Context-driven Text Visualization

Fall 2016
Sanjana Wadhwa
wadhwa.28@osu.edu

Aim:

The aim of project is to create visualization of publication data by IEEE. This can prove to be a helpful tool for students and faculty to traverse through the huge amount of publication data available on the web. Source: http://www.vispubdata.org/site/vispubdata/

Part 1: Data Processing

The main aim is to extract useful keywords from the abstracts of the given papers. Using statistical machine learning for Natural Language Processing, we have to calculate the correlation of publications from the text.

Techniques for Extraction of Keywords:

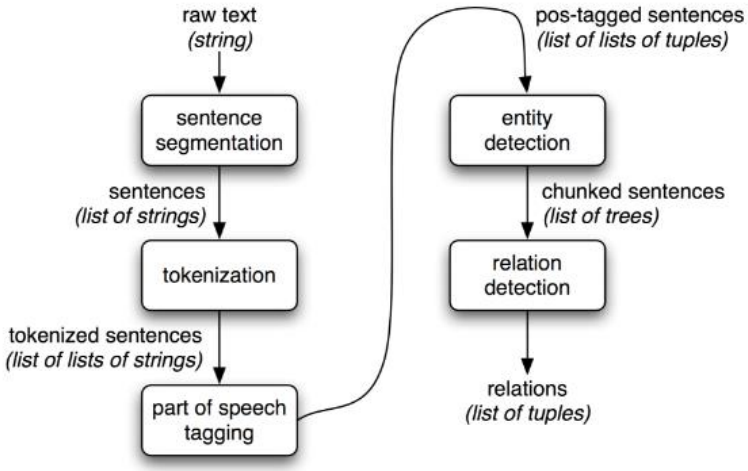
- Use author given keywords for the visualization
- Use most frequent words from the abstract of the paper
  Assume words from same root to be the same, for example, “analyst”, “analysis”
  - Reference: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.723.8579&rep=rep1&type=pdf

  Give more weight to words in sentences that are in introduction and conclusion. Also, sentences that represent any results, or special inferences should be given more weight
- Using NLTK from Python [http://www.nltk.org/book/ch07.html], it would be very easy to identify keywords and existing relations between them to create a more meaningful and robust visualization. This is known as the Information Extraction System. The information extraction system in NLTK works as follows:
  1. The text of the document is split into sentences using a sentence segmenter. This is then broken into keywords using a tokenizer.
  2. Next, each broken part is tagged with part-of-speech tags.
  3. The next step is where the keywords are extracted from each sentence.
  4. Lastly, relation detection is used to search for likely relations between different entities in the text.

  Chunking:

  For keyword detection, chunking can be very useful. Part-of-speech tags are very important for chunking. POS tagging Is followed by chunking. It omits whitespace and selects a subset of the tokens. The pieces produced by a chunker do not overlap in the source text.

Chunkers exist for detection of named entities, and other can be constructed using rule-based systems, such as the RegexpParser class provided by NLTK; or using machine learning techniques, such as the ConsecutiveNPChunker.



Information Extraction from structured data in NLTK

```
>>> def ie_preprocess(document):
...     sentences = nltk.sent_tokenize(document) ❶
...     sentences = [nltk.word_tokenize(sent) for sent in sentences] ❷
...     sentences = [nltk.pos_tag(sent) for sent in sentences] ❸
```

Sample Code for Preprocessing of IE System in Python

Calculate Correlation of Papers:

- Using common references of papers from the dataset
- Using most common keywords
- More correlated if have common authors
- Published in same conferences
- Published from same university or organization
- Published around same time

Part 2: Visualization

Keypoints for the Visualization of the Node-Edge Tree:

- Nodes as name and link of paper
- Clustering nodes together according to similarity
- Weight of edges could represent similarity of papers
- Create horizontal, scrollable timeline to show correlation with time
- Height of node can be proportional to number of citations
- Length of each node along the timeline can show how long the paper was relevant for

The final motive is to create a real-time, interactive visualization using WebGL (JavaScript), and extensions such as PhiloGL. WebGL could render a more flexible visualization.

