Slot Filling through Statistical Processing and Inference Rules

Vittorio Castelli and **Radu Florian** and **Ding-jung Han**
{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 2010, 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 the two tasks. The system presented here is an extension of the system participating in last year's evaluation.

1 Introduction

This paper describes the IBM systems for entity linking and slot filler extractor used during the TAC-KBP evaluation, with emphasis on the slot filler extractor. While a system for entity linking was used in a significant fashion, we did not participate in the entity linking evaluation. The 2010 system is an extension of the IBM system participating in the year 2009 TAC-KBP evaluation.

We have continued to use hybrid approaches, using 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. We have also started to investigate statistical approaches for filling the slots, by implementing a statistical post-filtering model (based on the Maximum Entropy principle), described later in the paper.

The entity linking module is pretty much unchanged from the last year's model – we will only describe it briefly in this report, as it it presented fully in (Bikel et al., 2009). In short, the system uses a combination of trigram fast-match and in-depth, contextual string similarity to assert if the given string is in the knowledge database, and to find the best matching entry.

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.

After the TAC evaluation, we have implemented a maximum-entropy high-pass filter, which is designed to increase the precision of the system, without hurting the recall too severely. We extract features from system's output and learn a low-pass filter that tries to eliminate wrong answers (low-pass on $P\left(Wrong|\text{filler}\right)$). The preliminary results of its effectiveness are encouraging.

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 6 presents the numerical results obtained by the system in the official evaluation and some observations related to the evaluation process. Finally, Section 7 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(Han, 2010). 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 evolved independently of TAC-KBP from that of ACE (NIST, 2008a), from which it departs in two ways. First, it provides a broader spectrum of entity types: there are 36 entity types and 17 event types, versus 7 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 sequential algorithm that combines a stack decoder(Jelinek, 1969)

with a cascaded MaxEnt model (Kambhatla, 2004). The stack decoder efficiently explores multiple hypotheses on existence and types of relations among the mention pairs within a sentence. The cascaded model is used by the stack decoder to analyze individual mention pairs and mention-event mention pairs: it yields estimates of the probability of existence of a relation and of the different relation types should such relation exists. As the stack decoder analyzes mention pairs in a predefined order, it explores a tree of possible hypotheses, where all nodes at the same depth corresponds to the same mention pair (or entity mention-event mention pair) and each edge corresponds either to a different allowable relation type or to the decision that no relation exists that links the mentions. The tree edges are decorated with the probabilities produced by the cascaded classifier, and at each step the path in the tree with the highest probability is further explored. The cascaded model is implemented as a multi-stage classifier that analyzes the entity mention pair or entity mention-event mention pair selected by the stack decoder. 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.

Note how the relation features depend on the specific path through the hypothesis tree currently being explored by the stack decoder, while the others do not. Thus, only relation features need to be recomputed if the same pair of mentions is analyzed while exploring different branches of the hypotheses tree. We found that using a stack decoder yields a gain of 2 to 3 F points (depending on the selected operating point) over using just the cascaded model with the same feature.

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. Although we did not enter the formal entity linking evaluation, we nevertheless used this approach to link slot values to KB entities as necessary, as well as for redundancy reduction.

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

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

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.

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), the fast match consists of a fuzzy name match, which can be viewed as an information retrieval problem. Using the 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:

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

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 $c_{\text{inclusion}}$:

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

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 documents 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 the name string and the reference document associated with the query ("query document") 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.

If the query entity is specified using an acronym, we further perform query expansion using the query document. We identify the mentions of the query entity that appear in the document, and we identify all the mentions that are coreferent with the acronym, and we select the named mentions that are not acronyms. We perform document retrieval using both the acronym and the names that appear in the selected mentions. This has two benefits: the first is to reduce the chance of retrieving information about entities that share the same acronym as the

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

query but are completely unrelated. An illustrative example is the query CC: from the query document we can infer that the acronym CC refers to the (UK) Competition Commission, an independent public body which regulates mergers, regulations of major industries, and markets; however, according to Wikipedia, the acronym applies to at least 14 different companies or organizations, none of which is the Competition Commission. The second benefit is that we gain the ability to retrieve documents where the organization does not appear with its acronym; for example, the UK Competition Commission is rarely referred to as CC in the news.

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 org:top_members/employees; however, we can perform reasoning on a set of structures to fill this slot. Consider the sentence ... said William Rhodes, chairman of New York-based Citibank. Our system first identifies two PERSON mentions, William Rhodes and chairman, and the query mention Citibank. It also detects that William Rhodes and chairman corefers each other, and the chairman and Citibank are connected by the KLUE relation EMPLOYEDBY. By considering all of these facts the system concludes Mr. Rhodes is an employee of Citibank, and his title is one of the "top" positions within that organization.

The example above shows 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 the "closest" n entities (based on the numbers of the relations connecting an entity to the query entity).

