A distant supervised learning system for the TAC-KBP Slot Filling and Temporal Slot Filling Tasks

Guillermo Garrido, Bernardo Cabaleiro,
Anselmo Peñas, **Álvaro Rodrigo** and **Damiano Spina**
NLP & IR Group at UNED
Madrid, Spain
{ggarrido,bcabaleiro,anselmo,alvarory,damiano}@lsi.uned.es

Abstract

This paper describes the system implemented by the NLP GROUP AT UNED for our first participation in the Knowledge Base Population at the Text Analysis Conference (TAC-KBP). For this Slot Filling Task, our approach was to design a distant supervised learning system, which was then specialized for the Regular Slot Filling and Full Temporal Slot Filling subtasks.

From the initial Knowledge Base and source document collection distributed by the organizers, we automatically gathered training data for supervised slot classifiers. We used a rich document representation, augmenting syntactic dependency trees with named entity recognition, coreference resolution, temporal events annotation and semantic graph transformations.

1 Introduction

For the first UNED participation in the KBP Slot Filling Task, we developed a simple distant supervision approach, following the paradigm described by Mintz et al. (Mintz et al., 2009), which has also inspired other participants in earlier editions of the task (Agirre et al., 2009; Surdeanu et al., 2010).

In this paper, we describe the general architecture, design and implementation of the system; we give a preliminary analysis of the results, describe the problems and difficulties that we have encountered and propose lines of inquiry we plan to follow in future work. We participated in two of the subtasks: the Regular Slot Filling Subtask and the Full Temporal Slot Filling Subtask.

In Figure 1, we depict the data flow of our system. As pointed out in (Ji and Grishman, 2011), the slot filling task combines Question Answering (QA), Information Retrieval (IR) and traditional Information Extraction (IE). The QA/IR aspect of the task is having to answer to a query, obtaining the response from a document collection, while as in general IE, the queries involve a fixed set of relations of entities. Thus, the architecture of many of the systems that have taken part in the KBP slot filling task, and of our own, includes three phases: (1) QA/IR passage/document retrieval; (2) IE answer extraction; and (3) answer aggregation.

For the retrieval of candidate documents, that are used both in training and in the application of the system, we used a straightforward IR approach, that is described in Section 3. The IE component is the core of our system; it exploited a rich document representation, described in Section 4; the general distant supervised learning approach is detailed in Section 2; this section also details how answers were aggregated to produce a set of runs for evaluation.

The focus of our development was to tackle the Temporal Slot Filling subtask; this conditioned many of our engineering decisions. With a few adaptations, we also addressed the Regular Slot Filling subtask. The specifics of the the Regular Slot Filling subtask are described in Section 7, while the participation in the Full Temporal Slot Filling subtask is detailed in Section 6.

Finally, preliminary results are reported in section 8, and our conclusions in section 9.

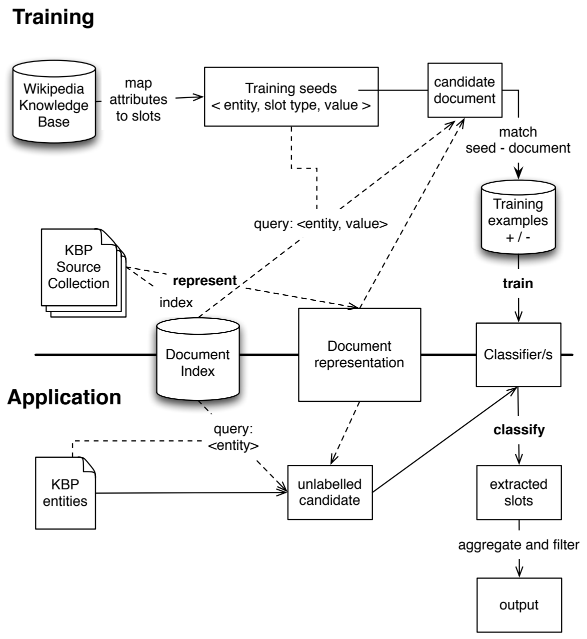

Figure 1: Scheme of the system data flow.

2 Slot Classification

In this section, we describe the general architecture of our system.

2.1 Automatic acquisition of training data

From the Wikipedia-derived Knowledge Base (KB, from now on) made available by the task organizers, we extract a set of relation triples or seeds; each seed is a triple: $< entity, slot\_type, value >$ where the entity corresponds to an article's title, and the slot_type is one of the target slots of the task. We automatically obtain training examples by matching the seeds to the documents in the KBP source collection, and use those examples to train a multi-label classifier.

The attributes of the KB do not exactly match the target slot types of the task. Using the mapping provided by the task organizers, we translated from (infobox class, attribute) to KBP slot type[1]. Unfortunately, this mapping can be ambiguous, as a particular Wikipedia infobox attribute can contain, for different instances, a value that maps to either none, one, or many KBP slots.

[1]This KB has been obtained from Wikipedia infoboxes; there are several Wikipedia *infobox classes*, and each summarizes the key facts about an entity, as an (attribute, value) pair.

