Combining Flat and Structured Approaches for Temporal Slot Filling or: How Much to Compress?

Qi Li, Javier Artiles, Taylor Cassidy, and Heng Ji

Computer Science Department and Linguistics Department,
Queens College and Graduate Center,
City University of New York,
New York, NY 11367, USA
{liqiearth,javart,taylorcassidy64,hengjicuny}@gmail.com

Abstract. In this paper, we present a hybrid approach to Temporal Slot Filling (TSF) task. Our method decomposes the task into two steps: temporal classification and temporal aggregation. As in many other NLP tasks, a key challenge lies in capturing relations between text elements separated by a long context. We have observed that features derived from a structured text representation can help compressing the context and reducing ambiguity. On the other hand, surface lexical features are more robust and work better in some cases. Experiments on the KBP2011 temporal training data set show that both surface and structured approaches outperform a baseline bag-of-word based classifier and the proposed hybrid method can further improve the performance significantly. Our system achieved the top performance in KBP2011 evaluation.

1 Introduction

There are many relations between named entities that may change over time (e.g. a person's residence, an organization's top employees, etc.), and these changes are expressed in the usage of temporal expressions in text. The TempEval evaluation campaigns [15] studied Temporal Information Extraction (TIE) concentrating on the identification of temporal expressions, events and temporal relations, but these tasks did not tackle the problem of finding the specific start and end dates for a given relation or event. In order to solve this problem a TIE system should detect whether a temporal expression actually concerns a certain relation, and in that case, the kind of role this temporal expression plays (i.e. whether it expresses the beginning of the relation, its end, or a time in between). Temporal information about a single relation can be scattered among different sentences and documents and presented with varying degrees of precision (e.g. a specific date, a range of dates such as a month, season, or year). To address these problems a system needs to detect coreferential mentions of the entities involved in a relation and aggregate the collected temporal information into a single answer. The NIST TAC Knowledge Base Population (KBP) 2011 track [9] included a pilot Temporal Slot Filling (TSF) task. In this task systems extract the start and end dates of a relation from a collection of documents. A relation involves an entity, a type of slot and a particular fill for that slot (e.g. "Nicole Kidman" is the slot fill for the entity "Tom Cruise" in the relation *spouse*) .

As is the case for many NLP tasks, one of the main challenges of TSF lies in capturing relationships within long contexts. The context surrounding the entity, slot fills and temporal expressions is often too long and diverse to be generalized with surface features. By using syntactic parsing we can *compress* long contexts based on their underlying structure and capture common syntactic patterns. However, NLP tools that try to provide a deeper representation of the text can introduce many errors. Furthermore, in cases where there is a short context, surface features tend to provide a more appropriate generalization. One of the earliest works in IE asked "where is the syntax?" [8] and concluded that, although incorporating structure into an Information Extraction process would be necessary to overcome performance plateaus, only a conservative approach to parsing would be accurate enough to improve IE results without introducing too much noise. Our work re-visits the general hypothesis that using just the right amount of structure can improve IE results by evaluating the impact of features defined in terms of multiple levels of structure.

Our general approach to the TSF task is to decompose it into two problems: the classification of a temporal expression in the context of a query and its slot fill(s); and temporal information aggregation, in which the classified temporal expressions are combined to produce the start/end dates of a relation expressed by a given query and its slot fill. To this end, we have developed and tested two approaches to the temporal classification problem: a structured approach and a flat approach. The structured approach captures long syntactic contexts surrounding the query entity, slot fill and temporal expression using a dependency path kernel tailored to this task. The flat approach exploits information such as the lexical context and shallow dependency features. We show that these two approaches are complementary and thus can be combined through a simple temporal information aggregation process. We also show that the performance of each approach is highly correlated with the length of the example contexts. Our proposed approaches outperform a number of baselines, and achieved the top performance in the KBP2011 evaluation.

2 Related Work

The need for structural representations is acknowledged in many Natural Language Processing fields. For example, the shortest path between two vertices in a dependency parsed graph has been used to capture the syntactic and semantic relation between two words [2,16,18].

