CPSC-340 Machine Learning and Data Mining 2009

Homework # 4

Due Wednesday, Dec 2 1pm.

NAME:_____

Signature:_____

STD. NUM: _____

General guidelines for homeworks:

You are encouraged to discuss the problems with others in the class, but all write-ups are to be done on your own.

Homework grades will be based not only on getting the "correct answer," but also on good writing style and clear presentation of your solution. It is your responsibility to make sure that the graders can easily follow your line of reasoning.

Try every problem. Even if you can't solve the problem, you will receive partial credit for explaining why you got stuck on a promising line of attack. More importantly, you will get valuable feedback that will help you learn the material.

Please acknowledge the people with whom you discussed the problems and what sources you used to help you solve the problem (e.g. books from the library). This won't affect your grade but is important as academic honesty.

When dealing with python exercises, please attach a printout with all your code and show your results clearly.

Figure 1: 3s and 8s from the MNIST handwritten digit data set.

1. **L2 boosting for the MNIST data**

   Using the MNIST data set, we will use L2 regression to train a linear least squares model for distinguishing '3's from '8's.

   First, we will select just the 3s and 8 from the PyTables MNIST database. If you haven't yet gone through the "Dealing With Data in Python" tutorial, you will first need to build the PyTables database for MNIST as described in the final section. It is assumed that that you created the PyTables database from MNIST as described there.

```
from numpy import *
from tables import *

# load data.  you may need to alter this if your DB is in a
# different directory
try:
    h5f = openFile('mnist.h5', 'r')
    X  = []
    Y = []
    for row in h5f.root.data.train.where('(label==3) | (label==8)'):
        # interpret images as vectors
        X.append(row['image'].reshape(-1) / 255)
        # label is 1 for digit '8' and -1 for digit '3'
        Y.append(1 if row['label']==8 else -1)
finally:
    # make sure we close the file even if something
    # goes wrong
    h5f.close()

# convert from lists to arrays
```

```
X = array(X)
Y = array(Y)
```

Use L2 boosting linear regression to learn a model to distinguish 3s from 8s. Use with 100 learners of the form $\{j, \beta\}$, where j is the pixel and β is the learner weight. It is not necessary to fully optimize over β – a fast approximation involving a few different values is sufficient.

Store the learners at each iteration, and generate an image that shows which pixels were selected. Pixels that are indicative of 8s should be brighter, 3s darker, and unselected pixels medium-grey.

```
??? setup ???
F = zeros(Y.shape, dtype=float32])
js = []
betas = []
for _ in xrange(100):

    ??? train base learner and update F ???

    js.append(???)
    betas.append(???)

selected = zeros(28**2) + .5
selected[js] += array(betas)
imshow(selected.reshape(28, 28))
gray()
show()
```

Hand in your code and the image generated by boosting. In one or two sentences, explain why the image looks like it does.

2. **Unsupervised learning [Optional exercise - recommended for final]**

   (i) 2D clustering:

   For this exercise, we will see how clustering evolves in a 2-dimensional domain.

   Using the algorithms presented in class, implement functions for k-means and EM. Both functions should return a list of the means at each time step.

```
def kmeans(X, k=3, steps=8):

    NX, NA = X.shape
    means = ??? initial means ???
    mhist = [means.copy()]
    for t in xrange(steps):
        ???

        mhist.append(means.copy())
    return mhist


def EM(X, k=3, steps=8):

    NX, NA = X.shape

    # initialize xi, p(c), sigma, mu
    xi = ???
    pc = ???
    sigma = ???
    mu = ???
    mhist = [mu.copy()]
    for t in xrange(steps):
        ???

        mhist.append(mu.copy())
    return mhist
```

   Now we'll use NumPy's sampling methods to generate some random data.

```
from numpy.random import *

# distribution of clusters -- draw points from 2nd cluster
# half the time
pc = array([.3, .5, .2])

# cluster means
```

```
mu = array([[.2, .2], [.4,.8], [.8, .2]])

# cluster variances
sigma = array([[.07, .07], [.1,.1], [.05, .1]])
X = []
for i, nsamp in enumerate(multinomial(100, pc)):
    for _ in xrange(nsamp):
        while True:
            # sample from Gaussians, and reject if outside
            # a 0-1 box
            x = [normal(mu[i,0], sigma[i,0]),
                 normal(mu[i,1], sigma[i,1])]
            if 0 < x[0] < 1 and 0 < x[1] < 1:
                break
        X.append(array(x))
X = array(X)
```

You can then plot the mean evolution with a series of subplots:

```
for i, cluster in enumerate([kmeans, EM]):
    mhist = cluster(X)
    figure(i+1)
    for m in xrange(9):
        subplot(3,3,m+1)
        plot(X[:,0], X[:,1], 'ko')
        plot(mhist[m][:,0], mhist[m][:,1], 'ro')
```

Submit your plots for k-means and EM. Try initializing EM's `mu` and `sigma` differently and rerunning. What do you observe?

(i) Country clustering:

Now, we'll cluster a set of countries according to their similarity.

The web page for the book *The Elements of Statistical Learning* has a number of data sets, including one for country similarity, based on a 1990 survey of political science students. Download the 'Countries' data file from `http://www-stat.stanford.edu/ tibs/ElemStatLearn/` (under 'Data'), and save as `countries.txt`. You can load the data using the NumPy `fromstring()` function:

```
X = array([fromstring(line, sep=' ') for line in \
       open('countries.txt').readlines() if len(line) > 1])
```

Modify your EM and k-means functions so that instead of returning the history of means, they return a list, in which element n is the cluster number for X_n. Then for each of EM and k-means, using 5 clusters, show which countries are assigned to which clusters. The code will probably look something like this:

```
k = 5
cnames = "BEL,BRA,CHI,CUB,EGY,FRA,IND,ISR,USA,USS,YUG,ZAI".split(',')
emclust = EM(X, k=5)
for c in xrange(k):
    print 'EM cluster %d: ' % c,
    print ', '.join([cnames[i] for i in xrange(len(cnames)) if clust[i]==c])

kmclust = ...
```

Hand in the cluster assignments for EM and k-means.

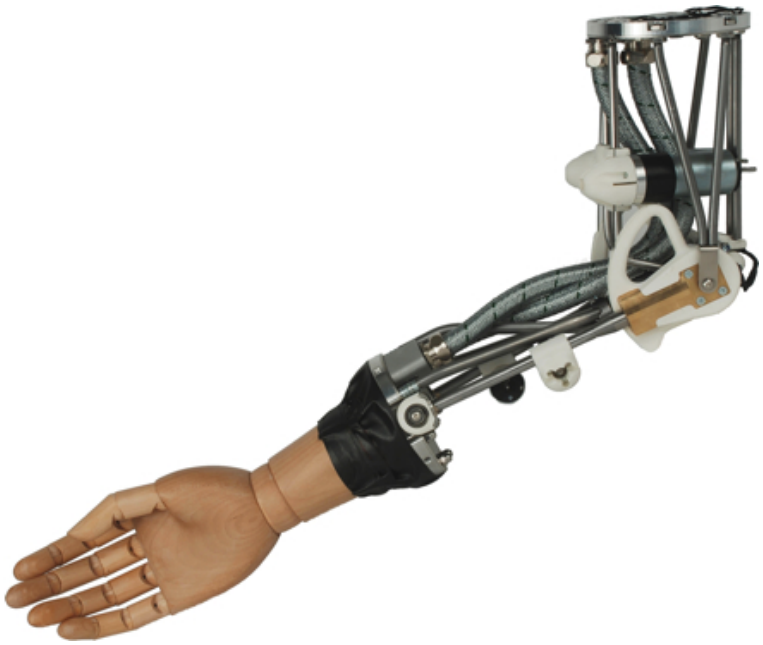

Figure 2: The ISELLA robotic arm.

3. Neural networks

Kinematics (and *inverse kinematics*) are an important part of robotics. Given a set of joint angles in a robot arm, for example, what is the position of the end of the arm? What joint angles are required to position an arm so that it can reach a given point? These are critical problems, and they are not at all easy to compute rigorously, since they are highly nonlinear. As human beings, however, we solve approximations to these problems constantly. We don't need to invert matrices or perform tensor products to answer a phone or open a door. We know how to do these because we have learned models of the relationships between motor functions and outcomes. When we hit the same problems in AI and robotics, we can sometimes train machines to learn the relationships as well.

We will use neural networks to learn the relationship between a robot's joint angles (the inputs x_1, x_2) and the positions of the end of the arm (the targets y_1, y_2). To train, download the robot data from `http://www.inference.phy.cam.ac.uk/mackay/Bayes_FAQ.html#Data`. There are two files: `Cinputs5` and `Ctargets5`.

To load the ASCII data as NumPy arrays, you can use the `fromstring()` function:

```
inputs = array([fromstring(line, sep='\t') for line in \
           open('Cinputs5').readlines()])
targets = array([fromstring(line, sep='\t') for line in \
           open('Ctargets5').readlines()])
```

Using whatever coding framework you like, implement a system to train and test a neural network and train on the `inputs`, `target` pair.

Using the learned model, create meshes to visualize the two outputs. Assuming your test function looks something like `testNN(x1, x2, model)` and returns `y`, we can generate plots with this code (adapting as necessary for your implementation):

```
x1 = arange(-2, 2, .05)
x2 = arange(0, 4, .05)
X1, X2 = meshgrid(x1, x2)
Y1 = zeros(X1.shape)
Y2 = zeros(X1.shape)
nx1, nx2 = X1.shape
for i in xrange(nx1):
    for j in xrange(nx2):
        Y1[i,j] = testNN(X1[i,j], X2[i,j], w1)
        Y2[i,j] = testNN(X1[i,j], X2[i,j], w2)

for i, Y in enumerate([Y1, Y2]):
    ax = subplot(1,2,i+1)
    cs = ax.contour(X1, X2, Y)
    clabel(cs)
    ax.set_title('Y%d'%(i+1))
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.axis([x1[0], x1[-1], x2[0], x2[-1]])

show()
```

(a) What are the train and test errors?

(b) Repeat the experiment by adding a ridge regularizer. State the value of the regularizer you chose (and why you chose it) and the train and test errors.

Hand in your code and the contour plots for both cases.