Modeling Common Real-Word Relations Using Triples Extracted from n-Grams

Ruben Sipoš, Dunja Mladenić, Marko Grobelnik, and Janez Brank

Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
{ruben.sipos,dunja.mladenic,marko.grobelnik,janez.brank}@ijs.si

Abstract. In this paper, we present an approach providing generalized relations for automatic ontology building based on frequent word n-grams. Using publicly available Google *n*-grams as our data source we can extract relations in form of triples and compute generalized and more abstract models. We propose an algorithm for building abstractions of the extracted triples using WordNet as background knowledge. We also present a novel approach to triple extraction using heuristics, which achieves notably better results than deep parsing applied on *n*-grams. This allows us to represent information gathered from the web as a set of triples modeling the common and frequent relations expressed in natural language. Our results have potential for usage in different settings including providing for a knowledge base for reasoning or simply as statistical data useful in improving understanding of natural languages.

1 Introduction

Solving non-trivial problems using a computer can usually be augmented by including background knowledge. When humans try to solve complex problems they use a large pool of background knowledge attained during their life. However, computers do not have this knowledge nor is it yet available in a computer-friendly format. A possible solution is hiring an expert to construct an ontology that we can use when needed. On the other hand, increasing amounts of publicly available data (especially with the growth of web and conversion of legacy data into digital form) are unlocking new possibilities for automatic knowledge acquisition.

Our motivation is similar to that in [1]. We believe that currently available sources of real-world text are large enough and can be used to attain general world knowledge represented as triples (possibly connected into an ontology). This can then be used for various applications ranging from improvements in automatic text analysis to extending existing ontologies and reasoning over them (e.g., textual entailment [1]). The approach we are proposing in this paper consists of two major components: triple extraction and construction of generalized models of common relations. The advantages of the triple extraction approach described in this paper are efficiency (more than 100 times faster compared to full deep parsing) and quality of extracted triples when applied on text fragments. The second component is used for constructing models which describe a larger set of concrete triple instances with more general terms. This reduces the noise and creates informative triples which can be saved in a widely accepted format such as RDF[1] and used as additional knowledge in various applications.

Because our approach works with text fragments we used Google's *n*-grams[2] dataset for evaluating performance of triple extraction and model construction. This dataset is interesting due to its size (it contains a few billion *n*-grams) and its origin (it was computed from Google's web index). New challenges posed by this dataset required the development of new approaches that can deal with highly noisy data and can process *n*-grams instead of complete sentences.

Related Work. There exists a plethora of related work that deals with similar topics. However, in all cases their research focuses on a more specific problem, only deals with some of challenges presented here or presents possible alternatives.

Triple extraction can be approached from various angels. The most common and intuitive is by using full deep parsing and constructing triples from parse trees or similar structures. Such approaches include [2] (which we used in our comparison in section 4.2) and [3, 4, 5]. The main problem with using Stanford Parser, Minipar and similar tools in our case is that we deal with text fragments and not complete sentences as those tools expect. Moreover, those approaches are too slow and can not be used to process large datasets such as Google's *n*-grams. Some related work [6, 7] uses different sets of handcrafted rules to extract triples. Nevertheless, those rules are different from our patterns because they are more specific and work with different underlying structures. There are also some other methods which take an entirely different approach to triple extraction, for example [8] (using kernel methods, can only extract relations it was trained on), [9] (using statistics about frequently occurring nouns near verbs marked by a POS tagger) and [10] (extracting only subject and predicate and adding possible objects later).

In many of the related papers we can find (similar) approaches to normalization. In [6, 10] they analyze noun phrases to find head words. In [11] they remove attributes (i.e. they keep only head words) and lemmatize words to base forms. We do all that; mark head words, treat attributes separately and lemmatize all words.

We can also find various approaches for generalizing extracted relations. Some of them are graph-based [12, 13, 14] while others define generalization more in the form of textual entailment [1, 11]. Compared to our work, they are meant for different settings than ours and can not be used in our case. One similarity that does exist is the use of WordNet (or some other domain specific ontology) for finding concept abstractions. Another possible approach, described in [17], tries to find generalizations in a way similar to learning selectional restrictions of predicates.

