Digit Recognizer Project

Statistical Pattern Recognition, Bahar 1403

1 Overview

In this project you implement and test a classifier for recognizing digits 0..9 in a scanned document image. First, write code to find digit regions and measure features of these regions. Then implement and test a digit recognizer that classifies regions based on their measured features. Your final project report should describe

- your feature set and classifier design: what type of classifier did you choose, and why
- your training and test data
- the implementation of your classifier: describe code you wrote; describe how you configured classification code provided by a toolbox
- classifier performance: provide an estimate for P(error) and discuss the accuracy of this estimate
- strong and weak points of your classification approach; avenues for improving the classifier in future work.

Digit recognition has important practical applications such as reading zip codes for automated mail sorting. Years of research and development have led to high-performance noise tolerant digit classifiers that perform well on a large variety of fonts and handwritten digits. That level of performance is far beyond what you are expected to achieve in this course project.

This project exposes you to the challenges of classifier design, allowing you to experience the wide range of choices in feature selection, type of classifier, training, and testing. For a successful project you do *not* have to achieve super high recognition rates on noisy data with a big variety of fonts. You may report P(error) as low as a few percent or as high as 40%- 50%. A lot depends on the test data that is used: students who venture into testing with many different fonts or with handwritten digits naturally end up with a higher P(error). Don't obsess about getting the highest possible recognition rate, but instead focus on gaining insights and useful experience as you work on this project. Describe these insights in your project report: what aspects of classifier implementation went as expected and what were the surprises? What did you learn? Would you do anything differently next time? What challenges did you encounter in testing your classifier, and how was the performance? If you had another year to work on the classifier, how would you go about improving it?

2 Training and test data

Use your own created or a public database of digit images such as:

- MNIST provides 70,000 handwritten digits http://yann.lecun.com/exdb/mnist; this is a subset of a larger database available from NIST. This website http://cis.jhu.edu/~sachin/digit/digit.html provides a subset of the MNIST data in a format convenient for classifier training: 10 separate files, one per class, with 1000 training samples per class.
- The Chars74K dataset (Character Recognition in Natural Images) includes an EnglishFnt dataset with both digits and alphabetic characters. These are typeset, and include italic, bold and normal.
- Tell me about additional datasets you find, so that I can add them to this list.

3 Choice of programming language or classification environment

Use the Python programming language.

- If you write code from scratch thorough understanding of every detail of the classification algorithm. In evaluating projects I definitely take into consideration that it is time-consuming and difficult to write code from scratch. In such a case, you may restrict your implementation to relatively simple classification algorithms.

- If you use the implemented classifiers from a python library, you can easily choose from a menu of classification algorithms, compare various classifiers, and try classifier combination. The downside is that you won't understand the classification algorithms in as much detail as if you coded them yourself. Very important is that you understand the ideas and algorithms you use! In your project report write a brief description of each type of classifier you used (e.g. what is a *Random Decision Forest*). Describe the parameters for each classifier. What they mean and how you set them. Describe the testing protocol you use -- for example, describe what *cross validation* is and what parameters you set.

4 Digit Recognizer Part 1: Feature extraction

Write a program that takes an image as input, finds the connected dark regions in the image, and measures features of these regions. You can also try different methods of feature extraction.

4.1 Image file formats

An N by M image contains N*M pixels. A grayscale image typically has pixel values between 0 and 255, so one pixel can be stored per byte. In a binary image (black and white) each pixel requires only one bit, allowing 8 pixels to be stored per byte.

Many different image file formats are in use, including GIF, JPEG, BMP, RFF, TIFF, PBM. These files contain more than just a list of pixel values because such a list isn't enough to reconstruct the image. For example, a list of 10,000 pixel values might be a square image with 100 rows and 100 columns or a rectangular image with 500 rows and 20 columns. Each of the image file formats starts with header information: the number of rows and columns in the image, the number of bits per pixel, and perhaps other information related to file compression, comments, image tagging.

