Entity Linking and Slot Filling through Statistical Processing and Inference Rules

Dan Bikel and **Vittorio Castelli** and **Radu Florian** and **Ding-jung Han**
{dbikel,vittorio,raduf,dbhan}@us.ibm.com
IBM TJ Watson Research Center
1101 Kitchawan Rd,
Yorktown Heights, NY 10598

Abstract

Information extraction is one of the fundamentally important tasks in Natural Language Processing, and as such it has been the subject of many evaluations and competitions. The latest such evaluation, the Knowledge Base Population (KBP) part of the Text Analysis Conference 2009, is focusing on two aspects: entity linking and slot filling. This paper presents the design and implementation of the hybrid statistical/rule-based IBM system that performs these two tasks.

1 Introduction

This paper describes the IBM systems for entity linking and slot filler extractor used during the TAC-KBP evaluation. Due to the paucity of training data for the two tasks and the relatively short time available for this task, we have adopted hybrid approaches, where we used statistical classifiers to perform information extraction, including parsing, semantic role label detection, mention and entity detection and recognition, relation detection and recognition, and time normalization, to extract important clues from text, and then implemented a rule-based mapping on top of the rich annotations to perform the individual tasks. As training data becomes available for this task, the system is well positioned to be converted to a fully statistical system, by converting the rules into features and training the system with one of the many available training procedures (*e.g.*, Maximum Entropy, SVM, etc).

In processing the data for entity linking, we made heavy use of DBpedia (v 3.2), which contains automatically extracted relations from the Wikipedia, and used the SPARQL query language (Pérez et al., 2006) to query it. The similarity measure we used to assess the closeness of the entity candidate to an entry in the knowledge database has two components, one related to spelling (name aliases, Wikipedia redirections, edit distance) and the other to contextual similarity between the candidate's document and the database wikipedia entries. This measure was tuned on a development set.

For the slot filling task, we implemented an inference engine on top of the automatically derived relations, using a maximum-entropy cascaded model. Around 120 simple inference rules were manually created (e.g. if *A* is sibling of *B* and *C* is a parent of *A*, then *C* is a parent of *B*). The process of filing slots for a particular entity consists of (1) searching the database for documents that contain the entity, (2) run the inference engine on the returned documents, and (3) filter the identified values, by accumulating evidence across documents. For some of the slot types, the answers can be found directly in arguments of relations (such as AGE, Date-of-Birth, Date-of-Death), while for others we used the inference engine to produce the answers. For instance, in the sentence "John was one of the three Americans who died in the Monday crash." - we use the fact that John is in the relation 'part-of-many' with 'Americans', which are the object of an 'event-violence' that happened 'Monday' to obtain that John died on Monday, which was resolved to be 20081223.

The remainder of the paper is organized as follows: Section 2 presents the data processing that was performed on the TAC corpus, serving as input to the entity detection and slot filing systems. Section 3 describes the design and implementation of the entity linking task, while Section 4 presents the slot-filling system. Section 5 presents the numerical results obtained by the system in the official evaluation and some observations related to the evaluation process. Finally, Section 6 concludes the paper.

2 Data Processing

For the TAC-KBP evaluations we annotated data based on an IBM-developed framework for mention, event, coreference, and relation annotation. We call this framework *Knowledge from Language Understanding and Extraction*, or *KLUE* for short. KLUE is a general-purpose component of our NLP toolkit, serving as a basic building block for our TAC-KBP system. We extended parts of the KLUE framework, in particular its entity taxonomy, to accommodate the TAC-KBP tasks.

In this section we describe the KLUE aspects relevant to the TAC-KBP evaluation. First, we discuss the KLUE mention detection and coreference resolution framework and the derived statistical model we used to preprocess the data. Then we describe the relation framework and the resulting relation statistical model.