Temporal IE has recently attracted intensive research interests. For example, the TempEval [15,17] shared tasks has aimed at the extraction of relations between events and temporal expressions from single documents. Various approaches have been developed for this task, which can be roughly categorized into flat or structured approaches: (1) **Flat approaches**: [3] built a supervised learning model to classify a pair of event triggers in a sentence based on syntactic and semantic features at the lexical level based on tense and aspect. [19] used similar features, using a Markov Logic based joint inference framework for temporal relations. [10] also exploited cross-event joint inference, but they used shallow dependency features to build local formulas without considering the deeper syntactic structure. (2) **Structured approaches**: [1] designed syntactic and semantic features based on syntactic treelets and verbal government for temporal relation classification. [14] used sentence level syntactic trees to propagate temporal relations between syntactic constituents. [5] introduced a type of feature called *tree position* that classifies nodes on a syntactic dependency tree based on their position in the tree relative to that of a target node.

Our work advances temporal IE in the following three main aspects: (1) it extends the notion of temporal relation from that of a pair <event, time expression>, or <event>, to a 3-tuple <entity, relation/event, time expression>, allowing us to capture temporal information of varying degrees of uncertainty; (2) We represent the contexts surrounding the tuple elements, using both flat and structural features; (3) it extends from single-document extraction to cross-document extraction so that we are able to effectively combine the advantages from flat and structured approaches through cross-document information aggregation.

3 Experimental Setup

3.1 Task Definition

The goal of KBP2011 temporal slot filling task is to add temporal information to selected slots for a given entity query from a large collection of documents. The slot types considered on this task are: spouse, title, employee_of, member_of, cities_of_residence, state or provinces_of_residence and countries_of_residence for people, and the top_employees/members slot for organizations. There are two subtasks: full and diagnostic. For the full temporal task, the system is given an entity name and a document where this entity is mentioned and is expected to find the relevant slots in the document collection, augmented with temporal information as described below. For the diagnostic temporal task, the system is given the entity and a set of slot values with their types and the documents that support them. For this task the system should determine the temporal information for each slot value, based only on the information in the provided support document. In order to investigate the capability of various approaches to temporal information extraction, we conduct experiments in the diagnostic setting.

3.2 Temporal Representation

The KBP2011 temporal representation model consists of a 4-tuple whose elements are dates (day, month and year), $< t_1, t_2, t_3, t_4 >$. A tuple represents the set of possible beginnings and endings of an event. t_1 and t_2 represent the lower and upper bounds, respectively, for the beginning of the event, while t_3 and t_4 represent the lower and upper bounds for end of the event.

Given a slot-filling query name *Jose Padilha*, its slot fill *Film Maker* for the slot type *per:title*, a diagnostic temporal slot filling system may discover a temporal tuple $< -\infty, 2007-12-26, 2007-12-26, +\infty >$ to represent the temporal boundaries.

3.3 Scoring Metric

We use the official scoring metric $Q(S)$ for the task. This metric compares a system's output $S =< t_1, t_2, t_3, t_4 >$ against a gold standard tuple $S_g =< g_1, g_2, g_3, g_4 >$, based on the absolute distances between t_i and g_i:

$$Q(S) = \frac{1}{4} \sum_i \frac{1}{1 + |t_i - g_i|}$$

When there is no constraint on t_1 or t_3 a value of -∞ is assigned; similarly a value of +∞ is assigned to an unconstrained t_3 or t_4.

Let $\{G^1, G^2, ..., G^N\}$ be the set of gold standard tuples, $\{S^1, S^2, ..., S^M\}$ the set of system output tuples, where for each unique slot fill i, G^i there is the 4-tuple $< g_1, g_2, g_3, g_4 >$, and S^j is the 4-tuple $< t_1, t_2, t_3, t_4 >$. Then Precision, Recall and F-measure scores are calculated as follows:

$$Precision = \frac{\sum_{S^i \in C(S)} Q(S^i)}{M} \quad Recall = \frac{\sum_{S^i \in C(S)} Q(S^i)}{N} \quad F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Where $C(S)$ is the set of all instances in system output which have correct slot filling answers, and $Q(S)$ is the quality value of S. In the diagnostic task, precision, recall, and F_1 values are the same since we are provided with correct slot filling values as part of the system input.

