

Using patterns over ontology terms to detect inconsistencies in large text collections (extended abstract)

Christian Wartena¹, Anjo Anjewierden², and William van Dieten¹

¹ Telematica Instituut, Enschede, The Netherlands,

{Christian.Wartena,William.vanDieten}@telin.nl

² Universiteit van Amsterdam, The Netherlands, anjo@science.uva.nl

Abstract. Inconsistencies in customer communication becomes a real threat for organizations. We have investigated possibilities to detect possible inconsistencies in the content of a large pension fund. An important step in the solution we propose is finding all statements on a certain product property. In this paper we show how this can be done using patterns over ontology terms.

1 Introduction

Many organizations are faced with the problem of a growing amount of texts that have to be managed. One problem that arises is that inconsistencies can emerge more easily as the amount of texts grows and constantly changes. Inconsistencies can have far reaching consequences if an inconsistency implies faulty information in manuals or customer product information. But even harmless errors might create an impression of sloppiness and can cause harm to the reputation of an organization.

In a project with the *Algemeen Burgerlijk Pensioenfonds (ABP)*, the largest Dutch pension fund, we have investigated possibilities to support documentalists to automatically find inconsistencies. We have implemented a prototype of a tool that first classifies a text to find a set of documents with which the text should be consistent. In the second stage it searches patterns to find statements on certain properties of the products of ABP. The relevant fragments finally are presented to the user who has to decide whether they are consistent.

In this paper we will discuss how the search for relevant pieces of text can be done using pattern recognition. We start describing the specific situation of ABP in some more detail and sketch our overall solution. In section 3 we describe how patterns on ontology terms can be defined and used to find fragments that have to be checked for inconsistencies. In section 4 we compare the used techniques to related work. Finally, we discuss the results in the conclusion.

2 Inconsistencies

We have composed a corpus of about 900,000 words consisting of internally used product descriptions as well as public information, like web sites, leaflets and circular letters.

ABP has implemented a standard procedure for the definition of new products and the creation of all communication, system consequences and procedures that come with a product. Somewhat simplified we can say that first a detailed formal description of the product is made, called a *materieboek* (loosely: “book describing the matter”).

Subsequently, all software designs and communication is derived from the *materieboek*. For communication to customers the formal nature of a *materieboek* creates a source for inconsistencies. First, the meaning of a *materieboek* needs to be “translated” into natural language such that customers can understand it, and secondly because the different forms of communication (letters, website) are written by different teams they can be inconsistent as well. In some cases a *materieboek* is still subject to changes when the first letter has to be written. An example of an inconsistency is that one text says the claim to payment of a widow’s pension starts the day after the death of her husband and another text stating that the claim starts on the first day of the first month after his death. Possible types of inconsistencies are described in more detail in [3]. ABP currently has a procedural approach to finding inconsistencies: “groups of readers” check all communication manually.

In order to manage the growing amount of products, ABP has developed the notion of generalized product and generalized product concepts (GPC). The latter can be seen as parameters of the single generalized product. The GPCs, however, are not defined very formally. In the following we will say that two documents related to the same product are consistent if all statements on each GPC are consistent. Inconsistencies between documents related to different products thus don’t exist. Regrettably, the relation between documents and products is only made explicit for *materieboeken*. Moreover, many letters, web sites etc. relate to several products.

Meziane and Rezgui [4] describe a method to avoid inconsistencies in textual documents. The essence of their proposal is to find documents that are similar to a certain document that is changed, added or simply checked for inconsistencies. Whether the found documents are consistent or not has to be judged by humans.

Searching for similar documents in our corpus did not work very well for various reasons. We had more success classifying text into classes related to a small number of similar pension products. (The ideal of classes consisting of the texts related to one product could not be reached since many products are too similar.) In order to make it possible to compare a document with all other texts in the same class we try to find all statements on a certain GPC in a document and all other documents in the same class. A statement on a GPC is found by searching instances of a number of patterns over words and concepts that are defined for that GPC.

