Document Similarity Project

I added a small test directory for your final program Run your program on `/u1/junk/cs617/MiniTest`. The `y1.*` files should all be similar, but different from the `y2.*` files.

Outline of the Steps

1. Put all of your thousands of document files into your `Files` subdirectory. (Nov 14)
2. Find all 3-word shingles that appear in more than one document. Save a dictionary that has the shingles as keys and the shingle's position (in some fixed ordering of the shingles) as values. Write your program with the directory containing the files as the first command line argument and with the name of the output file as a second command line argument. Use pickle and follow the usual file name conventions (pickle file names end in `.pkl`). (Nov 18)
3. Write a function that uses the dictionary obtained in the previous step, reads a document and returns a binary (0-1) array (or list) as the value of the function. For each 3-shingle in the document, check to see if it is in the dictionary, if so put a one in the output array at the position of that 3-shingle. Name the function `charfunc`, which has two parameters: a filename and a dictionary. (Nov 29)
4. Create the big numpy array whose shape is `(rows,columns)` where `rows` is the number of shingles in the dictionary and `columns` is the number of document. Save the array using `np.save`. Numpy save file names end in `.npy`). (Nov 30)
5. Load (using `np.load`) the big array. Compute an array of minhashes. Each minhash will use one hash function h to permute the rows of the array. The minhash value in each column is the position of the first one in the rows $h(0)$, $h(1)$, $h(2)$, etc. Debug your minhash values by printing the rows $h(0)$, $h(1)$, $h(2)$, etc., and verifying the location of the first one. (Dec 2)
6. Experiment with the number of rows per block, and the number of blocks. (Dec 5)
7. Select parameters for your final program. When I run your final program on a directory containing a set of documents, your final output will be a

list of similarity groups, i.e., a group number followed by the files in that are in that group. (Dec 9)

Save your programs in your Project directory. In that directory make a subdirectory for each date: `nov18`, `nov29`, `nov30`, `dec02`, `dec05` and `dec09`. Name your program for each with the date following by ".py". I will deduct points if your program is too similar to another student's program. Each step is worth 25 points. If you are late, 5 points will be deducted for each late day. Due times are all end-of-day.

Don't forget to include a README file in the `dec09` directory to tell me how to run your system. The README should give me the command to run your system on my directory of files. So the input to the whole process is a directory containing a set of text files and the output lists groups of files that your system judges to be similar.