“Smarter Data Collection for Microsoft Civic Graph

All of my work can be found in this GitHub repository.

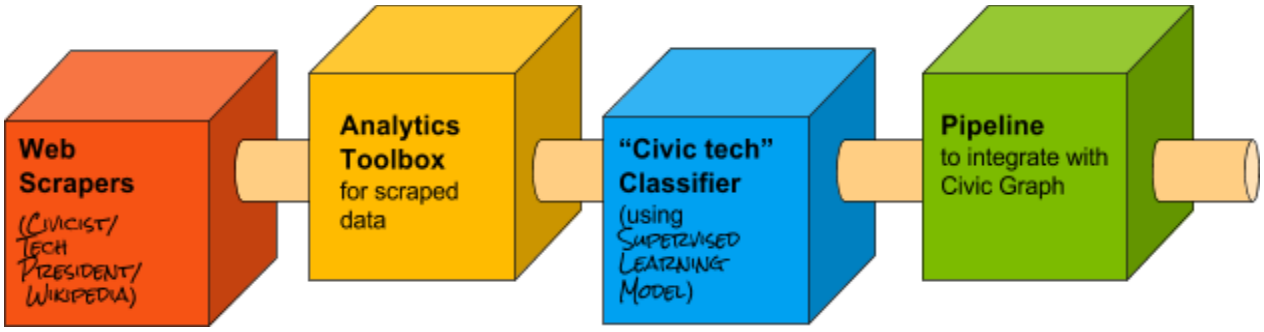
In this document:

- Overview
- Technical Documentation for: **Scrapers, Analysis, Classifier, Pipeline**
- Next Steps
- Thoughts and Ideas about: tools to consider using in future, incorporating additional external data sources into Civic Graph

Overview

In an effort to make Civic Graph a little bit *smarter*, I developed **four** building blocks to help improve the quality (i.e. accuracy, completeness) of the data stored as well as automate aspects of the data collection process. They are:

1. **Web Scrapers**
2. **Analytics Toolbox**
3. **Classifier**
4. **Pipeline** to integrate scrapers, classifier, analysis with existing Civic Graph



Technical Documentation

Web Scrapers:

I developed Python scripts to fetch all content from the Civicist and Tech President archives (published 2004-2015) as well as a script that pulls content from Wikipedia.

Source	Relevant scripts	Libraries used	Notes/Status

Civicist Archives	**civ-link-parse.py:**
	- Parses Civicist Archives
	- grabs all article URLs in Archive
	- Writes article URLs to text file
	--- 	
	civ-article-parse.py:
	- Read article URLs from text file
	- Iterate through list of URLs and extract
	  - Article title
	  - Article author
	  - Article date published
	  - Article content
	- Append all extracted content to string variable and write to text file
		- BeautifulSoup	Active
			Accurate
			Not automated

TechPresident Archives	**tp-link-parse.py:**
	- Parses TechPresident Archives
	- grabs all article URLs in Archive
	- Writes article URLs to text file
	--- 	
	tp-article-parse.py:
	- Read article URLs from text file
	- Iterate through list of URLs and extract
	  - Article title
	  - Article author
	  - Article date published
	  - Article content
	- Append all extracted content to string variable and write to text file
		- BeautifulSoup	*TechPresident's Archives spanned 2004-2015. I discovered that over time the structure of the archive source code changed. There were 1411 pages total. I ran the script three times, extracting the article links in three batches.*

Wikipedia	**wikipedia_parser.py:** - Prompts user to enter a topic (and press ENTER) - Fetches Wikipedia page contents for that topic if a page	- BeautifulSoup - Python's wikipedia package	Active Accurate

	exists. Extracts: - Page summary - Page Categories - Relevant links (pages associated with the topic) - Loops and continues to prompt for new query until user exits program		

Analysis ("Analytics Toolbox")

I experimented with several open source sentiment analysis/natural language processing tools. Based on the results of my experimentation, I decided to use spaCy and nltk.

