Using Textual Entailment with Variables for KBP Slot Filling Task

Ofer Bronstein, Erel Segal-haLevi
Department of Computer Science, Bar-Ilan University
Ramat-Gan, Israel
oferbr@gmail.com, erelsgl@gmail.com

Abstract

This report presents our extension of the textual entailment paradigm, in which variables are incorporated in the hypothesis, and are filled with extracted information during the entailment recognition process. This paradigm is the basis for a novel approach of using variables in textual entailment, with various potential future applications. We experimented with this approach in the TAC 2012 Knowledge Base Population challenge, participating in the Cold Start task. Our system was based on a variant of our open source system for textual entailment recognition – BIUTEE[1]. Being in preliminary stages of our work, the system does not yet scale well, so no results are incorporated in this report.

1 Introduction

The textual entailment (TE) paradigm deals with understanding whether one text in natural language can be inferred by reading another text. In the original formulation of the TE problem, given a pair of natural language text (T) and natural language hypothesis (H), a system shall decide whether T entails H, or not (Dagan and Glickman, 2004).

In the pats TE has been offered as an appealing approach for information extraction (IE) validation. For example, it was used for building the datasets for the various Recognizing Textual Entailment (RTE) challenges, as detailed in (Dagan et al., 2009). There, extractions from various IE systems were used as input for textual entailment systems, which in turn had to determine whether the extraction was valid or not by deciding whether entailment holds between the original text and a natural language sentence that embedded the extraction in an appropriate template. While filling the templates was performed manually in the first rounds of the challenge, in the more recent rounds, as described in (Bentivogli et al., 2010) and (Bentivogli et al., 2011), it was done automatically, allowing a much larger and richer ability to validate IE extractions.

We propose to extend the TE problem formulation, by allowing variables in the hypothesis, thus having a **template hypothesis**. For example, such hypothesis could be of the form *"{var1} was born in {var2}"*. During the entailment recognition process, the variables are filled with relevant phrases from T. This paradigm can be applied to IE; instead of just validating existing extractions, the extraction process itself is performed using textual entailment mechanisms. The variable assignments are the extractions.

It is important to note that unlike most common methods for performing IE, our approach is relatively **unsupervised**. While the system's classifier is trained on a general-purpose corpus, the only work that needs to be done when given the required relations is to manually represent them as template hypotheses. For instance, to represent the relation *"X was born in Y"*, we can write the aforementioned template hypothesis *"{var1} was born in {var2}"*. This hypothesis is actually no more than an example of a natural language representation of the relation. So when given relations, all we need to do is write "example" sentences for them. Thus, the system is easily operable with any set of user-given relations, requiring very little effort.

In order to experiment with our approach to IE, we found a convenient test bed in the TAC 2012 KBP challenge. TAC introduced the Knowledge Base Population (KBP) challenge, promoting research in IE. This year a new task was presented – Cold Start – focusing on systems that incorporate both Entity Linking (determining whether two phrases in the text refer to the same real-world entity) and Slot Filling (extracting information regarding those entities, ac-

[1] Bar Ilan University Textual Entailment Engine: www.cs.biu.ac.il/~nlp/downloads/biutee

cording to a list of binary predicates given in advance). Participants were provided with a corpus of roughly 27,000 documents extracted from the web, and a predefined list of predicates. For the purpose of our experiments we put most of our emphasis on the slot filling part (using the paradigm described above), and handled entity linking in a rather shallow manner.

Since this is a preliminary work, and our system's current ability to complete a slot filling task is very limited (as detailed in section 5), this report mostly deals with the general concept, rather than final results. Using the significant experience we gained while participating in the challenge, we are continuing to develop our methodology much further, and could hopefully report meaningful results in the future.

