Digging Deeper Reaching Further

Libraries Empowering Users to Mine the HathiTrust Digital Library Resources

Module 4.2 Performing Text Analysis: Basic Approaches with Python Instructor Guide

Further reading: go.illinois.edu/ddrf-resources

Narrative

The following is a suggested narrative to accompany the slides. Please feel free to modify content as needed when delivering workshops. Additional information and examples that are provided to aid the instructor's understanding of the materials are in *Italic text* – whether to present them to workshop audiences is entirely up to the instructor.

Slide M4.2-1
This lesson, which focuses on supporting advanced researchers, explains how to read an HTRC Extracted Features file and analyze it using a Python script run from the command line on their computer.

Slide M4.2-2
In this module, we will review text analysis strategies for the more advanced researcher, including exploring text analysis methods in more depth than we've done so far. We will focus mainly on using the HTRC Extracted Features dataset for doing exploratory data analysis. For this, we'll learn a bit more about using Python and will use a Python library, the Feature Reader, to explore word counts in our workset. Finally, we will see how Sam analyzed his Creativity Corpus in our case study section.

Slide M4.2-3

By the end of this module you will have used a Python to create a list of top adjectives in a

volume and a graph that visualizes word count across a volume so that you can get a feel for the features of the volume.

Slide M4.2-4

First, let's look at some key approaches to text analysis. One key approach is **Natural Language Processing (NLP),** which is using computers to understand the meaning, relationships, and semantics within human-language text. Generally, for natural language processing, full text is needed. It is not a bag-of-words method. Some common, specific methods under NLP are:

- **Named entity extraction**, which uses computers to learn about what names of people, places, and organizations are in the text.
- **Sentiment analysis**, which uses computers to explore what emotions are present in the text.
- **Stylometry**, which uses computers to speculate who wrote the text based on language style.

Slide M4.2-5

Another key approach to text analysis is **Machine Learning**, which is training computers to recognize patterns in text without explicit human programming. Machine learning can either be unsupervised (with minimal human intervention) or supervised (with more human intervention). Here are some common, specific methods that are based on machine learning:

- **Topic modeling**, which explores the thematic topics present in the text. Remember that topic modeling is a bag-of-words approach.
- **Naïve Bayes classification,** which explores the categorization of texts. It involves determining what categories does a certain text belong to, based on the categories that the researcher has named.

Slide M4.2-6

Let's review what we just covered. On the screen are each of the examples we looked at in Module 1, the introduction. Can you identify which broad area of text analysis each example falls in and what the specific method was? These broad areas/methods are defined in the previous two slides that we just went through.

*Note: you can review the projects online (*http://go.illinois.edu/ddrf-research-examples)

(If time runs short, cut the time down for this activity.)

Slide M4.2-7

Let's orient ourselves in the text analysis workflow. There is an additional step between preparing the data and performing the analysis, and that is translating the text into features. You need to get the text into a form that a computer can crunch – often that involves some kind of abstraction into counts or other numeric representations.

Slide M4.2-8

This concept of translation to features is especially important in the HTRC, where a dataset called Extracted Features has been made available.

- Extracted Features dataset is one way the HTRC tries to facilitate text analysis outside of pre-built tools to give researchers more flexibility for their work.
- The dataset is a downloadable, structured dataset consisting of 5 billion pages within 13.6 million volumes of content, pulled from the entire HT corpus.

The link to the Extracted Features dataset webpage is shown on the slide:
https://analytics.hathitrust.org/datasets#ef

Slide M4.2-9

The "features" that are extracted are page- and volume-level data and metadata pulled from the raw text. They position the researcher at a point where they can begin their analysis. The text has been tokenized, and some things like counts and part-of-speech tagging is already done. It's a form of non-consumptive access that allows the researcher to analyze the entire corpus through the extracted content without having access to the full, expressive text.

Slide M4.2-10

At the volume-level, the features are mostly pulled from the bibliographic metadata.

- They include things like title, author, and language that are included in the library catalog metadata.
- They also include the unique identifiers for the volume, such as the HathiTrust ID, which is consistent across HathiTrust tools and services, and the OCLC number.
- You can see an example of volume level metadata in the Extracted Features JSON format on the slide.

Slide M4.2**-**11

For each page in the volume, there is also metadata.

- This includes the sequence of the page in the volume, as well as computationally-inferred metadata, such as the number of words, lines, sentences, and empty lines on the page, as well as the language for the page in question.
- You can see an example of the metadata included for each page within an Extracted Features file.
- For the language, “en” here means English.

Slide M4.2**-**12

As we’ll see, Extracted Features pull out metadata for each section of each page: header, footer, and body.

- The image here on the screen shows the different sections of two pages from *Public papers of the presidents of the United States (Gerald R. Ford volume, book 2)*.
- Extracted Features help the researcher navigate the para-textual information to get to the pieces most relevant to them.

Slide M4.2**-**13

As we’ll also see, Extracted Features can provide counts of characters occurring at the beginnings and end of lines. This can help identify tables of contents, indices, and title pages, as well as to differentiate poetry from prose, because they all have notable different patterns in the types of characters that occur at the beginnings and ends of lines. For example, we can notice some patterns here from the table of contents of *Public papers of the presidents of the United States (Gerald R. Ford volume, book 2)*. Or, for another example, English-language poetry from some time periods is highly likely to start each line with a capital letter, and end each line with a lowercase letter or punctuation mark.

Slide M4.2**-**14

The page-level features are further broken down by section of the page, the boundaries of which are computationally determined. This makes it so a researcher can drop headers and footers from their data easily.

- These features include line counts, empty line counts, and sentence counts, and counts of the characters that begin and end lines.

- There are also part-of-speech-tagged tokens and their frequencies. The part of speech codes are from the Penn Tree Bank. And homonyms are counted separately: For example, you can differentiate between Rose the name, rose the flower, and rose the verb.
- On the slide, there's an example of the body section for a page in an EF file.

Slide M4.2**-**15

- Using the HTRC Extracted Features, a researcher can do things like identify the parts of a book from the computationally-derived descriptive metadata.
- They can also perform any algorithm that relies on the bag-of-words model, including topic modeling and Dunning's log-likelihood. Because word order is not preserved, the EF datasets cannot be used for things where that info is important, like in sentiment analysis.
- The researcher could also try a machine learning project to validate the bibliographic metadata. We'll see an example of this later.

Slide M4.2**-**16

Let's transition now to talking about how more advanced researchers approach text analysis, keeping in mind the methods we discussed earlier and the HTRC Extracted Features.

- Some experienced researchers are going to want more control over their text analytic methods than they can get using built-in algorithms and tools.
- They may be uncomfortable with being unable to finely-tune their research processes in plug-and-play tools.
- They will likely mix-and-match their approaches by creating an individualized toolkit of methodological strategies.

Slide M4.2**-**17

This toolkit is highly dependent on the individual. Generally, the toolkit will require the researcher to have an understanding of statistics in order to be useful. A researcher may also rely on expert collaborators to wield some of the tools. The toolkit itself is made up primarily of command line tools and programming languages the researcher uses to process and analyze the textual data.

Note: If asked some common tools and programs include: Command line skills, Python and/or R, Visualization tools like Gephi or d3.js, or packages such as Stanford NLP or MALLET.

Slide M4.2-18

Two common command line tools are MALLET and Stanford NLP.

- MALLET runs from the command line to do topic modeling and text classification.
- Stanford NLP also runs from the command line to do natural language processing.
- These tools must be downloaded to be used. You don't need to know Java to use them, that's just the language they are written in.

Slide M4.2-19

As mentioned, the researcher may also need programming languages. While programming-based text analysis is more "do-it-yourself" than using pre-built tools, no one writing code is starting at ground zero.

For example, a researcher might borrow or modify someone else's scripts.

Even if a person is writing a script from scratch, they will rely on so-called libraries, which are collections of shared code. In general, you download the existing code in the form of "packages" and these packages are collections of "modules." You could think about the package like the screwdriver kit and the modules as the screwdriver heads. Each of the heads (modules) performs a specific task.

The kinds of programming tasks that a package for text analysis might do includes preparing, reading or loading, and analyzing text with preset routines.

If there are questions: Note that "library" is a less formal term in Python and general refers to the distribution of prewritten code. May consist of one or more packages and modules.

Slide M4.2-20

This tweet from Ben Miller, a professor at the University of Pittsburgh, sums up the process experienced by many emerging programmers of writing Python code and using libraries.

- You have a problem you want to solve that you think code might help solve.
- You try for a long time to write a script on your own. You get frustrated.
- You Google, and you find an existing library that does the thing you want to do!
- Then you spend a long time figuring out how to use the library.

This is a "problem" that experience solves, but it's helpful to remember to check first (using Google or a website called StackOverflow) to see if there is an existing library that does what you are trying to accomplish.

Slide M4.2-21

Packages need to be installed. Once a package is installed, it is then used in a script or program. The installation process is most easily done using a package manager. We're not going to spend a lot of time discussing the details of what package managers do, but it's important to know that they handle the download and installation process for you.

- Pip is a common package manager when dealing with Python, and it's generally included when you install Python. That means you don't have to do any extra set-up to be able to use pip once you have Python going on your computer.
- The basic syntax for pip installing something is to execute a command on the command line that looks like "pip install <the name of the thing you want to install>". Pip then goes out, gets the thing it's supposed to install, and downloads it for you.
- Other package handlers include Homebrew and Conda. Conda especially is good for setting up a programming environment for Python-based data analysis, such as Jupyter notebooks.

Slide M4.2-22

Once the package is installed, it is then used in a script or program. Here's the script we are going to run later in the activity, and let's take a slightly closer look:

- At the top, we see that it imports several modules, which are components of packages we installed. In particular, On line 3 we see that "sys" is imported as one of the modules, and then on line 12 we see that it is called so that we can ask for command-line arguments for this script. Note that we don't have to explain what sys does because all that information is stored elsewhere on our computer. We are able to reuse its functionality without building or defining it. Sys is a special module that comes included in Python, so we didn't even have to install it!
- Other modules you will have to install, and we'll talk about that process next. First though, notice in line 1 that we see "from htrc_features import FeatureReader" which is importing just the FeatureReader module from something called "htrc_features." It's important to note that we are being specific about the module we want - you import only what you'll need because libraries are very large and if you are installing them from a 3rd party, modules across packages may share the same name.

Slide M4.2-23

In Python, common text analysis packages are SciKit Learn, which does machine learning, and Pandas, which is a general data science package, and the Natural Language Toolkit (NLTK), which does natural language processing.

Here's an example that might get included in a text analysis script, 'nltk.words_tokenize()'. Can you guess what this does?
It will chop the text into tokens. *Remember that tokens are generally single word fragments of text, where punctuation is discarded.*

Slide M4.2**-**24
The HTRC Feature Reader is a Python library for working with HTRC Extracted Features.

- You can analyze the features without the library, but using the library allows us to incorporate some already-written code into our analysis.
- Because it's a specialized library and is not standard in Python, it will need to be installed.
- We'll also need to have the package Pandas installed in order to use the Feature Reader because it makes use of some Pandas functions. Remember that Pandas is a Python library for working with data. Luckily, it's included in Python Anywhere, so we won't need to install it.

Slide M4.2**-**25
Let's return to our sample reference question and think about how one might use extracted features in the analysis. We can look at only the adjectives for the volumes in your political science workset to get a feel for the content of the volumes.

Slide M4.2**-**26
In this activity, we will run a Python script to create a list of the most-used adjectives in your political science workset.

- The files we're going to be working with are Extracted Features files for 1970s presidential speeches. For later, you'll also need to have Extracted Features files of similar volumes from the 1930s on hand.
- You will need to have the files (and directories) in PythonAnywhere as shown on the slide. In the next slide, we will show you how to check if everything is in place.

Screenshots are for backup only. Instructor will demo live in workshop.

Slide M4.2**-**27
First, let's all make sure we have the files and directories we need.

- After unzipping the “activity_files.zip” file and moving the contents to your user directory in our hands-on activity in module 2.2, the 1930 and 1970 directories and other needed files should already be in your PythonAnywhere user files.
- To check if the files and directories are there, go to your PythonAnywhere dashboard and click on the “Browse files” button. On your “Files” page, you will see your directories on the left and your files on the right. You should have two directories “1930/” and “1970/” listed on the left.
- Click on each directory to the files inside. There should be 5 json.bz2 files in the “1930/” directory and 16 json.bz2 files in the “1970/” directory.

The subsequent screenshots are for backup only. Instructor will demo live in workshop.

Slide M4.2**-**28

The script we’ll run incorporates the HTRC Feature Reader.

- In order to run it on our computer (or in PythonAnywhere) we need to have the library installed so that the computer know how to implement the Feature Reader- specific functions.
- Remember that pip is our package manager.
- Our command primes pip first, then asks it to install something for us. In PythonAnywhere, we need to use the --user flag to note that we want to install the library under our user account. Then we tell pip what we want to install, in this case it’s the htrc-feature-reader.

Type command in your Bash console: *pip install --user htrc-feature-reader*

Slide M4.2**-**29

Let’s take a little time to examine the script that we’ll be using. You can see it on the screen, or open the file in PythonAnywhere – remember the file is called “top_adjectives.py.” Here in the script, we again can see the way the relevant modules are called at the top of the script.

- We see the Feature reader, Pandas, and something called “sys” that allows us to pass arguments to the script are all imported. We also have “glob” imported, which helps with directory paths.
- We can also see the Penn TreeBank adjective code “JJ” that shows we are only pulling out adjectives.
- Does this script require an argument? (*Yes, in line 6)*

Slide M4.2-30

In Bash shell, type command and run: **python top_adjectives.py 1970**

- Our argument is “1970” – and it fills in the directory path for us so that we can look at all the files in that directory.
- Hit ”enter” to run. It may take a while to process.

Slide M4.2-31

See the output of the script, which are the adjectives and the total number of times they occur over the volumes in the 1970 presidential speeches and papers in our workset.

- What do you see? Anything that strikes you as interesting?

Slide M4.2-32

Now, working with your neighbor, output the top adjectives in our 1930s volumes. How do you need to change your command to get the desired results? What differences do you see in the two outputs? Similarities?

If you want a challenge, see if you can modify the script to search verbs or another part of speech.

Instructor organizes activity.

- *To output top adjectives in our 1930s volumes, use 1930 as the argument when running the script in place of 1970. Command should be changed to:*
  `python top_adjectives.py 1930`
- *For the challenge: open the file, and replace JJ with another code from the Penn Tree Bank (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html), such as VB for verb.*

Slide M4.2-33

Let’s dive more deeply into the idea of visualization as an exploration strategy.

- Exploratory data analysis is an approach to becoming familiar with a dataset
- The idea behind exploratory data analysis is that it is difficult to grasp the contents of a dataset by just reading a spreadsheet or looking through files, especially when one is working at a large scale.
- There are whole books written about exploratory data analysis. This discussion barely scratches the surface.

- There are several general strategies for exploratory data analysis that involve visualizing the dataset:
    - Plot raw data
    - Plot simple statistics
    - Compare plots to look for patterns

Slide M4.2**-**34

We'll discuss visualization more in the next module, but for now, here is a list of common graphing libraries in case you want to make note of them.

- In Python, there is the pyplot function in the library matplotlib, and there is also a library called ggplot.
- Other, non-Python libraries include ggplot2 in the programming language R and d3.js, a JavaScript library for visualizations.
- It's not important that you know or remember these now – just that you've seen the names to know they exist.

*One reason some researchers like to use **R** versus Python is because of its visualization capabilities. Some find it easier to visualize with R, and the package many use is ggplot2. There are wars in the data science and programming communities about R versus Python and ggplot versus ggplot2. It's not important to understand the details of this preference-battle, but it might be useful to know that opinions about which is better are strongly held.*

Slide M4.2**-**35

Now let's try our hand at a little exploratory data analysis. We'll return to our sample reference question and plot raw data in an HTRC Extracted Features file.

Slide M4.2**-**36

For this activity, you will create visualizations using the visualization function in Pandas, called pyplot, to see the word count over a volume based on its Extracted Features file. We'll also try our hand at a few other different exploratory techniques.

The files that you will need are shown here on the slide.

Slide M4.2**-**37

First, have a look at the first script we are going to run. You can see it on the screen, or open the file in PythonAnywhere - remember the file is called "word_count.py."

- Notice at the top we are importing the modules and libraries that we'll call to achieve our visualization. Are there any in that list you are unsure about?
- Then see that the script is asking for an input – those are the arguments expected by the script – and in this case it's the name of the file.
- Finally, you can see at the bottom of the script that it will output an image file as png.

Slide M4.2**-**38

Then, open your shell to run your script. The script file is word_count.py. Can you input the command on your own? Or you can find it on the screen if you need a hint.

Slide M4.2**-**39

Remember that the script saved our visualization as an image file. Let's go to our files in PythonAnywhere to see it. The file name should be words.png. Click on the download icon to the right of the file name to open the file to see what you produced.

The graph shows how many words appear on each page in the volume. Page numbers are on the horizontal axis, and the word count is on the vertical axis. For example, we can see the 600th page has about 350 words.

Is it what you expected?

Slide M4.2**-**40

Now, let's try to look at the word count of another volume by modifying the script. Edit, save, and then run the file, and discuss the process and results with your neighbor.

If you want a challenge, see if you can modify the script to search adjectives or verbs.

How to modify the script to look at the word count of another volume:

- *To make the script look at another volume also in directory 1970, change the path to another volume in line 10 on the script. This means changing the part in **bold** in this line of code:*

  *path = glob.glob('1970/**mdp.49015002203033.json.bz2**')*
- *You can go to the "1970/" directory from the "Files" tab on your Dashboard to find a new volume, and copy and paste the name of the new volume to the script. For example, change line 10 of the script to: path = glob.glob('1970/ mdp.49015002221761.json.bz2')*

- *If you want to the script to look at another volume under the 1930 directory, you will also need to switch the "1970/" part in the initial code to "1930/". For example, change line 10 of the code to: path =glob.glob('1930/*miua.4925052,1928,001.json.bz2*')*

Slide M4.2-41

Even basic counts can lead to sophisticated research outcomes. Earlier in the workshop, we saw an example of machine learning research done by Ted Underwood. He used the HTRC Extracted Features in this research. He classified texts in the HT corpus by genre.

Even basic counts can lead to sophisticated research outcomes. Earlier in the workshop, we saw an example of machine learning research done by Ted Underwood. He used the HTRC Extracted Features in this research. As a reminder, he classified texts in the HT corpus by genre.

- Using machine learning methods, Ted identified (at the page level) unigrams (single words) that are prominent in the genres of prose fiction, drama, and poetry. The genres are assumed to be non-overlapping, discrete categories.
- These genres were chosen because they are easier to map than more specific subgenres, where categories start to blur, such as "detective fiction."
- Ted released his derived dataset, which are comma-separated-value files of word counts by genre by year and made them available for others who want to incorporate them into their own analysis.
- Even though Ted wasn't working with full text as his original data, he was able to do sophisticated research.

Slide M4.2-42

Let's see how Sam made use of HTRC Extracted Features.

- Once he had his creativity corpus prepared, he performed topic modeling on only those pages he had identified as containing a form of "creative."
- That way he was able to see what themes or topics appeared around the concept of creativity in the literature.
- He then graphed the topics to see how their prevalence changed over time.

Slide M4.2-43

Here are the topics that decreased in prevalence from 1940 to 2000. They include:

- god, christ, jesus, creation, word

- species, animals, natural, plants, soil
- nature, mind, creative, world, human
- invention, power, creative, own, ideas

From these topics, Sam was able to argue that the usage of “creative” changed over the course of the twentieth century be become less related to words like “generative” or “productive”.

He first experimented with topic modelling using a common technique called Latent Dirichlet Allocation (LDA), which is the method described in the last module that is used by most out-of-the box tools. There was a potential problem with using it, though. Typically in LDA, the order in which texts are sampled is either chronological (starting from the beginning of the list) or randomized to control for time. Sam wanted to be to use time as factor, because he wanted the model to be able to identify topics specific to their particular eras, but he did not want older topics to be drowned out by the massive number of works published in the latter-years of his period of study. With help, he developed code for temporally weighting the training sample (chronologically, by decade, randomizing within each decade) in order to soften the temporal bias without entirely removing it.

Slide M4.2**-**44

Next we can see the topics that increased in prevalence from 1940 to 2000. They include:

- advertising, media, marketing, sales, television
- economic, development, capital, economy, production
- poetry, language, poet, poets, poems
- social, creative, study, development, behavior

Using these results, Sam made the argument that the usage of creative became more related to art or imagination.

Slide M4.2**-**45

Let’s wrap up this module with a group discussion. In groups, please think about the following questions: In what ways can librarians support advanced text analysis research? What additional skills would you need to learn in order to do so?

Short discussion of around 5 minutes. If there is clearly not enough time and instructors would like to save more time for the hands-on activities with Bookworm in the next module, cut this discussion completely.

Slide M4.2**-**46

That’s all we have for this lesson, and we will be happy to take any questions from you.

Slide M4.2**-**47

(*Show references so attendees know where to find them later.)*