4 Approach Overview

Our approach to the TSF problem consists of three main steps: (i) find all the contexts where the entity and the slot value are mentioned; (ii) classify each temporal expression in those contexts according to its relation with the entity/slot value pair; (iii) aggregate all the classified temporal information in a coherent 4-tuple. In Figure 1 we summarize the system pipeline. Our system takes as input an entity, slot type and slot value as well as the source documents where the slot value was found.

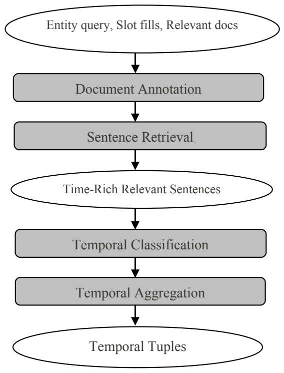

Fig. 1. General Temporal Slot Filling System Architecture

Each source document is fully processed using the Stanford NLP Core toolkit [7] to tokenize, detect sentence boundaries, detect named entities (including temporal expressions), build coreference chains and analyze the syntactic dependencies within sentences. The annotated output is used to find sentences that mention both the entity and the slot value. Finding these sentences by string matching provides only very limited coverage, so we use named entity recognition and coreference results to expand this set of relevant sentences. We look at the coreference chains that contain the provided slot value or entity name and we select sentences that mention both, according to the coreference chains.

Each temporal expression in these sentences is then represented as a classification instance and labeled as belonging to one of the following classes: start, end, hold, range and none. Finally, for each particular entity/slot value, all of its classified temporal expressions are aggregated in a single 4-tuple.

4.1 Temporal Classification

Classification is applied to label temporal expressions that appear in the context of a particular entity and the slot value as "start", "end", "hold", "range" or "none". Suppose our query entity is *Smith*, the slot type is *per:title*, and the slot-fill is *Chairman*. Table 1 shows a description of each class along with its corresponding 4-tuple representation:

Table 1. Description of Temporal Classes

Class	Temporal Role	Four tuple
Start	beginning of the slot fill	$< t_a, t_b, t_a, \infty >$
End	end of the slot fill	$< -\infty, t_b, t_a, t_b >$
Hold	a time at which the slot fill is valid	$< -\infty, t_b, t_a, \infty >$
Range	a range in which the slot fill is valid	$< t_a, t_a, t_b, t_b >$
None	unrelated to the slot fill	$< -\infty, \infty, -\infty, \infty >$

The next two subsections describe the two classification approaches we have tested.

4.2 Flat Approach

The flat approach uses two types of features: window features and dependency features. A window feature value for the query entity, slot value, and a target temporal expression is extracted from each example. This value is a set containing all tokens that occur in the normalized sentence within 4 tokens in either direction of any instance of the normalized token in question.

Two dependency feature values for the query entity, slot value, and a target temporal expression are extracted from each example, resulting in two sets of tokens for each normalized token T. One set contains all tokens that any instance of T governs, the other set contains all tokens governed by any instance of T. Before a feature value set for a normalized token T is created, punctuation marks, duplicate consecutive normalized tokens, and instances of T itself are removed.

Example (1) is from the evaluation set, for the query, attribute = *per:title*, entity = *Makoni*, slot fill = *minister of industry and energy development* .(1') is its normalized version.

(1) In 1981, Makoni was moved to the position of minister of industry and energy development, where he remained until 1983.

(1') In DATE, TE was moved to the position of TA , where he remained until TD.

Table 2 shows the feature values extracted from (1').

Table 2. Feature Values for (1)

Feature	Value
TE Win	be, move, to, in, DATE, the
TA Win	of, to, remain, position, the, where, until, he
TD Win	remain, where, until, he
TE Governs	-
TA Governs	-
TD Governs	-
TE Governed by	move
TA Governed by	position
TD Governed by	remain

For two feature values U, V, let K_T be the normalized size of their intersection

$$K_T(U, V) = \frac{|U \cap V|}{\sqrt{|U|^2 + |V|^2}} \tag{1}$$

Let F denote the flat features. Then for any $G \subseteq F$, let K_S be the kernel function for a pair of examples, and $x.i$ the feature value for the i^{th} feature value type for example x:

$$K_S(x, y) = \sum_{i \in G} K_T(x.i, y.i) \tag{2}$$

With these features we trained a classifier using Support Vector Machines (SVM) [6].