Tool/ Library	**Analysis performed**	**Relevant files**	**Notes/Status**
AlchemyAPI	- Entity Extraction (NER) - Keyword Extraction - Taxonomy extraction - Category Extraction	**alchemyapi.py** - Parses Civicist Archives - grabs all article URLs in Archive - Writes article URLs to text file - Run "python alchemyapi.py" and enter API key. Mine was: ff8f993db5ee0b907a3e41f19bbd57b8b4cbc24a --- **cg_categorize.py** - Reads in file containing names of entities in Civic Graph and their associated descriptions - Attempts to categorize entities in Civic Graph based on descriptions using AlchemyAPI's "category" function --- **civictech_NER.py** - Extracts data from Civicist/ TechPresident article content including: - Keywords - Taxonomy	*First tool that I used. After doing further analysis with other tools (i.e. nltk, spaCy), I learned that this tool is not very accurate, and it only extracted a fraction of the named entities. It was a useful exercise in teaching myself what kind of information one can extract from raw text using NLP libraries.* Scripts retired.

		- Entities * - Categories - Function-	

nltk	- Text tokenization
	- Entity Extraction (NER)
	- Concordance
	- Shared contexts for keywords
	- Frequency distribution
	- Collocation of common terms
		nltk_basic_analysis.py
		- Reads in raw text file and tokenizes text by sentence
		- Does semantic analysis on each token, extracting:
		  - All Named Entities
		  - Unique entities
		  - Concordance for a given keyword
		  - Shared contexts for two or more words
		  - Frequency distribution of named entities
		  - collocations of commonly occurring terms
		--- 	
		reverse-madlibs.py
		- Reads in raw text file and tokenizes text by sentence
		- For each token:
		  - Extracts Named Entities
		  - Computes all possible pairings between entities in that token and stores as a list of tuples
		- Iterates through tuple list and prompts user to enter relationship between the two entities in a given tuple
		- Stores user response to external file
			Accurate and easy to use but VERY slow on large datasets.
			
			Active

spaCy	- Text tokenization - Part of speech (POS) labeling - Entity Extraction (NER) - Frequency distribution	**spacy_experimentation.py** - Reads in raw text file and tokenizes text by sentence using spaCy library --- **spacy_NER.py** - Reads in raw text file and tokenizes text by sentence using spaCy library	Accurate and super fast.

		- For each token:
		  - Extracts Named Entities
		  - Counts up entity frequencies and stores in dictionary {k: entity_name, v: freq}
		  - Sorts dictionary in ascending order by value and prints resulting ordering	

General Analysis	Compute set intersection between two lists (e.g. Civic Graph entities and entities extracted from blog content)	**cg_queries.sql**
		- SQL queries Civic Graph database to extract:
		  - Entity names, nicknames and id's from entity table
		  - All connection types (collaboration, data, investment) from respective tables
		--- 	
		blog_graph_overlap.py
		- Reads in two csv files:
		  - entities pulled from Civic Graph database
		  - Entities extracted from Civicist/ TechPresident blog content
		- Computes set intersection between Civic Graph and blog entities
		--- 	
		entities_analysis.R
		- Reads in two csv files:
		  - entities pulled from Civic Graph database
		  - Entities extracted from Civicist/ TechPresident blog content
		- Computes set intersection between Civic Graph and
			blog_graph_overlap.py and **entities_analysis.R** essentially do the same thing.

		blog entities
		
		entity_diff.py
		- Reads in two csv files:
		  - entities pulled from Civic Graph database
		  - Entities extracted from Civicist/ TechPresident blog content
		- Computes set intersection between Civic Graph and blog entities	

textacy	N/A	Did not use this.	Did not use this.
Stanford nlp tools	N/A	Did not use this.	Did not use this but apparently not as robust as nltk, textacy, or spaCy.

