MERCATUS CENTER
George Mason University

QuantGov

QuantGov: An Overview

Patrick A. McLaughlin, Oliver Sherouse, and Stephen Strosko

March 2019

THE QUANTGOV ARCHITECTURE

Goals

The architecture used to create QuantGov has been designed to be modular, flexible, and extensible, targeting both the programmer and the academic. We want a system where it is not only possible but straightforward to add in or swap out either collections of text or machine learning algorithms to create new and innovative projects. To encourage others to build on our system, we have decided to adopt an open-source strategy. In addition, our code is inspired by the Unix philosophy, which encourages the writing of small programs that work together.

Three Components

There are three components in the QuantGov architecture:

1. A *corpus* is the file structure and scripts that retain and manage a collection of text documents. A corpus will clean and organize text, generate metadata that describes the text, and provide a driver that scripts can use to read through the text in an ordered way. Some corpora also feature a download script that fetches the documents in an automated fashion from a website or a nonlocal source.
2. An *estimator* is the file structure and scripts that retain and manage a classification, regression, or unsupervised learning task. Estimators should evaluate and tune candidate algorithms, train the chosen candidate, and provide a script that can perform an estimation on a given corpus.

3. A *project* combines corpora and estimators together to create a dataset. Projects provide a single point from which to run a corpus and estimator, and they may perform some finishing or cleanup work on the final product.

For example, the RegData 3.0 project uses two corpora (one representing the US *Code of Federal Regulations* [CFR] for analysis and one representing the *Federal Register* to train the candidate algorithms) and five resulting estimators (one for each level of North American Industry Classification System [NAICS]).

Snakemake for Workflow Management

Corpora, estimators, and projects all use the program Snakemake to manage workflow. Snakemake allows us to write rules that define how to create output files from input files. In addition, Snakemake allows users to create clear separation of tasks without mentally keeping track of every moving part—making the program perfect for managing QuantGov workflows.

Most importantly, Snakemake ties together the pieces of a project without having to hard-code information that connects the pieces. Therefore, Snakemake files can easily be generalized and used for multiple projects. In practice, Snakemake files for QuantGov projects tend to read in Python scripts that download and clean text while outputting the cleaned documents, resulting metadata, and analysis results.

The Snakemake files in the standard project are fully commented and can be taken as examples and used for custom projects. The Snakemake documentation also provides convenient tutorials.[1] Snakemake can be installed using pip or conda commands.

THE QUANTGOV CORPUS

Basic Structure

A corpus represents a set of documents and implements the corpus driver interface. The root directory for a corpus should contain the following:

- A makefile to manage the workflow of the corpus (Snakemake is recommended).
- A module named "driver.py" that implements the corpus driver interface (see below).
- A subdirectory named "scripts" containing the scripts needed to obtain, organize, and prepare the documents that make up the corpus, as well as to generate any metadata for the corpus.
- A subdirectory named "data" that holds any intermediate data generated during preparation of the corpus.

One file that each corpus should generate is a CSV file in the data directory named "metadata.csv." This file should contain any additional information about individual documents that may be relevant. For example, the metadata.csv generated by the CFR corpus includes which agency and department authored each individual CFR part, as well as the restriction and word count for each part. In essence, the metadata file contains any relevant data that are not determined by an estimator.

The Corpus Driver Interface

Each corpus should contain a python module named "driver.py." This driver serves two important functions. First it specifies how the corpus should be indexed. An index is one or more values that, taken together, uniquely identify each document in the corpus. An index can be as simple as an ID number, or it can be more descriptive. For the CFR corpus, each document, representing a single subdivision called a part, is represented by three pieces of metadata: the year of the CFR edition that contains it, the title that contains it, and the part number.

The names of the components of the index are stored in a module-level constant named INDEX and are always a tuple,[2] even when the index only has one component. Thus, for the CFR, INDEX = ('year', 'title', 'part'), and for a simple corpus using a document numbering system, INDEX = ('id',).

The second important feature of the driver is that it provides a function named "stream." The stream function should return an iterable—in most cases, a generator—that emits the index value (or values) and text of each document in the corpus. Thus, the first item emitted by driver.stream() in the CFR corpus might be (1975, 1, 1), "Text of the 1975 CFR, Title 1, Part 1".

Drivers may implement other features (such as only streaming a subset of documents based on the index), but these types of features are nonstandard, and estimators should not expect them as a matter of course.

Corpus Metadata

Relevant metadata will vary from corpus to corpus. Metadata can be generated from one of two sources: from the text itself or from additional external information. In the first case, the best practice is to write scripts that understand the corpus driver interface and can therefore be used in other corpora. An example of this approach can be seen in the "get_wordcount.py" and "get_restriction_count.py" in the QuantGov generic corpus. In the second case, the external resources should be stored in a databank kept separate from the corpus itself, which the corpus scripts treat as read-only. An example of this approach is the agency attribution in the CFR corpus, which relies on a set of documents separate from the main CFR text.

Writing a New Corpus

The easiest way to write a new corpus is to fork the most similar official corpus (see below) and modify it to represent the relevant body of text. There are three principal problems to solve in the creation of a new corpus:

1. How can the text be obtained and, if necessary, translated to plain text? Plain text is text that is not computationally tagged, specially formatted, or written in code. It is normally stored in a TXT file.
2. What is the logical unit of analysis for the corpus?
3. How can the text be organized to most usefully reflect the unit of analysis?

The first problem will determine the scripts needed for downloading or otherwise obtaining and cleaning the text from its published format. The second will determine the index for the corpus and identify what makes an individual document appropriate to be served through the driver. The third will determine how the driver is actually implemented.