This report is organized as follows: Section 2 provides an overview of textual entailment and our approach for implementing it, manifested in the BIUTEE system. Section 3 describes our extension of the textual entailment problem formulation, which introduces variables to the hypothesis. Section 4 explains how a TE engine with variables can be used to solve a generic slot-filling problem. Section 5 details the specific implementation of the system we built for the Cold Start challenge, based on a variant of BIUTEE. Section 6 describes some directions for future work.

2 Background

In various NLP settings it is required to identify that a certain semantic inference relation holds between two pieces of text. For instance, in passage retrieval for question answering, it is needed to mark text passages as candidates from which a satisfying answer can be inferred. In paraphrase recognition it is necessary to identify that the meanings of two text fragments are roughly equivalent. And with more relevance to our current case - in information extraction (IE), a system may be given a template with variables (e.g., "X was born in Y") and has to find text fragments from which this template, with variables replaced by instantiations from the text, can be inferred.

A generic formulation for the inference relation between two texts is given by the Recognizing Textual Entailment (RTE) paradigm (Dagan et al., 2005). In this setting, a system is given two natural language text fragments, termed "text" (T) and "hypothesis" (H), and has to recognize whether the hypothesis is entailed by (inferred from) the text.

One of the main approaches for recognizing such textual inferences is to explicitly transform T into H, using a sequence of transformations (Bar-Haim et al., 2007; Harmeling, 2009; Mehdad, 2009; Wang and Manning, 2010; Heilman and Smith, 2010; Stern and Dagan, 2011), . This approach has been the basis to our open-source textual entailment recognition system, BIUTEE, described in detail in (Stern and Dagan, 2011). This system provides state-of-the-art linguistic analysis tools and exploits various types of manually built and automatically acquired knowledge resources, including lexical, lexical-syntactic and syntactic inference rules. The system utilizes parse-based representations of the text and hypothesis, where the parse-tree of H is explicitly generated from that of T by applying a sequence of tree transformation operations. This sequence is called a proof. Some of the transformations may utilize the knowledge resources, following (Bar-Haim et al., 2007), while others are on-the-fly operations (such as inserting a new node or moving a sub-tree to a different location in the tree) that complement the proof in cases of some inevitably missing knowledge.

(Bar-Haim et al., 2007) also suggest using **template hypotheses**, which are similar to our hypotheses with variables. However, they implemented only a special case (H is a predicate word with 2 argument slots, and T's are not given but rather searched in a corpus). Additionally, they tested it only on a small number of verbs, and didn't pursue it later. We want to make it part of the RTE paradigm, for T-H pairs, and make it part of a standard RTE system.

3 Textual entailment with Variables

In the original formulation of the TE problem, the **input** is a pair of natural language sentences T and H, and the **output** is a binary classification – either T entails H, or not[2]. We suggest a novel formulation, in which the input H may contain **variables**, and the output is all possible assignments to the variables, extracted from T, such that T entails H with the assignments. As an example, consider the following text:

$$T = \textit{"Isaac is the son of Abraham"}$$

The original TE formulation allows us to check this text against hypotheses such as:

$$H1 = \textit{"Isaac is Jacob's son"}$$
$$H2 = \textit{"Isaac is Abraham's son"}$$
$$H3 = \textit{"Jacob is Abraham's son"}$$

[2]There are variants to this formulation, for example, replacing the binary output with a probability that T entails H, or allowing ternary output – entailment, un-relatedness and contradiction. However, the input still consists of two fully-specified texts.

A perfect TE system, based on the original formulation, would return "true" for the pair (T,H2), and "false" for the pairs (T,H1) and (T,H3).

Our new TE formulation allows us to check this text against a single hypothesis with two variables:

Hv = "{var1} is {var2}'s son"

This hypothesis represents an infinite set of hypotheses, parameterized by the variables "{var1}" and {var2}". H1, H2 and H3 are members of this infinite set.

A TE system, based on our new TE problem formulation, should return a set with a single assignment:

{"{var1}=Isaac, {var2}=Abraham"}

The returned set may contain more than one element. For example, if:

T = "Isaac is the son of Abraham and Sarah"

there are two possible assignments:

{"{var1}=Isaac, {var2}=Abraham", "{var1}=Isaac, {var2}=Sarah"}.