Contributions. The main contributions of this paper are the following:

- We present a scalable approach that can be used even with very large datasets such as Google's *n*-grams.
- Our method for triple extraction is more than 100 times faster than methods which use deep parsing. Moreover, the proposed method gives vastly better results when used on text fragments.

[1] http://www.w3.org/RDF/

[2] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13

- We can construct general models that describe a set of concrete triple instances and thus represent common knowledge implicitly contained in text.
- Evaluation of our results showed that the constructed models are meaningful. We can expect them to be useful for automatic ontology construction and as additional knowledge for various reasoners.

The paper is structured as follows. We first present our novel approach to triple extraction from text segments that does not require deep parsing. Next we describe an approach for constructing models which describe relations represented as triples in a more general way. We experimentally compare our approach to triple extraction to a commonly used existing approach. We also propose a methodology for evaluating generated models and apply it to obtain quality of models constructed from triples. We conclude with a discussion of the results, possibilities for their use and ideas for future work.

2 Triple Extraction

In this section we will present the proposed approach to triple extraction used in our pipeline. Main design goals were scalability and ability to process text fragments. For example, this allows us to process large datasets based on *n*-grams such as Google's n-grams used in our evaluation.

Text fragments represented as word *n*-grams differ from sentences in some important properties. We have to consider how they were created if we want to adapt existing approaches for triple extraction to work with them. Text fragments as addressed in this paper never contain words from two different sentences. Our (very basic) filter removes all text fragments containing punctuation marks before the triple extraction step. In this way we ensure alignment of text fragments inside sentence and avoid difficult (and usually less useful) cases which contain words from several sentences or clauses if they are separated with a punctuation mark. Secondly, even short text fragments are enough to cover a lot of possible simple sentences. In the best case scenario text fragments will still resemble sentences although maybe cropped at the beginning and end. Such text fragments can be viewed as simple sentences consisting of a subject, predicate and object. This limits useable text fragments to those consisting of at least three words. In the experiments presented here, text fragments consist of at most five words, therefore we can extract at most one subject-predicate-object triple and two attributes.

The main motivation for developing a new method, instead of using deep parsing, was the time complexity of deep parsing and specificities of text fragments in comparison to complete sentences. Using deep parsing as a basis for triple extraction even on a lot smaller datasets than ours might be unfeasible due to time constraints. We wanted a significantly faster approach than the one using deep parsing even if that meant possibly lowering the quality of extracted triples. Moreover, approaches based on deep parsing usually expect complete sentences. Training sets for parsers such as Stanford Parser[3] and OpenNLP[4] consist of annotated sentences. However, we are dealing with text fragments and therefore might get worse results than expected.

[3] http://nlp.stanford.edu/
[4] http://opennlp.sourceforge.net/

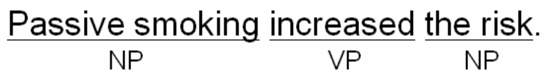

Fig. 1. Text fragment matching pattern noun phrase (NP), verb phrase (VP), noun phrase (NP)

Table 1. Patterns used for triple extraction listed by priority

Patterns represented as sequences of phrase types
NP, VP, NP
NP, VP, PP, NP
NP, ADVP, VP, NP
NP, ADVP, VP, PP, NP
NP, ADJP, VP, NP
NP, ADJP, VP, PP, NP
NP, PP, VP, NP
NP, PP, VP, PP, NP
NP, VP, ADVP, NP
NP, VP, PRT, NP

Our idea for an alternative approach to deep parsing is simple. Full deep parsing is slow; therefore we try to extract triples using only information provided by POS tags and chunking into phrases. This way we can avoid the most time consuming step – building a parse tree. Even though we have incomplete information we can still find the subject, predicate, object and their attributes.

We manually constructed patterns for triple extraction based on a sample of text fragments with added part of speech tags (POS tags) [15]. We incrementally expanded our set of patterns by tagging a set of text fragments with SharpNLP[5], manually noting the patterns, removing already covered examples and repeating this a few times. It turns out that even a very small sample (less than a hundred text fragments) already contains almost all patterns we found.