Basic Visualization in D3.js:

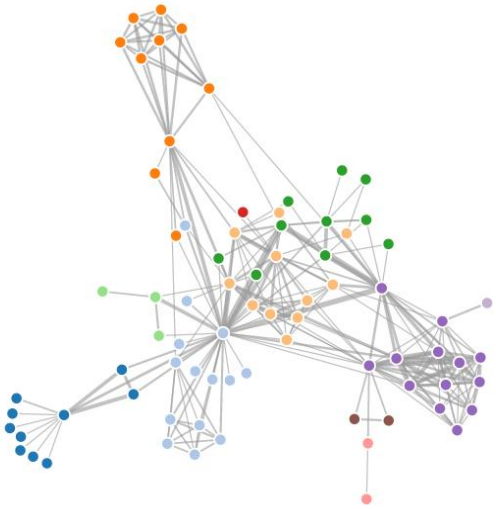
- Using Python on given excel file to extract node-pair for edges of the graph.
  JSON file:

```
{
  "nodes": [
        {"id" : "146388.0", "group": 1},
        {"id" : "146392.0", "group": 1},
        {"id" : "146389.0", "group": 1},
        {"id" : "146371.0", "group": 1},
        {"id" : "146413.0", "group": 1},
        {"id" : "146378.0", "group": 1},
        {"id" : "146368.0", "group": 1}
        …
        ],
  "links": [
        {"source" : "7192699.0", "target": "5652433.0", "value":1},
        {"source" : "7192699.0", "target": "729568.0", "value":1},
        {"source" : "7192699.0", "target": "663916.0", "value":1},
        {"source" : "7192699.0", "target": "6634131.0", "value":1},
        {"source" : "7192699.0", "target": "6875998.0", "value":1},
        {"source" : "7192699.0", "target": "4658161.0", "value":1},
        {"source" : "7194845.0", "target": "6327257.0", "value":1},
        {"source" : "7194845.0", "target": "6634103.0", "value":1}
        …
        ]
}
```

  Links to Python code:
  http://web.cse.ohio-state.edu/~wadhwas/DataViz/nodes.py
  http://web.cse.ohio-state.edu/~wadhwas/DataViz/links.py

- Using Force-directed Tree layout of D3.js, plotted this data, link to JavaScript page:
  http://web.cse.ohio-state.edu/~wadhwas/DataViz/BasicVis.html

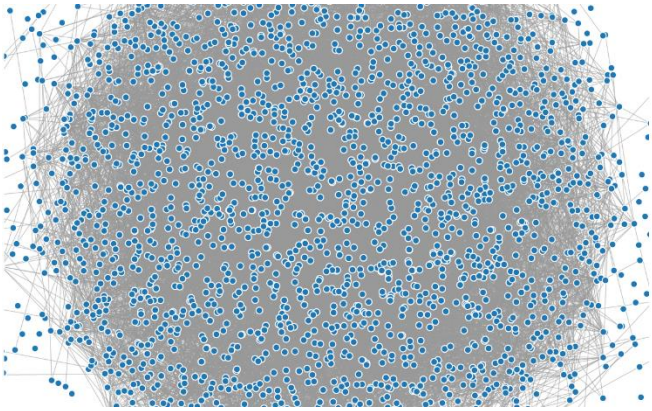
  Force-directed Graph:
  A force directed graph uses weight values to calculate charge between two nodes of the graph. A thicker edge represents more weight of the edge. For the simple case, all weight values are one, and all nodes belong to the same group.
  Reference: https://bl.ocks.org/mbostock/4062045



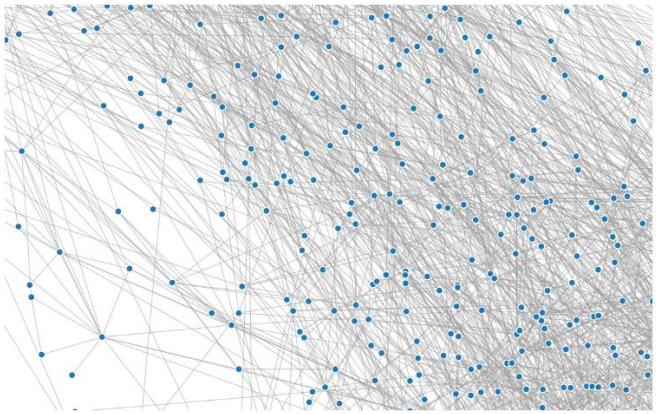
Sample Force-directed Graph

Drawbacks with D3.js:

- Very slow with generation of the tree since the data size is large!
- Initial stages look like:



- Fully rendered graph looks like:



Gives absolutely no context about the data! Local viewing can be improved with use of fisheye distortion, but holistic view is still bas, hence requirement for better processing and visualization of data.