Semantic Network Analysis Pipeline

Design Document

March 29, 2017

Collaborators

Levi Oyster

Eric Pak

Sponser Martin Cenek

Table of Contents

Overview

1. Natural Language Processing
2. Undirected Graph Network Generation
3. Network Analysis
4. 3-D Network Visualization
5. Hosting website

Description

1. Natural Language Processing
2. Undirected Graph Network Generation
3. Network Analysis
4. 3-D Network Visualization
5. Hosting website

Details

1. Natural Language Processing
2. Undirected Graph Network Generation
3. Network Analysis
4. 3-D Network Visualization
5. Hosting website

Overview:

/////////////Believe this is wrong based on his description of this section///////////////////

Generation of 3D visualization for a compiled body of individual semantic networks requires multiple components tied together to achieve this goal. First an individual document must be analyzed through natural language processing so it broken down by language rules. The resulting output is then transformed into a undirected graph network based on the distance of one word from another. All of these undirected graph networks are then combined into a larger more comprehensive network for complex network analysis and layout with each individual graph representing a time interval. This newly formed comprehensive graph is then transformed into a 3-Dimensional visualization using the layout determined for node placement while using the established time intervals as the 3rd dimension for placement. The final aspect of this project is tying all of these components together in a pipeline to link one components out put directly to the next components input.

Description:

///////////////////Something thats an overview, but not an overview…./////////////////////

Details:

A. Natural Language Processing

B. Undirected Graph Network Generation

C. Network Analysis

Using the Gephi 0.8.2 developer toolkit in conjunction with the Gephi Circular Layout plugin, Network analysis is done without use of the Gephi User interface, and is done directly through terminal commands. All objects listed below are Gephi toolkit Objects.

Gephi 0.8.2 api

https://gephi.org/docs/api/

1. Project Initialization
2. Construct Project to make use of toolkit features.
3. Construct a Workspace from the Project, a workspace is a Container for all data.
4. Construct an ImportController to handle import of Undirected Graph Network files, in this case file type .dl.
5. Construct a GraphModel, handles structure of entire graph as object.
6. Construct an AttributeModel, allows for graph manipulation based on Attributes.
7. Construct a DynamicProcessor, this allows for the use of time intervals when files are imported.
8. Construct a Graph from the GraphModel, manipulations of the graph are sometimes called directly on the graph
9. File Import
10. Take a directory of files as a command line argument and then iterate through the files to import them individually as separate time slices. If the argument is identified as single file and not a directory, still process file.
11. Load each file into a Container using the ImportController.
12. Using a regular expression, extract the file date out of the imported file’s file name.
13. Use file date to establish file date as a timeslice in the overall time intervals using the DynamicProcessor. The Gephi toolkit’s DynamicProcessor recognizes the individual nodes in a file and checks them against those already imported into the overall graph, if they exist already then the file date is used to append to the time internals of the existing node.
14. Insert file date into an array of dates, these will be printed to file on export.
15. Size Nodes
16. Construct a RankingController to be able to rank nodes by a given measure.
17. Using the AttributeModel in conjunction th, apply sizing of nodes based on either Betweenness Centrality or Closeness Centrality.
18. Color Nodes by Modularity Class
19. Construct a PartitionController to be able to partition groups of nodes by a given measure.
20. Construct a Modularity Object to be able to get the modularity classes of nodes.
21. Partition the graph using the PartitionController by passing the Modularity class column and then color nodes based on their modularity class.
22. Perform Layout by Modularity Class
23. Construct a NodePartitionFilter to be able to manipulate nodes a specified partition.
24. In a for loop, iterate through all partitions in the graph and store the percentage of nodes in portion into an array, sort the array.
25. Iterate through the array of percentages of, starting from the highest percentage, activate only that partition and perform the CircularLayout from the CircularLayout plugin on the active partition. Order the nodes by Betweenness Centrality or Closeness Centrality, and the nodes will placed in a circle by this measure. Translate the X and Y positions of the nodes based on the partitions index in the percentage array. The positions are translated in eight directions based the index’s remainder when divided by eight. Each iteration of eight indices, shifts the nodes farther away. The lower the index, the closer to center a partition of nodes is, this is to show the larger communities closer to the center for easy identification as main communities. Deactivate the partition and iterate to the next.

Figure1. Partition Placement

Numbers represent the ordering of the index.

8 9

| /

7 | 1

\ | /

6——— ———2

/ | \

5 | 3

|

4

1. Activate all partitions so the entire structure is visible.
2. Export Graph and Filedates
3. Construct an ExportController to export a variety of files.
4. Use the ExportController to export a .gexf file containing all graphical analysis information and graph structure.
5. Use the ExportController to export a .pdf file of the 2D visualization for debugging purposes.
6. Export a .txt file of the file dates using a Java Library PrintWriter.

D) 3-D Network Visualization

Module is made completely in Java 8, is intended to produce files compatible for Partiview.

1. File Import
2. Import fileDates.txt and immediately set the greatest possible Z value to the greatest date value in the file dates, the dates are converted into integers.
3. Import completeLayout.gexf, copying all data of each node in gexf file into nodes of 3-D graph, most importantly the lables, X and Y coordinates, and modularity class of a node. The time intervals of a node imported are compared to the dates in fileDates.txt, if an internal overlaps a discrete date in file dates, that date is used a Z value in which the node will be present, represented as a separate node from other occurrences of itself in the graph.
4. Processed nodes are appended to overall graph structure.
5. Generate Meshes
6. For each discrete file date, group those nodes with the respective Z value as a Layer of nodes.
7. For each community of nodes, set a centroid at the center of the circle of the community.
8. For a given time range, connect communities of nodes through time with the same modularity class if nodes are present within that range, creating Meshes, also known as Noodles due to their cylindrical structure.
9. When a community in a layer has less nodes than the nodes of the same modularity class in the next layer, join the nodes to the next closest node by X and Y value.
10. Update Mesh’s With Modularities from Individual Files Representing Separate Time slices.
11. Import an individual .gexf file representing each layer in to total graph, matching the file date to the Z value of the layer. Update the modularity classes of nodes at that given layer to the individual .gexf files modularity class for that node.
12. Rearrange nodes in each layer about their centroids based on similar modularity classes, grouping different modularities together.
13. Split noodles through layers of time so that different modularities represent different noodles of nodes.
14. Export Partiview Compatible Files
15. Export node.speck file for Partiview to make use of node data.
16. Export edges.speck file for Partiview to make use of edge data.
17. Export mesh.speck file for Partiview to make use of mesh data.

E) Website

Website is composed primarily of PHP, using CodeIgnitor to provide key functionality such as user accounts and network configuration. HTML, CSS and Javascript are also used in its implementation.

1. User Accounts
2. Hosting accounts is built directly into CodeIgnitor, use CodeIgnitor provided classes for registration, login, logout, and forgotten passwords.
3. Every account has a set of folders to host files for each page of the website: Natural Language Processing, Undirected Graph Network Generation, Network Analysis, and 3-D Network Visualization. User can Delete or Download from these folders on their respective pages.
4. Batch Processing Files
5. Javascript is used to check all check boxes for files to allow batch deletes and downloads. For Natural Language Processing and Undirected Graph Network Generation, this can also be used for batch processing.
6. Each page of the website performs batch processing of files by submitting command line arguments to the respective .jar files of the page with the input files from the folders owned by current logged in user when the user clicks submit.
7. Each .jar file outputs files to different locations in the file system, and batch transfers are used to move output files to their proper destination folders.