For instance, the infobox in the article "Liverpool F.C", belongs to the infobox class "Infobox Football club", and has the attribute "founded", which value is "1892 (by John Houlding)". This value should be splitted and mapped to two KBP slots: `org:founded` (1892) and `org:founded_by` (John Houlding). Solving this kind of ambiguities could be done by implementing heuristics customized for each (infobox class, attribute) (Surdeanu et al., 2010), and we have not yet attempted it.

Another source of ambiguity is that the value of an Infobox attribute might contain a list of valid values, rather than a single one. For instance, the value of (Julia Roberts, spouse) is:

"Lyle Lovett (1993-1995)
Daniel Moder (2002-present)"

In such cases, we have attempted to split the value and produce two seeds for the `per:spouse` slot, one with value "Lyle Lovett" and another with value "Daniel Moder".

The values are sometimes noisy and our procedures are not sufficiently fine-grained to always obtain clean values. We only performed very simple cleaning, such as separating values by end-of-lines. We leave for future work the task of obtaining cleaner seeds from the Knowledge Base.

2.2 Gathering of distant training examples

From a seed triple: $< entity, slot\_type, value >$, we retrieve candidate documents that contain both the entity and value, within a span of 20 tokens, using the IR engine. A certain amount of extra noise is allowed as the tokens in entity and value can appear in a different order.

These documents are represented as augmented dependency parsing graphs (see section 4).

The entity and value are matched to the document representation. If a node matches the entity and another node matches the value, the seed is used for training. Otherwise, it is discarded. As mentioned above, the seeds we are working with are noisy. The aim of this matching step is to reduce the effect of this issue; we also have to take into account that there is a trade-off between the tightness of this seed filtering and the size of our training set (and less data would damage our performance).

The matching procedure uses the descriptor of the node to compare against the string text of the entity and value. For the entity, a string comparison is performed, allowing for small differences by using the Levenshtein distance with a positive matching threshold for the ratio of similarity of 0.8.

The comparison algorithm for the values is slightly more complex. We test each of the nodes connected to the identified entity node in a BFS traversal of the graph representation, limited to paths of at most length 10. We have manually defined an expected Named Entity (NE) type for each KBP slot type[2]. If the node value is not identified with the same type, the node is not considered for the matching. This NE type filtering procedure has also been applied in the past (for instance in (Surdeanu et al., 2010)). The NE type filtering we applied in the Regular Slot Filling and Temporal Slot Filling subtasks differed slightly. In sections 6 and 7, we report on them.

2.3 Training

Each example was represented by binary features; some of them were inspired by previous work (Surdeanu and Ciaramita, 2007; Mintz et al., 2009; Riedel et al., 2010; Surdeanu et al., 2010), and others were unique of our graph representation.

We established a threshold on the features sparseness: features appearing in less than 5 training examples were discarded.

Table 1 summarizes our choice of features, some of which were only applied to the Temporal Slot Filling Subtask.

2.4 Classification process

To apply the classifier to the task of detecting the values for the slots, we followed a simple approach:

- Using our IR engine, we retrieved the subcollection of documents containing the query entities (limiting to 100 documents per query).
- We replicated the analysis of the document to obtain the same graph representation.
- We searched for the node or nodes matching the query entity, and from each of them: We started a BFS search, limited to paths of length at most 10, were each of the traversed nodes was proposed as a candidate value. We use the same feature generation to represent each of the $< entity\_node, value\_node, document >$ triples as an unlabelled example. We run the classifier or classsifiers on the set of examples, to label them as either positive for any of the slots_types (classes), or with the negative label (unclassified).

In the case of Temporal Slot Filling (Section 7, we used a battery of binary classifiers (extractors). For the Regular Slot Filling, we experimented with a single multi-class classifier (Section 6).

2.5 Answer aggregation

The classification process yields a predicted class label, plus a real number indicating the margin (linear distance from the example to the SVM hyperplane boundaries). We performed an aggregation phase to sum the margins over distinct occurrences of the same value. The rationale for this is that when the same value is labelled as positive in more than one example, we should accumulate that evidence.

At this point, we performed very basic filtering, discarding candidates that did not have the expected NE type, as defined for us for the slot, and used a closed list of places to distinguish between cities, countries and regions.

For those slot types that accept a single response, we chose the best (highest margin) extracted by the system. For those that accept a list as response, we chose the three highest extracted.

The system did not involve manual annotation, other than the NE expected types list and this geographic locations list.

3 Retrieval of candidate documents

The candidate documents retrieval phase consists of a standard IR approach. We decided to use a simple IR approach, without taking into account the ambiguity problem, in order to have a baseline for this phase. Ideally, an entity linking component should be put in place.

We use the Lucene function ranking with the default parameters and we use the entity name as query. The first 100 documents are considered for the searching of slot values phase. This threshold

[2]The NE annotation of the texts was obtained from the Stanford NE Recognizer (Finkel et al., 2005).