2.1 Mention Detection and Coreference Resolution

TAC-KBP is a natural evolution of the ACE program, which included an entity detection and coreference resolution task. The KLUE entity taxonomy departs from that of ACE (NIST, 2008a) in two ways. First, it provides a broader spectrum of entity types: there are 36 entity types and 17 event types, versus seven main ACE types. Second, it is shallower: while ACE defines entity subtypes, KLUE refrains from doing so.

These differences proved to be beneficial for the TAC-KBP evaluation. Adopting a diverse entity-type set provided us with almost all the entity types required for the slot-filling task; the few that were not originally covered were easily added to KLUE.

We trained a mention detection and a coreference model using internally annotated data. Both models are akin to those described in (Florian et al., 2004; Bikel et al., 2008). Mention detection consists of identifying spans of text that refer to specific entities and labeling each span with the entity type. Our mention detection system relies on a sequential detection algorithm centered around a maximum entropy (henceforth MaxEnt (Berger et al., 1996)) Markov model. The mention detection model yields Precision=0.7631, Recall=0.8097, and F-measure=0.7857. Coreference consists of grouping together mentions of the same entity or event. It is performed by selecting the partition of the document mentions that maximizes an approximation of the posterior probability over the space of partitions. Our coreference system uses a MaxEnt model to approximate the posterior probability.

2.2 Relation Detection

Relations are "links" or "connections" between entities or between an entity and an event supported by textual evidence. Our KLUE relation framework is an extension of the ACE relation framework (NIST, 2008b), and was developed independently of the TAC-KBP slot-filling task. Like in ACE, we only annotate relations between entity pairs supported by sentences or portions thereof. Thus, we do not support relations between three or more entities (*e.g.*, father, mother, and children could be linked by a *family* relation, but these types of relations are not part of KLUE), and we do not annotate relations that span more than one sentence. Unlike ACE, we support relations between an entity and an event anchor.

Relying on 36 entity types and 17 event types allowed us to define a rich set of relations. Specifically, we annotated 47 types of relations, substantially more than the 17 categories jointly defined by type and subtype in ACE. As in ACE, KLUE defines a relation mention as an association between two entity mentions, and a relation as a collection of relation mentions.

Our slot-content extractor makes use of a few characteristics of relations. A subset of the KLUE relations are symmetric, notably `colleague`, `competitor`, `near`, `overlaps`, `partner`, and `relative`, while the majority are asymmetric. A handful of relations are transitive, namely `locatedAt`, `partOf`, and `partOfMany`. Some KLUE relations map in a straightforward fashion to TAC-KBP slots, notably `basedIn`, `bornAt`, `bornOn`, `capitalOf`, `citizenOf`, `diedAt`, `diedOf`, `diedOn`, `dissolvedOn`, `educatedAt`, `employedBy`, `founderOf`, `foundedOn`, `managerOf`, `memberOf`, `parentOf`, `populationOf`, `residesIn`, `shareholdersOf`, and `subsidiaryOf`, in addition to several relation types mentioned above. Other TAC-KBP slots correspond to special cases of the general KLUE relations `affectedBy`, `agentOf`, `hasProperty`, thus mapping is not automatic. The slots that do not map to relations are filled through an inference engine described in §4. KLUE relations that are used in this inference process include: `before`, `instrumentOf`, `ownerOf`, `participantIn`, `playsRoleOf`, `productOf` and `timeOf`.

KLUE relations have similar attributes to those of ACE relations. In addition to the *type*, relations have an argument *order*, which, for non-symmetric-relations, denotes whether the leftmost mention is the first or second argument; a *tense*, which denotes whether the text describes, a past, present or ongoing, future relation; a *modality*, which denotes whether the relation is asserted or unspecified; and a *specificity* attribute, which denotes whether both arguments are specific entities or at least one of the arguments is generic (as in "John partnered with one of his colleagues"). Unlike ACE, KLUE relations do not have a *subtype*.

