Random Indexing

Brian Kundinger

Overview

- Final piece: Random Indexing

Review of Indexing

- In the literature, indexing is when we remove certain agreement patterns from consideration for matching after making the comparison vectors
- Jody indexed on first and last name, meaning that she chose a threshold for name similarity, and then set all record pairs with name agreement below that threshold to be nonmatching
- Used to reduce comparison space and avoid false matches

BK's Thoughts on Indexing

- If your model isn't able to avoid these types of matchings, you probably need a different model
- In Jody's code, indexing gave modest computational savings. Instead of summing over $n_A \times n_B$ record pairs to update m and u, she only had to sum over record pairs that met the indexing threshold
- Uses human judgement

Random Indexing

- Let P be the number of unique agreement patterns, and denote a particular agreement pattern by h_p for $p \in \{1, \ldots, P\}$. Let $|h_p|$ be the total number of pairs of p^{th} agreement pattern in the data, and let $|h_p|_j$ be the number of record pairs of the p^{th} agreement pattern between $j \in B$ and the records in A.
- Remember that when sampling Z_j, all record pairs of agreement pattern h_p are equally likely. Without knowing anything about the agreement pattern itself, this tells us that any agreement pattern for which $|h_p|_j$ is large is unlikely to be a match.

Data Storage

- Since the overwhelming majority of record pairs are of agreement patterns with mostly $0's$, these pairs contribute the most the the data storage burden.
- All contributions to posterior udpates themselves are governed by $|h_p|_j$, not the record labels themselves. So instead of storing arbitrarily large number of record labels, why not store only some small number R of them?
- Even if remove certain records from consideration, I maintain their probabilities and weights through $|h_p|_j$. This similar to the *u correction* concept from the McVeigh paper (I think!)
- Choosing R is up to the modeller, but much less room for error than choosing fields and thresholds to index on. Also, the goal is not avoid false matches (the model takes care of that), but instead data storage.

Example

- I simulated comparison vectors for a 4000×4000 linkage problem
- The comparison matrix Γ produced by `BRL` is 610 MB.
- After hashing, the object is only 68 MB. Pretty good.
- After running random indexing, the object was only **9 MB.** I ran `parlr`, and got the exact same results!

How does this help?

- I still need to create the comparison vectors in the first place. This is the unavoidable downsize of the Fellegi-Sunter methods, and there is no way to avoid this.
- However, I can create comparison vectors for smaller chunks of the data, reduce the peices through hashing and random indexing, combining results as I go along, without ever needing to store the entire $n_A \times n_B$ matrix of comparison vectors. In fact, no matter how large n_A is, we store at most $n_B \times P \times R$ record pairs.
- Processing these smaller chunks can be done in parallel.
- Should allow us to conduct *significantly* larger record linkage tasks.

Next Steps

- This whole concept is ready to go! I will have a (very rough) first draft ready for you by the next meeting.
- These mechanics will lay the foundation (with slight modifications) for the linkage cluster model I've been developing. This is on hold until the first `parlr` paper is done.