The returned set may also be empty, which means that T does not entail Hv under any assignment, e.g., when:

T="Isaac lived in Israel"

Our new TE problem formulation has potential applications in many extraction-based tasks, for example: spoken language understanding, question answering, open information extraction and slot filling. In this report we focus on its application for slot filling.

3.1 Typed variables

A hypothesis variable may have a **type**, which constrains the possible assignments to this variable. In general, a variable type can be defined by any boolean function. Several examples are:

- Surface-form types, such as {number} or {date}. These can usually be computed using regular expressions.
- Part-of-speech types, such as {noun} or {verb}. These are usually computed using a POS tagger.
- Named-entity types, such as {person} or {organization}. These are usually computed using a NER system.
- Ontology-related types. For example, {hyponym:engineer} may match any word that is a hyponym of "engineer", such as "programmer", "metallurgist", etc.[3].

In the example of the previous section, it makes sense to define {var1} and {var2} to be of type "person". In other cases, a hypothesis may contain variables of different types, for example:

Hv = "{person} was born {number} years ago."

3.2 Extending BiuTee to support hypothesis variables

BiuTee's analysis of a T-H pair can be divided into 3 steps (Stern and Dagan, 2012):

1. **Preprocessing** – converting T and H into syntactic dependency trees. This step uses standard linguistic tools such as a sentence splitter, tokenizer, POS tagger, and a syntactic parser..
2. **Proof generation** – finding a set of transformations that starts with T's tree, uses linguistically-based transformations over dependency trees, and ends with a tree that totally covers H's tree. Each transformation has a cost, and the engine looks for the transformation sequence (a.k.a. a "proof") with the lowest cost.
3. **Response generation** - The cost of that best proof is compared to a pre-determined threshold (usually found during training), and BiuTee returns a positive "entailment" decision if and only if the cost is lower than that threshold.

To make BiuTee support hypotheses with variables, we had to address these 3 steps:

1. The linguistic tools we use in the **preprocessing** step work only with natural language sentences; they cannot handle sentences with variables. These are all third-party tools, and we do not want to change them, so we devised a method that allows us to use the existing preprocessing stage as a black-box. For each type of variable that we want to support, we prepare in advance a set of "examples" – words of that type. Given Hv, a hypothesis with variables, we preprocess it in the following way:

- Replace each variable in Hv with a distinct example according to the variable's type.
- Send the resulting sentence, which is now a legal natural language sentence, to the standard BiuTee preprocessor. The result is a syntactic dependency tree.

[3] These specific examples were found in the WordNet ontology

- Traverse the dependency tree, find all instances of the words we used as examples, and replace them with the corresponding variables.

This workaround will fail when the hypothesis Hv contains some of the example words besides the variables. To reduce the probability of this failure, we use as example rare words, for example, for the "{noun}" type, one of our examples was "conciliabule".[4] Such words may still appear in the hypothesis, for example, "The conciliabule was held in a {noun}". We leave this improbable case for future work.

2. For the **proof generation** step, we had to extend BiuTee's set of transformations. We added transformations from nodes in the text tree to nodes in the hypothesis tree that contain variables. As we mentioned in the previous section, each variable type is defined by a computable binary function. It is straight-forward to use this function in order to create a transformation. For example, for the {noun} type, we created a transformation of the form "If the text-node is tagged with POS=NOUN, transform it to a hypothesis-node whose lemma is {noun}".

3. For the **response generation** step, in case the cost of the best proof is above the threshold, we return an empty set of assignments, as there is no entailment. In case the cost is below the threshold, we need to recognize the assignments to the variables. To get the assignment of a variable, we need to start from that variable's node in the final tree, and trace back to the node in the original text tree, which was transformed to that variable node. Fortunately, BiuTee already keeps these traces, making it straight-forward to extract the variable assignments from the best proof.