Classifier:

1. **Input:** Raw text data
2. Tokenize training and test data
3. Label training data
4. Train classifiers on labeled training data
5. Test classifiers on test data
6. **Output:** For each token, returns an accuracy score for each label 0.0 <= score <= 1.0 corresponding to label relevance.

Functionality	**Relevant scripts/functions**	**Notes/Status**
Tokenize training data	*readfile_tokenizedata()* function within **algorithm.py**	Active

Labeling training data	**label_training_data.py**
	- Read in each training dataset as text file
	- For each text file, generate labels for training data associated with a given label
	  - Funding
	  - Data
	  - Employment
	  - Collaboration
	  - Location
		Active

Training classifiers	*svm()* function within **algorithm.py**	Active
Preprocess/ tokenize test data	*svm()* function within **algorithm.py**	Active *Tokenization on test data not fully implemented*
Run classifiers on test data	*accuracy()* function within **algorithm.py**	Active

Pipeline:
View full-size diagram here.
Editable version can be accessed here.

Next Steps (Task List)

General tasks

- Cron job for all tasks within Mad Libs system.

Scrapers

- Auto scraper for Civicist/TechPresident

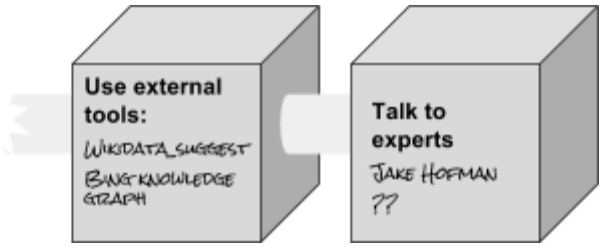
Classifier

- Choose an accuracy threshold (**e.g.** >= 87%)
- Make sure we are training the classifiers on the right data
- Structure the output of Classifier to fit schema of Civic Graph database
- Make sure we are training the classifiers on the right data
- Find the right source for our test data (is it Civicist/TechPresident? Something else?)
- Finish and tweak classifiers
- Talk to ML/NLP expert to get feedback (e.g. Jake Hofman)

Integration with Civic Graph Database

- Within **structure_and_post.py**, I wrote a pseudo-algorithm for:
    - structuring the output data from Classifier
    - Adding new table to database schema
    - POSTing structured data to Civic Graph database
    - *Note: Within this script, I describe all steps to integrate classified data with Civic Graph database.*

Thoughts and Ideas

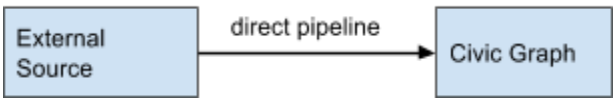


Useful tools to consider in future:

- Some tools think could help us to gather additional data once we have a solid method for classifying civic tech relevance.
    - Run **wikipedia_parser.py** on all "relevant" civic tech entities
    - Use Python's wikidata_suggest module (*To install:* "pip install wikidata_suggest")
    - Use the Bing Knowledge Graph tool

Suggestions for incorporating additional external data sources into Civic Graph:

- We have already identified potential sources that we can pull data from to incorporate into our Civic Graph database. These include:
    - DigitalSocialEU
    - Foundation Data
    - Civic Tech field guide
- As we expand this list, it will become increasingly necessary to think about how to utilize these resources to improve the quality of the data currently stored in civic graph (i.e. so that it is accurate and complete) and to think about how we can leverage these sources to identify new entities to add to Civic Graph. An obvious approach for incorporating external sources into the civic graph framework is to have direct pipelines from these sources to Civic Graph:



- You must assume that all of the data contained in the external source is relevant to civic tech and thus qualifies as being relevant to Civic Graph. Arguably, a better approach is to have an intermediate check where you aggregate and classify the external data before adding it to Civic Graph. For example, Create a "civic tech relevance score" and for each datapoint in the external source, if it generates a relevance score above some threshold, then pipe it to the Civic Graph database: