OpenBook Publishers

Digital Humanities Pedagogy: Practices, Principles and Politics

(visit book homepage)

Cover
Contents
Index

10. Teaching Computer-Assisted Text Analysis: Approaches to Learning New Methodologies

Stéfan Sinclair and Geoffrey Rockwell

Using a computer to analyze a text intimidates many humanities students, but the reality is that text analysis is becoming a fundamental and naturalized part of how we operate in a digital society. Text analysis is what enables Google to compile and index tens of billions of web pages so that our search terms produce results; it is fundamental to building IBM's Watson, a computer-system that was able to beat two of the top human *Jeopardy!* players of all time; it allows smartphone developers to build predictive texting capabilities; it also enables a humanist to study the relationship between Agatha Christie's dementia and the richness of her vocabulary over the course of her writing career.[1] Significant transformations of how we handle the written record are occurring as more and more of it is digitized and made available for computer analysis. Analytics are no longer an exotic preoccupation of digital humanists and computational linguists: humanities students need to

understand automated methods if only because we are surrounded by their use—in everything from our email to the news.[2] This chapter will therefore:

- Briefly describe what text analysis is;
- Make the case that analytics should be taught;
- Discuss how it can be integrated into humanities courses;
- Discuss recipes as a way of introducing students to text analysis; and
- Introduce the idea of notebooks for advanced students.

Our goal is to start by making the case for teaching text analysis, then to provide ideas as to how it might be taught, and to end with reflections on advanced support in the form of notebooks—where the analysis becomes a form of research writing.[3]

What is Text Analysis?

Computer-assisted text analysis or text analysis for short, is the use of computers as an aide in the interpretation of electronic texts. A concording tool, for example, can help an interpreter find all the passages in a text where a certain word appears and act as an index to help the interpreter find passages. It can also go further and present these passages with in a Keyword-in-Context (KWIC) display, where one line of context for each occurrence is presented in a new gathered text for the convenience of the author. Thus we can say that text analysis actually consists of two processes:

- **Analysis**, in which the computer breaks apart the text into basic units like words; and,
- **Synthesis**, in which the computer counts these units, manipulates them and reassembles a new text.

The counting and synthesis can become quite sophisticated and go beyond finding. For example, statistical techniques can generate clusters of words for visualization to help you figure out for what to search.

Why bother with text analysis tools? After all, most word processors and web browsers can search texts quickly. In the section of the Text Analysis Developer's Alliance (http://tada.mcmaster.ca/) wiki on "What is text analysis?" we describe what text analysis systems do thus:

- **Text analysis systems can search large texts quickly**. They do this by preparing electronic indexes to the text so that the computer does not have to read through the entire text. When finding words can be done so quickly that it is "interactive," it changes how you can work with the text—you can simultaneously and

serendipitously explore without being frustrated by the slowness of the search process.

- **Text analysis systems can conduct complex searches**. Text analysis systems will often allow you to search for lists of words or for complex patterns of words. For example, you can search for the co-occurrence of two words.
- **Text analysis systems can present the results in ways that suit the study of texts**. Text analysis systems can display the results in a number of ways; for example, a Keyword In Context display shows you all the occurrences of the found word with one line of context.[4]

The issue of scale is important to text analysis. Computers allow us to interpret texts that are so large that we couldn't study them with traditional reading practices. With Google Books (http://books.google.com/) we can search across a million books, more than we could ever read and digest. Thus we can imagine interpreting new collections of texts that we wouldn't before have imagined or dared to interpret. We can bring interpretative questions to these new texts, formalize them for the computer and get results back that we can interpret instead of reading the whole. Franco Moretti, for example, talks about using computers to perform "distant reading,"[5] being able to consider textual evidence inclusively—perhaps even exhaustively—rather than our close reading practice that tends to be exclusive. Stéfan Sinclair's innovative text analysis tool HyperPo (http://www.hyperpo.org/), which led to Voyant Tools (http://www.voyant-tools.org/), was designed to provide an interactive reading environment that does not pretend to answer questions so much as extend the users' ability to read and proliferate representations of texts.

Why Teach Text Analysis?

Until recently, text analysis tools looked like an obscure research area for those interested in programming and statistical techniques. Analytics have, however, become common on the web, and the scale of information available makes the need to teach students about computer-aided interpretation more urgent. Word clouds and Wordles have, for example, become popular on blogs and web pages. Some news sites, including that of the *New York Times*, have started building custom interfaces to help readers analyze transcripts of events, such as the Republican presidential candidate's debate on October 21, 2007. This analytical toy (Figure 1) lets users search for words and see which candidate's used the word(s) and when in the debate.

