Homework 1, due Tuesday February 16

COMS 4721 Spring 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 Euclidean distances, nearest neighbor queries, and so on. If in doubt about what is okay to use, just ask.

For efficiency, you should use vector operations (rather than, say, a bunch of for-loops). See `http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html` to learn how to do this in MATLAB.

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]`, 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.

Since the above process involves some randomness, you should repeat it independently several times (say, at least ten times). Produce an estimate of the learning curve plot using the average of these test error rates (that is, averaging over the repeated trials). Add error bars to your plot that extend to one standard deviation above and below the means. Ensure the plot axes are properly labeled. Submit the plot and your source code.

Problem 2 (Naïve Bayes; 20 points)**.** Download the "20 Newsgroups data set" `news.mat` from Courseworks. The training feature vectors/labels and test feature vectors/labels are stored as `data`/`labels` and `testdata`/`testlabels`. Each data point corresponds to a message posted to one of 20 different newsgroups (i.e., message boards). The representation of a message is a (sparse) binary vector in $\mathcal{X} := \{0,1\}^d$ (for $d := 61188$) that indicates the words that are present in the message. If the j-th entry in the vector is 1, it means the message contains the word that is given on the j-th line of the text file `news.vocab`. The class labels are $\mathcal{Y} := \{1, 2, \ldots, 20\}$, where the mapping from classes to newsgroups is in the file `news.groups` (which we won't actually need).

In this problem, you'll develop a classifier based on a Naïve Bayes generative model. Here, we use class conditional distributions of the form

$$P_{\boldsymbol{\mu}}(\boldsymbol{x}) \;=\; \prod_{j=1}^{d} \mu_j^{x_j}(1-\mu_j)^{1-x_j} \quad \text{for } \boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathcal{X}$$

for some parameter vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_d) \in [0,1]^d$. Since there are 20 classes, the generative model is actually parameterized by 20 such vectors, $\boldsymbol{\mu}_y = (\mu_{y,1}, \mu_{y,2}, \ldots, \mu_{y,d})$ for each $y \in \mathcal{Y}$, as well as the class prior parameters, π_y for each $y \in \mathcal{Y}$. The class prior parameters, of course, must satisfy $\pi_y \in [0,1]$ for each $y \in \mathcal{Y}$ and $\sum_{y\in\mathcal{Y}} \pi_y = 1$.

(a) Give the formula for the MLE of the parameter $\mu_{y,j}$ based on training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. (Remember, each unlabeled point is a vector: $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}) \in \{0,1\}^d$.)

(b) It turns out the MLE is not a good estimator for the class conditional parameters, especially if the estimate turns out to be zero or one. An alternative to the MLE is the following estimator based on a technique called *Laplace smoothing*:

$$\hat{\mu}_{y,j} \;:=\; \frac{1 + \sum_{i=1}^n \mathbb{1}\{y_i = y\} x_{i,j}}{2 + \sum_{i=1}^n \mathbb{1}\{y_i = y\}}$$

This ensures that $\hat{\mu}_{y,j} \in (0,1)$—never zero nor one.

Write a function that takes as input a (sparse) matrix of training feature vectors `X` and a vector of labels `Y` (as `data` and `labels` above), and returns the parameters `params` of the classifier based on this Naïve Bayes generative model. Use the Laplace smoothing estimator for the class conditional distribution parameters, and use MLE for the class prior parameters. Naturally, you should not use or look at any existing implementation (e.g., such as those that may be provided as library functions).

Also, write a function that takes as input the parameters of the above classifier `params`, and a matrix of test feature vectors `test`. The function should output a vector of predictions `preds` for all test feature vectors.

Train and evaluate a classifier using this code and the data from `news.mat`. Submit your source code, and report the training and test error rates in your write-up. Save the learned classifier for part (c)!

(c) The classifier you learn should have the following form: $\boldsymbol{x} \mapsto \arg\max_{y\in\mathcal{Y}} \alpha_{y,0} + \sum_{j=1}^d \alpha_{y,j} x_j$ for some real numbers $\alpha_{y,0}, \alpha_{y,1}, \ldots, \alpha_{y,d}$ for each $y \in \mathcal{Y}$. Compute the values of these $\alpha_{y,j}$'s from (the parameters of) your learned classifier from part (b).

For each class $y \in \mathcal{Y}$, report the vocabulary words whose indices $j \in \{1, 2, \ldots, d\}$ correspond to the 20 largest (i.e., most positive) $\alpha_{y,j}$ value. Don't report the j's, but rather the actual vocabulary words (from `news.vocab`).

Problem 3 (Cost-sensitive classification; 10 points)**.** Suppose you face a binary classification problem with input space $\mathcal{X} = \mathbb{R}$ and output space $\mathcal{Y} = \{0, 1\}$, where it is c times as bad to commit a "false positive" as it is to commit a "false negative" (for some real number $c \geq 1$). To make this concrete, let's say that if your classifier predicts 1 but the correct label is 0, you incur a penalty of $\$c$; if your classifier predicts 0 but the correct label is 1, you incur a penalty of \$1. (And you incur no penalty if your classifier predicts the correct label.)

Assume the distribution you care about has a class prior with $\pi_0 = 2/3$ and $\pi_1 = 1/3$, and the class conditional densities are $\mathrm{N}(0, 1)$ for class 0, and $\mathrm{N}(1, 1/4)$ for class 1. Let $f^\star \colon \mathbb{R} \to \{0, 1\}$ be the classifier with the smallest expected penalty.

(a) Assume $1 \leq c \leq 1.5$. Specify precisely (and with a simple expression involving c) the region in which the classifier $f^\star$ predicts 1.

(b) Now instead assume $c \geq 10$. Specify precisely the region in which the classifier $f^\star$ predicts 1.

Problem 4 (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? Explain your answer succinctly (but precisely).

(b) If you could paint each ball in the urn with any color (from $\mathcal{C}$), what would you do to maximize the probability from part (a)? Explain your answer succinctly (but precisely).