4.3 Structured Approach

Dependency Path Representation. In the structured approach, we exploit collapsed dependency parsed graphs generated from the Stanford dependency parser [12] to capture relevant grammatical relations and discover syntactic patterns. Figure 2 shows a part of the dependency graph obtained from the sentence, *"In $\underline{1975}_{[time\ expression]}$, after being fired from Columbia amid allegations that $\underline{he}_{[query\ entity]}$ used company funds to pay for $\underline{his}_{[query\ entity]}$ son's bar mitzvah, $\underline{Davis}_{[query\ entity]}$ founded $\underline{Arista}_{[slot\ fill]}$"* . In this example, *Davis* is the query entity, the slot type is *per:employee_of*, *Arista* is the slot fill, and *1975* is the time expression.

We extend the idea of shortest path on a dependency graph (see Section 2) to include three items: query entity, slot fill and time expression. Each instance is represented by

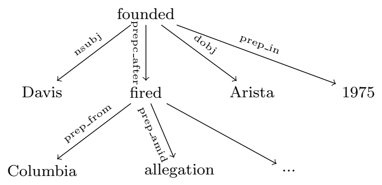

Fig. 2. Dependency parsed graph of the sample sentence

three paths: (i) the path between query entity and temporal expression (P_1), (ii) the path between slot fill and temporal expression (P_2); and (iii) the path between query entity and slot fill (P_3).

We found that two modifications in the graph allow us to obtain more informative paths. To capture phrasal verbs, take for example *took over* as one node in the path instead of only using *took*, we change two vertice linked by *prt* dependency into one vertex, where *prt* indicates a phrasal verb relation. Second, consider dependency path between *he* and *president*, sentence "*he was president of ...*" and "*he is president of ...*" produce same path, because *he* and *president* are linked by *nsubj*, where *nsubj* indicates subject relation. To address this issue, we reshaped the dependencies around copula verb such as *is* and *become* to those of common verbs.

Each shortest path P_i is represented as a vector $< t_1, t_2, ..., t_n >$, where t_i can be either a vertex or a typed edge in the dependency graph. Each edge is represented by one attribute, which is formed by combining the corresponding dependency type and direction. More formally, attribute $a \in \mathcal{D} \times \{\leftarrow, \rightarrow\}$, where $\mathcal{D}$ is the set of dependency types, and the arrow is directed from the governor to the dependent word. Vertices, on the other hand, may contain different levels of features, which can be found in Table 3.

Table 3. Features of Vertices

Feature	Description
Word	The original word token from the sentence. E.g., "Davis *founded*$_{[founded]}$ Arista"
Stem	Stemmed form of the word token. E.g., "Davis *founded*$_{[found]}$ Arista"
Entity type	Person, Location, Organization. E.g., "fired from Columbia$_{[Organization]}$"
Semantic class of trigger words	Each class contains trigger words of event subtype in Automatic Content Extraction 2005 corpus[1], and some manually collected slot type sensitive key words, E.g. if slot type is *per:spouse*, then the word *marry* belongs to one semantic class while *divorce* belongs to another semantic class.
Part-of-speech	Part-of-speech tag of original word

[1] `http://projects.ldc.upenn.edu/ace/`

For example, in the sentence of Figure 2, there exists *prep_in* dependency from *founded* to *1975*. *prep_in* represents prepositional relation between these two words, meaning that the action *founded* happened at *1975*.

When we search the shortest path between two nodes, we consider all mentions of the query entity and the slot fill in a sentence. For this reason there could be more than one candidate for each P_i. We define the following simple but effective strategy to choose one path among all candidate paths. If some candidate paths contain predefined trigger words, we choose the shortest path with trigger words. Otherwise, we choose the shortest path among all candidates.

Figure 3 shows three shortest paths that result from the sentence of Figure 2. These paths not only contain lexical features such as words, but also syntactic relations. In the resulting representations, informative patterns are distilled while some irrelevant information, as well as misleading words such as *fire*, are discarded.