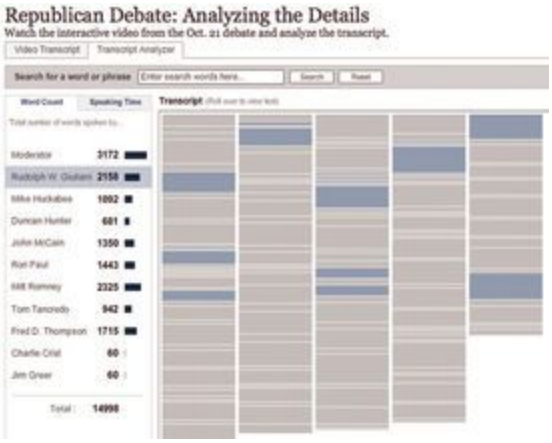


Figure 1. Transcript Analyzer, "Republican Debate: Analyzing the Details," *New York Times*, October 21, 2007, developed by Shan Carter, Gabriel Dance, Matt Ericson, Tom Jackson, Sarah Wheaton and Jonathan Ellis.[6]

As such analytical toys proliferate and as users embed simple analytics in their own web texts students will need to be taught basic analytical literacy so that they can interpret these visualizations just as we (should) teach students to read basic graphs in the news.[7] A further reason is that students need to understand how others are using analytics to study them. Companies and other organizations use analytic techniques to mine the data we freely share on social sites, like Facebook.[8] By trying text analysis, students should learn not only about literature, but also about the potential—and the limits—of surveillance analytics.

Integrating Text Analysis into a Course

Once convinced that it is useful to integrate text analysis learning into courses, one must then consider how to do it. Here are three models that go from a simple and short text analysis assignment—that asks the student to provide the text and the tool, to a more complex model—that requires students to create their own text.

Using an Analytical Tool

The easiest way to integrate text analysis into teaching is to provide access to a pre-populated text analysis tool, by which we mean a tool for interactive reading that already has an indexed text loaded. This way, students do not need to worry about finding an electronic text, preparing it for analysis and loading it. Instead, they can concentrate on thinking through how to use the tool to analyze the text. It is also

easier to support a text/tool combination with which you are familiar. The unfortunate reality of these text analysis tools (in the humanities) is that they are research tools developed by academics, and therefore not as robust as commercial tools. If students are to focus on the text, a combination of text and tool which is known to work (and that can be shown and taught) is needed.

One way to create such an assignment is to use an existing online resource like Mark Davies' Time Corpus (http://corpus.byu.edu/time/), which comes with analytical tools with a linguistic bent. Another way to do this would be to prepare a text yourself (making sure you have the necessary permissions), load it in Voyant Tools, and then ask Voyant to export a stable URL for the tool and text combination.[9] Generally speaking, if you can find a URL for the text you want—on Project Gutenberg (http://www.gutenberg.org/), for example—you can create a Voyant tool/corpus combination for your students. This allows you to provide an analytical environment customized to the text that you are teaching, rather than having to adapt your course to whatever existing text/tool combination is out there.

Once you have a tool/text combination available for students, you need a suitable assignment to encourage students to use the tool. Some ideas for assignments include:

- Provide students with specific patterns of words to search for and to follow through the text in preparation for a tutorial in which you want to discuss the themes associated with those words. This assignment also allows you to discuss the broader issue of whether patterns of words accurately track subtler themes.
- Provide students with a theme or themes and ask them to prepare a short written summary of how the theme unfolds in the text. They may require help imagining how to follow a theme through a text. You can point them to a thesaurus, where they can generate words to search for, count and follow. Encourage students to look also for words indicative of contrasting themes—for example, if they are looking at how women are portrayed in a text, they might also look at how the men are discussed in order to construct an argument by comparison. Simple comparisons of word counts can be interesting and provoke hermeneutical questions—for example, 'what does it mean if, in a corpus of videogame reviews, the words "man," "princess," and "alien" occur far more often than "woman"'?
- Related to thematic analysis, students may be asked to look at the structure of themes by using a distribution graph to see where certain themes are dealt with in the text. Does a theme appear to frame the work appearing at the beginning and end? Are there themes that co-occur or repel each other? This only works with a coherent text, in which there is some logic to the order of the parts of a corpus—for example, a collection of short stories that are not in any order will not show structure, but a collection in diachronic order might. Students can be asked to present a

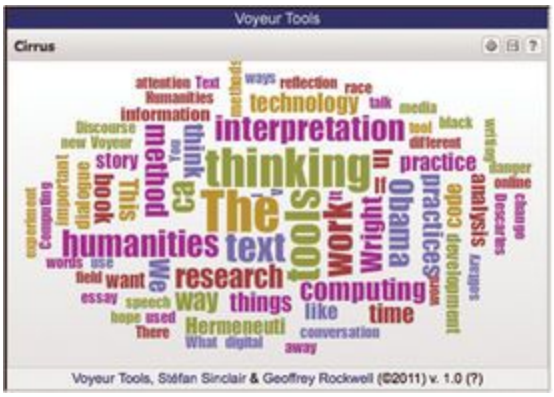
distribution graph in a tutorial to other students, using that visualization to make a point about the text.[10]

