Developing an OCR system

COM2004/3004 Assignment

Due: 3:00pm on Wednesday 13th December

Contents

1. Objective

- To build and evaluate an optical character recognition system that can process scanned book pages and turn them into text.

2. Background

In the lab classes in the second half of the course you will be experimenting with nearest neighbour based classification and dimensionality reduction techniques. In this assignment you will use the experience you have gained in the labs to implement the classification stage of an optical character recognition (OCR) system for processing scanned book pages.

OCR systems typically have two stages. The first stage, document analysis, finds a sequence of bounding boxes around paragraphs, lines, words and then characters on the page. The second stage looks at the content of each character bounding box and performs the classification, i.e., mapping a set of pixel values onto a character code. In this assignment the first stage has been done for you, so you will be concentrating on the character classification step.

The data in this assignment comes from pages of books. The test data has been artificially corrupted, i.e. random offsets have been added to the pixel values to simulate the effect of a poor quality image.

3. The task

Your task is to design a classifier that:

1. uses a feature vector containing no more than 10 dimensions;
2. operates robustly even on low quality, `noisy' image data.

4. What you are given

You have been given data for training and testing your systems and some code to get you started.

4.1. The data

The data is stored in a subfolder named `data` and is split into data for training and data for evaluation. The data comes from pages from novels. There are 10 pages for training and 6 pages for testing. The testing pages have progressive amounts of noise added to them, i.e.,test page 1 is the best quality and test page 6 is the poorest quality. For each page there are three files.

1. a `png` format image file containing an image of the page. You should be able to view these files in any standard image viewing software.
2. a file ending in the extension `.bb.csv` . This is a comma-separated variable file giving the bounding box coordinates of each successive character on the page. Each line represents the position of a single character.
3. a `label.txt` file giving the correct ASCII label for each character on the page. There is a direct correspondence between the lines in the `.bb.csv` file and the `.label.txt` file.

4.2. The code

The code is organised into four python files: `train.py` , `evaluate.py` , `utils.py` and `system.py` . The first three of these should not be changed. Your task is to rewrite the code in `system.py` to produce a working OCR system.

In brief, the files have the following function:

- `train.py` -- this runs the training stage. It will read the complete set of training data, process it and store results in a file called `model.json.gz` in the data folder. It uses functions in `system.py` that you will need to modify and extend.
- `evaluate.py` – this runs the evaluation code. It will first read the results of the training stage stored in `model.json.gz` . It will then perform OCR on the test pages and evaluate the results. It will print out a percentage correct for each page. Again, it uses functions in `system.py` that you will need to modify and extend.
- `utils.py` – these are utility functions for reading image and label data and for reading and writing the `model.json.gz` files.
- `system.py` -- the code in this file is used by both `train.py` and `evaluate.py` . It stores the dimensionality reduction and classification code and is the part of the software that you need to develop. The current version has dummy code which will run but which will produce poor results. The dummy dimensionality reduction simply truncates the feature vector to be 10 elements long (i.e., the first 10 pixels of the image). The dummy classifier outputs the first label in the list of valid labels regardless of the input.

Your task is to write a new version of `system.py` . Your solution must not change `train.py` , `evaluate.py` or `utils.py` . Once you are finished you will run `train.py` to generate your own version of `model.json.gz` . You will then submit the `system.py` along with the `model.json.gz` file. The program `evaluate.py` will then be run by the assignment assessors with the code and data that you have submitted. It will be run on a new set of test pages that you have not seen during development. The performance on these unseen test pages will form part of the assessment of your work.

5. How to proceed

The steps below should help you get started with implementing the system. Steps 3 to 6 are not necessarily sequential. Read through this section carefully before considering your approach.

Step 1: Read and understand the code provided

The code provided does all the file handling and feature extraction for you. However, it is important for you to understand how it is working so that you can develop your solution appropriately.

Step 2: Test the code provided

Check that you can run the code provided. Open a terminal in CoCalc. Navigate to the directory containing the assignment code,

```
cd com2004_labs/OCR_assignment/code/
```

Run the training step,

```
python3 train.py
```

Then run the evaluation step,

```
python3 evaluate.py dev
```

The code should print out the percentage of correctly classified characters for each page. The dummy code will produce results in the range 3% to 5% correct for each page.

