1_notmnist

January 30, 2016

1 Deep Learning

1.1 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 dataset to be used with python experiments. This dataset is designed to look like the classic 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 [33]: # These are all the modules we'll be using later. Make sure you can import them
         # before proceeding further.
         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
```

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://yaroslavvb.com/upload/notMNIST/'

        def maybe_download(filename, expected_bytes):
          """Download a file if not present, and make sure it's the right size."""
          if not os.path.exists(filename):
            filename, _ = urlretrieve(url + filename, filename)
          statinfo = os.stat(filename)
          if statinfo.st_size == expected_bytes:
            print('Found and verified', filename)
          else:
            raise Exception(
              'Failed to verify' + filename + '. Can you get to it with a browser?')
          return filename

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

```
Found and verified notMNIST_large.tar.gz
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 [4]: num_classes = 10

        def extract(filename):
          tar = tarfile.open(filename)
          root = os.path.splitext(os.path.splitext(filename)[0])[0]  # remove .tar.gz
          print('Extracting data for %s. This may take a while. Please wait.' % root)
          sys.stdout.flush()
          tar.extractall()
          tar.close()
          data_folders = [
            os.path.join(root, d) for d in sorted(os.listdir(root)) if d != '.DS_Store']
          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 = extract(train_filename)
        test_folders = extract(test_filename)
```

```
Extracting data for notMNIST_large. This may take a while. Please wait.
['notMNIST_large/A', 'notMNIST_large/B', 'notMNIST_large/C', 'notMNIST_large/D', 'notMNIST_large/E', 'not
Extracting data for notMNIST_small. This may take a while. Please wait.
['notMNIST_small/A', 'notMNIST_small/B', 'notMNIST_small/C', 'notMNIST_small/D', 'notMNIST_small/E', 'not
```

```
In [21]: i = Image(filename='notMNIST_small/A/QXJyaWJhQXJyaWJhU3RkLm90Zg==.png')
         display(i)
```

Now let's load the data in a more manageable format.

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. The labels will be stored into a separate array of integers 0 through 9.

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

```
In [6]: image_size = 28  # Pixel width and height.
        pixel_depth = 255.0  # Number of levels per pixel.
```

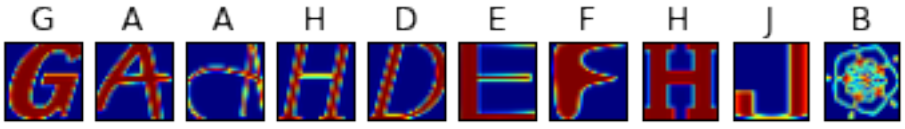
```
        def load(data_folders, min_num_images, max_num_images):
          dataset = np.ndarray(
            shape=(max_num_images, image_size, image_size), dtype=np.float32)
          labels = np.ndarray(shape=(max_num_images), dtype=np.int32)
          label_index = 0
          image_index = 0
          for folder in data_folders:
            print(folder)
            for image in os.listdir(folder):
              if image_index >= max_num_images:
                raise Exception('More images than expected: %d >= %d' % (
                  image_index, max_num_images))
              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[image_index, :, :] = image_data
                labels[image_index] = label_index
                image_index += 1
              except IOError as e:
                print('Could not read:', image_file, ':', e, '- it\'s ok, skipping.')
            label_index += 1
          num_images = image_index
          dataset = dataset[0:num_images, :, :]
          labels = labels[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))
          print('Labels:', labels.shape)
          return dataset, labels
        train_dataset, train_labels = load(train_folders, 450000, 550000)
        test_dataset, test_labels = load(test_folders, 18000, 20000)
```

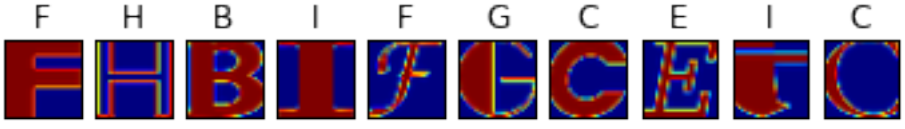
```
notMNIST_large/A
Could not read: notMNIST_large/A/SG90IE11c3RhcmQgQlROIFBvc3Rlci50dGY=.png : cannot identify image file <
Could not read: notMNIST_large/A/RnJlaWdodERpc3BCb29rSXRhbGljLnR0Zg==.png : cannot identify image file <
Could not read: notMNIST_large/A/Um9tYW5hIEJvbGQucGZi.png : cannot identify image file <_io.BufferedRead
notMNIST_large/B
Could not read: notMNIST_large/B/TmlraXNFRi1TZW1pQm9sZEl0YWxpYy5vdGY=.png : cannot identify image file <
notMNIST_large/C
notMNIST_large/D
Could not read: notMNIST_large/D/VHJhbnNpdCBCb2xkLnR0Zg==.png : cannot identify image file <_io.Buffered
notMNIST_large/E
notMNIST_large/F
notMNIST_large/G
notMNIST_large/H
notMNIST_large/I
notMNIST_large/J
Full dataset tensor: (529114, 28, 28)
```