Official QuantGov Corpora

Official QuantGov corpora are branches of the corpus repository, available on GitHub.[3] The generic QuantGov corpus is listed on the QuantGov platform page.[4]

Submitting a New Official Corpus

Complete corpora may be considered to be added as official QuantGov corpora. If accepted, a new branch will be created to which a pull request can be made on GitHub. Additions to the official corpora are at the sole discretion of the QuantGov team. Please email info@quantgov.org with any questions about adding a corpus to the official QuantGov corpora.

THE QUANTGOV ESTIMATOR

Basic Structure

An estimator is the file structure and scripts that retain and manage a classification, regression, or unsupervised learning task. The root directory of an estimator should contain the following:

- A subdirectory named “data” containing the results from the evaluation and training of the estimator, alongside any intermediate data.

- A subdirectory named "scripts" containing the scripts needed to evaluate candidate models, train the selected candidate using a trainer corpus, and analyze a target corpus using the trained model.
- A snakefile to manage the workflow of the estimator. The snakefile should implement the estimator interface (see next section).

The Snakefile Interface

The snakefile for an estimator should use the following variables, defined in the "config.yaml" file (located in the root directory alongside snakefile):

- "trainer_corpus" should be a path to the corpus needed to evaluate and train the estimator.
- "folds" should be the number of times to fold the training data for testing.
- "scoring" should be the method for testing the trained estimator.

Writing a New Estimator

The easiest way to write a new estimator is to fork the most similar official estimator (see below) and modify it for the relevant task. Generally, the tasks for writing an estimator are the following:

- Extract features from the training document.
- Test one or more algorithms for the estimation task, possibly tuning a set of parameters for each algorithm.
- Select an algorithm from the candidates tested and train it using the full trainer corpus.
- Construct a pipeline for feature extraction and analysis of other corpora, using the corpus driver interface.
- Additional help on training a custom estimator can be found in the official QuantGov documentation.[5] It should be noted that while making a custom estimator may seem challenging, the QuantGov Python library and platform can automate most of the steps.

Official QuantGov Estimators

Official QuantGov estimators are branches of the estimator repository, available on GitHub.[6] The generic QuantGov estimator is listed on the QuantGov platform page.[7]

Submitting a New Official Estimator

Complete estimators may be considered to be added as official QuantGov estimators. If accepted, a new branch will be created to which a pull request can be made. Additions to the official estimators

are at the sole discretion of the QuantGov team. Please email info@quantgov.org with any questions about adding an estimator to the official QuantGov estimator.

THE QUANTGOV PROJECT

Basic Structure

A QuantGov project brings together corpora and projects to create a new dataset. The root directory for a project should contain the following:

- A subdirectory named "data" that holds data generated in the creation of the dataset.
- A subdirectory named "scripts" containing any scripts needed to clean, combine, or polish metadata and analyses from the corpora and estimators.
- A snakefile to manage the workflow of the project.

Writing a New Project

The easiest way to write a new project is to fork the most similar official project (see next section) and modify it for the relevant task. The snakefile for a project should make use of the estimator interface and of the corpus feature of providing a combined metadata file in the "file data/metadata .csv." Projects should use the snakefile modularization to ensure that the desired files are created and up to date before putting them in their final formats.

Official QuantGov Projects

Official QuantGov projects are branches of the project repository, available on GitHub.[8] The generic QuantGov project is listed on the QuantGov platform page.[9]

Submitting a New Official Project

Complete projects may be considered to be added as official QuantGov projects. If accepted, a new branch will be created to which a pull request can be made. Additions to the official projects are at the sole discretion of the QuantGov team. Please email info@quantgov.org with any questions about adding a project to the official QuantGov project.

ABOUT THE AUTHORS

Patrick A. McLaughlin is the director of Policy Analytics and a senior research fellow at the Mercatus Center at George Mason University. His research focuses primarily on regulations and the regulatory process. McLaughlin created and leads the RegData and QuantGov projects, deploying machine learning and other tools of data science to quantify governance indicators found in federal and state regulations and other policy documents. The resulting database is freely available at QuantGov.org and has facilitated pioneering empirical research by numerous third-party users on the causes and effects of regulation.

Oliver Sherouse is a regulatory economist in the Office of Advocacy at the US Small Business Administration. He was previously the Policy Analytics lead in the Program for Economic Research and Regulation at the Mercatus Center.

Stephen Strosko is a Python developer and project coordinator for Policy Analytics at the Mercatus Center. He specializes in regulatory research and, notably, has worked on the RegData, Quantgov, FRASE, and RegData Canada projects.

NOTES

1. Snakemake, "Installation," accessed March 14, 2019, https://snakemake.readthedocs.io/en/stable/getting_started/installation.html.
2. W3Schools, "Python Tuples," accessed March 14, 2019, https://www.w3schools.com/python/python_tuples.asp.
3. GitHub, "QuantGov / corpus," accessed March 14, 2019, https://github.com/QuantGov/corpus.
4. QuantGov, "The QuantGov Platform," accessed March 14, 2019, https://quantgov.org/tools/.
5. QuantGov Documentation, "Tutorial: Training Your First Estimator," accessed March 14, 2019, http://docs.quantgov.org/tutorial/first_estimator/.
6. GitHub, "QuantGov / estimator," accessed March 14, 2019, https://github.com/QuantGov/estimator.
7. QuantGov, "The QuantGov Platform."
8. GitHub, "QuantGov / project," accessed March 14, 2019, https://github.com/QuantGov/project.
9. QuantGov, "The QuantGov Platform."