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 Classifying Geo-political Entities (GPE)

One of the differences in this year's evaluation is the requirement of classifying a GPE filler; e.g., instead of having just one slot type per:residences, we have in this year per:countries_of_residence, per:stateorprovinces_of_residence and per:cities_of_residence. We performed this classification task by simply linking a GPE filler to a KB entity using techniques described in §3, and searching for keywords in the first sentence in the document accompanying the entity in the KB. If a KB entity cannot be found, we take advantage of any relevant KLUE mention associated with the filler and classify that instead. For example, although "outskirts of Mumbai" cannot be linked to any KB entity as is, but because our IE system detected the existence of KLUE relation ISLOCATEDAT between the filler and another mention "India", we proceed to classify the latter as a country; therefore we conclude that the filler must be a city.

4.4 Producing final output

In order to produce the final result, the extracted fillers first undergo a merging process so duplicates can be merged together and counts can be gathered. The merged lists of fillers are then sorted descendingly based on the instance counts, and only the top-n fillers are kept (n was set differently for different list-valued slots, and was set to 1 for all single-valued slots). Following we describe

the merging process in more details:

- We first remove fillers that are exact matches to the query strings.
- A filler is also removed if it already appears in the KB: this is done by checking if both the filler and a value in the KB for the query entity can be both linked to the same KB entity, using techniques described in §3.
- All of the fillers are then sorted lexicographically, and adjacent fillers are matched and if they match, they are merged into an equivalence class. The merging process goes on until a fixed point is reached.
- Finally we pick the filler that has the largest instance count in an equivalence class to be its representative, and link it to a KB entity: if the linking is possible, we compare the KB entities of two equivalence classes and if they match, we further merge the two classes.

Special logic is also implemented for certain slot types. For example, for time-related fillers (*e.g.*, `per:date_of_birth`), we compare the normalized times to remove duplicates.

5 Filtering Out Bad Fillers

The rule-based slot filler extractor described in the previous section is prone to false alarms. Since the official evaluation, we have implemented a filtering stage that attempts to weed out false alarms without substantially increasing the number of misses. The crucial assumption is that we can use data and learn to distinguish between "good" inference paths and "bad" inference paths. Assume that the query entity is "International Business Machines", and consider the fragment "Sam Palmisano, the current chief executive officer of IBM, ...".

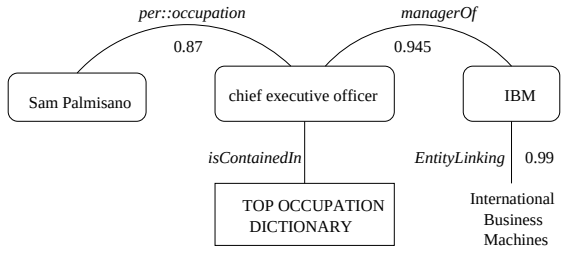

Figure 2: The information used by the slot filler extractor to deduce that "Sam Palmisano" is a company::top_employee of the query entity "International Business Machines".

The rule-based slot filler relies on the cross-document entity-coreference system based on the entity-linking component to identify with high certainty that "IBM" and "International Business Machines" are mentions of the same entity. The relation extractor detects a `managerOf` relation mention between the person mention "chief executive officer" and the organization mention "IBM". The rule-based slot filler also detects that "chief executive officer" is the occupation of the person mention "Sam Palmisano"; this is based on the coreference system linking "Sam Palmisano" and "chief executive officer", and on the mention detection system identifying "Sam Palmisano" as the name of a person, and "chief executive officer" as a nominal mention of a person, having role `occupation`. Finally, "chief executive officer" appears in a dictionary of top occupations. It is apparent from this example how even a simple case can be the result of a complex set of inference steps. We expose the inference chain to the statistical filter, which uses it to analyze the outputs of the rule-based slot filler. The statistical filter is based on a MaxEnt model. It uses features obtained by decomposing the inference chain into steps, quantizing the probabilities associated with individual inference steps when available, and constructing appropriate combinations of these features. For example, the link between "International Business Machines" and "IBM" would be captured by the feature `entity.Q_xdc`, which states that the query entity is matched by the cross-document-cofererence component to a mention in the document. Similarly, the feature `rel.G_managerOf_H_SCORE=HIGH` captures the fact that the relation "chief executive office" `managerOf` "IBM" is detected with high posterior probability.

Using the 100 queries provided by LDC for the task, we extract features from the slot fillers output by the system. These features were used to train a Maximum Entropy classifier that predicts whether a filler is correct (C) or wrong (W). This filter can be applied to the output of the system in two ways:

- For each filler, if the classifier labels the answer as *Wrong*, eliminate the filler from the output.
- Eliminate any filler that is labeled as *Wrong* and on which the classifier is confident (the probability $P\,(W|\text{filler})$is above a threshold).