The next step in our system is to use a kernel function to generalize these paths and represent them in a high dimensional feature space implicitly.

$$\text{Davis} \xleftarrow{\text{nsubj}} \text{founded/found/VBD/Start-Position} \xrightarrow{\text{prep\_in}} 1975$$

$$\text{Arista} \xleftarrow{\text{dobj}} \text{founded/found/VBD/Start-Position} \xrightarrow{\text{prep\_in}} 1975$$

$$\text{Davis} \xleftarrow{\text{nsubj}} \text{founded/found/VBD/Start-Position} \xrightarrow{\text{dobj}} \text{Arista}$$

Fig. 3. Three Shortest Paths from Figure 2

Kernel Function. Following previous work [11] and [2], we present a string kernel function based on dependency paths. The main idea is to use the kernel trick to deal with common-substring similarity between dependency paths, and to extract syntax-rich patterns from dependency paths.

Let x, y be two instances. We use $l(P)$ to denote the length of a dependency path P, $P[k]$ to denote the set of all substrings of P which have length k, and a substring $a \in P[k]$ is a substring of P with length k. For example, if P is "*ABC*", then $P[2]$ ={"*AB*", "*BC*"}. The kernel function of x and y is defined as follows:

$$K_s(x, y) = \sum_{i=1}^{3} K_p(x.P_i, y.P_i) \tag{3}$$

$$K_p(P_x, P_y) = \sum_{k=1}^{Min(l(P_x),l(P_y))} \sum_{a \in P_x[k], b \in P_y[k]} \prod_{i=1}^{k} c(a_i, b_i) \tag{4}$$

Where K_p is a kernel function on two dependency paths P_x and P_y which sums the number of common substrings of feature paths in P_x and P_y with length from 1 to the maximum length. In $c(a_i, b_i)$ we calculate the inner product of the attribute vectors of a_i and b_i, where a_i and b_i are elements of two paths respectively. The final kernel function K_s does the summation of the partial results of the three dependency paths (query entity-slot fill, query entity-temporal expression, slot fill-temporal expression).

Consider the following example containing two dependency paths P_x and P_y between an entity (E) and a temporal expression (T) in two different sentences.

$\text{E}\xrightarrow{\text{nsubj}}\text{founded/found/VBD/Start-Position}\xleftarrow{\text{prep\_in}}\text{T}$
$\text{E}\xrightarrow{\text{nsubj}}\text{joined/join/VBD/Start-Position}\xleftarrow{\text{prep\_in}}\text{T}$

For instance, if we consider substrings of length 5 we find the following two matches:

$\text{E}\xrightarrow{\text{nsubj}}\text{VBD}\xleftarrow{\text{prep\_in}}\text{T}$
$\text{E}\xrightarrow{\text{nsubj}}\text{Start-Position}\xleftarrow{\text{prep\_in}}\text{T}$

By counting the common substrings for the remaining lengths (1 to 4) we can obtain the final result: $K_p(P_x, P_y) = 26$.

A problem of Equation (4) is that K_p has a bias toward longer dependency paths. To avoid this bias, we normalize K_p as in [11]. This normalization scales the feature vector $\phi(P)$ in the kernel space to $\phi'(P) = \frac{\phi(P)}{|\phi(P)|}$:

$$K'_p(P_x, P_y) = \frac{K_p(P_x, P_y)}{\sqrt{K_p(P_x, P_x) \cdot K_p(P_y, P_y)}} \tag{5}$$

A deviation from related work in [11] and [2] is that we count common substrings from m to maximum, rather than a fixed length. Furthermore, we only consider contiguous substrings in K_p because each substring feature in the kernel space is treated as a pattern. Non-contiguous substrings with the same length can be safely discarded as different patterns.

Although it's not easy to enumerate all substrings explicitly, like many other kernel functions, K_p can be efficiently computed by using dynamic programming in polynomial time complexity. Here, we applied a variant of the Levenshtein Distance algorithm to calculate K_p. Given the representation and kernel function, SVM model was applied to train a classifier.

4.4 Temporal Aggregation

In order to produce the final 4-tuple for each entity/slot value pair, we sort the set of the corresponding classified temporal expressions according to the classifier's prediction confidence. We initialize a 4-tuple to $< -\infty, +\infty, -\infty, +\infty >$ and then iterate through that set, aggregating at each point the temporal information as indicated by the predicted label (see Section 4.1). Given two four-tuples T and T', we use the following equation for aggregation.

$$T \wedge T' = < max(t_1, t'_1), min(t_2, t'_2), max(t_3, t'_3), min(t_4, t'_4) >$$

At each step we modify the tuple only if the result is consistent (i.e. $t_1 \leq t_2$, $t_3 \leq t_4$, and $t_1 \leq t_4$).