For example, if the hypothesis contains a "{noun}" variable, and the text contains a noun, then the proof-finder will, eventually, use a transformation that converts the noun in the text to "{noun}". It will record a link to the original text node from the resulting "{noun}" node. After the proof is complete, if the engine decides that there is entailment we go to that "{noun}" node, follow the link to the original text node, and retrieve the original noun.

Unfortunately, currently BiuTee returns only a single best proof, so we can get at most a single assignment. We plan to handle sentences with two or more assignments in future work.

[4] Meaning "secret meeting of conspirators". Taken from http://phrontistery.info

4 TE with variables for Slot Filling

The slot filling task is defined by a closed set of predicates. Each predicate is described informally in the task definition. To solve a slot-filling task using a TE system, we should create, for each such predicate, a small set of hypotheses with variables. For example, for the "parent-of" predicate, we can use the following hypotheses:

{person1} is the parent of {person2}
{person2} is the child of {person1}

The hypotheses can be created manually, based on the description of the predicate in the task definition. This requires only little effort – less than writing the description itself. Besides the creation of the hypotheses, no other predicate-specific action is needed. Particularly, it is not required to train the system for each specific predicate.

Theoretically, a single hypothesis should suffice, since a perfect TE engine could decide that the other hypotheses entail it. However, current TE engines are not perfect, so we assist them by supplying several different hypotheses.

A similar scheme was used by (Bentivogli et al., 2010) and (Bentivogli et al., 2011) in the KBP validation tasks of RTE-6 and RTE-7.

Given a text sentence T, and a set of hypotheses with variables {Hv1, Hv2,... }, we can run our TE engine on the pairs (T,Hv1), (T,Hv2), etc. For each pair, we get a set of zero or more variable assignments, and use them to create an instantiation of the corresponding predicate, to fill the target knowledge base.

5 System Architecture

This section details our implementation of the system we used in the Cold Start challenge. Its core is our BiuTee-variant that implements textual entailment with variables as detailed above.

First, we manually created hypotheses (with variables) and specified a mapping between them and the required predicates. As mentioned earlier, we could specify several hypotheses per one predicate, to better capture its meaning.

Second, we trained the BiuTee-variant with a general purpose training set (we used the RTE6 Development Set). This is a weakness of the current implementation, that is subject to future work, as detailed in Section 6.

After that came the major step of the system – processing each sentence in the corpus. First of all the sentences were filtered. Our observation was

that sentences that were too short were usually non-meaningful (such sentences consisted a major part of the given corpus), so as a heuristic they were filtered out. Additionally, sentences that were too long required a substantial amount of time to process (disproportionate to their potential value), so as a second heuristic they were filtered out as well. Since recognizing textual entailment is quite a heavy task, further filtering was required. We performed a shallow filtering, requiring that a candidate sentence would include words appearing in the hypothesis, or any lexical expansion of them. The expansion was based on the same knowledge resources used in our BIUTEE-variant, in our case – WordNet(Fellbaum, 1998) and CatVar(Habash and Dorr, 2003). This is a common step in information extraction, when some form of shallow filtering needs to be used to handle massive amounts of text.

After having a much smaller set of candidate sentences, we performed two operations on each candidate: (1) We applied Stanford's Named Entity Recognizer on the sentence, to get all the entities and their types (recognizing an entity's type was one of the challenge's requirements). (2) The core process – we provided the candidate sentence to our BIUTEE-variant as Text, and our manually defined hypotheses as Hypotheses (one hypothesis each time). The result of this process was a collection of all predicates that adhere to hypotheses that were entailed, linked with their respective slot instantiations (or an empty collection if none of the hypotheses were entailed by the candidate sentence). All extractions were written to the task's submission file.