Our patterns describe the subsequence of phrases. The simplest pattern is: noun phrase (NP), verb phrase (VP), noun phrase (NP). A text fragment matching this pattern is shown in Figure 1. Constructing a triple is trivial: the first NP chunk represents the subject, the VP chunk is the predicate and the second NP chunk is the object. All patterns are listed in Table 1. A pattern does not have to match the whole text fragment, but it has to occur in it as a subsequence. For example, the pattern NP, VP, NP matches the text fragment consisting of NP, NP, VP, NP but not NP, PP, VP, NP. To resolve conflicts if multiple matches are possible we assign each pattern a priority.

After deciding on the subject, predicate and object, we have to choose the main word in each part. Because we did not compute the parse tree we rely on the POS tags. We decided to select the first word with an appropriate POS tag in each part as a main word. For example, word marked with a DT POS tag is certainly not the main word. Set of POS tags used for selecting main words was selected based on the descriptions of those tags.

[5] http://www.codeplex.com/sharpnlp

3 Modeling Triples

Our method for modeling triples consists of two phases. During the first phase we use machine learning to determine concept coverage. This is necessary due to some anomalies in our background knowledge. Models can still be constructed even if we skip this phase, but their quality might drop. The second phase uses information from the first and constructs models describing input triples.

In the previous chapter we describe an approach to extracting triples from text fragments. In order to construct more general models we want to abstract triples and find abstractions with the largest set of supporting instances. We used background knowledge which provides us with hypernymy relations and therefore enables concept abstraction. Then we extended this to triples and developed a method for finding the most interesting abstractions.

3.1 Background Knowledge

In order to construct our models we need a way to generalize concepts. One possibility is to use a hypernym. The desired level of abstraction and number of steps we have to traverse in the concept hierarchy depends on the specific goals we want to achieve.

We are using WordNet[6] as our background knowledge. More specifically, we are using hypernymy relations in WordNet for generalization of concepts. However, the tree representing hypernymy relations in WordNet has different levels of detail in different branches. This is mainly due to the fact that different parts of WordNet were constructed by different people. For some topics there were domain experts available, for others maybe not. This resulted in different levels of detail in different parts of WordNet. This means that the same number of steps taken while generalizing some concept might not yield the same amount of generalization if it happens in different parts of WordNet. Therefore, we have to find a way to compensate for this and normalize hypernymy tree to the same amount of generalization per one step in all parts of the tree.

Let us suppose that the words in the input text fragments have a nice and intuitive distribution. If we assign words as instances to possible concepts in WordNet hypernymy tree we expect that less detailed branches will have higher density of assigned instances. To compute the density of instances in different parts of hierarchy we use an unsupervised learning method. Every triple is represented as a bag-of-words. In comparison with using whole text fragments, we use only main words and leave out all additional attributes which might add noise. Additionally, we expand each bag-of-words with related words. Specifically, we add words which are hypernyms of the words from the triple. We limit ourselves to 1-2 steps of generalization using WordNet; alternative solutions are left for future work. This expansion increases the overlap between examples and reduces sparsity. In particular, this introduces some overlap between examples containing related words.

For unsupervised learning we used the *k*-means clustering method due to its simplicity and reasonable speed. The distance measure was cosine similarity, as commonly used for measuring similarity on text data. Computed centroids are used for

[6] http://wordnet.princeton.edu/

determining coverage of concepts in WordNet. We can assume that the majority of cluster mass exists near the centroid and ignore the examples farther off. Next, we can map areas of high density from bag-of-words space onto nodes in WordNet hierarchy.

Instance density information can help us determine the right amount of steps required when generalizing some concept using WordNet. We believe that in areas of lower density we have to make more generalization steps to achieve the same level of abstraction in comparison to more densely covered parts of hierarchy. On the other hand, when dealing with concepts from more densely covered parts of hierarchy we have to be more conservative because those concepts are more widely used and can easily cover larger amount of concrete instances.

An alternative approach to this would be using just the number of instances in a node to determine density in that part. However, we hope to achieve some improvement over that by using clustering and thereby being more robust when dealing with high level of noise (as is the case in our dataset).

3.2 Constructing Models

Some of the findings discussed in Section 3.1 can be used for generalizing words where we strive to select still interesting but more abstract concepts. Now we have to extend this to work with triples consisting of three main words. Naïve approach that tries to generalize all three words independently soon faces too big combinatorial complexity if applied to large datasets.

