**Categorization and Analysis of Digital Texts**

Huandong Chang, Mingwenchan Xu, Ruixian Li, Frank Li

**Structure:  
1. Introduction**

**2. Algorithms Analysis and Implications**

**3. Further Steps**

**4. Visualization Attachment**

**Part 1: Introduction**

Digital humanities is a computer-driven field of study involving algorithmic analysis of data in many different forms. This provides opportunities a much larger, more objective, and thorough analysis that can be done orders of magnitudes faster than by human power. This application of computing technology to humanities manifests itself in many ways and across many fields, from digital archiving to finding cultural trends to textual analysis.

This project will demonstrate a method of applying digital humanities to expedite the analysis of texts. Using OCR’ed text (another application of digital humanities) of the Scarlet and Black articles from the 60’s, the project compared the frequency that various academic subjects occur in articles across time periods. Analyzing this trend can help researchers determine the change in discussion of academic topics across time periods, as well as compare the frequency of discussion across topics. This project also had a focus on providing concise, well-designed visualization and analysis for the results of categorization using well-written, easily-adaptable code, which could then be used as a boilerplate or example for future research projects involving this kind of textual analysis and categorization.

**Part 2: Algorithms Analysis and Implications**

This program utilizes three major algorithms: First, the actual categorization algorithm. Second, the visualization algorithm (Visualization attached at the end). And finally, the analysis algorithm. In order to collect the texts into a usable form (list of strings), we created and used a simple directory-to-string procedure that fetched the text from each file. In order to classify the files by year, we used regular expressions pattern matching to classify files by detecting patterns in the filename to create a hash table where the values are lists of strings.

The categorization algorithm itself is designed in a few layers. Basic helper procedures `add-counts!`, `tally-matches`, and `update-values!` were made to perform individual hash modifications and calculations based on values produced by the main algorithm. `tally-matches` is the main procedure call, being used to determine the value each article returns based on word matches. A bias protection limiting the impact of a single word (limiting the maximum tally-count of a single word to 5) is included in the function, and the efficiency of the overall program is highly dependent on the efficiency of `tally-matches.` `tally-matches` is used by the algorithm by applying each string of text to be analyzed to each match string in a hash table sorted by subject. While this kind of organization may sacrifice efficiency, it can be easily modified for more keywords, or even added as a parameter to the overall program to generalize the program.

`count-occurrences` applies `tally-matches` with every match word of each subject through `add-counts!`, which continuously updates a second hash table of counts with updated values for each subject. The result is a multiply-recursive function, although the time-to-compute appears to be linear based on some anecdotal analysis. This one-dimensional categorization algorithm is applied using `map` to a list of strings in the procedure `compile-occurrences`. The procedure `compile-occurrences` applies `count-occurences` to multiple lists of strings, compiling the result into a single value. This function determines the ranges based on an input of a list of lists, where the elements of the outer list are a range of numbers corresponding with years between 1961-1970. Thus, the resulting list’s elements correspond with a given time range. Although this is specific to the data we are analyzing, this can easily be adapted for general use, since, fundamentally, `compile-occurrences` takes lists. We have written a convenient algorithm that allows users to input two-element lists representing these ranges, rather than typing out the lists manually. The overall program uses an algorithm `occurrences->percents` to convert the tallied values of `count-occurence` to a percent of the total tallies. Storing the data in hash tables allows us to easily manipulate the complete sets for other uses, such as certain statistical procedures.

The visualization algorithm takes the results of the categorization procedure `count-occurrences` (which we convert to percents, using `occurrences->percents`), a list of hash tables, and creates an interwoven histogram displaying the results. The building block of the visualization algorithm is the function `make-histogram`. We used this procedure to allow us to generate histograms with unique colors and names (Racket’s plot library allows numbers to be used in place of color symbols or strings), while being interwoven in the final chart. This allows us to recursively generate histograms based the data of our hash tables as in our main `visualize` procedure. The `visualize` procedure takes an input skip (which our overall program ensures is over the number of histograms), ensuring that our final histogram is clear and organized. The parameter `skip` is passed down to each recursive call `rc`, while `rv` recurses through each element of `list-of-percents`, each of which correspond with a time range. This time range is indicated by a corresponding reference list `period-strings`, a list of strings generated by the overall program based on the input ranges.

This visualization algorithm can be very effectively generalized for many procedures. Using the general `make-histogram` procedure, `visualize` can be easily adapted to draw a variety of interwoven histograms with different parameters, making the visualization procedure just as easily generalized as the categorization algorithm.

The analysis algorithm tracks the changes across time of relative frequency based on an input list of tally counts (converted percents) to a user-determined sensitivity. The main procedure `analyze-trends` applies a helper procedure `analyze-subject-trend` to each category/subject. It passes down a category (which is a hash key), for which the `analyze-subject-trend` procedure determines significant changes across time periods, based on a sensitivity input (a decimal form of a percent, e.g. .10 for 10%). `analyze-subject-trend` then returns a string with significant changes and the changing values. `analyze-trends` displays each string to the terminal, in the same way that the histogram formed by `visualized` is displayed.

Although the overall program does not return a value, it is to be noted that each main algorithm in this program can be easily modified to return numerical values that can be used with other programs.

This analysis algorithm, while simplistic, allows researchers to quickly find possible points of study in a huge data set. Rather than providing a full suite of statistical tools, this analysis algorithm was intended to provide an example of how powerful the digital humanities can be when it comes to analyzing text. This tool can be easily adapted for implementation into automated applications or even a larger digital humanities operation.

A simple example of what can be done even with this demonstrative algorithm: Using the overall program `analyze` (which applies all of the algorithms automatically), we created a report of subject frequency every two years from 1961-1966 with a sensitivity of 10%. The automatically-displayed messages provided points of interest when analyzing the graph.

For example, our report indicated a 17% increase in other-humanities occurrences from the years 64-65 to the years 65-66. Looking at the graph, we found that this increase was the final difference in a trend of increasing in the field since 61-62. Likewise, we found that mentions of social-sciences decreased significantly across this period. However, it is obviously not perfect. The automatic analysis did not find the significant decrease of art occurrences over the same time period. Analyzing larger periods (3+ year) resulted in similar analyses with similar behavior.

In general, this was a pretty conservative model. The trends found by the model were generally supported by visual analysis, while visual analysis also found a lot of other trends. In this field, it may be more important to have conservative algorithms than to have riskier ones, since when analyzing even larger sets of data, it is very difficult to manually check the results of data. Thus, having a higher certainty of a results validity is more important.

**Part 3: Further Steps**

For next steps, it is important to refine the efficiency of this algorithm. While having a generalizable algorithm is important, this algorithm will be slow for datasets that are magnitudes larger. If this algorithm will be used for those, it would need significant modifications. In addition, adding tests for different statistical conditions can also be important. These calculations can be taken from any statistics textbook, and can provide a valuable tool for digital humanists categorizing a variety of texts. Different forms of analysis can also be demonstrated.

**Part 4: Visualization Attachment**

