OCR Project

Due March 25 at midnight

For this project , if you choose, you may work with a partner. You may also choose to work alone. No more than 2 people per group. Note that you must BOTH WORK ON ALL THE PROJECT. Equally, “My partner bailed” is not an excuse for not turning in the project .

If you choose to work with a partner, only one of you needs to turn in the project , but both names must be on the project (of course).

Project: OCR

(100 pts)

We humans are quite good at generalizing. We can look at text that’s on a page that is splattered with mud (or, more likely, coffee), and we can most likely read the text. We can read text that has been ripped, crumpled, faded, and even worn off the page. We’re that good. Computers, however, struggle with the problem of generalizing. And yet it is necessary to be able to take a page with text on it and to scan it in and convert it into recognized characters and words. Think of all the medical records that are currently sitting in files in medical offices. What if you need to know whether your mom was ever vaccinated against whooping cough? What if the CDC wants to determine if there’s an outbreak of a particular syndrome or type of cancer in a particular region of the country? Equally, many law offices need to scan in old cases and conclusions (clearly I’m not a lawyer – I’m not using the correct terminology) so that when researching current cases they can refer to previous conclusions to support their case. I’ve dealt extensively with the problem of scanning in documents and converting them into a form that can be used in a word processor (and hence read by a screen reader) when dealing with students who are blind or just nonvisual readers. These students regularly get handouts from professors who’ve had an article they’ve been Xeroxing since the 80’s. Without the ability to scan in the paper and then convert it to a form that can be read by a screenreader, the handout is the equivalent of a blank paper to these students.

The problem of taking text on a physical page and digitizing it and converting it into characters recognized by the computer is known as OCR – Optical Character Recognition. It usually consists of two parts: Taking a page and scanning it in to the computer (i.e., digitizing the page); and then taking the scanned in image and locating characters within the scanned in page. When the page is first scanned into the computer, it is usually a bitmapped image of the page – just numbers representing the amount of color at every dot on the page. At this point, the computer has no idea what characters may be on the page. OCR involves looking at the dots of color within the scanned image and trying to find patterns of color, and then relate those patterns to patterns of images.

Initially, OCR was done using individual files, with each file holding a pattern for a particular character in a particular font. While this can clearly get out of hand quickly (think of the number of fonts you have on your computer alone, and then throw in individual handwriting), we still use this method for certain problems in which we know the font that will be used. More frequently today we use a more sophisticated system of looking for characteristic patterns that represent characters. For instance, a capital letter A would be represented by 3 patterns – a forward slant, a backwards slant, and a horizontal line occurring between the top and bottom of the forward slant and backward slant.

Your problem:

The post office only has to identify zip codes. Thus their problem is limited to 10 characters – 0 through 9. For simplicity's sake, that's what we'll work on. In addition, we'll assume that every character is in the same font. Thus we have files representing the characters 0 through 9 (named zero.txt, one.txt, two.txt, etc.). These will be the files that will be used to identify characters in the various image files (You can download each of these number files from my web site. Store them in your OCRProj folder (parallel to the src folder)).

The general idea: Both the character files and the image files will be read into matrices (arrays of arrays). The image file will be "cleaned" (converted so that each entry in the matrix will either be considered black (represented by 1) or white (represented by 0). Once the image file is cleaned, each entry in the character file will be compared to each entry in the image file. If the two entries match, the matching score will increase. If they don't, the score will either drop or stay the same. Thus an overall matching score will be determined for the image file and the character file.

The image file will be compared to all 10 character files. The image will be determined to be the character whose file it best matches.

Note that the image file will be "noisy", meaning there will be certain dots that will be considered to be black (1) when they are just dirt specs or extra ink, or even just scanned in incorrectly. Equally some dots that should have ink may be considered white (0), possibly because of folds in the paper or over-xeroxing, etc. We're doing a best match.

Specifically:

I am giving you a class called NumFiles that will read in each of the character files into a private field of int[][] (i.e., a matrix of integers). The field will be named after the number it represents (e.g., zero, one, two,…). Each matrix is a 13x13 matrix of integers representing the black/white (0/1) pattern of a particular character.

The fields are private, so you must access them using the "accessor" called getMatrix, which takes as a parameter a String that holds the name of the number (again, zero, one, two, etc.).

Your job is to create an image class (completing the outline, below), and then to create a main class that creates an object from the image class and calls its FindBestMatch method to determine which number best matches the image file.