We can expect high combinatorial complexity due to ambiguity of mapping between words and synsets in WordNet. Sense disambiguation uses context to select the correct synset. However, our context is very limited and probably insufficient for acceptable quality of sense disambiguation. While humans can in most cases still identify the correct sense even when given only a triple with no or few additional attributes, it still is impossible to do using only computers. We concluded that we can not expect to solve this with any simple approach and will have to deal with ambiguities.

The method we are proposing for constructing model can be considered as unsupervised learning. To deal with the large amount of noise present in our data we largely rely on redundancy and reliable statistic information due to large amount of data.

We can not consider each part of a triple separately because of the time and space constraints; we have to find a more efficient solution. Besides, we have to check the majority of possible models with brute-force method because we do not have a good guide for choosing generalizations and appropriate models yet. We define word depth as the number of steps required to reach the tree root from that word in the WordNet hierarchy. We also assume that main word depths in a triple are usually similar because we believe that examples such as "Anna is drawing abstract entity" are rare. Searching for appropriate models is done in multiple passes; each pass at a specific triple depth, which is defined as a sum of main word depths. This defines the order of exploring of model space. To get a triple with a given triple depth we have to generalize main words in this triple to an appropriate word depth using WordNet.

The inability to disambiguate senses is an unavoidable constraint. Our approach uses a trivial solution to this. Every triple is converted to a set of triples where each one represents a possible combination of senses based on words from source triple. However, this drastically increases the amount of data we have to process. One possible solution would be using only the most frequently occurring senses and ignoring others. The current approach does not filter senses in any way because we wanted to construct a baseline first so we can compare results and measure effects of not using all possible word senses. Another possible solution described in [18] would be to use some of the additional information contained in extracted relations to disambiguate senses as some word senses occur more frequently in some relations than other.

WordNet contains only lemmatized word forms. That is why our input data requires lemmatization before we can use it. Simple rule based stemmers, such as Porter Stemmer, are useful for example when using bag-of-words representation of text in text mining [16]. We are using our own lemmatizer to improve the chances of finding a matching concept in WordNet for a given word. This lemmatizer is based on machine learning from correctly annotated examples. Experiments have shown that the use of lemmatizer is essential in achieving acceptable level of matching between words in triples and WordNet concepts.

We remove triples that contain stop-words. This applies only to main words and not additional attributes as we use only main words for constructing models. Also, we did not include some common stop-words in our list. For example, words such as "he" or "is" can be considered as stop-words but we keep them because they are building blocks for many informative triples. Still, we have to remove at least some very frequent words, otherwise we get a lot of useless abstractions such as "it is this" as the models with the highest support.

The algorithm given below describes the whole process of model construction we discussed in this section. The *BuildModels()* function iteratively explores possible candidates for models and keeps only the most promising ones. The *AbstractTriple()*, which is called from *BuildModels()*, is responsible for generalizing triples to a desired triple depth.

```
function AbstractTriple(wordSenses[3], desiredDepth[3])
//generalize all three parts of a given triple
newTriple[3] = new()
for i=0..2
  sense = wordSenses[i]
  depth = desiredDepth[i]
  //look up density information
  //for concepts in WordNet
  //that we previously computed
  density = GetDensity(sense)
  //set the max number of steps we can make
  if density < DENSITY_TRESH
    maxSteps = LOW_DENSITY_MAXSTEPS;
  else
    maxSteps = HIGH_DENSITY_MAXSTEPS;
  //check the constraints
  currentDepth = WordNet.getDepth(sense)
  if |depth - currentDepth| <= maxSteps
    newTriple[i] = WordNet.getHyper(sense, depth)
return newTriple
```

```
function BuildModels()
//we process data in multiple passes
//depending on triple depth
for depthSum=MIN_DEPTH..MAX_DEPTH
  foreach (xd, yd, zd) where xd+yd+zd == depthSum
    //process all triples
    foreach triple in allTriples
      //consider all possible senses
      foreach si in WordNet.getSenses(triple[0])
        foreach sj in WordNet.getSenses(triple[1])
          foreach sk in WordNet.getSenses(triple[2])
            //generalize triple
            model = AbstractTriple({si, sj, sk}, {xd, yd, zd})
            //and add it to candidate list
            mHash[model] += triple;
  //keep only the best model candidates
  ApplyTreshold(mHash, MIN_SUPPORT)
```

The largest part of time complexity is due to inclusion of all possible senses because we can not disambiguate word senses with so limited context. Space requirements of this algorithm depend mostly on hash table which is storing all current model candidates. The possibilities of parallelization are limited because we have to synchronize or distribute hash tables storing model candidates. To reduce the number of model candidates we set a threshold on the lowest acceptable number of supporting instances. The selection of this value depends on the target application in which we will use our models.

