Reading text in an image

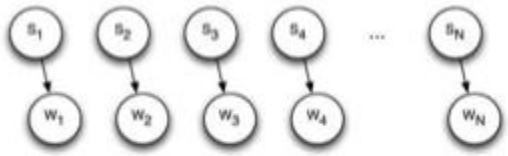

Figure 1: Simplified Model

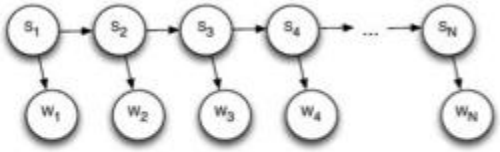

Figure 2: HMM

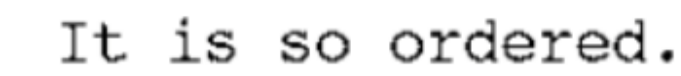

Figure 3: Our goal is to extract text from a noisy scanned image of a document.

To show the versatility of HMMs, let's try applying them to another problem; if you're careful and you plan ahead, you can probably re-use much of your code from Part 1 to solve this problem. Our goal is to recognize text in an image - e.g., to recognize that Figure 3 says "It is so ordered." But the images are noisy, so any particular letter may be difficult to recognize. However, if we make the assumption that these images have English words and sentences, we can use statistical properties of the language to resolve ambiguities. We'll assume that all the text in our images has the same fixed-width font of the same size. In particular, each letter fits in a box that's 16 pixels wide and 25 pixels tall. We'll also assume that our documents only have the 26 uppercase latin characters, the 26 lowercase characters, the 10 digits, spaces, and 7 punctuation symbols, (), .- !?'". Suppose we're trying to recognize a text string with n characters, so we have n observed variables (the subimage corresponding to each letter) O1, . . . , On and n hidden variables, I1 . . . , In, which are the letters we want to recognize. We're thus interested in P (I1, . . . , In | O1, . . . , On). As in part 1 , we can rewrite this using Bayes' Law, estimate P (Oi | Ii) and P (Ii | Ii−1) from training data, then use probabilistic inference to estimate the posterior, in order to recognize letters.

What to do. Write a program called image2text.py that is called like this:

python3 ./image2text.py train-image-file.png train-text.txt test-image-file.png

The program should load in the train-image-file, which contains images of letters to use for training (we've supplied one for you). It should also load in the text training file, which is simply some text document that is representative of the language (English, in this case) that will be recognized. (The training file from Part 1 could be a good choice). Then, it should use the classifier it has learned to detect the text in test-image-file.png, using (1) the simple Bayes net of Figure 1 and (2) the HMM of Fig 2 with MAP inference (Viterbi). The last two lines of output from your program should be these two results, as follows:

python3 ./image2text.py train-image-file.png train-text.txt test-image-file.png
Simple: 1t 1s so orcerec.
HMM: It is so ordered.