```
Mean: -0.0816596
Standard deviation: 0.454233
Labels: (529114,)
notMNIST_small/A
Could not read: notMNIST_small/A/RGVtb2NyYXRpY2FCb2xkT2xkc3R5bGUgQm9sZC50dGY=.png : cannot identify imag
notMNIST_small/B
notMNIST_small/C
notMNIST_small/D
notMNIST_small/E
notMNIST_small/F
Could not read: notMNIST_small/F/Q3Jvc3NvdmVyIEJvbGRPYmxpcXVlLnR0Zg==.png : cannot identify image file <
notMNIST_small/G
notMNIST_small/H
notMNIST_small/I
notMNIST_small/J
Full dataset tensor: (18724, 28, 28)
Mean: -0.0746363
Standard deviation: 0.458622
Labels: (18724,)
```

```
In [20]: import random
         def showProcessedRandom(dataset,labels,n): # shows size of the sample
             indices=random.sample(range(0,labels.shape[0]),n)
             fig=plt.figure()
             for i in range(n):
                 a=fig.add_subplot(1,n,i+1)
                 plt.imshow(dataset[indices[i],:,:])
                 a.set_title(chr(labels[indices[i]]+ord('A')))
                 a.axes.get_xaxis().set_visible(False)
                 a.axes.get_yaxis().set_visible(False)
             plt.show()

