The UC3M team at the Knowledge Base Population task

César de Pablo-Sánchez, Juan Perea,
Isabel Segura-Bedmar, Paloma Martínez

{cdepablo,jiperea,isegura,pmf}@inf.uc3m.es
Computer Science Department
Universidad Carlos III de Madrid
28911 Leganés, Spain

Abstract

The UC3M team participated in the two subtasks proposed in the Knowledge Base Population (KBP) task at TAC09. We adapted open-source systems to propose initial solutions to both tasks, Entity Linking and Slot Filling. In Entity Linking we have indexed entities and related documents in the Knowledge Base using Lucene. We experimented with different entity representations and TF-IDF similarity measures as a baseline. For the Slot Filling task, we reused Open-Ephyra QA system and combined several extraction strategies and sources like the Web. Our experiments confirm that the Web may be useful to locate more slot values in a local collection, but more accurate confidence estimation methods are needed to avoid updating the KB with spurious values.

1 Introduction

The KBP task has proposed the challenge of automatically updating of a large Knowlege Base (KB) by extracting new and timely information from a textual collection. The focus of the task has been directed to two of the most basic operations. The first subtask, Entity Linking (EL), consists on disambiguating individual textual references of entities to their representation on the KB. For example, a mention like *Michael Jordan* in a document could refer to several personal entities and the correct one in the KB should be identified. Besides, it could be that the mention refers to a completely different person which has no associated node in the KB yet. The second task, Slot Filling (SF), involves the acquisition of values for characteristic relations and attributes for the target entities. Besides, those slots values that refer to entities in the KB should be linked to the correct nodes too.

In the current setting, the KB has been generated automatically from structured information from Wikipedia. The KB contains 818.741 different entities of three different types; persons (PER), organizations(ORG) and geo-political locations (GPE). A number of generic slots for each of the types are also defined and outlined in the appendix. Slot values have been also filled automatically by mapping Wikipedia infofoxes to the generic slots. No normalization on the textual values has been carried but named values which corresponds to entities in the KB are transformed into links between entities. The mapping between the Wikipedia infoboxes and the generic slot values is also provided though Wikipedia infoboxes use a much larger and inconsistent categorization as it is community generated but no control or typing mechanism is enforced. The source document collection is mainly composed of newswire text from different press agencies. There are also a small number of documents with different genres, like broadcast news transcriptions and weblogs. The collection contains about 1.3 million documents that span from 1994 to the end of 2008.

We have participated in both subtask with the aim of establishing a foundation for future work in the task. Our focus has been on establishing a set of tools based on open-source software that we can improve in folowing participations with our own developments. We have submitted one run for the Entity Linking task and three in the Slot Filling task using different configurations. The two subsystems have not been integrated yet so we did not attempt to link filled slots to the KB. The following sections describe each of the systems and the results obtained in the task. Last section presents some general conclusions and future work we plan to address.

2 Entity Linking

Given a set formed by a list of entities organized in a knowledge base, a document corpus, and a list of [name-string, document] queries, the Entity Linking task will have to determine, for each of the queries, which of the entities in the knowledge base, if any, is being referred to by the name-string in the given document.

Each entity in the knowledge base contains title, name, type, id, some plain text, and several facts of different types in the form of a [name, value] pair, being all this data extracted from a Wikipedia snapshot. There are three types of entities: persons, organizations, and geo-political entities.

An additional [knowledge base, document corpus, query list] set is supplied for testing purposes. This set is substantially reduced compared to the full set, but it contains the solution for each query, so it could be used not only for testing but also for manual tuning or automatic learning purposes

2.1 Approach

Being our first incursion into the Entity Linking task, and due to the lack of time, the approach had to be as simple and flexible as possible, as most probably some of the ideas could not be implemented in time.

Our initial idea involved using a text indexing and searching tool to disambiguate by matching the information in each proposed document with the information in the knowledge base entities. Some heuristics could be used to create fields not directly present in the knowledge base entities, to unify field names, or to create new field values. Using a named-entity extraction tool could help to extract some information from the plain text. And finally, in case we had time and information enough, a learning algorithm, probably based on another external tool, could be implemented to help with the tuning tasks.

Once indices have been created, each query is solved using a two-phase algorithm. In a first stage, a list of candidate entities is obtained. For an entity to be a candidate, it needs to have some similarity in any relevant field with the name-string in the query. This similarity could also be with any synonym, alias or acronym extracted from the name-string itself or even from the document text.