Feature family	Feature name	Description
Syntactic dependency	path	dependency path between ENTITY and VALUE in the sentence [represented with the unigrams and bigrams of dependency labels, POS tags, NE tags]
Placeholders	X-annotation	NE annotations for the sentence fragment X
	X-pos	Part-of-speech annotations for the sentence fragment X
Lexical context	X-gov	Governor of X in the dependency path
	X-mod	Modifiers of X in the dependency path
Properties	X-has_age	X is a NE, and we have identified it has an age attribute.
	X-has_class-C	X is a NE, and we have identified it has a class C.
	X-has_property-P	X is a NE, and it has a property P
	X-has-Y	X is a NE, and it is in a relationship `to-have` with another NE, Y
	X-is-Y	X is a NE, and it is in a relationship `to-be` with another NE, Y
	X-gender-G	X is a NE, and it has gender G
Properties of the verb V (features for Temporal Slot Filling only)	V-tense	Tense of the verb V in the path.
	V-aspect	Aspect of the verb V in the path.
	V-polarity	Polarity (positive or negative) of the verb V in the path.

Table 1: Features included in the model. X stands for both the ENTITY and the VALUE sentence fragments (we generate a set of features for each). The verb features are generated from the verbs identified in the path between between ENTITY and VALUE, and appear only in our models for the Temporal Slot Filling subtask.

	PER	ORG	ALL
recall@10	0.44	0.38	0.41
recall@50	0.72	0.63	0.67
recall@100	0.79	0.71	0.75

Table 2: For the document retrieval, the threshold of 100 documents per entity was empirically obtained evaluating the average recall of the support documents of previous editions of the task, over Person and Organization entities and both together.

was empirically defined after a partial evaluation of the recall of the relations included on the supervised collection. For each entity, we have calculated the mean recall of the support documents in the search results. On average, a total of 75% support documents are covered in the first 100 results.

A more complete evaluation of the overall system is needed in order to understand the impact of this baseline approach, and it is left for future work.

4 Document Representation

We have experimented with the effect of using a rich document representation, that uses a graph structure, obtained by augmenting the syntactic dependency tree analysis of the document with semantic information.

A document D is represented as a *document graph* G_D; each of the nodes in the graph represents a *chunk* of text, which is sequence of words. Each node is labeled with a dictionary of attributes, some of which are common for every node: the words it contains, their part-of-speech annotations (POS), lemmas, and their positions in the phrase and in the sentence. Also, a representative *descriptor*, which is a normalized string value, is generated from the chunks in the node.

Certain nodes are also annotated with one or more *types*. There are three families of types:

- Events: Verbs that describe an action. They contain information about tense, polarity and aspect.
- Time expressions: Normalized representation of a time and date.
- Named entities: With additional information, such as gender or age.

Edges in the document graph represent relations between the nodes. There are four kinds of relations:

- Syntactic: Indicating a dependency relation between two chunks.

- Coreference: Indicating that two chunks refer to the same discourse referent.
- Semantic: Indicating semantic relation between two chunks, such as `hasClass`, `hasProperty` and `hasAge`
- Temporal: Indicating a temporal relation between events and time expressions.

The graph representation is produced by a succession of four steps, each of them adding a new layer of information. Schematically, the process is the following:

1. Process the document collection with Stanford CoreNLP.
2. Add Tarsqi Toolkit information to the data already extracted.
3. Collapse related named entities by coreference into a single *discourse referent* node, and normalize the graph.
4. Add semantic information.

This information generated from the documents is the source of the features we use for classification, that were summarized in Table 1.

In the following sections, we detail each of these steps.

4.1 Dependency Parsing, Named Entity Recognition and Coreference Resolution

Dependency parsing is included in the Stanford CoreNLP software (Klein and Manning, 2003), which provides syntactic dependency analysis and part of speech (POS) tagger. Stanford CoreNLP also has a named entity recogniser (Finkel et al., 2005). This tool classifies chunks in different kinds of entities, mainly persons, organizations and locations, but also numbers, ideologies, causes of death, criminal charges, titles and others. Note that Stanford CoreNLP classifies time NE too, but we use the Tarsqi Toolkit to resolve this issue. We also take advantage of Stanford CoreNLP to resolve coreference (Lee et al., 2011; Raghunathan et al., 2010).

4.2 Events and Temporal Information

Temporal information and events are extracted through the Tarsqi Toolkit (Verhagen et al., 2005). Time expressions are marked and normalized data is generated following TimeML TIMEX3 standard (Pustejovsky et al., 2007). Also some verbs are marked as events, and information about tense, polarity and aspect is provided. Moreover, Tarsqi Toolkit automatically generates temporal relations between expressions and events.

4.3 Collapsing referents of discourse and graph normalization

The aim of graph normalization is simplify data which does not provide any semantic difference, such as active and passive. In this way, we cluster referents of discourse into the same node, and then eliminate the redundant edges that may result of the operation. In clusters with a entity with a known name, all the pronouns are omitted. Also there is a process which relates time expressions to the corresponding event instead of its original syntactic relation. In this step, the document graph G_D is transformed into a *collapsed document graph*, G_C. Each node of G_C clusters together coreferent nodes, and represents a *discourse referent*. Thus, a node u in G_C is a cluster of nodes $u_1, \dots, u_k$ of G_D. There is an edge (u, v) in G_C if there was an edge between any of the nodes clustered into them: $u_1, \dots, u_k$ and any of the nodes $v_1, \dots, v_{k'}$.

