Higher Order Networks & BuildHon+  

Steven Krieg
The Problem

How do we represent big data as a network, while accurately preserving dependencies?
A Solution!

Raw event sequence data

First-order network

Count number of pairwise interactions as edge weights

Extract higher-order dependencies from raw event sequences

Higher-order dependencies

Construct HON based on the extracted rules

Higher-order network

Image from [2]
The Kernel: BuildHon

Algorithm used to construct the HON

Has 2 main steps:
1. Rule extraction
2. Network rewiring
Step 1: Rule Extraction

cur_ord = 1;
seqs = get_raw_sequences();
first_order = build_observations(seqs, cur_ord);
rules.append(first_order);

while (rules.last != empty AND current_order < MAX_ORDER) {
    next_cands = get_next_order_candidates(rules.last);
    next_ord_obs = build_observations(seqs, cur_ord, next_cands);
    next_rule = check_and_extend(rules.last, next_obs);
    rules.append(next_rule);
}
Step 1: Rule Extraction
1. Count the number of sequential node interactions at the first-order
   (basically the normal network)
2. Normalize the distributions for each pairwise interaction
3. For each fork node, add the preceding step and see how that changes
   the distribution of the sequence
4. If the change is “significant” (above a selected threshold), add a
   second-order dependency and repeat the process recursively to
determine higher orders

Step 2: Rewire the Network
...
where $L$ is the count of records in the raw data; 
$N$ is the number of unique nodes in the raw data; 
k is the maximum order of dependency; 
$R_i$ is the count of dependencies at order $i$

(*the theoretical upper bound is exponential but is not really helpful for real data sets, in which orders of dependency tend to follow an inverse power law)
Data Sets

- Synthetic web clickstreams (11 billion nodes)
  - Subsets with 1, 5, 10, 100 million nodes
- Global shipping data (3,415,577 voyages made by 65,591 ships between May 1st, 2012 and April 30th, 2013)
Implementation

- C++ implementation of Rule Extraction algorithm
- ~400 lines, not including header files & definitions
- (Close to the same as the original Python implementation, minus a couple utilities)
- Mostly vectors for fast iteration, plus one unordered_map (hash table)
Modules

`build_observations(seqs, cur_ord, next_candidates=None):`
for each c : next_candidates:
    // get all sequences of length cur_ord with target c
    // count all sequences and calculate out degree distribution

`get_next_order_candidates(rules_base_order):`
for each r : rules:
    if r.confidence < THRESHOLD:
        next_order_candidates.append(r)
check_and_extend(rules_base_order, next_order_obs):
for each r : rules_base_order:
    // get all candidate extensions from next_order_obs
    if (get_ext_significance(r, candidates) > EXT_THRESHOLD) {
        next_order_rules.append(candidates);
    }
return new_order_rules;
## Initial Results

<table>
<thead>
<tr>
<th>Number of Seq Pairs</th>
<th>C++ Exec Time (s)</th>
<th>C++ # Rules</th>
<th>Python Exec Time (s)</th>
<th>Python # Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m</td>
<td>22.32</td>
<td>440</td>
<td>26.01</td>
<td>212</td>
</tr>
<tr>
<td>5m</td>
<td>328.52</td>
<td>2,200</td>
<td>116.12</td>
<td>1160</td>
</tr>
<tr>
<td>10m</td>
<td>1321.26</td>
<td>4400</td>
<td>-- (crashed)</td>
<td>--</td>
</tr>
</tbody>
</table>
Future Work

● Scrap the Rule Extraction? :-(

● Possible parallel implementation of Network Rewiring
  ○ Giraph or Stinger?
  ○ Need to think more through use case

● Possible method for validating network representation