The second stage chooses the best entity from the candidate list, or returns NIL if none of the candidates is good enough. Once it is well tuned, this will be the most important part of the process. The simplest algorithm for this second stage is just a pass- through algorithm that returns the candidate entity that gets a higher score in the first stage. A threshold is be defined so that NIL is returned when the Lucene score is below that value.

Finally, in case the given queries contain information about solutions, statistics are collected and analyzed. This feedback helps to improve the process in either phase, as indices, algorithms, formulae and coefficients are adjusted to achieve better results.

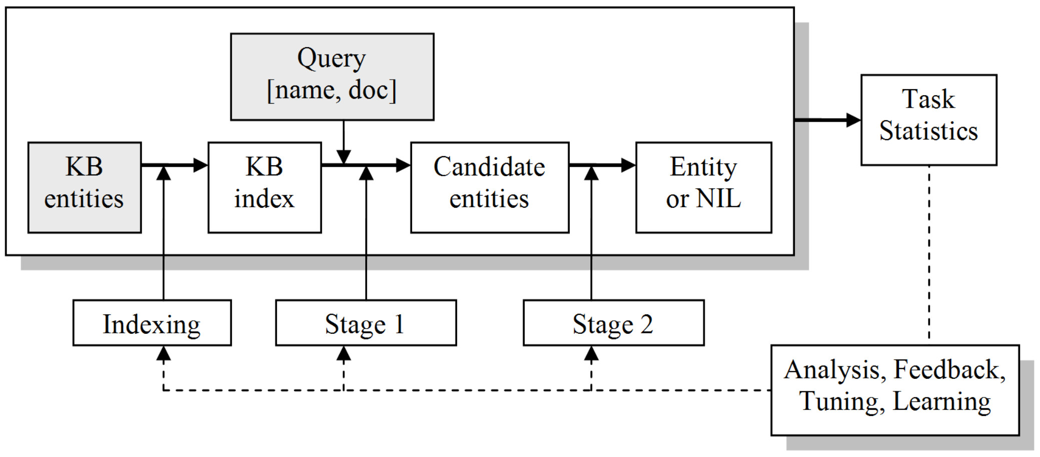

Figure 1: Entity Linking system design

2.2 System description

Our final system relies on two external tools: a text indexing and searching tool and a named-entity recognition tool.

The chosen text indexing and searching tool is the well-known Apache Lucene information retrieval open-source library [1]. Lucene can create indices of documents containing different fields, and perform text searching on them. For each query, it can return a scored list of matching documents, so it fits our needs quite nicely.

To find entity names in the text associated with both entities and articles, the Stanford NERC [2] [3] library is used. It returns all entity names found in a given plain text, classified as persons, organizations and geo-political entities, this is, the same types of entities the task will deal with.

The system is also configurable to deal with different [knowledge base, document corpus, query list] sets.

2.2.1 Indexing

Before any query can be run, the knowledge base entities and the article corpus need to be indexed using Lucene. A Lucene document is a collection of [field name, value] pairs which can be multi-valued. Searches are executed only against the knowledge base index; the article corpus index is created solely for storage purposes, as locating an article in the file system could be quite tedious given the amount of directories.

For each entity in the knowledge base, a document is created containing the following fields:

- Fields `wikiTitle`, `name`, `type`, `id` and `wikiText` are created from the analogous fields in the XML element that holds the entity.
- The complete list of [fact name, fact value] pairs are translated to fields.
- Field `search` contains all the possible name strings the entity could be referred to by. This includes:
  - The entity name

[1]http://lucene.apache.org
[2]http://nlp.stanford.edu/ner/index.shtml

run	non-NIL queries (1675)	NIL queries (2229)	total (3902)
UC3M1	38.41%	0.99 %	17.09%

Table 1: Micro-averaging Accuracy results for the EL-UC3M1 run

– Some relevant optional facts (`alias`, `birth_name`, `nickname`, `company_name`, etc.)
– Acronyms derived from the entity name.

2.2.2 Stage 1 - Obtaining a candidate list

The candidate list is obtained by simply running a Lucene search on the `search` field of the knowledge base. This search returns a scored list of documents, and all entities corresponding to documents whose score is above a given threshold are included in the candidate list. Most of the correctness of this stage relies on the creation of the `search` field during indexing. Final analysis should check that a very high percentage of solutions are included in the candidate list; if this fails, it means more name strings should be appended to the `search` field.

