## Homework 1, due Wednesday September 21 COMS 4771 Fall 2016

**Problem 1** (Nearest neighbors; 20 points)**.** Download the OCR image data set `ocr.mat` from Courseworks, and load it into MATLAB:

```
load('ocr.mat')
```

Or Python:

```
from scipy.io import loadmat
ocr = loadmat('ocr.mat')
```

The unlabeled training data (i.e., feature vectors) are contained in a matrix called `data` (one point per row), and the corresponding labels are in a vector called `labels`. The test feature vectors and labels are in, respectively, `testdata` and `testlabels`. In MATLAB, you can view an image (say, the first one) in the training data with the following commands:

```
imagesc(reshape(data(1,:),28,28)');
```

If the colors are too jarring for you, try the following:

```
colormap(1-gray);
```

In Python, to view the first image, try the following (ideally, from IPython or Jupyter Notebook):

```
import matplotlib.pyplot as plt
from matplotlib import cm
plt.imshow(ocr['data'][0].reshape((28,28)), cmap=cm.gray_r)
plt.show()
```

Write a function that implements the 1-nearest neighbor classifier with Euclidean distance. Your function should take as input a matrix of training feature vectors `X` and a vector of the corresponding labels `Y`, as well as a matrix of test feature vectors `test`. The output should be a vector of predicted labels `preds` for all the test points. Naturally, you should not use (or look at the source code for) any library functions for computing pairwise distsances or nearest neighbor queries. If in doubt about what is okay to use, just ask. Note that for efficiency, you should use matrix/vector operations (rather than, say, a bunch of for-loops).[1]

Instead of using your 1-NN code directly with `data` and `labels` as the training data, do the following. For each value $n \in \{1000, 2000, 4000, 8000\}$,

- Draw $n$ random points from `data`, together with their corresponding labels. In MATLAB, use `sel = randsample(60000,`$n$`)` to pick the $n$ random indices, and `data(sel,:)` and `labels(sel)` to select the examples; in Python, use `sel = random.sample(xrange(60000),`$n$`)` (after `import random`), `ocr['data'][sel].astype('float')`, and `ocr['labels'][sel]`.
- Use these $n$ points as the training data and `testdata` as the test points, and compute the test error rate of the 1-NN classifier.

A plot of the error rate (on the y-axis) as a function of $n$ (on the x-axis) is called a *learning curve*. We get an estimate of this curve by using the test error rate in place of the (true) error rate.

Repeat the (random) process described above ten times, independently. Produce an estimate of the learning curve plot using the average of these test error rates (that is, averaging over ten repetitions). Add error bars to your plot that extend to one standard deviation above and below the means. Ensure the plot axes are properly labeled.

What to submit: (1) learning curve plot, (2) source code (in a separate file).

[1] `http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html`

**Problem 2** (Prototype selection; 20 points)**.** Prototype selection is a method for speeding-up nearest neighbor search that replaces the training data with a smaller subset of *prototypes* (which could be data points themselves). For simplicity, assume that 1-NN is used with Euclidean distance. So a prototype selection method simply takes as input:

- the training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ from $\mathbb{R}^d \times \{0, 1, \ldots, 9\}$ (say), and
- a positive integer $m$;

it should return $m$ labeled pairs $\{(\tilde{\boldsymbol{x}}_i, \tilde{y}_i)\}_{i=1}^m$, each from $\mathbb{R}^d \times \{0, 1, \ldots, 9\}$.

Design a method for choosing prototypes, where the goal is for the 1-NN classifier based on the prototypes to have good test accuracy. Implement your algorithm; use it to select prototypes for the OCR data set, and evaluate the test error rate of the 1-NN classifier based on the selected prototypes. You should use the whole training data set as input (i.e., all $n = 60000$ data points), but vary the number of selected prototypes $m$ in the set $\{1000, 2000, 4000, 8000\}$. If your procedure is randomized, repeat it at least ten times (for each $m$) to properly assess its performance.

What to submit:

1. A brief description of your method (in words).
2. Concise and unambiguous pseudocode for your algorithm.
3. A table of the test error rates for the different values of $m$ you try. (Report averages and standard deviations if your procedure is randomized.)
4. Source code (in a separate file).

**Problem 3** (Probability; 10 points)**.** Suppose you have an urn containing 100 colored balls. Each ball is painted with one of five possible colors from the color set $\mathcal{C} := \{\text{red}, \text{orange}, \text{yellow}, \text{green}, \text{blue}\}$. For each $c \in \mathcal{C}$, let $n_c$ denote the number of balls in the urn with color $c$.

(a) Suppose you pick two balls uniformly at random *with replacement* from the urn. What is the probability that they have different colors? Briefly explain your answer.

(b) Suppose you want to maximize the probability from part (a). You are allowed to paint all of the balls in the urn, each with any color from $\mathcal{C}$. How many balls in the urn would you paint with each color in $\mathcal{C}$? Briefly explain your answer.