When asking students to write from text analysis, one of the challenges is helping them frame an argument that draws on analytic results. This, of course, creates an opportunity to discuss the rhetoric of using computers to demonstrate arguments and claims about a text. Do computer results change the character of the interpretation? Can you prove things with computing in ways that you cannot with human practices? One way to help students to write about results is to ask them to find a visualization that they believe is interesting and to base a short paper on it—including a discussion of how it was generated and what it means. TAPoRware (http://taporware.ualberta.ca) and Voyant Tools have visualization tools, but there are also other collections of tools, such as Many Eyes by IBM (http://www-958.ibm.com/). Many Eyes has the added virtue of a gallery of visualizations produced by others with their texts. Students can look at different types of visualizations and different uses across the same or different texts, helping them to form ideas about what they might generate themselves.

Building a Corpus to Study

The limitation of having students use prepared text/tool combinations is that they don't learn to find texts and build their own corpus. If there is time, students can be asked to develop their own research questions using their own texts. This works particularly well when students are encouraged to gather texts from non-literary sources that often exhibit less subtle uses of language, which means that simple analytical techniques are more likely to generate interesting results. Here is a suggested order of steps for teaching:

1. Ask students to identify a phenomenon they wish to study using text techniques. You could discuss appropriate phenomena in class after having read an introduction or example essay. Encourage students to choose a popular cultural phenomenon for which they might be able to find a variety of texts online.
2. Have students use Google (http://www.google.com/) to find a representative sample of 50 to 100 texts that deal with the phenomenon. They should not use the first 100 hits from Google, but make defensible choices. They should cut and paste the relevant text into a text editor and keep track of the URL where they found each fragment. They should clean up the text and edit out the HTML or XHTML. Most text analysis tools work well with plain text files, so students should be encouraged to save each individual fragment of their corpus as a text file with a ".txt" extension for use in these tools.

3. Students should then run the text through various tools, starting with tools that might suggest themes or anomalies to follow more carefully. We often start with a word cloud (such as the Cirrus Word Cloud included in the Voyant Tools, http://voyant-tools.org/tool/Cirrus/) or Wordle (http://www.wordle.net/), which presents the high-frequency words in the text graphically (Figure 2). The juxtapositions often suggest themes to follow through. A frequency-sorted word list is another tool that you can use to see lots of words. For students, looking for patterns in the high-frequency words can be a way into the text.
4. 

5. **Figure 2.** Cirrus Word Cloud, Voyant Tools, developed by the authors.
6. With words or themes of words that students want to follow carefully, they might look at the distribution of the word, at the words that co-occur with the target word and at concordances of words. Encourage students to take careful notes as they go along so they can recreate a result later. Students might keep a running journal of what they did and their results. This will help them when writing up a paper or presentation for the project.
7. Alternatively, you could have a discussion about what hermeneutical questions we can bring to a text and how text analysis tools can help us formalize a question. Students could be encouraged to have a set of questions and hypotheses before they even touch the computer, since this forces them to look for the tools that might help them answer their questions.
8. You can ask students to look at the Recipes we have developed (discussed below) to get ideas and to see examples.

The goal should be a paper or presentation about the phenomenon, not about text analysis, in which students describe the choices made in creating their corpus, discuss the questions asked through computing, discuss the results and how they were arrived at and present an original reading of popular culture through its texts online. We have found that, given time and support, students can gather surprising results about phenomena they are interested in.

Introducing Text Analysis

However you choose to weave text analysis into your teaching, it is worth introducing the subject explicitly, especially if you want students to think about how computers can be used to analyze data. Here are some ideas for introducing the topic that we have found useful:

- Have the students read a short **introduction** to text analysis. You could use materials on the Text Analysis Developers Alliance site (http://tada.mcmaster.ca/), such as the entry for "What is text analysis?" or more substantial discussions, like the collection of articles on the subject in the 2003 special issue of the journal *Literary and Linguistic Computing* (volume 8, number 2).
- Have students work through an online **workbook**, like the TACTweb Workbook,[11] which walks students through the subject with interactive panels that students can experiment with. Given the flexibility of the web you can even create your own tutorials and workbooks with text analysis tools woven in.[12]
- Have students look at **example essays** that present the interpretative results of text analysis. We have written such an essay, "Now Analyze That," which uses our tools to compare speeches on race by Barack Obama and his mentor Jeremiah A. Wright, Jr.[13] The essay has live Voyant panels embedded right into the essay where we try to make points from the analysis.

