Rivet Developer Guide

Author(s): James Baggs

Continuing Development of the Rivet System

Further changes to the Rivet system for the purposes of data mining are likely to be in either the domain of new expression generation or extracting semistructured data from different file formats. For both of these, it is important to be familiar with Rivet's current File-pattern data structure featured below:

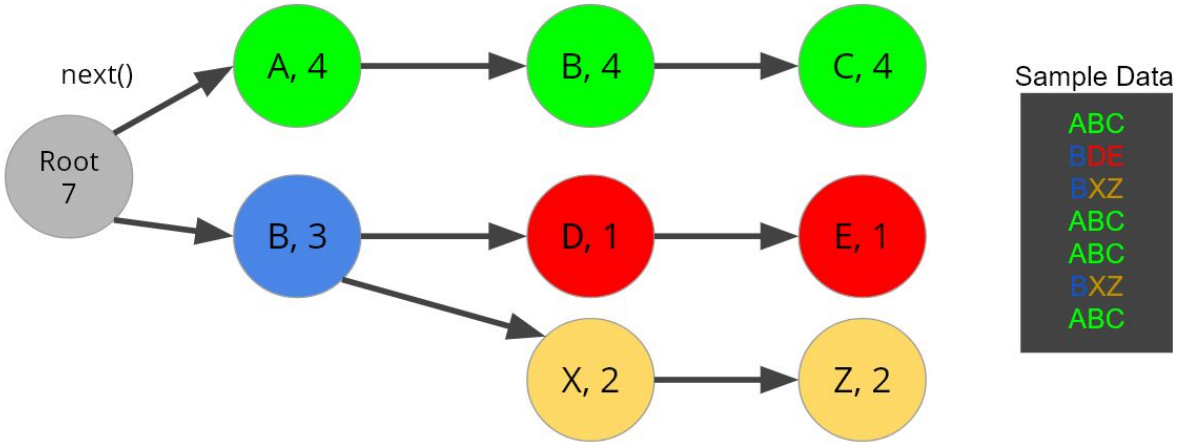

Figure 1 - Data Structure

Pattern Frequency Analysis and Expression Generation

While the Rivet system analyzes the frequency of patterns in a file on a line-by-line basis, it fails to analyze the frequency of certain patterns across lines. In the example above, the 2-gram pattern of B-then-D appears *five* times. This information is important when considering much larger sets of data, and when B or D are not pre-defined Rosie expressions, and are instead simple sets of characters, like punctuation, digits, or a-z. While the Rivet system is currently only concerned with producing an RPL pattern based on the most frequent line-by-line pattern, it is an important task to produce more Rosie Expressions for the Rivet system to match data against. Knowing which sets of characters, expressions, and patterns are commonly co-located can be important for the generating new expressions.

For example, using our system provides only an frequency of patterns line-by line, but with more information about how frequently tokens co-occur could potentially lead to the

discovery that a pattern like one below is very frequent across multiple lines of data in different columns:

<Hex><Hex>”.”<Hex><Hex>”.”<Hex><Hex>”.”<Hex><Hex>

This turns out to be an *IPv4* address format. A new feature for development would be automatically detecting this high-frequency pattern and suggesting it to a user as a potentially new pattern.

Semistructured Datafile type handling

The Rivet system currently assumes that each line of a file is a unique entry to be analyzed. While this is true for semistructured data such as CSV or XLXS which have row-based entries, and column-based fields, this is not true for all semistructured data. HTML and XML files in particular hold data inside of nested tags, where each row may be inside of a <tr> tag, and each field may be inside of a <td> tag.

The Rivet system will still provide improvements for data extraction for Rosie over Rosie’s brute force simply because it suggests which patterns to try first based off of the frequency found in the files, but this implementation is far from optimized, and can contain a lot of noise since it doesn’t know how to handle tags such as <p> in html (other than as a combination of <, p, and >)

A suggested approach for continuing development on the Rivet system is to create specific extraction rules for processing HTML, XML, or other types of nested semi-structured data, or to provide a pre-processor which removes their tags and formats them in a way more similar to row/column based files like CSV.

Another example of why certain types preprocessing would be necessary is shown below in Figures 1 through 3. In the example of the NCZipCodes.csv data which was obtained from a data.gov source, we found that our system did not handle a large number of missing values very well. As you can see from a small sample screenshot of the data in Figure 1, many columns have replaced what would normally be Integers with “**” to indicate missing data. Since this dataset had missing values spread widely over many feature columns (as opposed to having only one or two columns with missing data), the Rivet system recognizes these as unique patterns. On a sample of 587 lines from this file, 495 unique patterns were found (Figures 2 and 3). A system which preprocesses data to handle missing values before being passed to the Rivet system would make the overall data extraction more robust. We also think that this provides a good case for the n-gram approach, in which the position of values within the lines don’t matter as much as the datatypes of their neighbors. This n-gram approach would handle missing values more gracefully, since entire lines within files do not have to be uniform.

ney amounts are in thousands of dollars]

ZIP code [1]	Size of adjusted gross income	Number of returns	Number of single returns	Number of joint returns	er of head of house hold return	er with paid prepar er's signat
		(1)	(2)	(3)	(4)	(5)
27006	$75,000 under $100,000	740	120	590	30	460
27006	$100,000 under $200,000	1250	110	1100	40	760
27006	$200,000 or more	520	50	460	**	400
27007		860	280	500	90	570
27007	$1 under $25,000	320	170	100	50	190
27007	$25,000 under $50,000	220	80	120	40	140
27007	$50,000 under $75,000	160	30	130	**	120
27007	$75,000 under $100,000	80	**	70	**	60
27007	$100,000 under $200,000	80	**	80	**	60
27007	$200,000 or more	**	**	**	**	**

Figure 1: NCZipCodes.csv - Missing data values in csv format

```
james@James-PC:~/2017FallTeam11$ python ./src/Rivet.py -s 10 -p 0 NCZipCodes.csv
Start sampling...
[##################################################] 100.0%
Sampling complete:
Number of lines in input file: 5879
Number of lines to sample: 587
```

```
Pattern 491 matched 0.17% of sample file
Pattern 492 matched 0.17% of sample file
Pattern 493 matched 0.17% of sample file
Pattern 494 matched 0.17% of sample file
Pattern 495 matched 0.17% of sample file
```

Figures 2 and 3: Number of patterns extracted from NCZipCodes.csv is 495 with 587 lines sampled