Cold Start Knowledge Base Population with the Knowledge Resolver System for TAC-KBP 2015

Hans Chalupsky
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA, USA
hans@isi.edu

Abstract

This paper describes the Knowledge Resolver system (KRes) and its performance on the TAC-KBP 2015 Cold Start KBP task. KRes is a logic-based inference system aimed at improving statistical relation extraction by deduction and abduction inference towards the best document-level interpretation. For the 2015 evaluation we focused on transitioning our English Slot Filling system used in TAC-KBP 2014 to the more complex Cold Start Knowledge Base Population task. Our 2015 system has some slightly improved slot-filling coverage of 22 slots (as compared to 19 from last year, not counting inverses), and an improved $f1$=0.24 measured on a 50 query slot filling test set from TAC-KBP 2012 compared to 0.212 of our 2014 system. We used RPI's Name Clustering toolkit to perform cross-document entity detection. The integration of clustered entities with entities linked by the UIUC Wikifier did not get ready in time for the evaluation, hence, our submission is a baseline that we are currently improving upon by adding additional inference and linking functionality. Our combined (ALL) scores were $f1$=0.0938 (LDC MAX Micro) and 0.0517 (SF Micro). Our system produced good precision but low recall due to our only partial coverage of slot types and the preliminary nature of our entity linking component.

1 Introduction

This paper describes the Knowledge Resolver system (KRes) and its performance on the TAC-KBP 2015 Cold Start Knowledge Base Population task. KRes is a logic-based inference system based on the PowerLoom knowledge representation and reasoning system, aimed at improving statistical relation extraction by linking extractions from across a section or whole document, and then using deduction and abduction to combine alternative extractions into the best document-level interpretation. We call this *story-level inference* which we have applied successfully for relation extraction and question answering (Chalupsky, 2012). The TAC-KBP Cold Start task elevates this problem of integrating alternative interpretations to the level of a whole corpus or knowledge base, and we are now in the process of generalizing story-level inference to *knowledge-base-level inference* to support the goal of creating a more coherent and complete knowledge base from noisy individual document-level extractions.

This was our first participation in the TAC-KBP Cold Start KB track and our third participation in TAC-KBP overall. For the 2015 evaluation we focused on transitioning our 2014 English Slot Filling system used in TAC-KBP 2014 (Chalupsky, 2014) to the more complex Cold Start Knowledge Base Population task. Our 2015 system has some slightly improved slot-filling coverage of 22 slots (as compared to 19 from last year, not counting inverse slots added by the Cold Start task), and an improved $f1$=0.24 measured on a 50 query slot filling test set from TAC-KBP 2012 compared to 0.212 of our 2014 system. We exclusively used RPI's Name Clustering toolkit to perform cross-document entity detection. Its integration with entities linked by the UIUC Wikifier did not get ready in time for the evaluation, hence, our only submission is a baseline that we are

currently improving upon by adding additional inference and linking functionality. Our final combined (ALL) scores were $f1$=0.0938 (LDC MAX Micro) and 0.0517 (SF Micro) according to the latest scoring revision from February 2016. Our system produced good precision (0.53 for hop-0 queries) but low recall due to the preliminary nature of our entity linking and discovery component, as well as our only partial coverage of TAC-KBP slot types that effectively only addressed 22% of Hop-0+1 queries.

2 Approach

Figure 1 shows the overall architecture of our 2015 TAC-KBP Cold Start system. We begin with a set of NLP toolkits (Stanford's CoreNLP, CMU's SEMAFOR and the UIUC 2011 Wikifier) which are run exhaustively on the source document corpus. This produces an annotation database for each toolkit stored as a compressed archive of tool-specific annotation files (one file per corpus document). After that, the Knowledge Resolver relation detection and slot filling pipeline is run using CoreNLP and SEMAFOR annotations as input and performing pattern-based and statistical relation extraction. After relation arguments have been linked to names, the Slot Filler produces a set of KBP slot hypotheses for each document whose arguments are normalized names and values relative to that document.

In order to produce a properly connected knowledge base, slot arguments have to be converted into KB entities that are linked across documents. To that end, we convert all KBP slot hypothesis arguments into a set of named entities that can be clustered by RPI's Name Clustering toolkit. This is slightly different than its standard configuration, which uses NYU's ENIE named entity tagger to generate input entities. In addition, since Cold Start query entities might come from a source document where no relation was extracted by KRes, we collect named entities detected by CoreNLP from across the whole source corpus that match one of the extracted slot hypothesis arguments. The resulting set of named entities is then clustered by the Name Clustering toolkit based on document similarity.

In the next step, the KB Assembler uses the set of clustered entities and previously extracted slot hypotheses to generate a raw knowledge base. This step also performs some filtering and normalization such as eliminating slots with low confidence scores, addition of inverse slots, generation of canonical mentions and recording of provenance information. Due to time constraints, for the 2015 evaluation, this raw KB was what we submitted as our final and only system result.

The dotted line from Wikified Entities to the KB Assembler is a system component that was not ready at evaluation time and is currently being completed. Wikfied Entities can significantly improve the result of Name Clustering which is aimed primarily at named entities that do not have corresponding Wikipedia pages. Moreover, we are also currently working on a KB Refiner to further improve the output of the KB Assembler based on various constraint checking and inference processes.

2.1 Corpus Preprocessing

In the preprocessing phase, we run a set of NLP toolkits (Stanford's CoreNLP, CMU's SEMAFOR and the UIUC 2011 Wikifier) exhaustively on the source document corpus. This step is very similar to what we did in 2014 (Chalupsky, 2014), so we only briefly review these steps and components here. The main difference is that we run on a significantly smaller corpus, however, due to the short evaluation window for the Cold Start task, it is still necessary to run these steps on a high-performance computing cluster.

2.2 CoreNLP

We used Stanford's CoreNLP toolkit (version 3.3.1) and ran the whole pipeline on each source document to perform tokenization, sentence splitting, POS-tagging, lemmatization, named entity recognition, dependency parsing and within-document coreference resolution. CoreNLP is a very mature tool suite, but problems arise nevertheless. For example, documents that contain long lists of sports scores can trip up the parser and coref systems and lead to performance problems. Fortunately, CoreNLP comes with options to restrict the maximum sentence length and to continue even if processing lead to an error which is very useful when processing large document sets on a cluster.

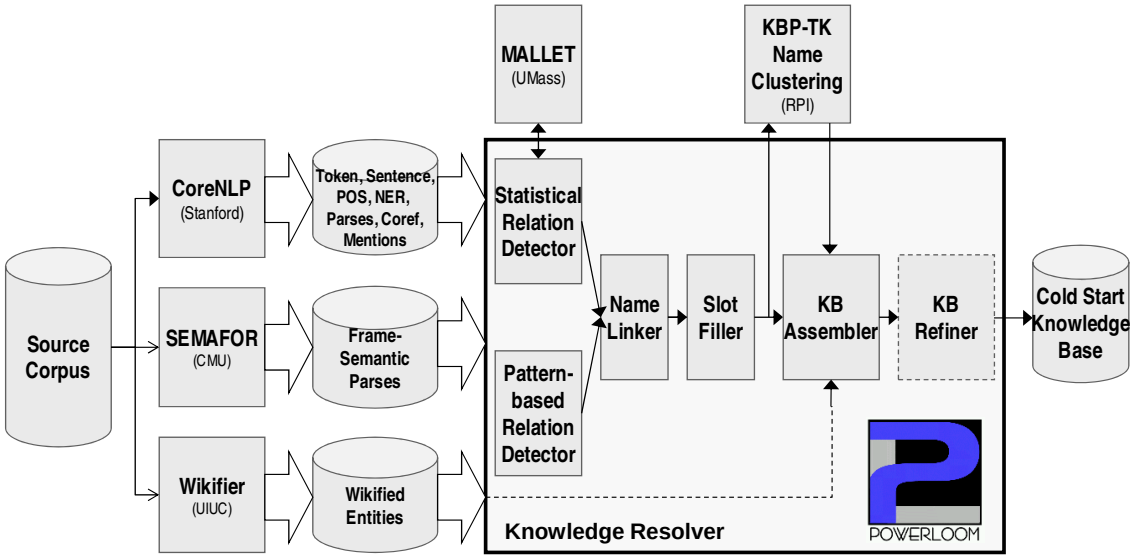

Figure 1: System architecture

2.3 SEMAFOR

SEMAFOR is a frame-semantic parser that was developed at CMU's Language Technology Institute (Das et al., 2014). Frame-semantic parsing is related to semantic role labeling but with a much richer set of semantic types and roles taken from FrameNet (Fillmore et al., 2003). For example, for the sentence "John has a job at a company" we get the following semantic frames which can be used immediately for the extraction of an employment relation:

```
Possession:                 has
Possession.Owner:           John
Possession.Possession:      job at a company
Being_employed:             job
Being_employed.Employer:    at a company
Businesses:                 company
Businesses.Business:        company
```

Identical to our 2014 system, we are using SEMAFOR 2.0 which is not the most recent version, but for which we have various existing tool and translation support already. SEMAFOR 2.0 does not directly support the processing of SGML-formatted documents. Instead it expects a sentence list and preprocessing with a tokenizer and the MST dependency parser before it can be run. To run it effectively on sets of documents, we use CoreNLP for sentence detection, then map multiple documents onto a single sentence list, run the SEMAFOR pipeline on this list, and then map the results back onto individual documents (with proper annotation offsets) and translate them into a PowerLoom format usable by KRes. We processed the 2015 source corpus this way which produces one PowerLoom file per input document.

2.4 UIUC Wikifier

We used the UIUC 2011 Wikifier (Ratinov et al., 2011) to help with cross-document coreference of slot arguments. The 2011 version uses the textual information from a 2009 dump of Wikipedia with 2011 redirects but no other resources (a newer 2013 version of the UIUC Wikifier (Cheng and Roth, 2013) also uses relational information from DBPedia, which is the reason why we did not use it instead). As mentioned above, the linking between Wikifier and Name Clustering outputs did not get ready in time and is currently being developed.

2.5 Knowledge Resolver Pipeline

The core of our 2015 KRes Cold Start system is built upon a slightly improved version of our 2014 slot filling system (Chalupsky, 2014). KRes is a logic-based inference system based on the PowerLoom knowledge representation and reasoning system,[1] aimed at improving relation extraction through

[1] http://www.isi.edu/isd/LOOM/PowerLoom/

the exploitation of richer semantic information and story-level inference.

The 2015 system adds relation extractors for three additional slots (`per:alternate_names`, `org:alternate_names` and `per:schools_attended`), and improves detection of `per:employee_or_member_of` by using a higher quality pattern-based extractor compared to the statistical extractor used before. We also further tuned title extraction and normalization. These improvements increased the score on a 50 query test set from the TAC-KBP 2012 slot filling task (which we did not participate in) from 0.212 to 0.24 using strict scoring on the gold standard from that evaluation. Using anydoc scoring that ignores the identity of the source document and more relaxed value scoring that looks for a minimal 20% overlap between value extents improves this score to $f1$=0.346.

KRes uses the Stanford CoreNLP toolkit for all core language processing tasks such as tokenization, POS-tagging, sentence detection, NER-typing, dependency parsing and coreference resolution. CoreNLP annotations (such as sentences, mentions, NER-types, parse trees, etc.) are then translated into a logic-based data model for the PowerLoom knowledge representation and reasoning system. SEMAFOR frame-semantic parses are handled similarly and translated into our logic-based text annotation data model.

We use an extended version of PowerLoom (compared to the publicly released version), that implements a variety of extensions relevant to NLP such as an extensive data model to represent text annotations, logic-based access to word lists and dictionaries, dependency tree matching, fuzzy string matching, various annotation translators, range indices for efficient annotation inclusion inference, and a number of other utilities. These extensions allow us to run the whole dependency pattern matching, feature generation, inference and result generation process via the PowerLoom inference engine.