Recipes

For students who want to go deeper into text analysis and master it for research purposes, we have developed a collection of "Recipes."[14] The idea of Recipes is to describe text analysis in terms of interpretative tasks that humanities researchers may want to do. Rather than starting with the technologies of analytics and their jargon, Recipes start with something a humanist may want to do, like identifying themes in a text. This allows students to start with the interpretative tasks they should understand.

We chose the name "Recipes" for these abstract tutorials because we wanted to understate the technological and use the metaphor of a cooking recipe to organize what users needed to know. The term was actually suggested by Stan Ruecker at the 2005 Canadian Symposium on Text Analysis, "The Face of Text," during a discussion about how tool rhetoric might be alienating. Like a recipe in a cookbook, each "Recipe" has **Ingredients** and **Utensils** (a list of what you need to compete the task), **Steps** (a sequence of interactions with the computer that will generate results relevant to the task), a **Discussion** of issues that may arise or opportunities for further exploration, a **Glossary** with definitions of text analysis jargon, a section for **Next Steps** or **Further Information** offering suggestions for other Recipes that

might follow and a concrete **Application Example** of the entire process on a specific text with specific questions (Figure 3).

Figure 3. Example “Recipe” Page.

There are currently 29 Recipes on the site (http://tada.mcmaster.ca/Main/TaporRecipes), though some cover pragmatics such as how to handle French texts. We found that we needed some utility Recipes so that the others could be written concisely. These Recipes are all freely available online and may be printed or saved in PDF. Instructions on how to contribute your own “Recipes” to the site are also available.[15]

With all these tools, Recipes and tutorials, the central issue when teaching text analysis is to get students to think methodically and to be able to explain how they developed their interpretations. This issue is not unique to text analysis—students will often have insights into a text that could be brilliant, but they are unable to explain how they arrived at that insight or explain how others might see it the same way. In other words, students often don’t know how to convince us of a perspective verbally/they don’t know how to express their perspective convincingly. When using text analysis tools, especially those that are interactive, students can likewise arrive at some display or result that they cannot recapitulate and, therefore, they don’t really understand. As we outlined above, one possible solution is to teach students to record their analysis carefully and to keep a running commentary of results for later use. Alas, this is like asking students to keep a record of their code when they are more likely to leap ahead and neglect to document their methodical work. With this problem in mind, the next section will deal with a way of thinking about text

analysis tools that offers a new perspective on the recording process, encouraging students to document what they do.

Literate Programming

We will now switch to a more theoretical discussion of where we are going with Voyant Tools. Specifically, we will introduce an idea—literate programming—from one of the key figures of computer science, Donald E. Knuth, which has always struck us as useful for thinking and documenting intellectual processes, including those of analytical research.

Literate programming is essentially a paradigm for writing a highly blended mix of computer code and documentation. Although computer code is almost always accompanied by some documentation, literate programming tends to favor documentation that is more prosaic and exegetical. The objective is not only to describe what the specific lines of code do, but also to provide some narrative context. In a seminal article, Knuth suggests that, “Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.”[16] For Knuth, it is not only the documentation style, but also the coding style that changes, which develops more symbiotically with the narrative. The discursive reasoning embodied in the documentation leads to better code.

A useful analogy is the way that mathematical proofs (or formal logic in philosophy) are expressed. There are equations, but these tend to be illustrations (and interruptions) of the narrative flow. It would be easier to understand the mathematical proof if one took away the equations than if one took away the prose and, as such, we might assert that the equations are there to help better understand the text. With computer programming, documentation is typically included to help better understand the code, but with literate programming the relationship is somewhat inverted in that the human reader is better served by reading the documentation than the code. The code is still crucial, but it is primarily intended for the computer to read. A brief example will help to illustrate the difference, even if the specifics here are contrived:

```
// CONVENTIONAL CODE

// send message to standard output
```

```
print(“Hello World!”);
```

```
[LITERATE PROGRAMMING]

[I’m trying to learn a new programming language and I’ll start by writing a very simple
program. It’s been a tradition since the 1970s to output the text “Hello World!” as a
simple program when first learning a language. This language has a “print” method
that takes as an argument, between parentheses, the quoted string “Hello World!”,
which will appear on the output console.]

print(“Hello World!”);
```

Not only is the literate programming version more verbose, it also makes the program more self-explanatory—it tells a story. The story may not be of interest to more advanced programmers but, just as this story may be of interest to other novice programmers, the stories told by more advanced programmers may be more enlightening to other advanced programmers. The verbose and explanatory style has the potential to serve an important pedagogical purpose.