4.4 Addition of semantic information

- Normalize meaning of copulative verbs.
- Normalize genitives.
- Use appositions and genitives to infer semantic class indicators, included age and property.
- Use information from pronouns to assign gender to a NE.

Graphic example Let us picture the graph representation process for a simple example. Let us say we had the sentence: "John was so tired that he fell". We would start from the well known syntactic dependency tree of the sentence, and after applying the steps 1 and 2 described above, we would have the Figure 2.

After applying the subsequent steps 3 and 4, we would have the Figure 3

5 Slot Temporal Restrictions

Task organizers define a 4 tuple of constraints to fill in the new temporal slot filling task. This constraints are T1, T2, T3 and T4, which represents that the slot

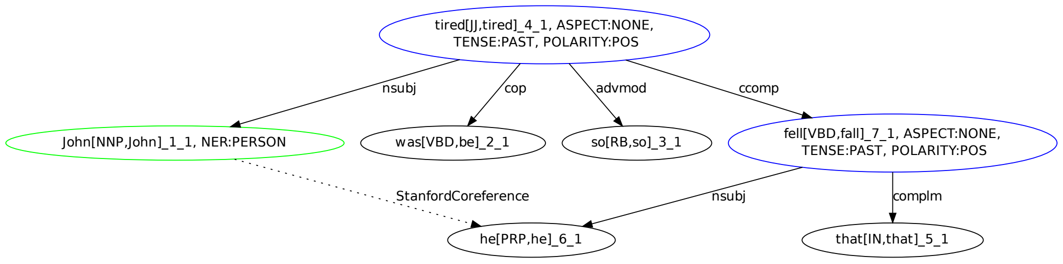
tired[JJ,tired]_4_1, ASPECT:NONE, TENSE:PAST, POLARITY:POS
nsubj
cop
advmod
ccomp
John[NNP,John]_1_1, NER:PERSON
was[VBD,be]_2_1
so[RB,so]_3_1
fell[VBD,fall]_7_1, ASPECT:NONE, TENSE:PAST, POLARITY:POS
StanfordCoreference
nsubj
complm
he[PRP,he]_6_1
that[IN,that]_5_1

Figure 2: Graph obtained, for the sentence "John was so tired that he fell", after applying step 1 (Dependency Parsing, Named Entity Recognition and Coreference Resolution) and step 2 (Events and Temporal Information).

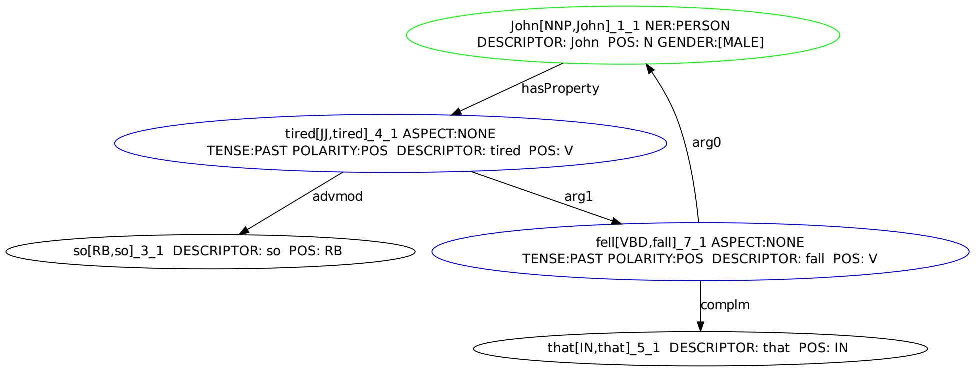
John[NNP,John]_1_1 NER:PERSON DESCRIPTOR: John POS: N GENDER:[MALE]
hasProperty
arg0
tired[JJ,tired]_4_1 ASPECT:NONE TENSE:PAST POLARITY:POS DESCRIPTOR: tired POS: V
advmod
arg1
so[RB,so]_3_1 DESCRIPTOR: so POS: RB
fell[VBD,fall]_7_1 ASPECT:NONE TENSE:PAST POLARITY:POS DESCRIPTOR: fall POS: V
complm
that[IN,that]_5_1 DESCRIPTOR: that POS: IN

Figure 3: Graph obtained after applying also steps 3 (Graph Normalization) and 4 (addition of semantic information).

value holds true in a period which starts between T1 and T2 and ends between T3 and T4. Besides, it is sometimes not possible to fill all the constraints. In this section, we describe how we gather the data to fill those constraints.

5.1 Temporal relations acquisition

The first step of the acquisition of temporal relations is to identify a syntactic pattern "Event - preposition - Time Expression" within the lexical context of the candidate entity and value in the document. To do so, we search the time expression that is closest to the shortest path between the entity node and the value node. This is done via a breadth-first-search in the graph, starting from the nodes in the path, that are visited first, in order from value to entity. The search is limited to a depth of 2.