4.2 Suggested steps for finding digit regions and measuring features

The following steps are suggested for digit recognizer part 1. You are welcome to assist each other on the programming aspects of this problem.

1. Find or write code for image I/O: you need to be able to read an image from a file into a 2D array
   - Write code to make a simple image modification, and make sure that you can display the result. For example, create a photographic negative of a gray-level image (with pixel values from 0..255) by changing each pixel value A[i,j] to the new value 255-A[i,j].

2. Threshold the image (in preparation for step 3). If your digit images are clean and have a lot of space between digits wide range of thresholds work well for these images. You can manually try a value (e.g. make all pixels with values ≥ 100 white, and make all pixels with values < 100 black) and display the result to check if it's ok. Note that some image environments use 0 black and 255 white while others use the opposite.

3. Write a program (or use an image utility) to find connected black regions in the binary image. Simplifying assumptions:
   - Each connected region is a symbol. In digit recognition we don't have to deal with multi- part symbols such as "i" "j" "=".
   - Segmentation problems do not occur. The above mentioned databases contain digit images which are widely-spaced. This avoids two types of segmentation problems that are common in noisier document images: (1) a symbol is split into pieces because of insufficient inking, and (2) several symbols are merged because they bleed together due to close spacing or too much inking. Segmentation problems are difficult to handle in real- world applications. The development of robust segmentation algorithms continues to be an active area of research.

   Finding regions does not require you to implement an edge detector. Instead, region growing can be done directly as follows, using a *flood fill algorithm.* Scan the image to find a black pixel. Then look in the neighborhood of this black pixel to find other black pixels that belong to the same region. Repeat this process until a complete region has been extracted. This can be coded using recursion, or keep a list of (row, col) values for pixels that still need to be visited to complete the current region. Recursive code for region finding is easy to write, but large regions can cause stack overflow. The regions in the above mentioned digit images are rather small, so should not cause stack overflow.

   Your program needs to keep track of which pixels belong to each region. One method is to create an output image where pixel value 0 is used for the background (not part of any region) and pixel value i>0 denotes that this pixel belongs to region i. As you process a black pixel in the input image, write value i into the output image and erase this pixel from the input image to prevent subsequent re-processing of the same pixel.

4. Write code to measure the following features for each connected black region.
   - Blackness ratio. This is "area of the region (number of black pixels)" divided by "area of the bounding box".

     The *bounding box* of a region is the smallest axis-parallel rectangle that encloses all the pixels belonging to the region. Let minX, maxX, minY and maxY be the extreme x and y values for the pixels belonging to the region. The left edge of the bounding box is at x=minX, the top edge is at y=maxY, etc. If you like, you can draw the bounding box in your output image; this can help with debugging.
     - Aspect ratio of the bounding box. This is (maxX - minX + 1) / (maxY - minY + 1)
     - Number of holes in the connected black region. You can reuse the region growing code you wrote for step 3, but switch the role of white and black pixels.

       Notice that the definition of black and white regions must be asymmetric. If you define the connectivity of black regions using an 8 neighborhood, then define the connectivity of white regions (holes) using a 4 neighborhood. *Using an 8 neighborhood* means that diagonally-touching black pixels (such as pixels A[3,4] and A[4,5]) are considered to be part of the same region. Using a 4 neighborhood, diagonally-touching pixels are not

considered part of the same region (unless they are connected some other way, e.g. by A[4,4] or A[3,5] also being part of the region).

- Two or more additional features of your own devising.

4.3 What to hand in for part 1 of project

For part 1, hand in

- Sample output showing that you are able to measure features of digit images. (Not long, please. Just enough to demonstrate that it is working.)
- A brief description of the features of your own devising.

5 Digit Recognizer Part 2: Classification

Create a digit classifier and measure its performance. You can use some of the features measured in Part 1, and you are welcome to add new features. You might find that some of the features from part 1 are rather weak and you would get better classification performance by omitting them. For example, you may find that the bounding-box aspect ratio does not provide a lot of discrimination information.