Crucially, this pedagogical purpose is not only related to how literate programs can teach others, but also how the literate programming style can help programmers teach themselves. Encouraging novice users to explain explicitly how the language structures work in the documentation helps them understand the structures for themselves. Explaining why one is doing something in a specific way may lead to reflection about other ways of doing it, some of which may be preferable. Indeed, part of the documentation may include an explanation of various approaches considered and attempted, as well as a justification of the approach chosen—typical documentation loses any trace of that intellectual work and focuses only on the code that remains.

The focus on expression and rhetoric in literate programming will be very familiar to most humanists: the value of what is expressed is calibrated by how it is expressed (the functional value of the program—whether or not it works—may be distinguished from the rhetorical value of the code). Knuth captures this succinctly in stating that the “practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style.”[17]

Literate programming is a good fit for the humanities since the first priority is prosaic expression (something familiar to all humanities students) and the actual code (which is foreign to most humanists) is secondary. One might even imagine an introduction to programming course in which students begin by writing a procedural

essay and learn to backfill the essay with snippets of actual code as the course proceeds. Such an approach would help to ensure that the programming component does not hijack the overarching humanistic impetus.

The primary challenge for humanities students is probably not learning a new programming language. The syntax can be time-consuming to learn and the formalism of a programming language can be frustrating at first—a missed semi-colon can invalidate several hundred characters of otherwise perfectly valid code (depending on the programming language used), whilst such details do not have such drastic effects in essay writing. The real intellectual challenge is learning to think algorithmically about texts: 'the digital texts are there, now what operations can I perform on them (and why would I want to)?'

For many humanists, the text essentially represents an analogue object, one that is continuous from start to end. Though readers are aware of various structural breaks such as paragraphs and chapters in prose, the text is still considered a singular object. The digital transforms that model by allowing the text to be divided into tiny bits and reconfigured in infinite ways. Common reconfiguration tasks in text analysis include filtering, counting and comparing. However, using computers to perform formal operations on texts does not require humanists to approach texts from a positivistic perspective: we can ask formal questions of texts in service of speculative or hermeneutic objectives. Stephen Ramsay calls this approach *algorithmic criticism*, and explains, "one would not ask how the ends of interpretation were or were not justified by means of the algorithms imposed, but rather, how successful the algorithms were in provoking thought and allowing insight".[18]

The most effective way of helping students think algorithmically about texts is to provide templates and examples. Recipes are a form of template: they describe steps to follow in relatively generic terms. The texts may be different, but the principles and methodologies can have commonalities across applications. Though enormously useful, Recipes can present two major disadvantages for students learning text analysis. First, a lot of technique and knowledge are implicit. Just as most cookbooks would not bother explaining the mechanics of how to, say, whisk eggs, text analysis Recipes should not be expected to explain the purpose and techniques involved in every step along the way. Second, and more importantly, Recipes tend to be teleological, focusing on a singular end-point rather than a process. Algorithmic criticism is not about monolithic breakthroughs made possible by computational methods, but rather about incremental and iterative explorations. Moreover, the intellectual wanderings tend to be idiosyncratic and thus resist any formalization into reusable Recipes.

Whereas Recipes provide conceptual templates with some instructions for achieving an objective, literate programming can be used as a model for providing

specific examples of following an intellectual process. Exposing this process is unusual in the humanities, since scholarly articles tend to be heavily synthesized and reorganized knowledge (which is why it is difficult to teach humanistic methodology based on scholarly articles). Literate programming, on the other hand, inverts the model and allows insights into the process that was followed.

It is worth noting that, despite its conceptual strengths, literate programming has remained relatively marginal in computer science and software engineering. Kurt Nørmark, who states, “it is probably fair to say that the ideas have not had any significant impact on ‘the state of the art of software development,’” outlines several factors that have prevented a wider adoption of literate programming.[19] One pragmatic consideration is that most programmers do not have ambitions to write literary code; writing code that works is more valued than writing documentation that explains. In any case, the literate programming model may actually work better in the humanities because the point is not to run a program (the functional concern) but to read the documentation (the code snippets are merely arguments). For the same reasons, literate programming has had more of an impact in mathematics, where capturing reproducible process and reasoning are essential.

There exist several development tools for those wishing to write works of literate programming. The original system developed by Knuth was called WEB (named in the 1980s before the advent of the World Wide Web) as it was an interweaving of documentation and code in the Pascal language. Other language-specific environments exist (such as CWEB for the C++ language and FWEB for the Fortran language), as well as tools that support any programming language (such as noweb, FennelWeb and nuweb). The documentation in these latter systems can be expressed in a variety of formats, including LaTex, HTML and troff. Similarly, just about any text editor can be used depending on one’s preferences (for syntax highlighting, code completion, etc.) Arguably, the most common use of literate programming is in mathematics and statistics. Sweave, for instance, allows developers to write literate programming in LaTeX and have code executed by R, a widely used open source software environment for statistical computing and graphics. Maple and MATLAB have similar mechanisms available. However, while these tools are not typically difficult to use, the technical barrier for a humanist just starting is significant: programmers are now required to learn two syntaxes (the programming language and the documentation format), and the compilation of documentation and executable code requires additional steps (a tool has to be run to generate the documentation and to extract the code).