Furthermore, we utilize 4-tuple aggregation to combine outputs from the flat classifier, which uses shallow syntactic features, with that of the structured classifier, which uses deep syntactic features. We hypothesize that these two systems are complementary when combined in this way. Given an input, we consider the output from the structured classifier T as the default output. If one element of the output equals $-\infty$ or ∞, then we combine it with output from flat classifier T' as final output.

5 Experiments

5.1 Automatic Training Data from Distant Supervision

Given the expensive nature of human-assessed training data for this task, we adapted a distant supervision approach [13] to obtain large amount of training data from the Web without human intervention.

We use Freebase[2] to gather not only instances of relations, but also the start and end dates of those particular relations. We can still follow the usual distant supervision assumption: given a context that mentions the query entity and slot fill it is likely that it will express the relation in the database. But our methods go beyond the usual distant supervision in that we incorporate an additional element, the temporal expression. We assume that we can label a temporal expression occurring in the context of the entity/slot fill pair by comparing it to the start/end temporal information that is stored in our database. We obtained through this method more than 40,000 training instances with no human intervention.

5.2 Overall Results

To evaluate the performance of different approaches, we use the KBP 2011 temporal slot filling training data as test set. This data set contains 430 query entity names, and 748 slot fills and corresponding temporal four-tuples. In the experiments, we used LIBSVM library [4][3] to train SVM classifiers.

Table 4. Overall Performance

System	Overall	Employee_of	City	State	Country	Memebr_of	Title	Top_members	Spouse
BoW	0.638	0.637	0.781	0.525	0.662	0.582	0.702	0.510	0.438
Structured	0.667	0.674	0.844	**0.675**	**0.766**	0.627	0.702	0.538	0.556
Flat	0.663	0.657	0.844	0.661	0.707	0.613	0.707	0.544	0.570
Combine	**0.678**	**0.681**	**0.865**	**0.673**	0.721	**0.628**	**0.720**	**0.545**	**0.862**

We compared the performance of the proposed combination approach against *Structured*, *Flat*, and *BoW*. The baseline *BoW* uses bag-of-words representation and linear kernel on top of sentence normalization to represent each instance. Table 4 shows overall performance with breakdown scores for each slot type. Compared to other approaches, *BoW* achieves the lowest performance. Although the advantage of the structured approach against the flat approach is subtle, the combined system outperforms both of them, and achieves the highest scores in 7 slot types. We conducted the Wilcoxon Matched-Pairs Signed-Ranks Test on a four-tuple basis. The results show that the improvement of the combined system is significant at the 99.8% confidence level when compared with the structured approach, and at the 99.9% confidence level compared with the flat approach.

[2] `http://www.freebase.com`

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

6 Discussions

For many NLP tasks including this new TSF task, one main challenge lies in capturing long contexts. Semantic analysis such as dependency parsing can make unstructured data more structured by *compressing* long contexts and thus reduce ambiguities. However, current core NLP annotation tools such as dependency parsing and coreference resolution are not yet ideal for real applications. The deeper the representation is, the more risk we have to introduce annotation errors. Furthermore, for certain types of slots such as "title", since the contexts are relatively short between the query and its slot fill (e.g. "Today[Time] President[Title] Obama [Query]..."), structured representation is not appropriate. Therefore we pursued a more conservative approach combining benefits from both flat approach (local context, short dependency path, etc.) and structured approach (e.g. dependency path kernel). We reported that the structured approach outperforms the flat approach in general except slot types involve shorter contexts. Furthermore, combining them through cross-document temporal aggregation can achieve higher performance than each approach alone.

For example, there is a long context between the query "Mugabe", the time expression "1980" and its slot fill "ZANU-PF" in the following sentence "ZANU, which was renamed **ZANU-PF** after taking over ZAPU, has been the country's ruling party and led by **Mugabe** since **1980**." The structured approach successfully identified "1980" as the starting date based on the short dependency paths among "ZANU-PF", "Mugabe" and "Mugabe".