You are free to choose the classification algorithm. For example, you might use one of the following:.

- A nearest-neighbor classifier, or a k-nearest neighbor classifier.
- Plot points in feature space and manually place decision boundaries.
- SVM
- Template matching
- Multiple classifier combination. Using an environment such as Weka or R, apply several types of classifiers and use voting to determine the overall classification result. Does the voting improve performance: is P(error) lower for the voting result than for the individual classifiers?
- You should not make neural network classifiers the main focus of your project. This is because neural networks were not covered in this course. However, if you are using several different types of classifiers in your project work, then you are welcome to include neural network classifiers. For example, you could implement several classifiers (including a neural network one) and compare performance. Or you could use multiple classifier combination, as mentioned above.

5.1 Measuring Performance of the Digit Recognizer

Some students test on only 20 digits, others test on over 1000. Recognition rates vary from around 50% up to 98% correct. I don't mark your project based on classification performance, as long as your classifier outperforms random guessing (i.e. is correct more than 10% of the time).

Typically, students report recognition results in the following form:

My classifier was correct on L out of M test digits (that is, N%)

I don't expect you to use thousands of test images, but be sure to discuss the statistical significance of your test results. Refer to the confidence intervals in Figure 9.10 of DHS to translate your L and

M values into a range of N values that is very likely (with 95% confidence) to contain the true correctness rate.

For large test sets it can be time-consuming to establish the ground truth (the correct answer) and to check the correctness of the classifier's response. One way around this is to use ten separate test files, where each file contains many instances of the same digit. If you have used those files for training, then you need to find other files for testing.

Sometimes students report recognition results for training data. It's ok to do this as long as you clearly label those results as "testing on the training data", and you have additional test data as well. If during your classifier design process you trained and tested, then retrained and retested using the same files, then your report should clearly state that your performance estimate may be optimistic due to "training on the test data".

If you are testing on a variety of fonts, or on handwritten digits, include a figure to show a small representative set of data. If you are dealing with very large font variations then it can be helpful to subdivide some of the classes. For example, character recognizers sometimes divide g into two classes: one class for the letter with two holes g g and another class for the letter with one hole g g (one hole). Analogously, a digit recognizer could use separate classes for fours with no holes Ч versus fours with one hole 4.

If you test on a variety of data, you might be able to make conclusions such as the following: recognition is best in the single-font case, degrades as more fonts are added, handles bold-face better than italic, and so on. That sort of observation and analysis is quite interesting even if you do not have time to test on sufficient amounts of data to give high-confidence estimates of the recognition rates.

If you like, your test data can include a few non-digit characters (such as alphabetic characters). How do these characters get classified? Can your classifier reject such input or report low confidence in classification?

5.2 What to Hand in for Part 2

Required report format: 2-4 pages of text (not counting figures and references) that succinctly presents the main points. If you wish, you can optionally include appendices to provide more detailed information. In my marking I will concentrate on your 2-4 pages of text, and will only read the appendices if your writing makes me eager to do so. I impose this strict page limit to give you practice in the vital skill of writing concise documents that convey the main ideas in an informative, convincing and engaging way.

Organize your project report so that the reader is instantly impressed by the quality of your work. This is an essential presentation skill, not just for writing course project reports, but also for writing theses, conference papers, journal papers, and grant proposals. Refer to my advice about technical writing on the course website.

The first paragraph of this document indicates topics you should include in your project report. Many students find that they don't have enough implementation time to get great classifier performance. Once you have a classifier that works somewhat reasonably well, then put your efforts into analyzing the strengths and weaknesses of this classifier, and describe that in your project report. Perhaps you can evaluate the effectiveness of your features. Maybe you can discuss strengths and weaknesses of your overall design. Maybe you can test how good the classifier is at handling a variety of fonts and font sizes, or bold and italic digits, or handwritten digits.