A more promising solution is offered by Mathematica, in which documentation is written using a styled editor (similar to writing in Microsoft Word) and code is clearly demarcated in separate blocks. As such, there is no need to learn a separate syntax

for the documentation. Also, critically, code blocks can be executed directly within the environment—removing the need to extract, compile and run them in separate steps. Mathematica notebooks (Figure 4) provide a very compelling model for literate programming: writing documentation is like working in a word processor, which strips away several technical barriers, allowing the new user to focus on learning the programming syntax. Though originally designed for mathematics and currently used in a wide range of scientific disciplines, Mathematica offers some powerful functionality for text analysis in the humanities. Its WordData function, for instance, enables a wide range of operations for English texts, such as looking up synonyms, phonetic transcription, part of speech, and base form (for example, larger and largest becomes large). These specialized text-processing functions, combined with extensive statistical calculations and graphic options, make Mathematica a convincing candidate for literate programming in the humanities. However, Mathematica's major obstacle is its price tag: Mathematica Professional costs upwards of $1,000 USD, though a student license for the Standard Edition is listed at $139 USD. The price tag is arguably justifiable, but probably prohibitive for most humanities programs.

Figure 4. A snapshot of a literate programming notebook in Mathematica. The author of the notebook, William Turkel, is explaining steps to fetch content and analyse it. Only one line of code is visible here, beginning with "response =".

This situation has led us to start developing our own literate programming interface within the Voyant Tools framework.[20]

Voyant Notebooks

Voyant Tools (http://www.voyant-tools.org/) is an online text analysis environment specially designed for humanists who wish to spend more time exploring their corpus than learning complicated statistical and analytical software. It allows

researchers to quickly and easily create their own corpus of texts in various file formats (such as HTML, XML, plain text, Microsoft Word, PDF, RTF and so forth) and begin using an extensive collection of tools to examine interactively and visualize characteristics of the texts. Though the interface strives to have a low technical bar of entry, it also enables more advanced operations. There are currently about 20 analytic tools available, ranging from a simple word cloud visualization to a graph of term distributions, from term frequencies tables to correspondence analysis graphs (Figure 5). Most importantly, these tools are designed modularly, which allows them to be used in different combinations and configurations and to interact in useful ways.

Figure 5. A sampling of visualization and analysis tools available in Voyant Tools.

One of the most innovative features of Voyant Tools is the ability to export tool results from the main site and embed live tool widgets—similar to YouTube clips—into remote digital content, such as blog posts or scholarly essays (Figure 6).[21] This may be considered a type of visual literate programming in that tool results—instead of code—are interspersed with essay argumentation.

Figure 6. An example of a live Voyant Tools widget embedded in a web essay.

Voyant Tools has proven useful in a variety of contexts, including courses in the digital humanities. An example is the historians studying the Old Bailey corpus of 200,000 trial accounts from London. However, its primary limitation is that users are constrained to the existing tools and methodologies that they impose.

Voyant Notebooks, in contrast, allows a much more flexible mode of interaction: the environment exposes the powerful backend system in Voyant Tools and enables users to write custom code to assist in exploring a corpus. User feedback of Voyant has indicated that humanists appreciate standard functionality, but very often wish to pursue a specialized process that would require custom tools.

Just like Voyant Tools, Voyant Notebooks is web-based, which makes it very easy to visit and start using—no downloads or configurations are required. The interface is essentially a web page with editable sections: the documentation sections use a styled editor (where no special syntax is required) and the code sections use a custom code editor (that supports syntax coloring, among other features).

Voyant Script, the programming language, is essentially Javascript enhanced by a library that facilitates communication with Voyant Tools. Voyant Tools is a mixture of interface panels for displaying tables, graphs and other output, as well as a back-end system that executes analytic operations. Many operations would be too slow and too memory-intensive to execute within the browser without making calls to the server. However, this distinction is hidden from the developer in Voyant Notebooks, client-side and server-side operations interact seamlessly. Although still in prototype stage—it is currently not publicly available—an example of Voyant Notebooks will help illustrate the system (Figure 7).

Figure 7. A screenshot from the prototype Voyant Notebooks interface. This shows styled documentation, code blocks and results blocks.

From a programming perspective, the real strength of Voyant Notebooks is that it not only allows all of the operations made possible by the core Javascript language, but also adds convenience methods and interface functionality from Voyant Tools (such as embedding a Cirrus visualization).