In this section we described an algorithm for constructing a set of triples that models concrete input instances. The final output is relations represented as subject, predicate and object which can be used for example as background knowledge in different applications.

4 Evaluation

4.1 Dataset

Evaluation of our approach was done on Google's *n*-grams dataset. This dataset contains word *n*-grams of lengths 1 to 5 and their frequency counts. It was computed from their web search engine's index and was published on 6 DVDs (about 26 GB of compressed textual data). The dataset includes only *n*-grams from the English index (but we can still find a few *n*-grams from other languages due to impreciseness of the methods used for language detection). They captured data in January 2006.

Text fragments represented as word *n*-grams were computed from more than 1 trillion words of source text containing more than 95 billion sentences. The dataset includes more than 1 billion unique 5-grams. Still, only *n*-grams with frequency count over 40 are included. All words that occur less than 200 times are mapped to the special token <UNK>. We believe that this dataset is sufficiently large to contain enough redundancy and useful information so we can build meaningful models on top of it.

Before we extracted triples from those text fragments we applied some filtering. Because the published dataset was not processed in any way (except thresholding frequency counts) it still contained a lot of random junk. We decided to filter away all text fragments containing border between two sentences, non alphanumeric characters and other similar things that are probably useless and would only complicate triple extraction. Our filter was very simple and rule based. For example, about 1/3 of 5-grams remained (8.79 GB of uncompressed binary data) after the filtering step. If later on we found that we filtered away some useful text fragments, we could still add additional rules to filter and keep those text fragments for further processing.

4.2 Triple Extraction Comparison

In this subsection we will present a comparison of triple extraction from text fragments using deep parsing with our approach presented in section 2. The main advantage of our method is efficiency. Processing 393,507,635 text fragments (5-grams) took about 1.2 million seconds (14 days of cumulative CPU time on 2.2 GHz Opteron 875) and created 63,814,809 triples. The time required for a single triple is about 3 ms. Our approach is almost 100 times faster if compared to deep parsing, which needs almost a second for one sentence (0.3 s per sentence according to results in [2] and our own tests). For example, if we used deep parsing we would need one year of CPU time on a 4-core processor and less than a week with our approach.

The second big advantage of our approach is that it was designed to work with text fragments. Tools for constructing parse trees used complete sentences as their training set and perform subpar when used with text fragments. A possible solution would be to artificially create text fragments from a training corpus and use it for training a new parser. Text fragments can always be created from unstructured text (although with some loss of information) if we want to use our method instead of some other approach designed for complete sentences (although doing this would be suboptimal). However, it is almost impossible to use methods designed to work with full sentences if we have only text fragments as they rely on too many assumptions which do not hold when working with text fragments.

The main disadvantage of our approach is the fact that it was designed for simple and short text fragments. If we use it for triple extraction from a complex sentence it will perform worse than approaches using deep parsing because we take into account only the local structure of sentence. However, some preliminary attempts to remedy this showed promising results. With minimal changes to patterns we can achieve roughly the same level of quality and recall as by using deep parsing.

We analyzed a random sample of 100 text fragments (5-grams) and manually reviewed the cases in which neither our approach nor approach using deep parsing returns a triple. In 81 cases neither approach successfully extracted a triple. In most of them (79) the text fragment was not even remotely similar to a sentence and therefore it was impossible to extract a triple from it. Most of those useless text fragments consisted only of nouns (thus we cannot find a predicate) or they started with a verb (and were thus missing a subject). Some other reasons were: the text fragment content was just noise, the text fragment was a fragment of a question, the main word was cropped and only attributes remained etc.

In the second part of comparison we wanted to evaluate the quality of extracted triples. We manually reviewed triples extracted from a random sample of text fragments (5-grams). The results of this evaluation are given in Table 2.