If an "Event - preposition - Time Expression" pattern has been found, it is then transformed into one relation from the set *within*, *throughout*, *beginning*, *ending*, *after* and *before*.

Next, we use the Tarsqi Toolkit to get temporal relations between events and temporal expressions as we explained in Section 4.2. This temporal relations may be *included*, *simultaneous*, *after*, *before*, *begun_by* or *ended*. We normalize the data to match this relations to the first ones.

5.2 Semantic Considerations

We need to consider the semantic of the event in order to decide the time period associated to a slot value. This step is important because the event could refer just to the beginnig, the ending, or both times of the slot value. This information helps us to decide what constraints must be managed. For instance, it is obvious that it is not the same to have the event *marry* than to have the event *divorce* at the time of deciding the temporal constraints associated to the `per:spouse` slot. We classify the way of giving time constraints in three temporal groups depending on slot type and event:

- *Start*: it refers to have information about just the starting period of the slot value (i.e. T1 and T2 constraints). Some examples of this group are the event *marry* in `per:spouse` slot and *born* in `per:*_of_residence`.
- *Finish*: this group is used when only information about the ending period of the slot value is considered (i.e. T3 and T4 constraints). Some examples of this group are the event *divorce* in `per:spouse` slot and *die* in `per:*_of_residence`.
- *Period*: this group is associated to situations where any part of the period can be set (i.e. T1, T2, T3 or T4 constraints). An example of this group is the event *live* in `per:*_of_residence`. This group is used as default when no event is detected.

5.3 Constraints Production

The final decision about the value given to each constraint depends on the temporal group and the temporal relation. The processing applied to the beginnig and ending groups are similar, but considering thay they refers to diferent constraints (T1-T2 for *start* and T3-T4 for *finish*): time relations "after" and "beginning" affects just to the starting point of a period (constraints T1 and T3), while "before" and "ending" only modifies the end of the period (constraints T2 and T4). When time relations "within" or "throughout" are found, the whole period (the two constraints affected in the temporal group) is fixed.

In case of the *period* group, it can be given the starting range of the slot value (denoted by T1 and T2), the range of the ending time (using T3 and T4), just the initial point (T1) or the final point (T4), or information about the time when the value was valid (denoted by T2 and T3).

Finally, we aggregate all the time constraints found for the same slot value accross different documents with the objective of narrowing the time ranges associated to a slot value. This processing consists in selecting the maximun T1 and T3 constraints among the available ones, and the minimun T2 and T4 constrains among the ones found.

6 Regular Slot Filling Subtask

6.1 Named Entity type matching

For each node that matches the NE condition for the slot, if any, we do: substitute punctuation by white space; collapse white space; wee give a positive matching if: the value string is contained in the descriptor; or the descriptor is contained in the value (and it is not an English stop word); or the Levenshtein distance between value string and descriptor is over 0.8

A source of trouble here is that for some slot types, there is not a perfect correspondence between the NE types obtained from the annotation and the actual *type* of the values that the human evaluators of the KBP Slot Filling task accept as correct.

Our initial approach was to force the NE types of every slot, but our first experiments showed that this condition was too restrictive. For the slots: `org:political/religious_affiliation`, `per:origin`, `per:religion`, we decided not to apply NE filtering.

For the slot type `per:title`, in particular, we accepted values annotated by the TITLE Stanford NE type or not annotated by any type.

This exceptions aimed at obtaining more and better positive examples, but also increased the number of examples to consider, as nodes with no NE annotation were tried for matching several of the slots, and as we will describe below, unmatching pairs are considered negative examples.

In the process of gathering positive examples, once fixed the entity node in the graph, for each candidate value that did not match the seed value, we generated a negative training example. Not only nodes with an identified NE type were considered, as we thought no NE nodes would be the right response to, for instance, the `per:title` slot. We kept 5% of the negative examples, as the set produced by this procedure was very large.

6.2 Choice of Classifier

We trained a SVM multi-class classifier with the positive and negative examples. We used the implementation[3] of the cutting-plane SVM algorithm (Joachims et al., 2009). We empirically evaluated a

[3]This implementation is freely available at `http://svmlight.joachims.org/svm_multiclass.html`

Collection	Positive Training Examples	Negative (unlabeled) examples	Slots with some example
KBP	18 491	5 242	42
Semi-supervised	17 682	-	32
Supervised	809	-	42

Table 3: Statistics on the training examples available for the Full Regular Slot Filling Task (collapsed version of the graphs). The training for the KBP Participation includes the positive examples of the semi-supervised and supervised collections, and a random sample of the negative examples found while processing both collections.

few configurations on the collection of known slots from earlier editions of the KBP Slot Filling Task, and decided to use a large value for the parameter C that controls the trade-off between training error and margin: $C = 10\ 000$. All other parameters were left in their default values.

6.3 Use of supervised seeds

Unfortunately, our elaborate automatic labeling procedure did not produce training examples for every slot type of the task. Of the 42 slots to be extracted, we had less than 2 training examples for up to 12 slots. We could not expect to extract any value for these slots.

