)))

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]

print("Validata dataset clean: %s, labels clean: %s, dataset: %s, labels: %s" % (len(valid_dataset_clean),
                                                                                 len(valid_labels_clean),
                                                                                 len(valid_dataset),
                                                                                 len(valid_labels)))


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())
```

```
train with overlaps: 200000, without overlaps: 200000
valid with overlaps: 10000, without overlaps: 10000
test with overlaps: 10000, without overlaps: 10000
Validata dataset clean: 8933, labels clean: 8933, dataset: 10000, labels: 10000
Time: 4.32s
valid -> train overlap: 1067 samples
test  -> train overlap: 1284 samples
test  -> valid overlap: 203 samples
```

Problem 6

Let's get an idea of what an off-the-shelf classifier can give you on this data. It's always good to check that there is something to learn, and that it's a problem that is not so trivial that a canned solution solves it.

Train a simple model on this data using 50, 100, 1000 and 5000 training samples. Hint: you can use the LogisticRegression model from sklearn.linear_model.

Optional question: train an off-the-shelf model on all the data!

In [58]:
```
from sklearn.linear_model import LogisticRegressionCV
clf = LogisticRegressionCV(solver='lbfgs', multi_class='multinomial')
from sklearn import cross_validation
from sklearn.metrics import accuracy_score


# 50, 100, 1000, 5000 train samples
# for split in [2.5e-4, 5e-4, 5e-3, 2.5e-2]:
for split in [50, 100, 1000, 5000]:
#for split in [5000]:
    # train_subset, _, train_labels_subset, _ = cross_validation.train_test_split(train_dataset, train_labels, test_si
ze=split, random_state=42)
    # np.random.shuffle(train_dataset)
    #train_subset = [x.flatten() for x in train_dataset[:split]]
    #print(Counter(train_subset))
    #train_labels_subset = train_labels[:split]
    random_index = np.random.randint(0, train_dataset.shape[0], split)
    train_subset = []
    train_labels_subset = []
    for i in random_index:
        train_subset.append(train_dataset[i].flatten())
        train_labels_subset.append(train_labels[i])
    clf.fit(train_subset, train_labels_subset)
    pred = clf.predict([x.flatten() for x in valid_dataset])
    score = accuracy_score(valid_labels, pred)
    print("Valid score: %s" % score)
    pred = clf.predict([x.flatten() for x in test_dataset])
    score = accuracy_score(test_labels, pred)
    print("Test score: %s" % score)
```

```
/usr/lib64/python2.7/site-packages/sklearn/model_selection/_split.py:581: Warning: The least populated class in y has only 2 members, which is too few. The minimum number of groups for any class cannot be less than n_splits=3.
  % (min_groups, self.n_splits)), Warning)
```

```
Valid score: 0.5257
Test score: 0.5691
Valid score: 0.6881
Test score: 0.7593
Valid score: 0.7907
Test score: 0.8661
Valid score: 0.8115
Test score: 0.8794
```