Step 3: Processing the training data

The function `process_training_data` in `system.py` processes the training data and returns results in a dictionary called `model_data`. The program `train.py` calls `process_training_data` and saves the resulting `model_data` dictionary to the file `model.json.gz`. This file is then used by the classifier when `evaluate.py` is called. So, any data that your classifier needs must go into this dictionary. For example, if you are using a nearest neighbour classifier then the dictionary must contain the feature vectors and labels for the complete training set. If you are using a parametric classifier then the dictionary must contain the classifier's parameters. The function is currently written with a nearest neighbour classifier in mind. Read it carefully and understand how to adapt it for your chosen approach.

Step 4: Implement the dimensionality reduction

You are free to use any dimensionality reduction technique of your choosing. PCA should perform well but is not necessarily the best approach. Start by looking at the function `reduce_dimension` in the existing `system.py` code provided. This function currently simply returns the first 10 pixels of each image and will not work well. It will need to be rewritten.

Step 5: Implement the classifier

You are free to use any classification technique of your choosing. A nearest neighbor classifier should work well but is not necessarily the best approach. Start by looking at the function `classify_page` in the existing `system.py` code provided. This function is currently just returning the first character in the list of valid labels regardless of the input. It will need to be rewritten.

Step 6: Error correction (Difficult)

There is potential to fix classification errors by using the fact that sequences of characters must form valid words. This can be done by checking the classifier outputs for the characters making up a word against a dictionary of valid English words. If the word doesn't appear in the list it is possibly because there has been a classification error. Errors can then be fixed by looking for the closest matching word. For example, if the classifier outputs the sequence, 'Comquter' this won't be in the word list, but it can be corrected to the closest match, i.e. 'Computer'. This simple approach is not without its problems, so feel free to experiment with this stage in order to come up with a better solution.

A suitable word list can be found at http://www-01.sil.org/linguistics/wordlists/english/.

This step is made more difficult by the fact that it may not be clear where a word starts and ends. You may try to infer this by looking at the spacing of the bounding boxes.

Additional rules

Some additional rules have been imposed that must be obeyed,

- The file `model.tar.gz` must not be bigger than 3 MB

- The `evalute.py` program should not take more than 120 seconds to produce a result when run on the CoCalc servers.
- You may make use of any code that has been developed in the lab classes (even code appearing in the solutions – but you may want to improve it!).
- You may not use code from the python scikit-learn module.

6. Submission

Deadline: 3:00pm Wednesday 13th December.

Submission will be via MOLE. You will be asked to submit the following.

- A copy of your `system.py`
- A copy of your data file `model.json.gz`
- A form (which will appear on MOLE) consisting of several questions asking you to:
  - report the performance of your system on the development set;
  - explain/justify the design of your feature selection;
  - explain/justify the design of the classifier;
  - explain/justify the design of the error correction code.

7. How your work will be assessed

The assignment is worth 50% of the module mark.

We will be looking at the Python code quality, the overall design and the general performance of your program. You will be awarded a mark out of 50 made up from the following five 10-mark components.

Code quality (10 Marks)

- Is the code well presented?
- Is it easy to read?
- Does it make appropriate use of Python's features?
- Is the code clearly documented?

Feature extraction (10 Marks)

- Has an appropriate feature extraction technique been employed?
- How has the choice and design of the feature extraction been justified?
- Has the chosen technique been well implemented?

Classification (10 Marks)

- Has an appropriate classification technique been employed?
- How has the choice and design of the classifier been justified?
- Has the chosen technique been well implemented?

Error correction (10 Marks)

- Has any attempt been made at error correction?
- Has the choice and design of the error correction code been justified?
- Has the chosen technique been well implemented?

Overall performance (10 Marks)

- Does the code run correctly?
- How does the performance compare to that achieved using a standard nearest neighbour and PCA approach.

The figures below give an indication of the approximate performance that you should expect using a basic nearest neighbour and PCA based approach.

Page	Score
1	98%
2	98%
3	83%
4	58%
5	39%
6	29%

8. Lateness penalty

There will be the standard 5% penalty for each working day late.

This is an individual assignment. Do not share your code with other students. Collusion will result in a loss of marks for all students involved.

(COM2004/3004 2017-18 Assignment Handout v1.0)