The very final stage was Entity Linking. Since this issue was required in the challenge, yet is not in the focus of our research, we utilized few simple manually built rules and heuristics (based only on the strings of the entities' names) to decide whether any pair of found entities is likely to represent the same real-world entity (checking for alternate spellings, abbreviations, names with common roots, etc.).

Since the system's run is quite slow, it had to be limited in various ways in order to handle the entire corpus. The most significant limitation was making the sentence-length-based filtering quite aggressive, and having it filter out a substantial amount of the corpus. Unfortunately, that filtering alongside other limitations (like dropping coreference resolution and training with a general dataset that was not fitted for this task), led to unsatisfactory results. As mentioned earlier, we hope to report meaningful results in the future.

6 Future Work

Our system yielded a fair amount of extractions, yet there is much room for improvement. As we wrote earlier, the challenge was a jump start for our group into this new line of research regarding variables in textual entailment, and their use in various applications such as information extraction. There are several possible topics for further research:

Unrestricted arity - in addition to binary relations, handle unary relations, ternary relations, etc.

Training for variables - in this system BIUTEE was trained using a general-purpose training set. The goal is to develop a methodology to train BIUTEE with awareness to the slot-filling setting.

Extending variable types - developing more mechanisms for type definitions, such as the aforementioned hyponym definition, which would allow any variable instantiation that is a hyponym of a specified phrase.

Multiple assignments - currently, our system finds, at most, a single assignment to the variables in the hypothesis. For example, if T = "Isaac is the son of Abraham and Sarah", and Hv = "{person1} is the son of {person2}", our system will return either {"{person1}=Isaac, {person2}=Abraham"} or {"{person1}=Isaac, {person2}=Sarah"}, but not both. To handle such cases, BIUTEE should be able to return more than one proof from T to H.

References

Roy Bar-Haim, Ido Dagan, Iddo Greental, and Eyal Shnarch. 2007. Semantic Inference at the Lexical-Syntactic Level. In *AAAI*.

Luisa Bentivogli, Peter Clark, Ido Dagan, Hoa T. Dang, and Danilo Giampiccolo. 2010. The Sixth PASCAL Recognizing Textual Entailment Challenge. In *Proceedings of TAC*.

Luisa Bentivogli, Peter Clark, Ido Dagan, Hoa T. Dang, and Danilo Giampiccolo. 2011. The Seventh PASCAL Recognizing Textual Entailment Challenge. In *Proceedings of TAC*.

Ido Dagan and Oren Glickman. 2004. Probabilistic Textual Entailment: Generic Applied Modeling of Language Variability. In *Learning Methods for Text Understanding and Mining*, January.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL Recognising Textual Entailment Challenge. In *Proceedings of MLCW*.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. 2009. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering*, 15(Special Issue 04):i–xvii.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. The MIT Press, May.

Nizar Habash and Bonnie Dorr. 2003. A Categorial Variation Database for English. In *NAACL*.

Stefan Harmeling. 2009. Inferring textual entailment with a probabilistically sound calculus.

Michael Heilman and Noah A. Smith. 2010. Tree Edit Models for Recognizing Textual Entailments, Paraphrases, and Answers to Questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1011–1019, Los Angeles, California, June. Association for Computational Linguistics.

Yashar Mehdad. 2009. Automatic cost estimation for tree edit distance using particle swarm optimization. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 289–292, Stroudsburg, PA, USA. Association for Computational Linguistics.

Asher Stern and Ido Dagan. 2011. A Confidence Model for Syntactically-Motivated Entailment Proofs. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 455–462, Hissar, Bulgaria, September. RANLP 2011 Organising Committee.

Asher Stern and Ido Dagan. 2012. BIUTEE: A Modular Open-Source System for Recognizing Textual Entailment. In *Proceedings of the ACL 2012 System Demonstrations*, pages 73–78, Jeju Island, Korea, July. Association for Computational Linguistics.

Mengqiu Wang and Christopher D. Manning. 2010. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1164–1172, Stroudsburg, PA, USA. Association for Computational Linguistics.