We extract KLUE relations from text using a cascaded MaxEnt model (Kambhatla, 2004). The cascaded model is a multi-stage classifier that analyzes all entity mention pairs and all entity mention-event mention pairs that occur in each sentence. The cascade consists of an existence model, followed by type, argument order, tense, modality, and specificity models in that order.

We trained our relation model with features that fall into five broad categories:

Structural features include the distance between the mention pair been analyzed and the number of intervening mentions.

Lexical features include the mention types and entity types, the non-stop-words between the mentions, PropBank-derived features, features that fire in the presence of lexical patterns, and in the presence of punctuation patterns.

Syntactic features include features computed from the parse tree, such as features extracted from the root of the subtree that covers the mentions being analyzed, features extracted when walking the parse tree from one mention to the other; features computed from the part-of-speech tags; and features that detect specific syntactic patterns, such as the existence of possessive constructions.

Semantic features are computed from the SRL labels of the parse tree nodes.

Relation features fire when the mentions being analyzed appear in relations with other mentions. More specifically, consider the mentions in a sentence, consider all the possible pairs, and construct an ordering. Pick a specific pair $(\mathcal{M}_1, \mathcal{M}_2)$, and consider all the pairs to its left in the ordering. If $\mathcal{M}_1$ or $\mathcal{M}_2$ appears in one such pair, and a relation exists between the mentions in that pair, then a relation feature fires.

From the viewpoint of the TAC-KBP slot-filling task, relation existence, relation type, and argument order are by far the most important attributes, while tense, modality, and specificity are for the most part ignored by the slot filling algorithm described later. Precision, Recall, and F-Measure on relation mention type are 0.7185, 0.6682, and 0.6924, respectively, and are dominated by misses and false alarms.

3 Entity Linking

The IBM entity linking system uses a two-phase approach: cross-document coreference followed by entity linking.

After mention detection and within-document coreference, a cross-document coreference component attempts to link all within-document entities to some entity in our database, a superset of the provided knowledge base (KB). If a link is found, the unique identifier of such an entity is transformed into a cross-document entity id. This first phase occurs on all entities in all documents in the corpus, and thus formally constitutes the last step in our data processing pipeline.

In the second phase, an entity linking query is processed by examining every occurrence of the query string in the context of the query document. If a mention (detected by our mention detector) overlaps an occurrence of the query string, and that mention was given a cross-document id from the KB in the first pass, then that id is output as the system response. Otherwise, the system attempts to link the query string in its context to a KB entity using the same cross-document coreference component of the first phase.

3.1 Data for the database

We constructed our database of entities from two sources: the provided KB and the dbpedia.[1] Crucially, dbpedia v3.2 includes an ontology with nodes that closely correspond to the entity linking task's definition of a PERSON, GPE and ORGANIZATION. Since the dbpedia is provided in the N-triples format, we were able to mine its information using the rich SPARQL query language. Figure 1 shows the basic statistics of both sources of data.

	KB	**dbpedia v3.2**
PER	116498	211029
GPE	114523	179842
ORG	55813	75627
UKN	531907	n/a
Total	**818741**	**466498**

Figure 1: Basic statistics for our entity database.

Given that both the KB and the dbpedia are culled from the Wikipedia, there is considerable overlap in the entities of each database: all but about 60,000 entities in the dbpedia are present in the KB. However, unlike the KB, the entities mined from the dbpedia all have types, so whenever a KB entity of type UKN also appeared in the dbpedia, we coerced its type to be that of the dbpedia's entry. We were thus able to transform 153,641 UKN entities from the KB to a "known" type.

3.2 Entity matching strategy

At bottom, entity linking is about similarity: we seek the best similarity metric so that given a name in context we can find the most similar entity in our database, or, if none appears similar, output NIL. Given the great size of our database, we need an efficient way to narrow down the search before we can apply a detailed similarity metric. Thus, we decompose entity similarity into two subproblems: a fast match followed by a "slow match".