In an attempt to overcome this, we added to the training seeds a set obtained *with supervision*; even though this contradicted our initial semi-supervised design. This supervised set is the set of the tuples: $< entity, slot\_type, value, doc >$, where entity is on of the query entities of the 2009 and 2010 editions of the KBP Slot Filling Task, and the slot type, correct value and support document are those annotated in the official evaluation key. This added 809 examples to our training, covering all slots, and producing a training dataset less disperse.

Statistics of the training examples are given in Table 6.3, for the collapsed version of the graphs[4].

6.4 Runs submitted

Let us describe the runs submitted to the task, that are also summarized in Table 4.

[4]The differences in the processing of the document graphs and the collapsed document graphs yields small differences in the number of training examples finally available; we only report those of the collapsed version.

	Collapse discourse referents	Lexical Context features	Semantic Transformations
UNED1		✓	
UNED2	✓	✓	✓
UNED3	✓		✓

Table 4: Runs submitted to the Regular Slot Filling Task. The marked components were used for each of the runs.

UNED1 In this first run, we used the document graph G_D without collapsing coreferent nodes.

UNED2 For the second run, we applied our full processing, using for representation the collapsed document graphs, G_C.

UNED3 This last run also used the collapsed document graphs, but differed from the previous in that we did not use the features marked as "Lexical context" in Table 1, that consider the actual words in the context of the entity and value nodes.

7 Temporal Slot Filling Subtask

The focus of our development was to tackle this subtask. Many of our engineering decisions, such as processing the documents for event detection (with the Tarsqi Toolkit), were taken because of this objective.

In this section we describe the particular adaptations of our general approach for the Full Temporal Slot Filling subtask.

7.1 Named Entity type matching

There are significant differences to how we performed this step for the Regular Slot Filling subtask. On the one hand, we enforced Named Entity type matching both of the candidate values and of the entities. As we explain below, we used a set of binary classifiers, or extractors, one for each of the slots to extract; for each particular slot classifier, only candidates with the expected entity and value types for the slot were used in the application phase.

7.2 Choice of Classifiers

For this subtask, we used a battery of binary classifiers; each of them was a SVM classifier with linear kernel (Joachims, 2002). We used the SVMLight implementation available at `http://svmlight.joachims.org/` We did not have time to tune the classifiers, so we left all parameters in their default values.

As we traverse the document graph looking for positive examples of a slot type, if we encounter a node that matches the expected NE type of the slot, but does not match the value in the seed, we generate a negative example for that slot classifier. As opposed to the sampling of negative examples we performed in the Regular Slot Filling subtask, here we kept all negative examples.

For this subtask, we did not use any supervised training data, as all of the target slots were covered by our semi-supervised training examples.

	Collapse discourse referents	**Use dates in document**	**Temporal resolution**
RUN 1		✓	
RUN 2	✓	✓	
RUN 3	✓	✓	✓

Table 5: Runs submitted to the Full Temporal Slot Filling Task. The marked components were used for each of the runs.

7.3 Runs submitted

For the three runs we submitted for the task, we used different versions of the underlying document graph representations, and different processing to generate the temporal constraints. The classifier features we used are common for every run, and are detailed in Table 1 and Section 2. In the following paragraphs, we describe the differences between each of the runs, these are also summarized in Table 5.

RUN1 In this first run, we used the document graph G_D without collapsing coreferent nodes. To generate the temporal constraints, we aggregate the dates of the documents that hold the solutions to fill the constraints T2 and T3.

RUN2 In the document representation used for this second run, we applied our full processing, using for representation the collapsed document graphs, G_C. To generate the temporal constraints, we simply aggregate the dates of the documents that hold the solutions to fill the constraints T2 and T3.

RUN3 In the third run, we also applied the collapsed document graphs, G_C. For the temporal constraints we apply the processing detailed in Section 5.

8 Results

8.1 Regular Slot Filling

The results of the Regular Slot Filling subtask have been poor. Therefore, we tried to evaluate where our losses have come from.

Candidate retrieval component After our empirical evaluation, we expected to be able to retrieve 75% of the documents relevant to the task queries. Comparing with the evaluation results, the figure was comparable, but lower: 71.6%. The recall of the IR module is low, and we are not addressing important issues, such as the possible ambiguity in the entities of the query (see Table 3).

Information Extraction component Both the precision and recall of our system were low, and the KBP scores were lower than the values we obtained in development. As we have seen, the quality of the automatically labeled examples is poor, and the noise introduced damages the performance of the overall system. The training collection we obtain is very disperse, as some of the slots contribute many of the examples and other very few. For up to six of the slot types, we did not extract positive training examples, or very few, a problem we tried to circumvent by considering in training the supervised examples from previous editions of the task. This goes against our self-supervised approach, and does not push the scores very far up, either. Augmenting the number of documents processed increases the performance of the system, but not greatly, as the problem of sparse training is not overcome.

Size of the training set To evaluate the effect of having more seeds, we have compared the number of training examples available in training for the Regular Slot Filling subtask against the *coverage* of the correct results in the response of the system. We define this *coverage* as the number of correct responses that were extracted by the IE component, before selecting from them the final response of the system. We say a response is correct if it matches exactly the response in the official evaluation key (this is a lower bound for the number of correct responses). The correlation coefficient between these two figures is 0.43; having a look at Figure 4 we see that some slots behave differently, but that a larger number of

	Supervised		Semi-Supervised	
Slot	Number of Seeds	Matches	Number of Seeds	Matches
org:city_of_headquarters	50	1	5608	4
org:country_of_headquarters	45	0	3856	5
org:founded	30	1	2191	0
org:parents	33	1	2912	4
org:stateorprovince_of_headquarters	36	0	2021	2
org:subsidiaries	100	1	320	0
per:age	52	5	1	0
per:city_of_birth	14	2	558	0
per:date_of_birth	12	1	59	0
per:employee_of	84	2	756	8
per:location_of_residence	139	6	819	0
per:member_of	49	4	1435	0
per:title	133	5	3031	3

Table 6: Correct matches using semi-supervised seeds (obtained from the KB automatically) or supervised seeds (obtained from the evaluation keys from past editions of the task). For slots with at least one correct match.

examples tends to correspond with better results.

Supervised seeds against semi-supervised seeds As noted in 6.3, we decided to merge our automatically extracted seeds with a set of *supervised* seeds, obtained from the keys of previous editions of the task.

To evaluate the contribution of semi-supervised and supervised seeds, we run the following experiment, after the participation in the task. We used a configuration of the system that is similar to the one described in Section 6.4 for RUN2, although the number of documents we have processed is larger. We train the system only with the supervised seeds, only with the semi-supervised seeds, and with both sets of seeds together (as in our participation in the task). We then checked the number of exact string matches with the official evaluation response key.

Using the semi-supervised seeds, we obtained 26 matches with the key. Using the supervised seeds only increased the number of matches to 31. In the submission of RUN2, we had 37 exact string matches.

The matches for each slot are differently distributed, as summarized in Table 6. With the supervised collection of seeds, which is smaller and better balanced, we have few correct results but spread evenly over the slots. With the poorly balanced semi-supervised collection, we get more matches for fewer slots[5]. This can be explained by the choice of learning method, a SVM multi-class classifier that divides the space of examples by cutting-planes.

It is interesting to note that some of the right predictions with the supervised seeds are for slots for which we have very few seeds in the semi-supervised collection. Therefore, the number of correct results that it is possible to get with distant supervision is similar to what it is possible to get with supervised seeds.

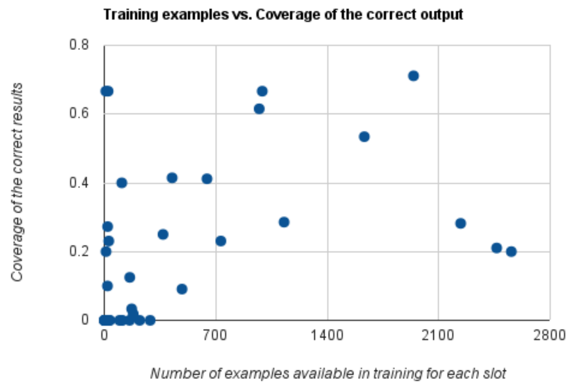

Figure 4: For each slot, we plot the *coverage* (how many values in the evaluation key were among the responses of our system, *before* the final filtering step) against the number of positive examples available in training.

Document representation From the Regular Slot Filing official evaluation scores, we can observe that

[5]We joined together the training for the slots `per:*_of_residence` into a single slot `per:location_of_residence`.

System	# Filled	Precision	Recall	F1
LDC (manual)	532	0.7041	0.6131	0.6555
BLENDER2	1139	0.1749	0.3261	0.2277
BLENDER1	1108	0.1750	0.3173	0.2256
BLENDER3	1153	0.1642	0.3099	0.2147
IIRG1	317	0.2505	0.1300	0.1711
UNED2	**156**	**0.2571**	**0.0657**	**0.1046**
UNED1	**165**	**0.2299**	**0.0620**	**0.0978**
UNED3	**156**	**0.2196**	**0.0561**	**0.0893**
Stanford 12	5094	0.0207	0.01724	0.0369
Stanford 11	4315	0.0211	0.01492	0.0370
USFD20112	327	0.0099	0.0053	0.0069
USFD20113	126	0.0020	0.0004	0.0006

Table 7: System ID, number of filled responses of the system, precision, recall and F measure.

collapsing the coreferent nodes has produced a slight boost of performance, and that having the lexical information of the words around entity and value also improved performance.

8.2 Temporal Slot Filling

The performance on relation extraction is an upper bound for temporal anchoring, attainable if temporal anchoring is perfect. Thus, we also evaluate the temporal anchoring performance as the percentage the final system achieves with respect to the relation extraction upper bound.

Results. We show in Table 7 results of all participants ranked by values of *F1*. The table contains information about the number of slots correctly filled by each system as well as *precision*, *recall* and *F1* values.