2.2.3 Stage 2 - Getting the best candidate, if any

The first algorithm that returns a possible best candidate in the list is also the simplest one: it just returns the entity that achieves a highest score in the previous stage. If this score is not above a given threshold NIL will be returned. This is of course not the best algorithm, but it is the one we used in Run #1. Being quite simple, this algorithm admits some improvement by applying a formula on the score based on the way the `search` field is created and the way Lucene calculates document scores, but we have not investigated this part. An easy way to tune this algorithm is adjusting the threshold according to the final statistics.

A second algorithm was implemented but not used in any submitted run. For each query, it calculates the overlap between the target entity facts and the document. Additional tuning of weight parameters need to be performed and was not complete to submit any run.

2.3 Results

Only one run based on the first algorithm described in stage 2 was submitted, as we were not in time for submitting a run based on the second algorithm. This run uses algorithm #1 for stage 2. The micro-averaged results obtained are:

Obtained results are not good, but numbers in the non-NIL chapter are by far better than expected given the simplicity of the algorithm used in stage 2. Using the second algorithm, after some improvements, should reward more acceptable results.

With regard to the external tools used, Lucene has shown to be a good tool for stage 1, and the tests done show it can integrate quite well in the algorithm for stage 2. Stanford NER has been able to extract some relevant information from plain text, but cannot be used as thoroughly as we intended when time is a concern.

Once we get larger training set and analyze the complete statistics, we can start working on improving the algorithms for stage 1 (by sketching out more heuristics to get better candidate lists) and for stage 2 (by learning weights for the second algorithm).

3 Slot Filling

The objective of the Slot Filling task is to harvest new values from the document collection for the predefined attributes of the entities in the KB. In this task each query is formed by a

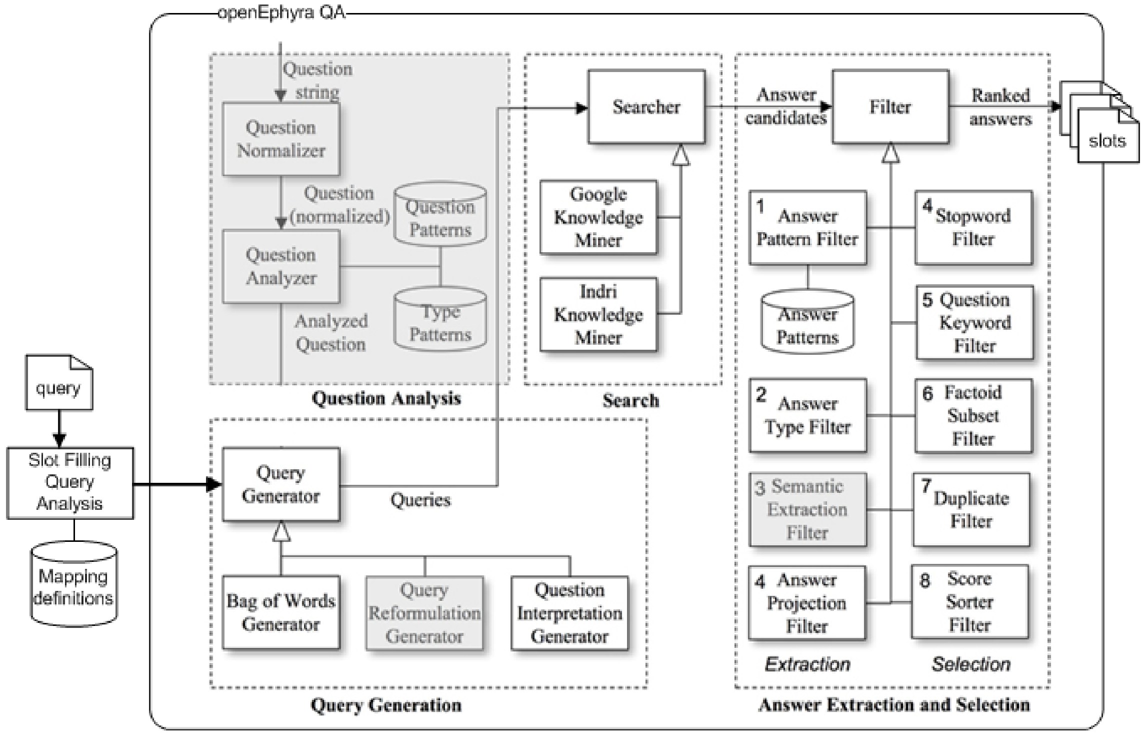

Figure 2: Slot Filling architecture (Shaded modules are not used)

target entity and additional information like their type, their unique ID in the KB (optional) and a document ID that provide additional context. Values for the slots defined for that type (see Appendix) should be filled at least something different is provided in the ignore field. The expected answer is a list of slot names and attributes values. Optionally some of this values could be linked to the Knowledge Base as they refer to other persons, organizations and geopolitical locations.

We approached the task from a Question Answering (QA) perspective and reused the open-source OpenEphyra [3] system to fill slot values. OpenEphyra is a modular QA systems which implements a pipeline architecture as depicted in Figure 2. It includes several modules and strategies for Question Analysis, Query Generation, Search and Answer Extraction and Selection. For example, a document retrieval module uses Indri[4] [6] for retrieving documents from a local collection while a compatible document retrieval module could query Google or Yahoo instead. There are three different Answer Extraction strategies implemented as filters and based on the use of Answer Types, Pattern Types and semantic extraction based on a Semantic Role Labeller. We decided to experiment with different system configurations and the existing extraction strategies.

A QA system is designed to answer questions in natural language like *What is the population of Zahle?* or *Who is the major of Zahle* but also open questions *Name beverages that are produced in Zhale*. In order to use a QA system for filling slots we found two different alternatives. The first option, using the QA as a black-box should consider to manually write question templates for each of the slots to fill. The second option, using the QA as a glass box reuses the internal

[3]http://www.ephyra.info
[4]http://lemurproject.org/indri

Question	–	
Keywords	John Doe	
Keyterms	–	
NamedEntities	[John_Doe/NEperson]	
Focus	John Doe	
AnswerTypes	NEdate	
QuestionInterpretation	TARGET	John Doe
	CONTEXT	–
	PROPERTY	DATEOFBIRTH
Predicates	–	
IsFactoid	true	

Table 2: Mapping a query for slot `per:date_of_birth` to question analysis

question representation to represent the slot filling requests. Both approaches have pros and cons, but we focused on the second as our initial interest was to introduce additional answer strategies. Even in a modular system like OpenEphyra the main disadvantage is that it requires detailed knowledge of the components. On the other hand, the disadvantages of the black box approach are also clear from an initial inspection. Some of the slots could require to generate several questions (e.g `gpe:parents`, `per:parents` or `org:alternate_names`) and then combine them. Moreover, we would have less control over the correct interpretation of the questions which could mean that it triggers an erroneous extraction process.

Once we opted for the glass-box approach, we defined a new QA pipeline that bypass the Question Analysis step. In their place a Slot Filling Query Analysis module takes a query and produce several analyzed questions, one for each slot to be filled. The OpenEphyra question analysis representation is reused and each slot is mapped to an Answer Type and a Pattern Type. Slot names are mapped to the Answer Extraction resources that are available in OpenEphyra and detailed in the Appendix. The Answer Type Filter applies NE type restrictions to select answers or slot values. It uses a mixture of machine learning NE recognizers (Stanford NERC tagger), regular expressions and name lists. The Answer Pattern Filter applies generalized relation patterns that have been acquired from the Web as described in [5]. No additional training was performed to acquire patterns beyond those used in the open-source distribution. The coverage of slots is relatively high (see Table 3) as the types has been derived and refined throughout TREC QA competitions. Finally, the Semantic Extraction Filter is described in [4] but was not used in any of the configurations. An example of how a slot request is transformed into a internal question representation is shown in Table 2. We have opted for the simplest representation which uses just one Answer Type, one Question Interpretation (Pattern type) and no additional keywords beyond the entity name. Unfortunately, we cannot guarantee that this is the most effective way to use the QA system or that we used and tuned the system correctly.

Other slot predicate characteristics that have been provided in the guidelines like values types (Names, Values or Strings) and cardinality (Single or List) are also used. For example, OpenEphyra generates the same rank of answers for single and list questions. The first top ranked answer is used to fill Single valued slots while a threshold is used to select the cut for a list of answers. We also included information on whether the slot value is linkable, though we did not integrate the Entity Linking module for the submitted runs. The document collection was also adapted to be indexed with Indri at the sentence level, which is the representation used by OpenEphyra to implement the Indri Knowledge Miner.