Following the approach taken in (Bikel et al., 2008), we do a fast match by only doing a fuzzy name match. Fuzzy name matching against a large database can be viewed as an information retrieval problem. Using the

[1]The dbpedia is available at `http://dbpedia.org/`.

open-source search engine Lucene, we index each entity's name and aliases by all its character trigrams, and then perform searches based on all character trigrams of query names. The names that have the most character trigrams in common with the set of trigrams of the query name will tend to have the highest scores. This approach works remarkably well at putting the correct entity within the top 50 hits, and can handle spelling variations. Since aliases are crucial to this method of fast match, we made extensive use of dbpedia in order to capture a wide variety of aliases for each entity, including its `redirects` dataset.

After narrowing the search space, our system performs a "slow match", attempting to match the query entity to the top hits from the fast match. The slow match relies on a more sophisticated name-matching technique, as well as a metric for evaluating context. A name similarity score is provided by SoftTFIDF from the SecondString (Cohen et al., 2003), using Jaro-Winkler as the secondary token-matching metric with a weight of 0.95.

Our context-matching score is based on cosine similarity; we call it a "cosine inclusion score". Let the context of query entity C_{query} as the set of non–stop words of all mentions in the current sentence plus those of the previous and following sentences. Let the context of a database entity C_{db} be the set of non–stop words from its infobox slot values (obtained both from the KB and the dbpedia). Our context similarity score $c_{\text{inclusion}}$ measures the overlap of the context words of a database entity with those of the query entity:

$$c_{\text{inclusion}} = \frac{\sum_{w \in C_{\text{query}} \cap C_{\text{db}}} \text{idf}(w)}{\sum_{w \in C_{\text{query}}} \text{idf}(w)}, \qquad (1)$$

where $\text{idf}(w)$ is the log of the inverse document frequency of term w.

The overall similarity score s is a simple weighted combination in log space of the name matching score ν and the context similarity score $c_{\text{inclusion}}$:

$$s = \log(\nu) + \alpha \cdot \log(c_{\text{inclusion}}) \qquad (2)$$

3.3 Entity linking

As described at the beginning of §3, we employ a two-pass strategy for entity linking. First, our system processes all mentions detected in a document with its coreference component, and then examines the mentions that overlap with the query name. If any has been linked to a KB entity, the system simply outputs that entity id. Otherwise, the coreference component examines each instance of the query name in context, employing the coreference component again, using its fast and "slow" matches described above.

One issue with the fast match is that it can ignore important cases of name ambiguity. For example, when the query name is `George Bush`, the top-scoring entity returned by the fast match is `George_Bush_(NASCAR)`. Our solution to this is to prefer "famous" entities. We accomplish this by using the large `pagelinks` dataset that is part of dbpedia, creating a new N-triples dataset that contains the number of incoming Wikipedia page links to each page. We use this number of incoming links as a proxy for fame; to wit, George Bush (the 43rd President of the U.S.) has 8133 incoming links, whereas George Bush the NASCAR driver has 2.

Accordingly, the entity linking system first attempts to link only among "famous" entities. We define "famous" as having a number of incoming page links greater than 100 (this parameter optimized on a small, held-out development set). Unless a famous entity in the database matches the query entity with a score higher than a specific threshold, the system tries again, this time attempting to match the query entity against all entities in the database. The process is illustrated in Figure 2.

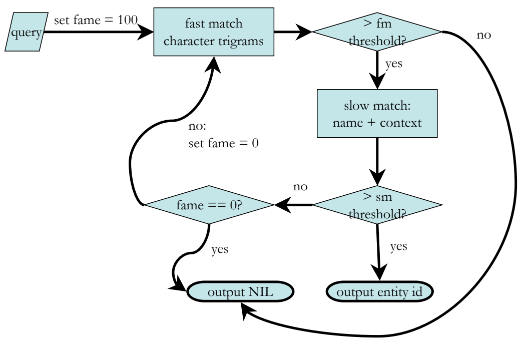

Figure 2: Two-pass entity linking strategy.

4 Slot Filling

There are three major components in the IBM system for the slot filling task. The first is our information extraction (IE) system described in §2. The second component is a search engine capable of finding the documents containing a specific entity. The third component is a slot filler extractor that works on the identified documents to infer the appropriate slot fillers for a given query entity.

4.1 Document Retrieval

For this task we have adopted the open-source Lucene search engine for identifying relevant documents for extraction. The preprocessed documents were indexed to support queries targeting (a) a specific KB id or (b) a specific entity type plus its spelling (PERSON, `Barack Obama`).[2] The search results consist of relevant docu-

[2]In fact, we used the same Lucene index created for the entity linking task, as described in §3.2.

ments and the relevant mentions (identified by our IE engine) within those documents.

At run-time when a query includes a KB node id, we query our search engine using both that KB node id and the name string. For a query without a KB node id, we use only the name string for searching.

In the latter case, we improve precision of document retrieval by checking the relevant mentions identified within the documents. For each of these relevant mentions, the system checks all the mentions in its coreference chain. If none of those mentions' text matches the query string, the system discards the document.

4.2 Slot Extraction

As discussed in §2.2 above, although many TAC-KBP slots can be filled by KLUE relations, there are still slots that require a combination of KLUE structures – mentions, coreference chains and relations – to produce their fillers. Our system therefore includes a third component to produce the final output.

For example, no single structure identified by our IE engine maps to slot `per:title:`; however, we can perform reasoning on a set of structures to fill this slot. Consider the sentence `The 44th President is Barack Obama.` Our system identifies two mentions, `President` of type OCCUPATION and `Barack Obama` of type PERSON. The two mentions are connected via coreference. The system is able to make use of the coreference chain and mention types to fill the `per:title:` slot of `Obama` with `president`. Next, consider the sentence `President Barack Obama addressed Congress`. There are still two mentions of type OCCUPATION and PERSON, and KLUE relates them via the `hasProperty` relation.

The above examples show that the rich set of KLUE structures may be combined with a small amount of inference to produce slot fillers. We designed three types of rules for slot extraction: COREF rules, RELATION rules and IRELATION rules ("inferred relation").

- COREF rules operate on pairs of mentions that corefer.
- RELATION rules are used when a chain of KLUE relations links two mentions.
- IRELATION rules are similar to RELATION rules, but may also include previously-extracted TAC-KBP slots. The advantage of this type of rules is that work done in extracting a slot can be re-used to infer another slot.

Since we allow recursion via IRELATION rules, it is vital to keep track of dependencies among the slot rules. If the system makes a deduction about one slot, it must update all deductions based on that slot. The system in effect performs non-monotonic reasoning, which makes these types of rules flexible and powerful.

Our system also fills slots for selected entities other than the query entity to aid its search. For example, to extract `per:parents` for a query person Q, it is potentially useful to extract `per:parents` for a non-query person S because if S is a sibling of Q and P is a parent of S, then P is also a parent of Q. Therefore, our system will first identify all entities that are weakly connected to the query entity via a relation, and then extract all applicable slot fillers for each of these entities.

In addition to the slot-specific extraction rules, our system also exploits two types of general inference rules on relations. The first type concerns the basic properties of KLUE relations, *viz.*, symmetry, transitivity and equivalence (we do not have reflexive relations). For example, if person A is a colleague of person B, then B is also a colleague of A by symmetry. The second type of inference rules encodes useful world knowledge. For example, if a person is a manager in an organization, then that person is also a member of that organization. These two types of rules are executed before any slot-specific rules.

4.3 Producing final output

We need filler ranking for selecting only one filler for the single-valued slots, and for filtering for the list-valued slots. For each filler candidate c_i

$$Score\left(c_i\right) = count\left(c_i\right) + \frac{1}{n} docCount\left(c_i\right)$$

where $count\left(c_i\right)$ is the frequency of the answer c_i, n is the total number of candidates, and $docCount\left(c_i\right)$ is the document frequency of the candidate c_i.

The final output for each slot is produced by the following steps:

- The system selects the candidate with the highest score for single-valued slots. For list-valued slots, the score is used to form a ranked list which is then processed to remove redundancy based on string match. Additionally, for time-related fillers (*e.g.*, `per:date_of_birth`), we compare the normalized times to remove duplicates.
- The system uses the same redundancy-elimination strategy to identify slot values that already appear in the KB. These values are not output.
- The system identifies slot values that are entities in the KB and links them using the entity linking component described in §3. Additionally, slot values that are linked to the same KB entries are considered duplicates and only one of them is kept.

5 Evaluation Results

The results of our entity linking system and our slot filler system on the evaluation data are summarized in Tables 1 and 2, respectively. Since there was very little annotated data available, the systems were tuned on such existent data. For the entity linking data, the development set consisted of internally annotated 120 examples, plus a small number of resources that were exchanged by the task participants.

For the slot filling task, we selected 30 frequent entities (10 of each type PER, ORG and GPE) in a set of 568 documents, from which we extracted roughly 1500 fillers (actual text spans of the fillers). These data drove the development of the rules described above, and allowed tuning of the various parameters of the system.

	Micro-average			Macro-average		
	No. Q	IBM*	all NIL	No. Q	IBM*	all NIL
KB	1675	63.5%	0%	182	42.3%	0%
NILL	2229	71.8%	100%	378	66.2%	100%
total	3904	68.2%	57.1%	560	58.4%	67.5%

Table 1: Results of the highest-accuracy IBM entity linking system on the evaluation queries.

Accuracy	IBM1	IBM2	Baseline (NULL)
Single-valued	0.816	0.816	0.847
List-valued	0.742	0.715	0.741
Overall	0.779	0.765	0.794

Table 2: Results of the IBM slot filler extraction system on the evaluation queries.

It should be noted that the results shown in Table 1 are obtained with a corrected system, which was submitted after the close of the entity linking evaluation cycle. During the slot filling evaluation period, we noticed that the formula computing the similarity of two mentions - Equation (2), was being computed wrongly as

$$s = \log(\nu) - \alpha \cdot \log(c_{\text{inclusion}})$$

This had the negative effect of penalizing the entities that have contexts in common with the query entity.

We also note that tuning the threshold for deciding whether an entity is in the KB or not is a very delicate matter, as it depends strongly on the test distribution. On our internal development test sets, we were seeing accuracies of over 90%. Also, when we took a random sample of 10% of the evaluation queries and re-evaluated our system but with that threshold turned up (to be more conservative and produce fewer false alarms), we saw our micro-average score jump dramatically from 68% to 75%. These results point to the ever-critical need of having development data that is representative of test data.

6 Conclusion

The Knowledge Base Population task, part of the 2009 Text Analysis Conference, is the latest in a long tradition of information extraction evaluations – which include the MUC conferences, the CoNLL 2002 and 2003 shared tasks, and the NIST-organized ACE evaluation. It facilitates and encourages progress by moving to more involved and realistic tasks – namely, cross-document coreference and slot filling. This paper presents the systems that produced the results submitted by IBM to this evaluation. The paper describes in details the hybrid statistical / rule-based models used by the systems, which make heavy use of automatically extracted mentions, entities (coreference information) and relations to quickly prototype and implement entity linking and slot filling.

For entity linking, we have implemented a system that utilizes information extracted from Wikipedia and DBpedia, and heuristic similarity measures based on character trigrams and context similarity to link the query entity with the KB. Its two-faceted design – composed of a fast search followed by a thorough search – allows it to investigate efficiently, yet in depth, the corpus to identify and score possible candidates.

The slot filling system is built on top of the information extracted by our in-house IE system, which identifies a large number of entity and relation types. This system has three components: a search engine, a slot extractor and scorer, and a summarization system. The mentions and relations extracted from documents found by the search engine are analyzed by an inference engine based on Horn clauses, built to map the basic relations to corresponding slots. The resulting output is filtered to produce the desired results (either single-valued or list-valued slots).

The systems were tuned on development sets that were both obtained from the community (for entity linking) and annotated in-house. We found that some of these data sets were not good representatives of the actual evaluation data, such as in the case of entity linking; this resulted in lower performance than we had anticipated. However, when the mismatch between the training and evaluation data was less pronounced, such as for slot filling, the resulting system was more competitive.

References

A. Berger, S. Della Pietra, and V. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Daniel Bikel, Vittorio Castelli, Radu Florian, Xiaoqiang

Luo, Scott McCarley, Todd Ward, and Imed Zitouni. 2008. IBM ACE'08 system description. In *Proceedings of ACE'08*, Alexandria, VA, May.

William W. Cohen, Pradeep Ravikumar, and Stephen Fienberg. 2003. A comparison of string metrics for matching names and records. In *KDD Workshop on Data Cleaning and Object Consolidation*.

R. Florian, H. Hassan, A. Ittycheriah, H. Jing, N. Kambhatla, X. Luo, N Nicolov, and S Roukos. 2004. A statistical model for multilingual entity detection and tracking. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 1–8.

Nanda Kambhatla. 2004. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 22, Morristown, NJ, USA. Association for Computational Linguistics.

NIST. 2008a. Ace (automatic content extraction) english annotation guidelines for entities. http://projects.ldc.upenn.edu/ace/docs/English-Entities-Guidelines_v6.6.pdf.

NIST. 2008b. Ace (automatic content extraction) english annotation guidelines for relations. http://projects.ldc.upenn.edu/ace/docs/English-Relations-Guidelines_v6.2.pdf.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2006. Semantics and complexity of sparql. *CoRR*, abs/cs/0605124.

Appendix: KLUE entities, events, and relations

KLUE defines 36 types of entities, namely: AGE, ANIMAL, AWARD, CARDINAL, DATE, DEGREE, DISEASE, DURATION, EMAIL, EVENT, FACILITY, FOOD, GEOLOGICALOBJ, GPE, LAW, LOCATION, MEASURE, MONEY, ORDINAL, ORGAN, ORGANIZATION, PEOPLE, PERCENT, PERSON, PERSONPEOPLE, PHONE, PLANT, PRODUCT, SUBSTANCE, TICKER, TIME, TITLEWORK, VEHICLE, WEAPON, WEATHER, and WEB.

KLUE defines 17 types of events, namely: EVENT-AWARD, EVENT-BUSINESS, EVENT-COMMUNICATION, EVENT-CRIME, EVENT-CUSTODY, EVENT-DEMONSTRATION, EVENT-DISASTER, EVENT-EDUCATION, EVENT-ELECTION, EVENT-LEGAL, EVENT-LEGISLATION, EVENT-MEETING, EVENT-PERFORMANCE, EVENT-PERSONNEL, EVENT-SPORTS, EVENT-VIOLENCE, and the catch-all category EVENT.

Finally, KLUE defines 47 types of relations: `affectedBy`, `affiliatedWith`, `agentOf`, `authorOf`, `awardedBy`, `awardedTo`, `basedIn`, `before`, `bornAt`, `bornOn`, `capitalOf`, `citizenOf`, `clientOf`, `colleague`, `competitor`, `diedAt`, `diedOf`, `diedOn`, `dissolvedOn`, `educatedAt`, `employedBy`, `founderOf`, `foundedOn`, `hasAttribute`, `instrumentOf`, `locatedAt`, `managerOf`, `memberOf`, `near`, `overlaps`, `ownerOf`, `parentOf`, `participantIn`, `partner`, `partOf`, `partOfMany`, `playsRoleOf`, `populationOf`, `productOf`, `relative`, `residesIn`, `shareholdersOf`, `siblingOf`, `spokespersonFor`, `spouseOf`, `subsidiaryOf`, and `timeOf`.