On the other hand, dependency parsing can produce errors. For example, it failed to capture the dependency relation between "September 2005" and "the Brookings Institute" in the following sentence "In **September 2005**, **Dichter** left office and became a research fellow at **the Brookings Institute** in Washington , D.C.". In contrast the flat approach can easily identify "September 2005" as the starting date for the query "Avi Dichter" to be a member of "the Brookings Institute" based on lexical features such as "became".

We also found that the gains by the structured approach are highly correlated with the compression rate, which is defined by (1 - the lengths of dependency paths among [query, slot fill, time expression] divided by the number of context words). For example, using structured approach they achieved much higher gains on residence slots (about 0.78 compression rate) than title (about 0.68 compression rate).

7 Conclusions and Future Work

In this paper, we presented a hybrid approach to diagnostic temporal slot filling task. We decompose the task into two steps: temporal classification and temporal aggregation. First, two approaches are developed for temporal classification: a flat approach that uses lexical context and shallow dependency features and a structured approach that captures long syntactic contexts by using a dependency path kernel tailored for this task. Following the hypothesis that these two approaches are complementary, we then combine them by aggregation as a hybrid approach. Experiment results show that the individual flat and structured approaches both outperform bag-of-word based classifier, and the proposed hybrid method can further improve the performance significantly. In the future we are particularly interested in conducting cross-query and cross-slot temporal reasoning to enhance the performance.

References

1. Bethard, S., Martin, J.H.: Cu-tmp: Temporal relation classification using syntactic and semantic features. In: SemEval 2007: 4th International Workshop on Semantic Evaluations (2007)
2. Bunescu, R.C., Mooney, R.J.: A shortest path dependency kernel for relation extraction. In: Proc. of the HLT and EMNLP, pp. 724–731 (2005)
3. Chambers, N., Wang, S., Jurafsky, D.: Classifying temporal relations between events. In: Annual Meeting of the Association for Computational Linguistics (ACL), pp. 173–176 (2007)
4. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001)
5. Cheng, Y., Asahara, M., Matsumoto, Y.: Naist.japan: Temporal relation identification using dependency parsed tree. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval 2007), pp. 245–248 (2007)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning, 273–297 (1995)
7. Finkel, J.R., Grenager, T., Manning, C.D.: Incorporating non-local information into information extraction systems by gibbs sampling. In: ACL (2005)
8. Grishman, R.: The NYU System for MUC-6 or Where's the Syntax? In: Proceedings of the MUC-6 workshop (2010)
9. Ji, H., Grishman, R., Dang, H.T., Li, X., Griffitt, K., Ellis, J.: An Overview of the TAC2011 Knowledge Base Population Track. In: Proc. Text Analytics Conference (TAC 2011) (2010)
10. Ling, X., Weld, D.: Temporal information extraction. In: Proceedings of the Twenty Fifth National Conference on Artificial Intelligence (2010)
11. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. The Journal of Machine Learning Research 2, 419–444 (2002)
12. Marneffe, M.C.D., Maccartney, B., Manning, C.D.: Generating Typed Dependency Parses from Phrase Structure Parses. In: LREC 2006 (2006)
13. Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. In: ACL/AFNLP, pp. 1003–1011 (2009)
14. Puşcaşu, G.: Wvali: Temporal relation identification by syntactico-semantic analysis. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval 2007), pp. 484–487 (2007)
15. Pustejovsky, J., Verhagen, M.: Semeval-2010 task 13: Evaluating events, time expressions, and temporal relations (tempeval-2) (2010)
16. Snow, R., Jurafsky, D., Ng, A.Y.: Learning syntactic patterns for automatic hypernym discovery. In: Advances in Neural Information Processing Systems, vol. 17, pp. 1297–1304 (2005)
17. Verhagen, M., Gaizauskas, R., Schilder, F., Katz, G., Pustejovsky, J.: Semeval2007 task 15: Tempeval temporal relation identification. In: SemEval 2007: 4th International Workshop on Semantic Evaluations (2007)
18. Wu, F., Weld, D.S.: Open Information Extraction using Wikipedia. In: Proc. of the 48th Annual Meeting of the Association for Computational Linguistics (2010)
19. Yoshikawa, K., Riedel, S., Asahara, M., Matsumoto, Y.: Jointly identifying temporal relations with markov logic. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pp. 405–413 (2009)