Target Type	Slots	Answer Type	Pattern Type
PER	20	19 (95%)	15 (75%)
ORG	14	13 (93%)	8 (57%)
GPE	8	7 (88%)	5 (63%)
TOTAL	42	39 (93%)	28 (67%)

Table 3: Slot predicates coverage statistics

Run name	Doc collection	Answer Strategy
UC3M1	local collection	Answer Pattern Filter, Answer Type Filter
UC3M2	local collection	Answer Pattern Filter
UC3M3	web and projection on local collection	Answer Pattern Filter, Answer Type Filter

3.1 Submitted Runs and Result Analysis

We submitted three different runs that have explored the use of different answer strategies and also constrast with the use of the web. Our first run uses the local news collection as information source and combines the Answer Pattern Filter (1) and the Answer Type Filter (2). The second run has a similar configuration but uses only the Answer Pattern Filter (1). The third run uses the Web as the source for slot values and later on, it projects the values to the local collection. We have used the Yahoo Knowledge Miner to retrieve snippets and answers. For the three runs, when no answer is found after running the QA system the slot value is filled with a NIL value. This situation could be that no answer strategy is available, that no documents were retrieved or that no answers were extracted at all.

Results for the three submited runs are presented in Table 4 together with a NIL-baseline. The NIL baseline assigns a NIL value to every slot value which corresponds to a system that do not perform any automatic update to the Knowledge Base. It has been turned out to be a very strict baseline because there are only a small percentage of the slots that had a new value in the collection. If a QA system is employed for the task of filling slots it is clear that a more elaborated NIL strategy is required. Besides, the estimation of the confidence of answers need to be improved too. On the other hand, for slots with new values our runs were able to obtain a reasonable level of recall. Nevertheless, a large margin of improvement is possible as only about a third of all the values are found in the single best performing run.

	single			list			combined
	all	non-NIL	NIL	all	non-NIL	NIL	
# slot queries	255	39	216	499	129	370	
run	*Acc*	*Recall*	*Recall*	$F-score$	*Recall*	*Recall*	*Acc*
UC3M1	0.220	0.179	0.227	0.428	0.181	0.514	0.324
UC3M2	0.243	0.077	0.273	0.493	0.131	0.619	0.368
UC3M3	0.149	0.308	0.120	0.168	0.292	0.124	0.159
NIL-baseline	0.847	0.000	1.000	0.741	0.000	1.000	0.794

Table 4: Results for the three submitted runs for SF task

By comparing runs UC3M1 and UC3M2 that use different answering strategies, we can conclude that the Answer Type strategy is a good back-off to retrieve additional slot values, specially for single slot values. Nevertheless, the uc3m2 obtains a better overall accuracy as it is more prone to produce NILs. It is not clear whether this is due to certains slots not having an appropiate matching. We expect that a more detailed analysis by slot provide addiotional information.

The comparison of runs UC3M1 and UC3M3, which use similar extraction strategies but different source collections is also interesting. Run UC3M3 extracts slot values by querying the web and those are then projected to the local collection. The improvement on the recall of non-NIL slots is very significant and greatly outperforms the other two runs. On the other hand as it proposes a much larger number of values it also generated more spurious values. It seems clear that this strategy should be combined with a more elaborated confidence estimation procedure in order to be effective for automatic filling of slots.

Finally, it is worth commenting the paradox that the runs that obtain better recall on the non-NIL attributes do also obtain worst results in the overall task though they provide much useful information. In our opinion, alternative evaluation measures should be investigated to balance the weight of NIL slots and reflect the advance on populating a knowledge base.

4 Conclusions and Future Work

We have presented our first attempts to develop a system for automatic Knowledge Base Population. Our system for Entity Linking indexes a KB representation using IR tools and provide a ranked list of candidates. This list can later on be reranked or filtered using contextual information and additional knowledge. We believe that such approach is appropiate for large KBs as the more expensive second stage is only applied to a small subset of the candidates. Nevertheless, we expect to apply machine learning to improve the reranking of candidates as well as the detection of entities that should not be linked.

The Slot Filling system is able to obtain a reasonable good recall for the discovery of new slots values, specially when the web is used as an additional source that helps to locate values. However, our QA system provides also a large number of spurious answers that would degrade quickly the quality of the Knowledge Base and therefore more effort should be devoted to estimate the confidence of extracted values. So far, we have used independent processes to extract values, we believe that the collective filling of relared slots could be use to reduce some of this errors.

In the reported runs we have used a simple configuration to reuse OpenEphyra components. There are plenty of features to explore including the use of additional keywords and name variations from the KB and the integration with the Entity Linking system that could improve results. And interesting comparison betweeen the actual approach and the black-box approach to reuse QA is also worth explored. Finally, we also expect to be able to extend our own pattern acquisition approach based on bootstrapping [1, 2] and compare with other extraction alternatives.