We can see that our results are low due to the upper bound that error propagation in candidate retrieval and relation extraction imposes upon this step: we have seen that our temporally anchoring alone achives 69% of its upper bound. This value corresponds to run UNED2 (whose way of extracting temporal information could be considered as a baseline for this task), showing its strength. The difference in performance with run UNED3 shows that this baseline is difficult to beat by considering temporal evidence inside the document content. There is a reason for this. The temporal link mapping into time intervals does not depend only on the type of link, but also on the semantics of the text that expresses the relation as we pointed out above. We have to decide how to transform the link between relation and temporal expression into a temporal interval. Learning a model for this is a hard open research problem that has a strong adversary in the baseline proposed.

On the other hand, a comparison of runs UNED1 and UNED2, which only differ on the document representation, shows us an slight improvement in results when using collapsed coreferent nodes. However, this observation is not kept when comparing runs UNED1 and UNED3, which differs on document representation and temporal anchoring. This is due to the fact that the temporal anchoring used for us as a baseline (employed in run UNED1) is able to clearly outperform the more complex temporal anchoring of run UNED3 despite the fact of using a less proper representation.

Comparative Evaluation.

We have compared also our approach with the other four participants at the KBP Temporal Slot Filling Task 2011. As shown in column *Filled* of Table 7, our approach returns less triples than other systems, explaining the low recall values. However, our system achieves the highest precision for the complete task of temporally anchored relation extraction. Despite low recall, our system obtains the third best F_1 value. This is a very promising result, since several directions can be explored to consider more candidates and increase recall. Table 7 also includes a row for the scores for LDC's manual annotation, that can be considered an upperbound for automatic systems.

9 Conclusions and future work

There is room for improvement for our system, as we could not tune each of the system components. Nevertheless, we can conclude with the following observations.

We hypothesized that distant learning would be enough for addressing the task of filling all the KBP slots. We have observed, nevertheless, a few shortcomings of this approach. The first is that the process of automatically extracting seeds from an infobox KB such as the one provided by the organization is complex. The ambiguities in the definition of the infobox attributes, and its far from perfect correspondence with the target slots have to be taken care of. Otherwise, the quality of the seeds obtained is worse than that of manually supervised seeds. Better quality seeds could be found substituting the provided initial KB with a cleaner input KB. A better mapping from the resource to the target slot types would allow having a comparable number of seeds for each slot, and subsequently, a well balanced dataset. We have observed that the having *more* training examples has an effect in the performance, but that it is less important than the effect of having *good* seeds.

We leave for future work the problem of comparing the approach of automatically obtaining seeds from a Wikipedia based resource against a bootstrapping method that would start from the manually supervised seeds.

For the temporal anchoring sub-problem, we have demonstrated the strength of the document creation time as a temporal signal. It is possible to achieve a performance of 69% of the upper-bound imposed by relation extraction by assuming that any relation mentioned in a document held at the document creation time (there is a *within* link between the relational fact and the document creation time). This baseline has proved stronger than extracting and analyzing the temporal expressions present in the document content.

We expected a larger impact of the representation in the performance of the system. A future line of research is to better exploit the document representation for training, instead of our simple binary feature generation process.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation, through the project Holopedia (TIN2010-21128-C02), and the Regional Government of Madrid, through the project MA2VICMR (S2009/TIC1542).

References

Eneko Agirre, Angel X. Chang, Daniel S. Jurafsky, Christopher D. Manning, Valentin I. Spitkovsky, and Eric Yeh. 2009. Stanford-UBC at TAC-KBP. In *TAC 2009*, November.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL 2005*, pages 363–370.

Heng Ji and Ralph Grishman. 2011. Knowledge base population: Successful approaches and challenges. In *ACL HLT 2011*, pages 1148–1158.

Thorsten Joachims, Thomas Finley, and Chun-Nam Yu. 2009. Cutting-plane training of structural svms. *Machine Learning*, 77:27–59.

T. Joachims. 2002. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL 2003*, pages 423–430.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *CoNLL-2011*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL 2009*, pages 1003–1011, Stroudsburg, PA, USA. Association for Computational Linguistics.

James Pustejovsky, Jessica Littman, and Roser Saurí. 2007. Arguments in TimeML: Events and entities. In *Annotating, Extracting and Reasoning about Time and Events*, LNCS, chapter 8, pages 107–126.

Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. 2010. A multi-pass sieve for coreference resolution. In *EMNLP-2010*.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In José Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6323 of *LNCS*, pages 148–163. Springer Berlin / Heidelberg.

Mihai Surdeanu and Massimiliano Ciaramita. 2007. Robust information extraction with perceptrons. In *ACE07*, March.

Mihai Surdeanu, David McClosky, Julie Tibshirani, John Bauer, Angel X. Chang, Valentin I. Spitkovsky, and Christopher D. Manning. 2010. A simple distant supervision approach for the tac-kbp slot filling task. In *TAC 2010*, November.

Marc Verhagen, Inderjeet Mani, Roser Sauri, Robert Knippen, Seok Bae Jang, Jessica Littman, Anna Rumshisky, John Phillips, and James Pustejovsky. 2005. Automating temporal annotation with TARSQI. In *ACLdemo'05*.