Assignment 4

Our last program will give you a chance to show off how much you’ve learned in this course, using a simple example of a larger problem—archival search.

Our search corpus consists of 25 text files—these include fiction, news articles, poetry, recipes—basically, some odds & ends. Your program will read them, develop a profile of each by extending the techniques we used in the authorship program, and allowing the user to interact with the program—when the user enters a search term, your program will return a ranked list of possible matches, with the best match first, and a short excerpt from the beginning of each match.

Your program will need to carry out several steps. You may find it easier to break into 2 parts, with the “preprocessing” portion below handled separately from your main (interactive) program.

PREPROCESSING:

To save time when interacting with the user, you will need to develop a ‘profile’ of each item in your corpus. You did some of this for the authorship program; re-use or modify code from that program as needed.

For each file in the corpus:

1. Read in each file, clean up (i.e. remove) punctuation as before, convert to a list of words (strings), and from that, generate a hash with the count of how many times each word appeared in that file.
2. *Remove* from consideration any word appearing in the list of stop words (attached). These are words that tell us little to nothing about the document itself—words like “the”, “on”, “in”, “itself”, and so on. Removing them reduces the amount of noise in our data.
3. Produce the relative frequency and find the negative of the base-10 logarithm, as in the authorship assignment, to get a frequency score for each remaining word in the file.
4. Save the hash to a file. Remember that each hash is associated with a particular file; you will need a way to track that. Whether it’s in a separate data structure or stored as a special entry in the hash itself is up to you. Or, you could have a hash-of-hashes; that is, a hash in which keys are filenames (strings) and values are hashes of words and counts from that file.

USER INTERACTION:

1. Conduct the pre-processing, or load the hashes from disk, as necessary.
2. Ask the user for a search word (or words).
3. For each word the user enters:
   1. For each file in the corpus:
      1. If the word appears in the file, retrieve its score as calculated above.
4. Rank results as follows:
   1. If the user entered more than one word: Rank in descending order based on number of matches. That is, if the user entered 3 words, begin with the cases where all 3 words were found in the file; then the files with 2 matches; then the files with 1 match.
   2. Within each group, estimate the quality of the match by finding the sum of the ratings, and ranking results in ascending (increasing) order.
      1. Remember, the rating is an estimate of the rarity of a term. A file where all 3 terms are used frequently is a better match than a match where 1 is used frequently and the other 2 are much rarer. But a file with 3 matches is still better than a file with 2 matches.
   3. For each result, show the first line from the file.

Development notes:

* Use functional design principles! Look for opportunities for composition. How will you control state in those cases it can’t be avoided?
* As usual, take it one step at a time. When designing functions, think of the output of each function being the input to the next function (see previous point).
* Don’t Repeat Yourself.
* Parameterize The Things.
* Monoids Are Everywhere.