Lab 3

CS 3793/5233 – Fall 2013 assigned October 9, 2013
Tom Bylander, Instructor due midnight, October 28, 2013

In Lab 3, you will implement a program to recognize handwritten digits. Your grade on the lab will depend on the behavior and accuracy of the program that is implemented. The initial program is lab3.zip, which you can download from the course web site.

NIST Environment

The NIST training dataset at `http://yann.lecun.com/exdb/mnist/` contains $60,000$ images of handwritten digits and their labels. Each image is 28×28 in grayscale. For reasons of time and space efficiency in this lab, a crude 8×8 bitmap was derived from each image.

The `Nist` program repeatedly does the following steps. It outputs a randomly selected (without repetition) bitmap as a string of 64 ones and zeroes. It then inputs a prediction, a single digit on a line by itself. After the prediction, it outputs whether the prediction is correct or incorrect, the correct label, and a summary of the error rate so far.

Running the main method in `Nist.java` will display a picture of the bitmap for you to classify.

Agent

Your task is to finish `Learner.java` so it will perform *online learning*, that is, its accuracy will improve as additional examples are presented to it. The suggested algorithm is naive Bayes.

The initial code will read in all the examples when `Interact.java` is run, but it always predicts 7. The main method in `Learner.java` will run the code for a small sample of clean bitmaps.

Naive Bayes

In the `Nist` environment, there are 10 possible outputs, the digits 0 through 9. Each example image is 64 bits. Your program should count:

- n: how many examples have been processed.
- n_d: how many examples for each digit d.
- $n_{d,i}$ how many examples for each digit d where bit i is equal to 1.

This is a total of 651 counts.

Naive Bayes is naive because it corresponds to a Bayesian network in which the target variable (the digits in this lab) is the parent of the input variables (the 64 bits in this lab) with no other edges. It computes $P(Y \mid X)$ (where Y is a digit and $X = X_1, \ldots, X_{64}$ are the 64 bits) as:

$$P(Y \mid X) = \frac{P(Y \wedge X)}{P(X)} = \frac{P(Y)P(X \mid Y)}{P(X)} = \frac{P(Y)P(X_1 \mid Y) \ldots P(X_{64} \mid Y)}{P(X)}$$

The denominator is the same for each digit, so computing it is not needed to determine which digit has the highest probability.

For the counts above, naive Bayes uses:

$$P(Y = d) = n_d/n$$

$$P(X_i = 1 \mid Y = d) = n_{d,i}/n_d$$

$$P(X_i = 0 \mid Y = d) = (n_d - n_{d,i})/n_d$$

To avoid underflow, usually naive Bayes computes the log of the probability of the numerator:

$$\log(P(Y)P(X_1 \mid Y) \ldots P(X_{64} \mid Y)) = \log P(Y) + \log P(X_1 \mid Y) + \ldots + \log P(X_{64} \mid Y)$$

To avoid zero probabilities, usually the probabilities are modified using Laplace's rule of succession. Because there are 10 digits:

$$P(Y = d) = (n_d + 1)/(n + 10)$$

Because each bit has two values:

$$P(X_i = 1 \mid Y = d) = (n_{d,i} + 1)/(n_d + 2)$$

$$P(X_i = 0 \mid Y = d) = (n_d - n_{d,i} + 1)/(n_d + 2)$$

Expected Results and a Contest

A correct implementation should have an error rate under 21% over all 60,000 examples. The output of my program on the `cleanImages` in `Learner.java` is shown at the end of this document.

There are many learning algorithms that are better than naive Bayes. Classical linear learning algorithms will classify an input $\mathbf{x} = x_o, x_1, \ldots, x_{64}$ by

$$y = \mathbf{w} \cdot \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots w_{64} x_{64}$$

where $x_0 = 1$ for all examples; w_0 is called the bias weight.

For this lab, you would need 65 weights for each digit, a total of 650 weights. Your prediction would be digit d corresponding to the highest y_d value.

Let l be the correct label for an example. Then, for each $d \neq l$, if $y_l - y_d < 1$, then

$$\mathbf{w}_d \leftarrow \mathbf{w}_d - \alpha \mathbf{x}$$

$$\mathbf{w}_l \leftarrow \mathbf{w}_l + \alpha \mathbf{x}$$

With an appropriate value for the learning rate α, it is possible to acheive an error rate well under 20%. The five best error rates in this lab (must be under 20%) will receive an additional 50, 40, 30, 20, or 10 points.

Testing Your Program

A correct implementation should produce something like the following output when the main method in `Learner.java` is run.

```
0000000000111100010000100100001001000010010000100100001000111100
0
correct (label is 0, error rate = 0/1 = 0.00)
0000000000001000000010000000100000001000000010000000100000001000
0
incorrect (label is 1, error rate = 1/2 = 50.00)
0000000000111100010000100000001000000100000010000001000001111110
0
incorrect (label is 2, error rate = 2/3 = 66.67)
0000000000111100010000100000001000111100000000100100001000111100
0
incorrect (label is 3, error rate = 3/4 = 75.00)
0000000001000100010001000100010001111110000001000000010000000100
3
incorrect (label is 4, error rate = 4/5 = 80.00)
0000000001111110010000000100000001111100000000100000001001111100
3
incorrect (label is 5, error rate = 5/6 = 83.33)
0000000000000000000000000000000000000000000000000000000000000000
1
incorrect (label is 6, error rate = 6/7 = 85.71)
0000000000111100010000100100000001111100010000100100001000111100
3
incorrect (label is 7, error rate = 7/8 = 87.50)
0000000001111110000000100000010000001000000100000010000001000000
2
incorrect (label is 8, error rate = 8/9 = 88.89)
0000000000111100010000100100001000111100010000100100001000111100
3
incorrect (label is 9, error rate = 9/10 = 90.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 19/20 = 95.00)
0000000000111100010000100100001000111100010000100100001000111100
0
incorrect (label is 9, error rate = 25/30 = 83.33)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 32/40 = 80.00)
0000000000111100010000100100001000111100010000100100001000111100
0
```

```
incorrect (label is 9, error rate = 36/50 = 72.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 39/60 = 65.00)
0000000000111100010000100100001000111100010000100100001000111100
0
incorrect (label is 9, error rate = 42/70 = 60.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 44/80 = 55.00)
0000000000111100010000100100001000111100010000100100001000111100
0
incorrect (label is 9, error rate = 47/90 = 52.22)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 49/100 = 49.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 74/200 = 37.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 99/300 = 33.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 124/400 = 31.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 149/500 = 29.80)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 174/600 = 29.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 199/700 = 28.43)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 224/800 = 28.00)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 249/900 = 27.67)
0011110001000010010000100011110001000010010000100011110000000000
4
incorrect (label is 9, error rate = 274/1000 = 27.40)
```