Given that Voyant Notebooks is web-based, the intent is to make it as easy as possible for users to find, share, rate, comment and create variations of notebooks. Each notebook will have a unique and persistent URL that allows a user to bookmark and distribute their work. Popular notebooks will be showcased in the hope they provide useful or exemplary examples of what is possible within Voyant Notebooks—the site will have its own social ecology. Above all, notebooks offer a form for sharing text analysis experiments that can both explain an insight and allow someone to recapitulate an experiment. This has potential for teaching students to think and explain what they are trying to do.

Text Analysis Tools and Resources

CATM or **Computer Aided Textual Markup & Analysis** (http://www.catma.de/) is a client-side tool developed by Jan-Christoph Meister. It lets you markup a text and use the markup in analysis, which encourages very close reading and interpretation.

DiRT or **Digital Research Tools Wiki** (https://digitalresearchtools.pbworks.com) is a wiki with links relevant for digital research, maintained by Lisa Spiro. A section of the wiki is devoted to "Text Analysis Tools," which includes links to many of the tools available.

Literary and Linguistic Computing (http://llc.oxfordjournals.org) is the journal of a number of digital humanities associations and published by Oxford University Press. A special issue on text analysis was published in 2003 (volume 8, no. 2), but there are articles on the subject in other issues.

TACT or **Text Analysis Computing Tools** (http://projects.chass.utoronto.ca/tact/) is a DOS-based text analysis and retrieval system developed at the University of Toronto by John Bradley and Ian Lancashire. **TACTweb**(http://tactweb.humanities.mcmaster.ca/) is a version of TACT that runs on the web.

Text Analysis Developers Alliance or **TADA** (http://tada.mcmaster.ca) runs a wiki full of resources and links relevant to text analysis. As a wiki it has some rough parts, but you can obtain a user account and edit it to suit your teaching purposes.

TAPoR or the **Text Analysis Portal for Research** (http://portal.tapor.ca/) is an advanced portal for text analysis tools that has links to web based tools and other resources.

TAPoR Recipes (http://tada.mcmaster.ca/Main/TaporRecipes) are the Recipes discussed in this chapter, which can be adapted to many tools and help students understand how they might use computing to tackle interpretative tasks.

TAPoRware Tools (http://taporware.ualberta.ca) are a set of "primitive" tools that can be used with smaller texts on the web. These tools emphasize individual processes, as opposed to the interactive reading philosophy of Voyant.

TextSTAT (http://www.niederlandistik.fu-berlin.de/textstat/software-en.html) is a simple text analysis tool.

TokenX (http://tokenx.unl.edu) is a web-based tool developed by Brian Pytlik Zillig that presents different views of a text. It has sample texts and can take XML texts.

Using TACT with Electronic Texts is a book by Ian Lancashire in collaboration with John Bradley, Willard McCarty, Michael Stairs and T. R. Wooldridge, originally published in 1996 by the MLA and now freely available in PDF

(http://www.mla.org/store/CID7/PID236). The book has chapters that serve as a manual for **TACT**, as well as chapters that can serve as a general discussion of text analysis or examples of analysis at work.

Voyant Tools (http://www.voyant-tools.org/) is a web-based tool that can handle large texts, developed by the authors Stéfan Sinclair and Geoffrey Rockwell.

Wordle (http://wordle.net/) is a tool for generating word clouds developed by IBM Many Eyes.

Footnotes

1 Watson is a DeepQA system developed by IBM to answer questions posed in a natural language. In February 2011, Watson successfully competed in three episodes of *Jeopardy!* For more about Watson, see the IBM Watson page, February 21, 2001, http://www-03.ibm.com/innovation/us/watson/. Ian Lancashire and Graeme Hirst conducted the research into Agatha Christie's Alzheimer's and her vocabulary, which the *New York Times* selected as one of the notable ideas for 2009. Ian Lancashire, "Vocabulary Changes in Agatha Christie's Mysteries as an Indication of Dementia: A Case Study," in *Forgetful Muses: Reading the Author in the Text* (Toronto: University of Toronto Press, 2010), 207–19.

2 The company Cataphora (http://www.cataphora.com/), for example, provides investigative services using their analytics. The company's website describes their services as follows: "For over nine years, we have pieced together the digital footprints left by individuals and organizations going about their daily lives. We have used these footprints to help clients understand the actions of their employees, and to mitigate the sometimes dangerous consequences".

3 Our research and development of tools was made possible with grants from the Social Science and Humanities Research Council of Canada and the Canada Foundation for Innovation.

4 Geoffrey Rockwell, "What is Text Analysis? A Very Short Answer," *WikiTADA*, Text Analysis Developers Alliance, April 30, 2005, http://tada.mcmaster.ca/Main/WhatTA. This wiki is generally a good resource for text analysis links, documents, and tutorials. Some are suitable for students, however, as a wiki the content is raw and parts are unfinished.