2.5.1 Pattern-based Relation Detection

The pattern-based relation detector is very similar to last year's version and described in more detail in (Chalupsky, 2013; Chalupsky, 2014). We only briefly describe some of its features and functionality here. After processing a document through CoreNLP, the pattern-based detector finds potential relation matches based on various dependency patterns. We developed patterns for nine of the TAC-KBP relations:

```
per:age
per:children
per:employee_or_member_of
per:other_family
per:parents
per:siblings
per:spouse
per:title
```

Dependency tree patterns are represented as PowerLoom list terms which are then interpreted by a pattern evaluation predicate. For example, the following pattern would match simple possessive constructs such as "John's father":

```
(listof dg-poss ^ family-relation-word)
```

Elements in the pattern can be dependency graph edge labels (such as `dg-poss`), named or unnamed PowerLoom relations (such as `family-relation-word` which accesses a small special-purpose dictionary for indicators of family relations), verbatim strings (such as prepositions) or the special root anchor constraint indicated by `^`. This pattern would then match the path from "John" to "father" in the following example dependency tree for "There he met John's father Frank":

```
(There
  RCMOD (met
          NSUBJ he
          XCOMP (Frank
                  NSUBJ (father
                          POSS John))))
```

More complex relationships can be expressed using the full expressivity of the PowerLoom logical description language. For example, the following `kappa` expression defines a more complex relationship between a dependency graph and two argument tokens *?x* and *?y*. This particular pattern would match constructions such as "he was appointed to Microsoft", or "he was named White House press secretary":

```
[kappa (?graph ?x ?y)
  (exists (?root)
    (and (dg-subj ?graph ?root ?x)
         (dg-passive-tree ?graph ?root)
         (plausible-person-token ?x)
         (employment-verb-pattern
```

```
   ?root  "A employs P as R")
(dg-dep ?graph ?root ?y)
(different ?x ?root ?y)
(plausible-organization-token ?y]
```

Such patterns can be named and call others recursively if necessary. Dependency labels such as `dg-subj` match all their more specialized versions, such as, for example, `dg-nsubj`. Using this approach, documents are pattern-matched with PowerLoom queries that retrieve all sentences and their dependency trees, match them for any of the defined relation patterns, and then assert any matches so that they can be further analyzed and refined in subsequent processing steps such as name linking and slot filling.

2.5.2 Statistical Relation Detection

For our 2014 system, a main goal was to exploit semantic information as extracted by the SEMAFOR frame-semantic parser to help with relation detection. SEMAFOR provides a very rich semantic vocabulary based on FrameNet, however, detected frames are not necessarily normalized in a way that is immediately exploitable by a relation detector. For example, minor syntactic variations can produce significantly different semantic representations. Let us look at four simple examples with more or less the same semantics:

1. John has a job at a company.
2. John works for a company.
3. John is employed at a company.
4. John is employed by a company.

Example 1 is the same we used above and correctly maps "job" onto a Being_employed frame with the correct association of the employer role. Example 2 maps "works" onto a Usefulness frame with "John" being what is useful and "a company" being the purpose of the Usefulness. Example 3 maps "employed" onto a Using frame with "John" being the instrument of the Using. Example 4 generates the same but additionally identifies "by a company" as the agent of the Using. While all of these interpretations are defensible in some form or other, this illustrates that a rule-based approach similar to what we use for the pattern-based relation detector would not be straight-forward and might have to account for a large number of variations. For this reason, we decided to use elements of SEMAFOR parses as features in a supervised learning approach.

To this end, we built statistical relation detectors for the following slot types:

```
per:alternate_names
per:cities_of_residence
per:countries_of_residence
per:statesorprovinces_of_residence
per:city_of_death
per:country_of_death
per:schools_attended
per:stateorprovince_of_death
per:origin
org:alternate_names
org:city_of_headquarters
org:country_of_headquarters
org:stateorprovince_of_headquarters
org:top_members_employees
```

Our relation detection approach first enumerates possible argument pairs for a relation in a given sentence. For example, for `org:top_members_employees` we enumerate all pairs of mentions of type organization and person. Mention types come from NER-types detected by CoreNLP, Wordnet, as well as gazetteers such as title lists. Potential relation argument pairs are restricted to be within a certain distance in the dependency parse tree generated by CoreNLP. We then use a binary maximum entropy classifier for each relation (based on MALLET) to identify instances of the relation. See (Chalupsky, 2014) for more details on features used.

2.5.3 Name Linking

The name linking component is more or less identical to previous versions and described in more detail in (Chalupsky, 2013). We only briefly describe some of its features and functionality here.

Answers to TAC-KBP queries always involve at least one named argument, the query name, and possibly a second named argument for the slot value as is the case for family relations. Identifying names and linking them to relation arguments is therefore a central part of the task. We chose to separate relation detection and name linking into two separate phases. For example, when processing the sentence "After John left his wife, Susan, ..." we would first detect

the spouse relation between "his" and "wife" and then link those arguments to their respective names "John" and "Susan" via pronoun coreference and an apposition pattern. Decoupling these steps instead of performing them in a single pattern match allows us to reuse the same mechanism across several patterns as well as statistically extracted relations. It also makes the name linking step explicit which allows us to more easily perform additional inference when choosing between alternative candidate arguments.

Given a relation argument x and the head n of a named mention, we look for a link between the two tokens as follows: we first check whether $x = n$, and, if that fails, for a connection via one of these dependency edges in this preference order: NN, APPOS, NSUBJ and DEP. Once a connection is found a name link is established which will prevent a name token to be linked to any other relation arguments in a sentence. In the next step, relative and personal pronouns are resolved to any named referents via coreference links detected by CoreNLP.

2.5.4 Slot Filling

In the slot filling phase, we use the pattern matches and name links established in prior phases to extract TAC-KBP slot values for named mentions found in a document. We additionally perform some value normalization here (e.g., to normalize age values or handle multi-element arguments), and we map relation types such as "cousin" onto the appropriate TAC-KBP slot (such as `per:other_family`). We also perform some simple inferences such as inferring employment from top employment and to handle inverse slots, e.g. infer `per:parents` from `per:children` and vice versa. Normalization for place names is still missing, which accounts for some redundancy and inexact match errors.

At this point, we have a set of KBP slot hypotheses for each document, that are basically elaborated relation mentions with associated provenance information, and whose arguments either are or are linked to names somewhere in the document. In our slot filling system, we would filter these slot hypotheses based on the names in evaluation queries (and their name variants). For the Cold Start system, we keep all slot hypotheses and write them to an intermediate per-document results file which will be used in subsequent steps to build a linked knowledge base.

2.6 Name Clustering

In order to produce a properly connected knowledge base, slot arguments have to be converted into KB entities that are linked across documents. To that end, we convert all KBP slot hypothesis arguments into a set of named entities that can be clustered by RPI's Name Clustering toolkit.[2] This is slightly different than its standard configuration, which uses NYU's ENIE named entity tagger to generate input entities. In addition, since Cold Start query entities might come from a source document where no relation was extracted by KRes, we collect named entities detected by CoreNLP from across the whole source corpus that match one of the extracted slot hypothesis arguments. The resulting set of named entities is then clustered by the Name Clustering toolkit based on document similarity.

2.7 KB Assembly

The KB Assembler uses the set of clustered entities and the previously extracted per-document slot hypotheses to generate a raw knowledge base. This step also performs some filtering and normalization such as elimination of slots with low confidence scores, generation of canonical mentions and recording of provenance information. Due to time constraints, for the 2015 evaluation, this raw KB was what we submitted as our final and only system result.