5 Acknowledgements

This work has been partially supported by the Regional Government of Madrid by means of the Research Network MAVIR (S-0505/TIC/000267) and by the Spanish Ministry of Education by means of the project BRAVO (TIN2007-67407-C3-01)

References

[1] César De Pablo-Sanchez and Paloma Martinez. Building a Graph of Names and Contextual Patterns for Named Entity Classification. In *European Conference in Information Retrieval 2009*, 2009.

[2] César De Pablo-Sanchez and Paloma Martinez. UC3M at WePS2-AE: Acquiring Patterns for People Attribute Extraction from Webpages. In *Second Workshop on Web People Search at WWW 2009*, 2009.

[3] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL '05*, pages 363–370, 2005.

[4] N Schlaefer, J Ko, J Betteridge, G Sautter, Pathak Manas, and Eric Nyberg. Semantic extensions of the Ephyra QA system for TREC 2007. *of the Sixteenth Text REtrieval Conference (TREC)*, 2007.

[5] Nico Schlaefer, Petra Gieselmann, Thomas Schaaf, and Alex Waibel. A Pattern Learning Approach to Question Answering within the Ephyra Framework. In *TSD '06*, pages 687–694, 2006.

[6] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. A language-model based search engine for xomplex queries (extended version), 2005.

A Mapping between slots and OpenEphyra extraction resources

PERSON					
KBP definition				Open Ephyra	
Slot name	Type	Card	Link	NE type	pattern type
per:alternate_names	N	L	0	NEperson	–
per:date_of_birth	V	S	0	NEdate	DATEOFBIRTH
per:age	V	S	0	NEduration	AGE
per:place_of_birth	N	S	1	NElocation	PLACEOFBIRTH
per:origin	N	S	1	NElocation	ORIGIN
per:date_of_death	V	S	0	NEdate	DATEOFDEATH
per:place_of_death	N	S	1	NElocation	PLACEOFDEATH
per:cause_of_death	S	S	1	–	CAUSEOFDEATH
per:residences	N	L	1	NElocation	–
per:schools_attended	N	L	1	NEeducationalInstitution	SCHOOL
per:title	S	L	0	NEprofession	PROFESSION
per:member_of	N	L	1	NEorganization	MEMBERSHIP
per:employee_of	N	L	1	NEorganization	MEMBER
per:religion	S	S	0	NEreligion	–
per:spouse	N	L	1	NEperson	SPOUSE
per:children	N	L	1	NEperson	ANCESTOR
per:parents	N	L	1	NEPerson	FATHER
per:siblings	N	L	1	NEperson	–
per:other_family	N	L	1	NEperson	–
per:charges	S	L	1	NEcrime	KILLED

ORGANIZATION					
KBP definition				Open Ephyra	
Slot name	Type	Card	Link	NE type	pattern type
org:alternate_names	N	L	0	–	ABBREVIATION
org:political/religious_affiliation	N	L	0	NEreligion	–
org:top_members/employees	N	L	1	NEperson	FOUNDER
org:number_of_employees/members	V	S	0	NEcount	NUMBEROFMEMBER
org:members	N	L	1	NEorganization	–
org:member_of	N	L	1	NEorganization	–
org:subsidiaries	N	L	1	NEorganization	FUNDING
org:parents	N	L	1	NEorganization	FUNDER
org:founded_by	N	L	1	NEperson	FOUNDER
org:founded	V	S	0	NEdate	DATEOFFOUNDATION
org:dissolved	V	S	0	NEdate	DATEOFEND
org:headquarters	N	S	1	NElocation	–
org:shareholders	N	L	1	NEorganization	–
org:website	S	S	0	NEurl	–

GEO_POLITICAL_ENTITIES					
KBP definition				Open Ephyra	
Slot name	Type	Card	Link	NE type	pattern type
gpe:alternate_names	N	L	0	–	–
gpe:capital	N	S	1	NElocation	CAPITAL
gpe:subsidiary_orgs	N	L	1	NEorganization	–
gpe:top_employees	N	L	1	NEperson	LEADER
gpe:political_parties	N	L	1	NEorganization	–
gpe:established	V	S	0	NEdate	DATEOFFOUNDATION
gpe:population	V	S	0	NEcount	POPULATION
gpe:currency	S	S	0	NEmoney	MONEY