         showProcessedRandom(train_dataset,train_labels,10)
         showProcessedRandom(test_dataset,test_labels,10)
```

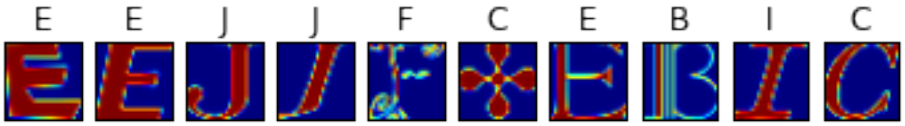


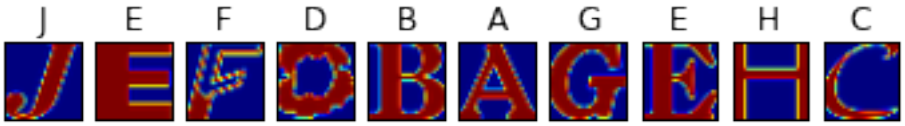


Next, we'll randomize the data. It's important to have the labels well shuffled for the training and test distributions to match.

In [18]:
```
np.random.seed(133)
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)
```

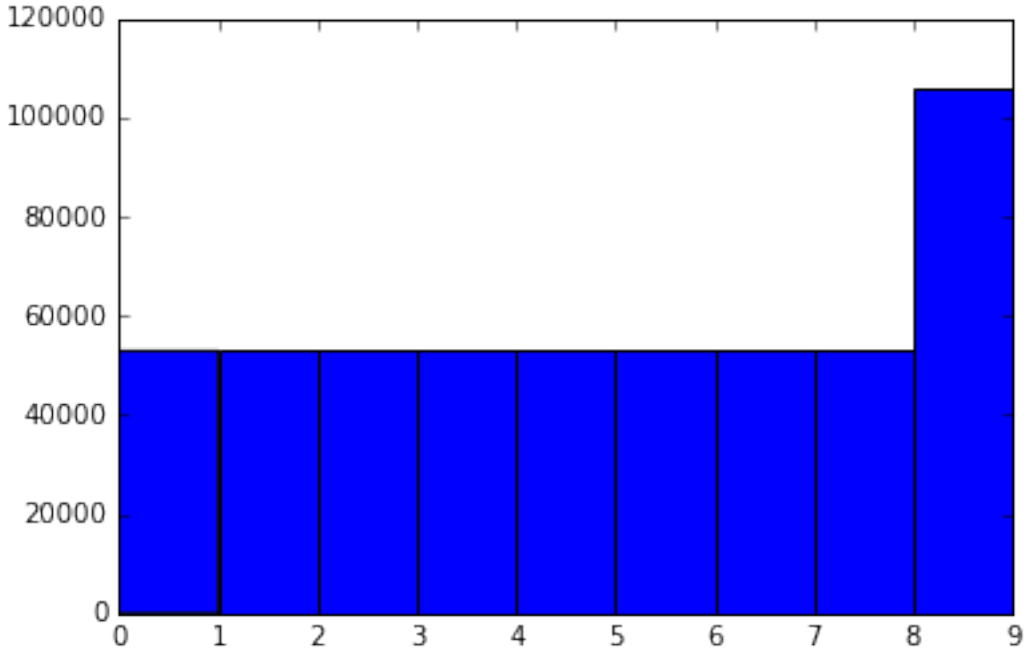
In [19]:
```
showProcessedRandom(train_dataset,train_labels,10)
showProcessedRandom(test_dataset,test_labels,10)
```





In [23]:
```
plt.hist(train_labels, 9)
```

Out[23]:
```
(array([  52909.,   52911.,   52912.,   52911.,   52912.,   52912.,
          52912.,   52912.,  105823.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.]),
 <a list of 9 Patch objects>)
```



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.

Also create a validation dataset for hyperparameter tuning.

```
In [24]: train_size = 200000
         valid_size = 10000

         valid_dataset = train_dataset[:valid_size,:,:]
         valid_labels = train_labels[:valid_size]
         train_dataset = train_dataset[valid_size:valid_size+train_size,:,:]
         train_labels = train_labels[valid_size:valid_size+train_size]
         print('Training', train_dataset.shape, train_labels.shape)
         print('Validation', valid_dataset.shape, valid_labels.shape)
```

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

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

```
In [25]: pickle_file = '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 [26]: statinfo = os.stat(pickle_file)
         print('Compressed pickle size:', statinfo.st_size)
```

```
Compressed pickle size: 718193881
```

```
In [ ]:
```

```
In [47]: from sklearn.metrics import classification_report, zero_one_loss
         train_sizes = [50, 5000, 100000]

         for N in train_sizes:
             print ('\nTraining on %d examples' % N)

             model = LogisticRegression(random_state=413, multi_class='multinomial', solver='newton-cg'

             model.fit(train_dataset[:N].reshape((N, -1)), train_labels[:N])
```

```
            pred = model.predict(test_dataset.reshape((test_dataset.shape[0], -1)))

            error_rate = zero_one_loss(test_labels, pred)
            print ('Error rate: %.2f%%' % (error_rate*100.0))