2.8 KB Refinement

The dotted line from Wikified Entities to the KB Assembler is a system component that was not ready at evaluation time and is currently being completed. Wikfied Entities can significantly improve the result of Name Clustering which is aimed primarily at named entities that do not have corresponding Wikipedia pages. Moreover, we are also currently working on a KB Refiner to further improve the output of the KB Assembler based on various constraint checking and inference processes.

[2] http://nlp.cs.rpi.edu/software/RPI-Software-NameClustering-1.0.0.tar.gz

Hop	Pos	TP	FP	Inex	Dup	Right	Wrong	Prec	Rec	$f1$
0	1268	86	41	16	10	76	67	0.5315	0.0599	0.1077
1	672	30	43	8	3	27	87	0.2368	0.0402	0.0687
ALL	1940	116	84	24	13	103	154	0.4008	0.0531	0.0938

Table 1: Evaluation Results: Micro-averaged LDC MAX Scores

3 Evaluation Results

We submitted results from one single run only which extracted slot fillers exclusively from the TAC KBP 2015 Cold Start source documents. No other external resources were used (however, the UIUC Wikifier implicitly uses the textual portion of a 2009 Wikipedia dump). As described above, given the incompleteness of our cross-document entity discovery and linking component, our results can be only considered an initial baseline.

Official evaluation results of our submission are summarized in Table 1 (as per the latest February 2016 scoring revisions). The table breaks out results by single and multi-hop queries and then in total. "Pos" shows the total number of correct answers for this slot type, "TP" and "FP" the total number of raw true and false positives reported by KRes, and "Inex" and "Dup" the number of inexact and duplicate values reported. "Right" and "Wrong" further refine all answers into one or the other category (e.g., duplicates and inexact matches are counted as incorrect), and the remainder of the columns report precision, recall, and $f1$ based on the refined answer counts.

The results have high precision, particularly for single-hop queries, but low recall. The single hop query results are most comparable to our previous slot filling results and show that we are at less than half of last year's performance of $f1$=0.22 (which would now be higher given our 13% improvements measured on the 2012 test set). While this year's source corpus was more difficult with about 80% of all documents coming from multi-post discussion fora and only 20% from newswire, a significant portion of the performance loss is due to our incomplete entity discovery and linking component. Preliminary results on 2014 Cold Start data show that improved entity linking based on Wikification alone improves the combined $f1$ by about 30%.

Another reason for the lower performance compared to previous slot filling results is that our limited coverage of only 22 out of 41 slot types (not counting inverses added by the Cold Start task) has a more significant impact. Only 59% of Hop-0 queries are addressed by our current set of extracted slots, and only 22% of Hop-0+1 queries. Given the importance of slot chaining in the full Cold Start KB construction task, the biggest potential for improvement might come from extending the coverage of slot types extracted by the Knowledge Resolver, as opposed to improving the quality of individual extractors.

Acknowledgments

This report is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory (AFRL) under agreement number FA8750-12-2-0342. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL or the U.S. Government.

References

H. Chalupsky. 2012. Story-level inference and gap filling to improve machine reading. In *Proceedings of the Twenty-Fifth International FLAIRS Conference*. AAAI Press.

H. Chalupsky. 2013. English slot filling with the Knowledge Resolver system. In *Proceedings of the 2013 Text Analysis Conference (TAC 2013)*. NIST.

H. Chalupsky. 2014. English slot filling with the Knowledge Resolver system. In *Proceedings of the 2014 Text Analysis Conference (TAC 2014)*. NIST.

X. Cheng and D. Roth. 2013. Relational inference for Wikification. In *EMNLP-2013*.

D. Das, D. Chen, A.F.T. Martins, N. Schneider, and N.A. Smith. 2014. Frame-semantic parsing. *Computational Linguistics*, 40(1):9–56, March.

C.J. Fillmore, C.R. Johnson, and M.R.L. Petruck. 2003. Background to FrameNet. *International Journal of Lexicography*, 16(3):235–250.

L. Ratinov, D. Roth, D. Downey, and M. Anderson. 2011. Local and global algorithms for disambiguation to Wikipedia. In *ACL-2011*.