Note that the above usages of the correctness classifier basically implements a low-pass filter on the quantity $P\,(W|\text{filler})$, with the first one being a special case of the second, where the threshold is 0.5.

Table **1** below reports the 10-fold cross-validation estimates of the accuracy of the filter, obtained on a data set containing **3747** positive and **4461** negative examples. The Table shows both standard accuracy and specific types of errors for the two types of using the model

System	Precision	Recall	F-measure
IBM1	28.0	27.0	27.5
IBM2	25.3	29.0	27.0
IBM3	31.0	25.9	28.2
Human	70.1	54.1	61.1
TopSystem	66.8	64.8	65.8
Top2System	66.5	18.7	29.2

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

as a filter, and for the baseline, which accepts all inputs as correct (we remind the reader that the input is actually the output of the system, which is intended to be correct).

The goal of the filter is to increase precision as much as possible, without reducing recall – in other words, reducing the false negative rate and increasing the true negative rate.

The work on this low-pass filtering system is very preliminary, and we don't report performance on applying it to the actual system output directly, yet.

6 Evaluation Results

We submitted 3 runs for the slot filling task. They differed slightly in terms of how many documents were considered for processing from the initial document search, and whether we filtered out entity types on which we had evaluated to have poor performance:

- IBM1 considered a large number of documents for processing (5,000), and had filtered output (2 slot types: employee_of and per:charges)
- IBM2 considered a large number of documents, and no filtered output
- IBM3 considered a smaller number of documents for processing (800) and had the same kind of filtered output as IBM1.

The results of our slot filler system on the evaluation data are summarized in Table 2. We have used only the evaluation data provided by LDC for this task, and the system has not accessed the Internet at any time during the evaluation. We have not used the community annotated data, though we plan to make use of it going forward. All three systems have performed similarly, with small differences in terms of precision and recall, with the system that has highest score being precision-driven. There is a considerable performance gap to the estimated human performance, and to the top performing system in the evaluation. Note that the top performing system exceeds human performance (rather surprisingly), and the next competing system (which is also precision-driven).

One immediate goal for us is to enable the low-pass filtering method described in Section 5 to further improve the precision of the system, and, going forward, to convert the rule-based system into a fully trainable system.

6.1 Slot Filling Inference Analysis

In order to measure the performance improvement from the different types of rules implemented for the *slot filling* task, we instrumented our system to be able to apply only specific types of rules; we devised 4 conditions of increasingly more complex rules:

1. Apply only rules directly derived from the information extraction system (basically, renaming relations extracted)
2. Apply rules derived from symmetry, etc
3. Apply rules from 2. and also rules derived by simple relation implications (e.g. *X is CEO of Y => X is employed by Y*)
4. Apply rules from 3, and rules derived from transitivity (relation chaining)
5. Apply rules from 4, and also rules derived from recursive reasoning (i.e. using rules defined at 4 and 5).

Figure 3 presents the results. One can observe the improvement in performance between levels 1-4 is very small, but the move from level 4 to level 5 results in a 30% improvement in performance, both for precision and recall.

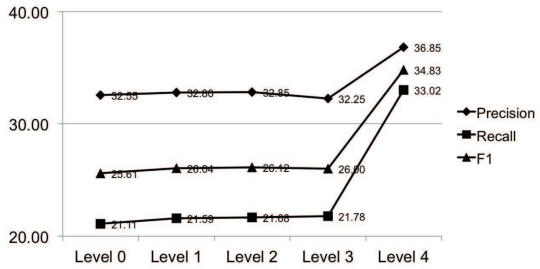

Figure 3: Effect of Inference on Performance

	Accuracy	True Positive Rate	False Negative Rate	False Positive Rate	True Negative Rate
Baseline	33.4	33.4	0	66.7	0
Filter default	73.3	19.4	12.7	14.0	53.9
Low-pass Filter	56.7	31.3	2.0	41.3	25.4

Table 1: Filtering Performance

7 Conclusion

The Knowledge Base Population task, part of the 2010 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 evaluations. It facilitates and encourages progress by moving to more involved and realistic tasks – namely, cross-document coreference and slot filling. This paper describes the hybrid statistical / rule-based models used by the IBM system, which make heavy use of automatically extracted mentions, entities (coreference information) and relations to quickly prototype and implement entity linking and slot filling.

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 system were tuned on development sets that were provided by NIST and LDC, and performed competitively, though we have preferred for it to have considerably higher precision.

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.

Dan Bikel, Vittorio Castelli, Radu Florian, and Ding-Jung Han. 2009. Entity linking and slot filling through statistical processing and inference rules. In NIST, editor, *Proceedings of the Text Analysis Conference*, November.

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.

D.-J. Han. 2010. KLUE annotation guidelines - version 2.0. Technical Report RC25042, IBM Research.

F. Jelinek. 1969. A fast sequential decoding algorithm using a stack. 13:675–685, November.

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.