COM2004/3004 Assignment 2 - Solving the Word Search Puzzle

DEADLINE: 3:00pm on Monday 12th December

Objective

To write a Python program for solving word search puzzles.

Background

In the lab classes you will be experimenting with nearest neighbour based character recognition. In this assignment you will build on the knowledge you have gained to design a complete word search puzzle solver. The program will take a rectified wordsearch grid image, a list of words and some classifier training data as input. It will output a marked-up image indicating where the words appear in the grid.

What you are given

In the assignment 2 folder in SageMathCloud you will find two files. The file `assignment2.pkl` is a file that contains the data that you need. You will also find a notebook containing some handy bits of code including a classify function that you can re-use.

`assignment2.pkl` is a Python 'pickle' file contains the following data,

- `train_data` – 699 training examples stored in a matrix,
- `train_labels` – the training data labels represented as integers,
- `test1` – a matrix representing an image of the word search grid to be processed,
- `test2` – same as test1 but with a lower image quality,
- `words` – an array of 24 words to be found in the grid (famous gardeners).

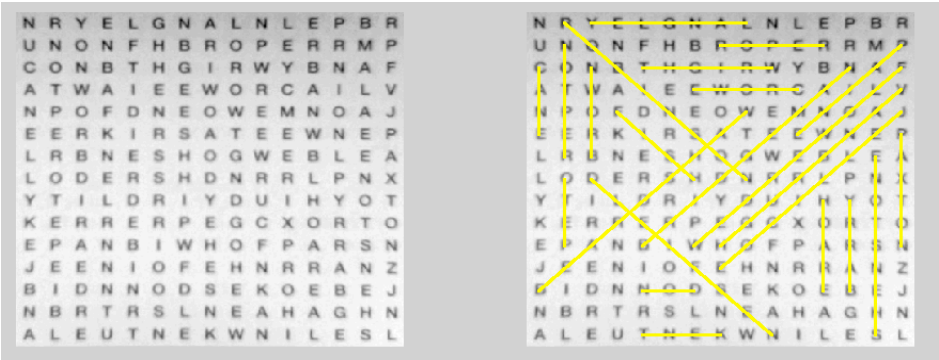
The image has been scaled so that each character is a 30 x 30 block of greyscale pixels, i.e., the characters can be represented by 900 pixel values (a 900-dimensional feature vector).

What you have to do

You are asked to write a Python function that can be called as follows,

```
wordsearch(test1, words, train_data, train_labels)
```

The function should find all of the words in the test image and display the result by drawing lines over the input grid to indicate the positions of the words, i.e., as in the figure below.



You are asked to submit results for the following **three test conditions**:

Trial 1: Using image **test1** and using all **900** pixel features for classification.
Trial 2: Using image **test1** and using a feature vector of at most **10** dimensions.
Trial 3: Using image **test2** and using a feature vector of at most **10** dimensions.

How to proceed

The steps below guide you through the process of implementing the system. They are merely a suggestion for one plan of attack. You may wish to approach the problem in another way.

Step 1: Preprocessing. `test1` is a 450 by 450 matrix of pixels. You will need to write a nested-loop to extract individual characters of size 30 by 30 pixels from the image. Each 30 by 30 character should then be reshaped to form a 900-element vector. There are 225 characters in the grid so you should have 225 feature vectors that can be stored in one large 255 by 900 element matrix.

Step 2: Classification. Study `classify.py` and make sure that you know how to use it. Then use `classify.py` and the training data provided to turn the matrix computed in Step 1 into a vector of 225 letter labels. Reshape this vector to form a 15 by 15 matrix of letter labels.

Step 3: Select a word to find. The words that you need to find have been given to you in a variable called, words. This variable is a list containing 24 strings.

Step 4: Perform the search. You now need to search the 15 by 15 matrix of letter labels for the sequence of characters in the word you wish to find. Remember, words can go forwards, backwards, up, down or in any diagonal direction. This is the challenging step, so think carefully. Remember that the letters in the matrix may not be correct because your classifier might have made mistakes.

Step 5: Display the result. Once a word has been found you should indicate where it is in the image. You will need to convert the row and column of the first and last letter of the word in the 15 by 15 letter grid into corresponding pixel coordinates in the image. You can then display the image using `matplotlib` and draw coloured lines over the top to indicate the word positions.

Step 6: Loop over all words. Write the necessary loop to find all 24 words and evaluate on image `test1`. This will provide you with the result for **Trial 1**.

Step 7: Dimensionality reduction. For **Trial 2** you can only use 10 dimensions. Use a dimensionality technique of your choosing to reduce the training and test data dimensionality to 10. Rerun your system on test1.

Step 8: Noise robustness. For **Trial 3**, rerun Step 7 but use the poorer quality image, `test2`. If the performance is poor think about how you can improve it. *Hint:* you may need to redesign Step 4 so that it can cope with a large number of classification errors. It should be possible to locate at least half of the words.

Submission

Submit a report containing an ***easily-readable*** *listing of the final version of your code*, and *screen shots* of the output of the three trials. For each trial report how many words have been found correctly. Your report should also contain *a written introduction to your solution (max 2 sides)*. The introduction should explain and justify the approach you have taken. A well-written introduction will make the code easier to read and easier to mark.

Submit the written report to reception in the usual manner. You should also upload your source code using the assignment upload tool on MOLE.

Deadline

3:00pm Monday 12th December.

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. Roughly equal weight will be given to each of these three aspects. Here are some things to think about when self-assessing your work before submitting it,

Code quality
Is the code well presented?
Is it easy to read?
Is the code well documented and clearly commented?

Design
Is the overall approach well conceived?
Does your design make good use of Python's strengths?
Has the program been decomposed into a sensible set of functions?

Performance
Does it run correctly?
Does it do what it is supposed to do?
Does it work reliably on the noisy data?

REMINDER

There will be the standard 5% penalty for each working day late. This is an **individual** assignment. Collusion will be dealt with severely and will result in a loss of marks for all students involved.

JPB, 10/2016