5 Franco Moretti, "Conjectures on World Literature," *New Left Review* 1 (2000): 54–68; "Graphs, Maps, Trees: Abstract Models for Literary History—1," *New Left Review* 24 (2003): 67–93; "Graphs, Maps, Trees: Abstract Models for Literary History—2," *New Left Review* 26 (2004): 79–103; and, "Graphs, Maps, Trees: Abstract Models for Literary History—3," *New Left Review* 28, (2004): 43–63.

6 You can still experiment with this Analyzer tool on the *New York Times* site, October 21, 2007, http://www.nytimes.com/interactive/2007/10/21/us/politics/20071021_DEBATE_GRAPHIC.html#video.

7 On embedding analytics, see Geoffrey Rockwell, Stéfan Sinclair, Stan Ruecker, and Peter Organisciak, "Ubiquitous Text Analysis," *The Poetess Archive Journal* 2, no. 1 (2010), http://paj.muohio.edu/paj/index.php/paj/article/view/13/.

8 For an accessible article on how social media data is being used, see Jeffrey Rosen, "The Web Means the End of Forgetting," *The New York Times Magazine*, July 21, 2010, http://www.nytimes.com/2010/07/25/magazine/25privacy-t2.html. There is also some evidence that the FBI in the United States developed large-scale analytical tools for surveillance purposes. See the Electronic Privacy Information Centre's documentation on the Carnivore and Omnivore systems

reported in 2000, January 19, 2005, http://epic.org/privacy/carnivore/. It is not clear, however, how effective such tools are, even if we can clearly see that they are pervasive.

9 When you load a text in Voyant Tools, you can ask for a URL to be able to access the corpus again by clicking on the floppy disk icon in the upper-right corner. It will give you a URL or the option to go to a layout manager where you can create a "skin" with exactly the tools in the Voyant portfolio that you want your students to use.

10 John B. Smith's article, "Image and Imagery in Joyce's *Portrait*: A Computer-Assisted Analysis," in *Directions in Literary Criticism: Contemporary Approaches to Literature*, ed. Stanley Weintraub and Philip Young (University Park: Pennsylvania State University Press, 1973), 220–27, is a nice early example of this type of analysis if you want an exemplar for students to read. Smith was a pioneer in the development of text analysis tools, with one of the first interactive text analysis tools called ARRAS. He also experimented with visualizing texts before we even called it visualization.

11 TACTweb was developed in the mid 1990s and is now a bit dated, but it is still online at a number of universities. For example, an installation of the workbook can be found at http://khnt.hit.uib.no/tactweb/doc/TWIntro.htm. For more on TACTweb and its use as a teaching tool, see Geoffrey Rockwell, Graham Passmore and John Bradley, "TACTweb: The Intersection of Text-Analysis and Hypertext," *Journal of Educational Computing Research* 17, no. 3 (1997): 217-30.

12 The Text Analysis Developers Alliance site maintains a list of other text analysis tutorials. See "Tutorials and Documentation," *WikiTADA*, Text Analysis Developers Alliance, May 6, 2005, http://tada.mcmaster.ca/Main/TATuts.

13 Stéfan Sinclair and Geoffrey Rockwell, "Now Analyze That: Comparing the Discourse on Race," *The Rhetoric of Text Analysis*, April 2, 2009, http://hermeneuti.ca/rhetoric/now-analyze-that.

14 These Recipes were developed by Shawn Day, under our supervision, as part of the CFI-funded TAPoR project (http://portal.tapor.ca/). While they point to specific tools in the TAPoR project, the Recipes were written in such a way to allow others to substitute different tools. We are currently in the early phase of developing a Methods Commons (http://hermeneuti.ca/methods), where people can post and comment on such methodological recipes.

15 Shawn Day, "Contributing Your Own TAPoR Portal Recipes and Exercises," *WikiTADA*, Text Analysis Developers Alliance, April 18, 2007, http://tada.mcmaster.ca/Main/RecipeStructure.

16 Donald E. Knuth, "Literate Programming," *The Computer Journal* 27, no. 2 (1984): 97, emphasis original.

17 Knuth, "Literate Programming," 1.

18 Stephen Ramsay, "Toward an Algorithmic Criticism," *Literary and Linguistic Computing* 18, no. 2 (2003): 173.

19 Kurt Nørmark, "Literate Programming: Issues and Problems," Department of Computer Science, Aalborg University, August 13, 1998, http://www.cs.aau.dk/~normark/litpro/issues-and-problems.html.

20 We had earlier developed for the TAPoR Portal a notebook-like feature, in which you had a blog to journal what you were doing and could save results. Though promising, its performance was often slow.

21 See also "Viral Analytics: Embedding Voyant in Other Sites," Voyant Tools, October 19, 2010, http://hermeneuti.ca/Voyant/embed.