In [1]:
```
import os
import struct
import numpy as np

import csv
import random
import math
import operator


def read(dataset = "training", path = "."):
    if dataset is "training":
        fname_img = os.path.join(path, 'train-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels.idx1-ubyte')
    elif dataset is "testing":
        fname_img = os.path.join(path, 't10k-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels.idx1-ubyte')
    else:
        raise ValueError, "dataset must be 'testing' or 'training'"

    # Load everything in some numpy arrays
    with open(fname_lbl, 'rb') as flbl:
        magic, num = struct.unpack(">II", flbl.read(8))
        lbl = np.fromfile(flbl, dtype=np.int8)

    with open(fname_img, 'rb') as fimg:
        magic, num, rows, cols = struct.unpack(">IIII", fimg.read(16))
        img = np.fromfile(fimg, dtype=np.uint8).reshape(len(lbl), rows, cols)

    get_img = lambda idx: (lbl[idx], img[idx])

    # Create an iterator which returns each image in turn
    for i in xrange(len(lbl)):
        yield get_img(i)

def show(image):
    from matplotlib import pyplot
    import matplotlib as mpl
    fig = pyplot.figure()
    ax = fig.add_subplot(1,1,1)
    imgplot = ax.imshow(image, cmap=mpl.cm.Greys)
    imgplot.set_interpolation('nearest')
    ax.xaxis.set_ticks_position('top')
    ax.yaxis.set_ticks_position('left')
    pyplot.show()
```

In [2]:
```
training_data = list(read(dataset='training', path='.'))
testing_data = list(read(dataset='testing', path='.'))

print len(training_data)
print len(testing_data)
```

```
60000
10000
```

In [3]:
```
tr_dt = np.zeros(shape=(60000,784))
tr_lbl = np.zeros(shape=(60000,1))
ts_dt = np.zeros(shape=(10000,784))
ts_lbl = np.zeros(shape=(10000,1))
```

In [4]:
```
for i in xrange(len(training_data)):
    label, pixels = training_data[i]
    tr_dt[i,:]  = pixels.reshape((1,784))
    tr_lbl[i,:] = label
```

In [5]:
```
for i in xrange(len(testing_data)):
    label, pixels = testing_data[i]
    ts_dt[i,:]  = pixels.reshape((1,784))
    ts_lbl[i,:] = label
```

In [6]:
```
import random

num_of_samples_for_training = 1500
num_of_samples_for_testing  = 250


indices_train = random.sample(range(0, 59999), num_of_samples_for_training)
traindata   = tr_dt[indices_train,:]
trainlabel = tr_lbl[indices_train,:]


indices_test = random.sample(range(0, 9999), num_of_samples_for_testing)
testdata   = ts_dt[indices_test,:]
testlabel = ts_lbl[indices_test,:]
```

In [7]:
```
print 'trainset   = ', traindata.shape
print 'testset    = ', testdata.shape
print 'trainlabel = ', trainlabel.shape
print 'testlabel  = ', testlabel.shape
```

```
trainset   =  (1500L, 784L)
testset    =  (250L, 784L)
trainlabel =  (1500L, 1L)
testlabel  =  (250L, 1L)
```

In [8]:
```
trainingSet = np.concatenate((traindata, trainlabel), axis=1)
testSet = np.concatenate((testdata, testlabel), axis=1)
print trainingSet.shape
print testSet.shape
```

```
(1500L, 785L)
(250L, 785L)
```

In [12]:
```
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)
```

In [11]:
```
from matplotlib import pyplot
import matplotlib as mpl
def show(image):
    fig = pyplot.figure()
    ax = fig.add_subplot(1,1,1)
    imgplot = ax.imshow(image, cmap=mpl.cm.Greys)
    imgplot.set_interpolation('nearest')
    ax.xaxis.set_ticks_position('top')
    ax.yaxis.set_ticks_position('left')
    pyplot.show()
```

In [13]:
```
import operator
def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

In [14]:
```
def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reve
    return sortedVotes[0][0]
```

In [15]:
```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
```

In [16]:
```
def main():
    predictions=[]
    k = 3
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        #print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1])
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: ' + repr(accuracy) + '%')
```

In [17]:
```
main()
```

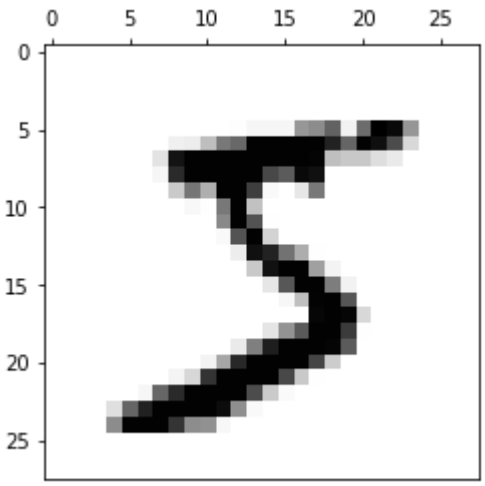
```
Accuracy: 90.0%
```

In [18]:
```
from sklearn import neighbors

knn = neighbors.KNeighborsClassifier(n_neighbors=3)

print('KNN score: %f' % knn.fit(traindata, trainlabel.ravel()).score(testdata, te
```

```
KNN score: 0.888000
```

In [12]:
```
label, pixels = training_data[0]
print(label)
print(pixels.shape)
show(pixels)
```

```
5
(28L, 28L)
```




In [ ]:
```

```