Table 2. Results of comparison between deep parsing and our approach

Method	**Triple quality**	**No. of triples**	
Deep parsing	correct	33	52 %
	useful	4	6 %
	forced	18	29 %
	wrong	8	13 %
	total	**63**	
Our approach	correct	89	62 %
	useful	33	23 %
	wrong	22	15 %
	total	**144**	
Both	same	14	40 %
	similar	20	57 %
	different	1	3 %
	total	**35**	
	no extracted triple	828	

First, we evaluated both approaches separately. We defined quality of extracted triples as following:

– **correct:** The extracted triple correctly describes the content of the text fragment. If we had to extract a triple manually we would select the same one. Example: “discussion document | is | a sample”.
– **useful:** The extracted triple does not correctly describe the content of the text fragment. However, it still represents a meaningful relation and remains relatively similar in meaning to the original text fragment. Example: “little advice | to help | you”.
– **forced:** Sometimes the approach using deep parsing extracts a triple from a text fragment that contains only nouns. It marks one of the nouns as verb although it is obvious to the human that that word represented a noun in the original context. Example: “Pictures Random | Thoughts | Rants Recreation”.
– **wrong:** The extracted triple is wrong and useless. The source text fragment in these cases is usually still similar to a sentence but is otherwise meaningless. Example: “order | to understand | these files”.

Next, we reviewed the cases in which both approaches extract a triple and compared the returned triples. We defined the following categories:

– **same:** Returned triples are identical.
– **similar:** Triples differ only in attributes; all three main words are the same.
– **different:** Triples differ in one or more main words.

The results we obtained show that our approach is successful. Using it we can extract 122 triples (correct and useful) while deep parsing returns only 37 triples (correct and useful). Furthermore, our approach returns only 15% of wrong triples while the other one returns 41% wrong triples. We can see that almost all triples extracted using deep parsing are included in intersection. Our approach misses and does not extract only 2 of the triples we get by using deep parsing. The new approach we are proposing for triple extraction from text fragments turned out to be better for our purposes as it is faster and extracts more and better triples than approach using deep parsing.

4.3 Models Evaluation

There is no standardized benchmark yet for what we are presenting in this paper. Moreover, it is quite hard to construct synthetic data that simulates real world data with high enough accuracy because we do not know all the properties of our data yet. We decided to manually evaluate models constructed from a subset of text fragments.

We are proposing an evaluation method that is based on very specific criteria for evaluating results. The main guidelines for selecting criteria were possible future applications of our results for automatic text analysis, ontology building, extending knowledge bases and reasoning. The criteria we used for evaluating the constructed models are:

- Appropriate level of abstraction:
  * All three parts of a triple should be on approximately the same level of abstraction.
  * Abstract concepts are useless (e.g. “entity”).
  * Instance names are too specific. If we wanted concrete instances we could just skip generalization and directly use extracted triples.
  * Commonly used concepts are preferred over rare ones.
- Usefulness of information:
  * Trivial relations with pronouns are useless (e.g. “it is this”).
  * Relation has to be meaningful.
- Ease of interpretation:
  * We prefer non ambiguous concepts.
  * Metaphors etc. are usually too hard to use.

We have to evaluate results using the previously listed criteria but we should take into account all of them simultaneously and not separately. We manually evaluated the most promising models with the following marks:

+1	conforms to most of the criteria
0	only about one half of criteria are satisfied
-1	most of the criteria are not satisfied

Because we decided to compensate the lack of sense disambiguation by taking into account all the possible word senses we wanted to evaluate in what amount do the constructed models include the correct sense. Two human evaluators manually reviewed models and decided if they use the correct word sense when we can discover it from the context (using our human knowledge). Results obtained during evaluation of sample A are presented in Table 3.

Table 3. Correctness of selected concepts in models constructed from sample A

	Number of models		
rater	correct	partially correct	Wrong
I	31	28	11
II	35	18	17

We are presenting results of evaluation of models constructed from sample A in the Table 4. We used the criteria described above. Most (9) of the models marked with -1 contained the concept "abstract entity" and were therefore considered useless (but we could easily avoid that by simple filtering or limiting ourselves to models with higher triple depth).

Table 4. Quality of models constructed from sample A

	Number of models		
rater	good (+1)	borderline (0)	bad (-1)
I	44	19	3+9
II	47	13	6+9

Table 5. Quality of models constructed from sample B

	Number of models		
rater	good (+1)	borderline (0)	bad (-1)
I	11	5	2+3
II	12	6	2+3

We performed our manual evaluation with two independent raters so we could check inter-rater agreement and verify the quality of our instructions for evaluating results. From the results presented in tables 3, 4 and 5 we can see substantial agreement in all choices except the "partially correct" and "wrong" selection of concepts in models. This disagreement is probably due to the vague definition of "partially correct".

To analyze the impact of sample selection on results we evaluated models for three more samples. Table 5 gives results of evaluating sample B where we gave negative mark three times due to too abstract models (they contained "abstract entity"). Samples A' and A'' gave a lot worse results than those we got for samples A and B. Almost two thirds of models were marked with -1 mostly because they contained concept "abstract entity".

Samples A, A' and A'' included 1000 triples. For a bit more than 50% of triples we could find matching synsets in WordNet for all three main words. We did not use the remaining triples for constructing models because we could not generalize them using WordNet. All three samples consist of small sequential groups of triples randomly selected from input file (which is partially sorted). If we chose entirely random triple we probably would not get any similar triples (within so small sample) and could not find useful generalizations (because they would contain only top level concepts such as "entity"). This does coincide with results (not presented in this paper) that we got from entirely random samples.

Selection of sample B was done in a different way. It contains 5000 triples. We randomly chose a few subjects (from the ones described by triples) and then randomly chose some triples containing that subject. In this way we assured at least some overlap between triples for the same reasons as in samples A, A' and A".

We evaluated results only at triple depths of 5, 10, 15 and 20 to shorten time needed for manual evaluation. This should not have any significant impact on evaluation results as it turned out that there are a lot of same models at different triple depths due to small sizes of samples. Models in samples A and B had about the same quality at different triple depths. Similarly, models constructed from samples A' and A" were bad at all triple depths.

Important bit of information we have to present is how we chose the most promising models from all the candidates. Even when we use really small sample size we get a large amount of candidates in hash table during the run of algorithm. This number is about 20,000 to 4,000,000 for sample B and 10,000 to 1,500,000 for smaller samples A, A' and A". Different numbers of candidates are due to different sizes of model space at different triple depths. Because candidates include all possible models for all possible senses we have to select only the most promising ones at the end. We decided to select the best ones for each triple depth separately because then we can observe the impact of different triple depth on constructed models. For every model candidate we also computed the number of instances that support it and use this number as a guide when choosing the best models. In evaluation we included 5-10 most supported models from selected triple depths.

From the presented results of evaluation we can conclude that the best models adequately satisfy chosen criteria. Also, we expect even better results when we will construct and evaluate models from all input triples (instead of only small sample). Small samples can be biased when randomly chosen from such large dataset depending on which triples we include: meaningful or not, with similar meaning or not, containing very specific or very general words etc. Additionally, we found out that we should add additional weights for some concepts because they have a very large negative impact on quality of results. For example, the word "he" is in a lot of cases represented with the concept of Helium although we know that most of the text contains the word "he" as a pronoun.

5 Conclusions and Future Work

In this paper we presented an approach to constructing generalized models describing triples extracted from text fragments. One of the challenges we tackled was the size of the dataset we used. Google's *n*-grams dataset is one of the largest publicly available datasets describing real world text. A lot of work is required to develop and implement algorithms that are fast enough so we can process such amount of data in a reasonable time. One important property of the dataset we used is that the source data for it was taken from the web. Therefore, we can observe some significant differences when we compare our extracted triples to manually or semi-automatically constructed triples currently available to the public. One major difference it the level of noise present in our dataset and consequently in extracted triples.

We described an approach for efficient triple extraction from text fragments. Compared to full deep parsing we achieved a major speedup (about 100 times) and better results. This will allow us to extract triples from other interesting data sources (e.g. Wikipedia articles) that are too large to process using deep parsing. We also presented an approach for constructing generalized models describing similar triples. Evaluation showed that constructed models were meaningful and could be used in many interesting settings.

Our approach still has to be more thoroughly tested. We hope that in the near future we will be able to test it indirectly by measuring the performance of applications using our data. This will allow us to fine-tune the parameters. The current choice of parameters was mainly based on the limited knowledge we have about the future applications.

In the future we want to extend our approach to work with wider choice of input data. Our design allows for easy extension and modification of various parts. We can use a different ontology (currently we use WordNet) for generalizing concepts and therefore cover other more specific domains. Moreover, we can add new or change existing patterns for triple extraction and by doing so modify our approach to work with longer text fragments or even whole sentences. By changing the POS tagger and chunker we can extract triples and construct models for other natural languages, too.

Acknowledgements

This work was supported by the Slovenian Research Agency and the IST Programme of the EC under PASCAL2 (IST-NoE-216886) and ACTIVE (IST-2008-215040).

References

[1] Clark, P., Harrison, P.: Large-Scale Extraction and Use of Knowledge from Text. In: Proc. Fifth Int. Conf. on Knowledge Capture, KCap 2009 (2009)

[2] Rusu, D., Dali, L., Fortuna, B., Grobelnik, M., Mladenić, D.: Triplet Extraction from Sentences. In: Proceedings of the 10th International Multiconference Information Society - IS 2007, pp. 218–222 (2007)

[3] Specia, L., Baldassarre, C., Motta, E.: Relation Extraction for Semantic Intranet Annotations. Knowledge Media Institute (2006)

[4] Sahay, S., Li, B., Garcia, E.V., Agichtein, E., Ram, A.: Domain Ontology Construction from Biomedical Text, pp. 28–34. CSREA Press (2007)

[5] Fundel, K., Küffner, R., Zimmer, R.: RelEx - Relation extraction using dependency parse trees. Bioinformatics 23, 365–371 (2007)

[6] Etzioni, M., Cafarella, D., Downey, S., Kok, A.-M., Popescu, T., Shaked, S., Soderland, D.S.: Web-scale information extraction in knowitall (preliminary results), pp. 100–110. ACM, New York (2004)

[7] Banko, M., Cafarella, M.J., Soderland, S., Broadhead, M., Etzioni, O.: Open Information Extraction from the Web, pp. 2670–2676 (2007)

[8] Zelenko, D., Aone, C., Richardella, A.: Kernel Methods for Relation Extraction. Journal of Machine Learning Research 3, 1083–1106 (2003)

[9] Kavalec, M., Svatek, V., Buitelaar, P., Cimmiano, P., Magnini, B. (eds.): A Study on Automated Relation Labelling in Ontology Learning. IOS Press, Amsterdam (2005)

[10] Schutz, Buitelaar, P.: RelExt: A Tool for Relation Extraction from Text in Ontology Extension, pp. 593–606 (2005)

[11] Soderland, S., Mandhani, B.: Moving from Textual Relations to Ontologized Relations. In: Proceedings of the 2007 AAAI Spring Symposium on Machine Reading (2007)

[12] Trampuš, M., Mladenić, D.: Constructing Event Templates from Textual News. In: Workshop on: Intelligent Analysis and Processing of Web News Content (2009)

[13] Leskovec, J., Grobelnik, M., Milic-Frayling, N.: Learning Sub-structures of Document Semantic Graphs for Document Summarization. In: Workshop on Link Analysis and Group Detection (LinkKDD), KDD 2004, Seattle, USA, August 22-24 (2004)

[14] Rusu, D., Fortuna, B., Mladenić, D., Grobelnik, M., Sipoš, R.: Document Visualization Based on Semantic Graphs. In: IV 2009 (2009)

[15] Bies, A., Ferguson, M., Katz, K., Mac-Intyre, R.: Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania (1995)

[16] Grobelnik, M., Mladenić, D.: Text Mining Recipes. Springer, Heidelberg (2009), `http://www.textmining.net`

[17] Ciaramita, M., Gangemi, A., Ratsch, E., Saric, J., Rojas, I.: Unsupervised Learning of Semantic Relations between Concepts of a Molecular Biology Ontology. In: IJCAI 2005, pp. 659–664 (2005)

[18] Pennacchiotti, M., Pantel, P.: Ontologizing Semantic Relations. In: ACL 2006 (2006)