Here is an outline of the image class:

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class ImgClass {
      /* has a field for a NumFiles object, that will contain all the
      matrices for each number.  This field's fields will be used for
```

```
	comparison to determine which number best matches the pattern in the
	image file.
	*/
	/*has a field for the image matrix, read in using the ReadMatrix()
	method, given below, that reads in the file of doubles containing the
	image of the number
	*/
	double[][] imgMatrix = ReadMatrix();
	/* has a field that is a matrix of integers the same size as the
	imgMatrix.  It will hold the black/white version of the image matrix.
	It is created by the method cleanup, which takes each value (double) in
	the imgMatrix and translates it to a 1 in the new matrix if the value
	is above .5, and to a 0 if the value is below .5.(you can read more
	about the cleanup method under its method description, below).
	*/

	public double[][] ReadMatrix() {
	/*This method reads in an image file of doubles, representing a
	character.  You must enter the name of the file to be read in, and the
	file must be in your OCRProj folder (parallel to the src folder).
	*/
	//This method returns an array of doubles, representing the image file.
	// Don't mess with this method.  It works. Don't argue with me.
	// Honestly, it works.
		String filename = JOptionPane.showInputDialog("Enter The file to
be read");
		File fl = new File(filename);
		double matrix[][] = new double[0][0];
		try {
			Scanner fn = new Scanner(fl);
			int row = 0;
			while (fn.hasNext()) {
				row ++;
				fn.nextLine();
			}
			fn.close();
			matrix = new double[row][];
			Scanner f2 = new Scanner(fl);
			row = 0;
			while (f2.hasNext()) {
				String line = f2.nextLine();
				System.out.println(line);
				String[] numarr = line.split(" ");
				int x = numarr.length;
				double[] arr = new double[x];
				matrix[row]= arr;
				for (int i = 0; i<numarr.length;i++){
					matrix[row][i] = Double.parseDouble(numarr[i]);
				}
				row++;
			}
			f2.close();
		} catch (FileNotFoundException e) {
			System.out.println("Error opening file. Please make sure that
you have your text file in the same folder as the NumFiles.class");
			System.exit(0);
		}
```

```
      return matrix;
}

public int[][] cleanup() {
/* this method cleans up" the image matrix of doubles, created from the
image file.
It creates and returns a brand new matrix of integers, the same size as
the image matrix of doubles.
It takes each value in the image matrix and translates it to a 1 in the
new matrix if the value is greater than .5, and translates it to a 0
otherwise.
*/
/*So, for instance, if the image matrix looks like this:
    { { 0.428, 0.671, 0.323, 0.281 }
      { 0.682, 0.031, 0.273, 0.511 }
      { 0.423, 0.888, 0.782, 0.123 } }
 The new matrix will look like this:
    { { 0, 1, 0, 0}
      { 1, 0, 0, 1 }
      { 0, 1, 1, 0 } }
*/

public String FindBestMatch() {
/*This method uses the method compareMatrices to compare the imgMatrix
to each of the num matrices in the NumFiles object.  It keeps track of
scores returned from compareMatrices.  The NumFile matrix with the best
compareMatrices score is considered to be the most likely character,
and a string representing that number is returned.  So, for instance,
if the comparison between the imgMatrix and the NumFiles object's three
field returns the highest score, the string "three is the best match"
is returned.
*/
/* Note that this method would've been much easier to write if I'd made
one field in my NumFiles class, and that one field was an array of
matrices.
*/

public double compareMatrices(int[][] num) {
/*This is the challenging method, and the heart of the OCR algorithm */
/* This method takes as input the img matrix and the num matrix.  It
loops through every possible place in which the num matrix could occur
within the img matrix and gets a score of how well the two matrices
match up. Note that the image matrix can be any dimensions, although we
can assume the dimensions are larger than the num matrix.  So, for
instance, if the image matrix is 5 x 9 and the num matrix is 4 x 4,
you'd have to look at each possible comparison:
```

```
1      0      0      0      0
0      0      0      0      0
0      1      0      0      0
0      0      0      1      0
1      0      0      0      0
0      0      0      0      0
0      0      1      0      1
0      0      0      0      1
0      0      0      0      0
```

```
Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
```

```
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
0       0       0       1       0
1       0       0       0       0
0       0       0       0       0
0       0       1       0       1
0       0       0       0       1
0       0       0       0       0

Then
1       0       0       0       0
0       0       0       0       0
0       1       0       0       0
```

```
    0       0       0       1       0
    1       0       0       0       0
    0       0       0       0       0
    0       0       1       0       1
    0       0       0       0       1
    0       0       0       0       0

    With the 4x4 matrix representing the character.
    */
    /*
    This will require loops inside of loops.

    The matching score for each possible matching matrix should be
    calculated as follows:
        •  If both the img matrix and the num matrix are 1, the score
           increases by 1.0.
        •  If the num matrix is 1 and the image matrix is 0, the score goes
           down by .25.
    •  If both the img matrix and the num matrix are 0, the score goes up
       by .25.
    •  And if the num matrix is 0 but the image matrix is 1, the score
       stays the same.

You must calculate the matching score for each possible matching matrix
within the imgMatrix, and return the best score of those scores.

Note that if you wish to create other methods that this method uses, that
is fine with me.

*/
```