Training on 50 examples
Error rate: 37.52%

Training on 5000 examples
Error rate: 15.69%

Training on 100000 examples

/usr/local/lib/python3.4/dist-packages/numpy/core/fromnumeric.py:2645: VisibleDeprecationWarning: 'rank
  VisibleDeprecationWarning)


        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-47-6d135021adcc> in <module>()
          7     model = LogisticRegression(random_state=413, multi_class='multinomial', solver='newton-c
          8
    ----> 9     model.fit(train_dataset[:N].reshape((N, -1)), train_labels[:N])
         10
         11     pred = model.predict(test_dataset.reshape((test_dataset.shape[0], -1)))


        /usr/local/lib/python3.4/dist-packages/sklearn/linear_model/logistic.py in fit(self, X, y, sampl
       1205                         max_squared_sum=max_squared_sum,
       1206                         sample_weight=sample_weight)
    -> 1207             for (class_, warm_start_coef_) in zip(classes_, warm_start_coef))
       1208
       1209         fold_coefs_, _, n_iter_ = zip(*fold_coefs_)


        /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in __call__(self, ite
        802             self._iterating = True
        803
    --> 804             while self.dispatch_one_batch(iterator):
        805                 pass
        806


        /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in dispatch_one_batc
        660                 return False
        661             else:
    --> 662                 self._dispatch(tasks)
        663                 return True
        664


        /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in _dispatch(self, b
        568
```

```
    569         if self._pool is None:
--> 570             job = ImmediateComputeBatch(batch)
    571             self._jobs.append(job)
    572             self.n_dispatched_batches += 1


    /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in __init__(self, bat
    181         # Don't delay the application, to avoid keeping the input
    182         # arguments in memory
--> 183         self.results = batch()
    184
    185     def get(self):


    /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in __call__(self)
     70
     71     def __call__(self):
---> 72         return [func(*args, **kwargs) for func, args, kwargs in self.items]
     73
     74     def __len__(self):


    /usr/local/lib/python3.4/dist-packages/sklearn/externals/joblib/parallel.py in <listcomp>(.0)
     70
     71     def __call__(self):
---> 72         return [func(*args, **kwargs) for func, args, kwargs in self.items]
     73
     74     def __len__(self):


    /usr/local/lib/python3.4/dist-packages/sklearn/linear_model/logistic.py in logistic_regression_p
    708             args = (X, target, 1. / C, sample_weight)
    709             w0, n_iter_i = newton_cg(hess, func, grad, w0, args=args,
--> 710                                      maxiter=max_iter, tol=tol)
    711         elif solver == 'liblinear':
    712             coef_, intercept_, n_iter_i, = _fit_liblinear(


    /usr/local/lib/python3.4/dist-packages/sklearn/utils/optimize.py in newton_cg(grad_hess, func, g
    181         # Inner loop: solve the Newton update by conjugate gradient, to
    182         # avoid inverting the Hessian
--> 183         xsupi = _cg(fhess_p, fgrad, maxiter=maxinner, tol=termcond)
    184
    185         alphak = 1.0


    /usr/local/lib/python3.4/dist-packages/sklearn/utils/optimize.py in _cg(fhess_p, fgrad, maxiter,
     86             break
     87
---> 88         Ap = fhess_p(psupi)
     89         # check curvature
     90         curv = np.dot(psupi, Ap)
```

```
    /usr/local/lib/python3.4/dist-packages/sklearn/linear_model/logistic.py in hessp(v)
    384         # r_yhat holds the result of applying the R-operator on the multinomial
    385         # estimator.
--> 386         r_yhat = safe_sparse_dot(X, v.T)
    387         r_yhat += inter_terms
    388         r_yhat += (-p * r_yhat).sum(axis=1)[:, np.newaxis]


    /usr/local/lib/python3.4/dist-packages/sklearn/utils/extmath.py in safe_sparse_dot(a, b, dense_o
    182         return ret
    183     else:
--> 184         return fast_dot(a, b)
    185
    186


    KeyboardInterrupt:
```

In [ ]: