Pace of Change:
A Preliminary YesWorkflow Case Study

Megan Senseney
January 2016
CIRSS Technical Report 201601-1

Center for Informatics Research in Science and Scholarship
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign

Problem Statement

YesWorkflow (YW) is a grassroots initiative that "aims to provide a number of the benefits of using a scientific workflow management system without having to rewrite scripts and other scientific software."[1] YW represents an improvement to workflow modeling by allowing scientists and scholars to bypass traditional workflow engines by utilizing a small suite of modeling annotations that can be integrated as comments directly into an existing script.

To date, YesWorkflow is still in its prototyping and development phase, but a handful of early adopters have already begun piloting the tool in a variety of scientific domains. To test the potential of utilizing YesWorkflow in a digital humanities context, this case study explores Professor Ted Underwood's research outlined in the 2015 article "How Quickly do Literary Standards Change", co-authored with Jordan Sellers and posted to FigShare[2] with a link to the project's GitHub repository.[3] In keeping with the GitHub repository, Underwood's project will be referred to as Pace of Change throughout this report.

Approach

Understanding the Use Case

The Pace of Change project begins by creating "two samples of poetry and fiction across the century 1820–1919: one drawn from reviews in prominent Anglo-American periodicals that reviewed literature selectively, the other drawn at random from HathiTrust Digital Library, which contains at least 146,000 English-language volumes of poetry and fiction in this period, many relatively obscure" (Underwood & Sellers, 2015). In brief, the goal of the experiment is to use the word frequencies from each of these files to train a machine to predict (based solely on word choice and frequency) whether a given volume was reviewed in a prominent periodical. Whether or not a volume is reviewed serves as an indication of literary prestige. Being able to predict whether or not a volume is reviewed indicates common traits within the language of poetry that are deemed prestigious or otherwise, thus suggesting long-term trends in literary reception.

The GitHub repository for this project contains a directory of 728 tsv files in a folder called "poems", a folder called "results" with two csv files, a csv file called "poemeta" (which provides bibliographic metadata about each of the files included in the "poems" directory), and eight Python scripts (four of

[1] https://github.com/yesworkflow-org/yw-prototypes
[2] http://figshare.com/articles/How_Quickly_Do_Literary_Standards_Change_/1418394
[3] https://github.com/tedunderwood/paceofchange

which are required to replicate the primary results of the experiment conducted in "How do Literary Standards Change?").

Analyzing the Code

The primary Python script is replicate.py. Underwood designed this script as a package that collects user-determined parameters and passes them to parallel-crossvalidate.py, which in turn calls metafilter.py to crosswalk between the metadata and poetry files (to determine which IDs to use) and also calls modelingprocess.py which conducts the leave-one-out predictions within the create_model fuction.

Applying YesWorkflow

Utilizing tools from the YesWorklow Github and concepts from McPhillips, Belhajjame, and Ludäscher (2015), the original plan for the case study was to create a prospective provenance graph using URI templates and then run the scripts with the goal of extracting facts about the run to a dlv file utilizing YW's extraction configuration. Using the extracted facts, I would then conduct a set of retrospective provenance queries against the data. As described further in the Progress and Findings sections, working through this case study uncovered some current limitations to the YW tool and some exciting new directions for development.

Progress

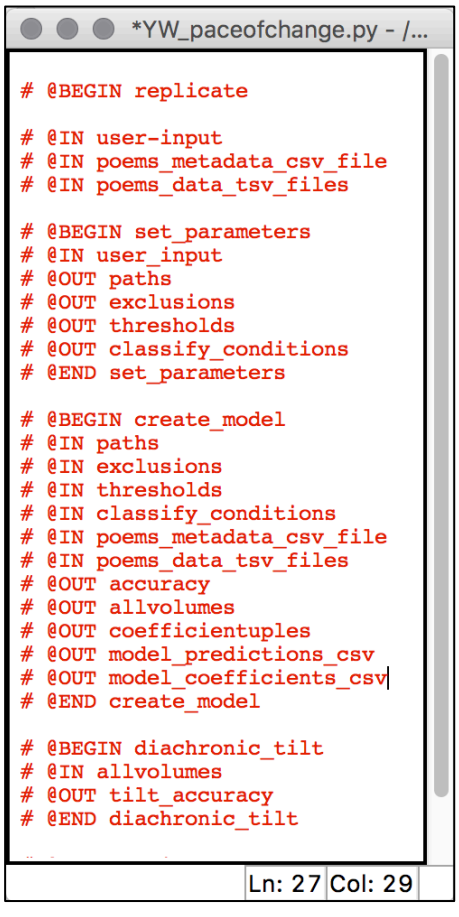

Fig. 1 YesWorkflow annotations for replicate.py

Prospective provenance

Since replicate.py serves as the project's main script, I used it to create a simple conceptual model of the core processes inherent to the project. Essentially, the script begins by prompting the user for input, which is used to set parameters for the remainder of the scripts' processes. These parameters are then passed to a complex function called create_model that resides in parallel_crossvalidate.py. The create_model fuction creates several outputs: two csv files and three values that are passed back to replicate.py. Accuracy is shared with the user in a print statement, and allvolumes is passed to another fuction called diachronic_tilt that also resides in parallel_crossvalidate.py. This function returns an adjusted tilt_accuracy, which is shared with the user in another print statement.

Due to the fact that I regularly toggled between the four Python scripts while preparing my conceptual model, I chose to model replicate.py in a standalone Python file called YW_paceofchange.py. This helps avoid confusion when reviewing the inputs and outputs of create_model, only some of which are explicitly named in the replicate.py script. From within the create_model function, parallel_crossvalidate.py retrieves the metadata file and poetry files. The output csv files are also written and saved as new files from within the create_model function in parallel_crossvalidation.py. Once the simple YW annotations were complete, I generated graphviz files and a pdf image from the command line using the YW tool:

```
lism01-mfsense2:YesWorkflow mfsense2$ yw graph YW_paceofchange.py -config graph.view=combined
 > YW_paceofchange.gv
lism01-mfsense2:YesWorkflow mfsense2$ dot -Tpdf YW_paceofchange.gv -o YW_paceofchange.pdf
lism01-mfsense2:YesWorkflow mfsense2$ open YW_paceofchange.pdf
lism01-mfsense2:YesWorkflow mfsense2$
```

The provenance graph is included in Figure 2 below.

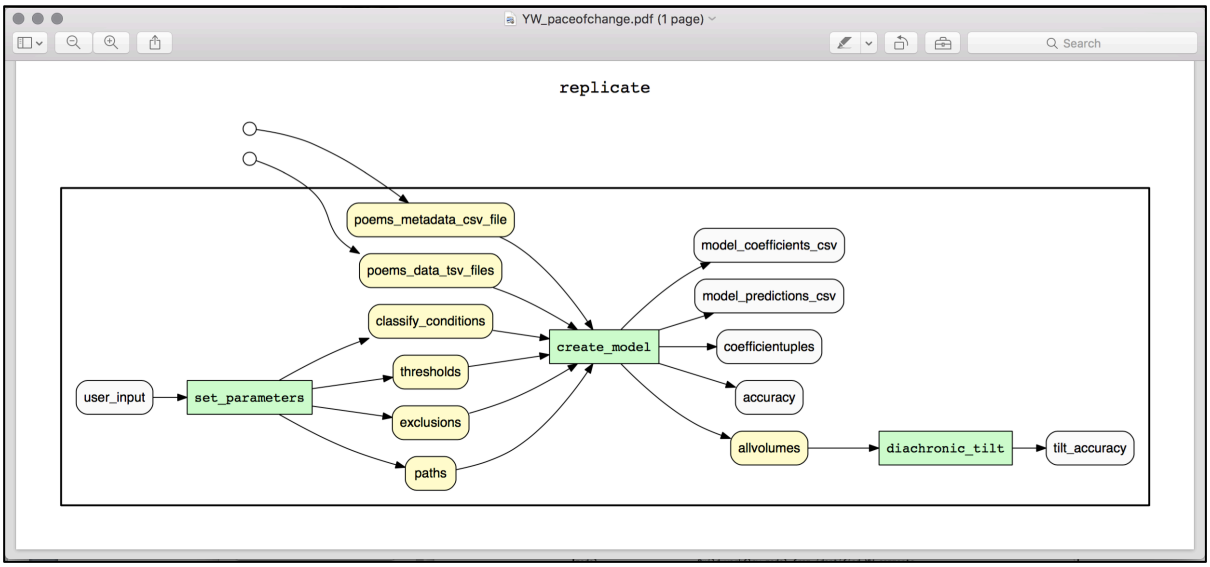

Fig. 2 A provenance graph of key fuctions in replicate.py

Create_model

While replicate.py provides the best high-level conceptual overview of the project, a number of interesting processing steps occur within the create_model function, including interactions with the other two required python scripts: metafilter.py and modelprocessing.py. Time permitting, I would have liked like to create a second provenance graph dedicately solely to the data and processing steps that occur within create_model.

Retrospective provenance

As stated above, the original project plan was to create a prospective provenance graph using URI templates and then run the scripts with the goal of extracting facts about the run to a dlv file utilizing YW's extraction configuration. Using the extracted facts, I would then conduct a set of retrospective provenance queries against the data. While analyzing Underwood's GitHub repository, however, I discovered some impediments to conducting meaningful retrospective provenance queries using YW in its current iteration. Because file names are not semantically meaningful in this dataset and most of the pertinent information about the files is stored in a separate csv, the URI templates currently used by YW do not lend themselves to meaningful queries with DLV. Further, to document which files were actually read over the course of a given run would require further modifications to original scripts or a modification to YW tool. Having reached an impasse, I met with YW developer Tim McPhillips and documented the results of our conversation in the Findings section rather than moving forward with my initial plan to conduct retrospective queries on a given run.

A further complication in conducting retrospective provenance queries is related to my current compute capacity. The statistical modeling process is also far more computationally intensive than I had originally suspected. In his readme file, Underwood writes "The most fragile part of this is probably the multiprocessing. We ran this so often that we needed to parallelize the process, but if that's not necessary

you could set processes=1 in line 369 of parallel_crossvalidate.py. Or, if you have more than 4 cores available, you could increase the setting." Out of curiosity, I attempted to run the program over the course of a weekend without adjusting the number of processes in parallel_crossvalidate.py and by Monday morning the script was still running without have completed the modeling process.

Findings

Modeling decisions

In preparing to generate a prospective provenance model, three core modeling decisions surfaced in light of this case study:

1. **Modeling across multiple scripts.** While replicate.py serves as a package for the set of scripts utilized by Pace of Change, manually tracking the thread of how the scripts interacted with one another proved challenging. My approach was to begin with the highest-level conceptual model possible and then identify the processes in which other scripts were invoked, usually by calling a function from within that script. Once I was within the body of the new script, I would use that function as my starting point to continue walking through the code. My thought is to use the replicate model as the primary package and then create separate models for the create_model and diachronic_tilt processes. If I were to present this visually I would use callout features to expose the internal workings of the more complex functions. This would not, however, provide any indication of which processes were drawn from a given script.

2. **Modeling user inputs.** I chose to represent the interactive feature of replicate.py in a simplified manner as "user_input". This provides information that the user input helps set important parameters, but it does not reveal which inputs are valid or how they impact the statistical model. To capture this information requires a combination of prospective and retrospective provenance.

3. **Modeling various kinds of data.** The current version of the prospective provenance graph colors intermediate or internal data in yellow and initial data or final outputs in white. In this case, the final outputs come in several forms. Two are shared with the user as print statements to the shell (i.e. accuracy and tilt_accuracy), two are saved as csv files (i.e., model_predictions_csv and model_predictions_coefficients_csv), and one represents a value returned from the function whose use is not immediately apparent in replicate.py. Data labels can also be configured in include file names where available, thus indicating which outputs result in new files. The color coding and label configuration are helpful components of the YW tool.

User-determined parameters

Replicate.py begins with an interactive session in the shell that prompts the user to determine whether they would like to model the full data set, break the dataset down into four quarter-century models, distinguish between American and British poets, and distinguish between male and female poets. These options determine the values for classify_conditions, thresholds, exclusions, and paths, which are passed from the set_parameters process to the create_model process. These parameters are used to determine the models to which a given file belongs and whether a file is used at all.

YesWorkflow does not currently record which files were *actually* read during any given run of a script, though a MATLAB extension for this feature is currently under development with further plans for an R

extension. An alternative approach would be to use to use provenance information collected via NoWorkflow in combination with the YesWorkflow tools and techniques.[4]

Another possible solution would be to write additional code into the script that generates a logging directory and creates a new empty file for every files that is processed during a given script run. Then the URI template could be mapped to the files in the logging directory. This approach raises possible concerns when working to replicate another scholar's pre-existing code. Whereas the YW annotations can be added as comments with minimal interference, adding new code represents a more significant and protentially problematic intervention.

At present, the best way to determine which files were used is to manually review the model_prediction_csv that is generated as part of the create_model process. Create_model will generate one or more of these csv files depending on the parameters passed to it. Each CSV contains the unique ID of the processed volumes, additional metadata pulled from poemeta.csv during the data processing, and new predictions about each volume that were generated by the statistical model.

CSV as Directory Index

Importantly for YesWorkflow, the poemeta.csv file serves as a directory index for the data to be analyzed, and user-determined parameters are compared against the metadata to choose a particular subset of files for analysis (based on a period of time or the authors' nationality or gender). Within poemeta.csv, each row captures a great deal of useful information about a given volume of poetry, including most significantly: a unique ID, dates, title, author, imprint, and the author's gender and nationality. The tsv file names are based on unique IDs, which in and of themselves are not particularly meaningful. Capturing file information in a new logging directory (as discussed above) will not lead to productive restrospective provenance queries if renamed files are still based on the file's identifiers. Thus, URI templates prove insufficient for cases that require an index file to provide metadata information.

Rather, this project would benefit from a new template for dealing with metadata this is stored outside the file name. An ideal solution would be to develop a csv file provenance extractor. Such an extractor would require a schema describing the contents of each row and a method for identifying the most meaningful fields within each row.

Further Thoughts

It is useful to note what kind of provenance *is not* being captured. In "How quickly do literary standards change?" Undewood spends a fair amount of time discussing sampling procedures for choosing which files to include in the poetry dataset. These procedures including a mix of manual selection and random sampling from the HathiTrust Digital Library. For the purposes on the GitHub repository, however, this might be considered an upstream process in that the selected dataset is provided without capturing the selection provenance. Similarly, each data file in the repository has already undergone pre-processing that transformed a full text data file into a word frequency table. While pre-processing is not fully covered in the context of this repository, Underwood provides a link to his algorithmic methods, which are described in a separate GitHub repository called "Understanding Genre in a Collection of a Million Volumes."

While the above limitations are based on the manner in which the code repository was scoped, a further limitation is based on provenance modeling decisions at the level of YesWorkflow. Whereas the YesWorkflow model focuses solely on data and processes as its core modeling concepts involved in causal relationships, competing models like PROV include a third concept: agent. According to the

[4] https://github.com/gems-uff/noworkflow

PROV Model Primer, "An *agent* takes a role in an activity such that the agent can be assigned some degree of responsibility for the activity taking place." [5] Due to the interactive nature of this script, which includes user inputs and computational processes, it may have been interesting to distinguish between the roles of the user and the roles of the system.

With that in mind, the most attractive feature of YesWorkflow is its effort to provide methods for provenance capture and representation that researchers can integrate directly into their code through structured comments. Provenance is best captured by the person who developed the project, but most provenance solutions require major script revisions or the use of external software. Having conducted a case study on another scholar's code repository, one of the greatest lessons learned from this project is the difficulty of working through another person's code to create provenance graphs secondhand. Any tool that lowers barriers for researchers to embed their own provenance research into their own protocols and procedures represents an important pragmatic step toward improved provenance capture.

Related Work and Context

This study drew upon two core themes: provenance and reproducibility. While the primary thrust of this work represents an extension of concepts covered in (McPhillips, Belhajjame, & Ludäscher, 2015), further work would benefit from deeper engagement around questions of statistical reproducibility (Stodden, 2015) and the use of datalog for provenance queries (Dey et al., 2012). This report also references competing tools and models for provenance capture such as NoWorkflow and PROV. These are just two examples of ongoing provenance research and development, but they provide an indication of the broader community of interest engaged in the topic.

Outlook

While this case study did not fully succeed in meeting its original goals, findings from the project are actively being integrated into ongoing YesWorkflow development priorities as well as proposals for external funding. The most immediate outcome of the project stems from a conversation with Tim McPhillips, the primary YW developer, that resulted in plans to develop the csv provenance extractor.

This case study represented a first attempt at considering YesWorkflow's efficacy for digital humanities projects based on computational analysis of texts. A larger and more complex assessment might include a study of the HathiTrust Research Center (HTRC). The HTRC utilizes a non-consumptive approach, in which users submit algorithms through a secure research environment to be run against a selected dataset but never gain full-text access to the selected data, to provide research access to restricted data. As such, this effort represents one of the more challenging use cases for humanities research data, a use case with clear implications for modeling best practices for curating data and capturing provenance. A planning grant proposal is currently under development to extend assessment of YesWorkflow and related tools and methods for capturing provenance in the HTRC infrastructure.

[5] http://www.w3.org/TR/prov-primer/

References

Dey, S. C., Köhler, S., Bowers, S., & Ludäscher, B. (2012). Datalog as a lingua franca for provenance querying and reasoning. In *TaPP*.

McPhillips T, Bowers S, Belhajjame K, Ludäscher B. (2015). Retrospective provenance without a runtime recorder. 7th International Workshop on Theory and Practice of Provenance. Edinburgh, Scotland.

Stodden, V. (2015). Reproducing statistical results. *Annual Review of Statistics and Its Application*, 2, 1-19.

Underwood, T. & Sellers, J. (2015). How quickly do literary standards change? FigShare. https://dx.doi.org/10.6084/m9.figshare.1418394