3 Patterns

This section describes the pattern language we have defined and implemented to be able to find phrases in documents. The pattern language has been implemented in an open source text analysis tool called tOKo [1]³. Design objectives of the pattern language were that it should be usable for non-technical users and that it provides primitives

³ See http://www.toko-sigmund.org/pattern_search.html for a full specification of the language.

for lexical, grammatical and “semantic” pattern elements. These design objectives are motivated by the primary use of the language: to find relatively short phrases in natural language documents (grammar) related to a certain domain (semantics as given by a user-defined ontology).

3.1 Specification

The language constructs referring to the content of documents are given below.

space A space matches any amount of white space.

integer The pattern # matches any integer.

word The pattern () matches any word.

literal A literal, syntactic, pattern is just written down as it is. The pattern **fruit** matches just that. And , matches a comma.

lemma A lemma is denoted by putting brackets around it: (**fruit**) matches all inflections of the natural language lemma for fruit and includes the plural.

wordclass Matches all terms that belong to the word class. For example, <**noun**> matches any inflection of any noun.

concept Wordclasses are one of the classifications linguists define for a set of terms.

In a particular domain, taxonomies of concepts can be used to create a customized classification. The user can, with the ontology editor part of tOKo, for example define that apple, banana and strawberry are all sub-concepts of fruit. The pattern [**fruit**] then refers to all possible lexical variations of the concept fruit and its sub-concepts and instances. Matching concepts to their occurrence in texts is achieved by allowing the user to specify lexical variants (synonyms, abbreviations, contractions, misspellings) and treating the label itself as if it were a (compound) lemma. [**fruit**] therefore matches both apple and apples, and perhaps even tomatoes.

The language elements make it possible to succinctly specify a wide variety of phrases in documents. Of particular interest is that the user can make the search results more specific by referring to concepts in the ontology she is developing. For example, the pattern <**verb**> <**article**> [**fruit**] matches activities (tasks) related to fruits, e.g. “peeled the apples”.

Given that the pattern language is intended to find relatively short phrases in natural language text there is little need for operators like the Kleene star. The language provides the following constructs to handle disjunctions, grouping and the notion of nearness:

disjunction The common | symbol is used for disjunctions: (**apple**)|(b**anana**).

grouping Curly brackets denote grouping: <**numeral**>{(b**anana**)} matches “two bananas”.

near A simple minded alternative for the Kleene star. The notation is ... which matches at most five tokens, the notation ...**n** (n is an integer) matches at most n tokens.

3.2 Representation and algorithms

This section provides a brief overview of the representation and algorithms used for pattern matching⁴.

⁴ A more detailed description will be in the final version of this paper.

The performance of any pattern matcher naturally depends on whether it succeeds in avoiding combinatorial explosions as much as possible. Compared to grammatical matchers, our pattern language has the matching of concepts as a potential bottleneck. Consider the following pattern **[colour] [fruit]**. When this pattern is expanded, assuming there are many colours and fruits in the ontology, hundreds or thousands of instantiated patterns must be matched.

tOKo represents a corpus as a set of documents and each document as an array of tokens (word, integer, space, other). Each token can be directly addressed by $\langle Doc, Index \rangle$, and $\langle Doc, Start, End \rangle$ is a consecutive sequence of tokens. The pattern matching algorithm consists of the following steps:

1. For all occurrences of non-trivial pattern elements a hash table is created of which the hash value is derived from $\langle Doc, Start \rangle$.
2. All occurrences of the first pattern element are stored in a set of tuples $\langle Doc, Start, End \rangle$.
3. For all tuples in the current set (initially the one in step 2), the next position, $\langle Doc, Start, End + 1 \rangle$, is looked up in the hash table of the current pattern element. If present, the tuple is replaced by $\langle Doc, Start, End_n \rangle$ (End_n is the last token of the current pattern element), otherwise there is no match and the tuple is removed from the set of potential matches.
4. Step 3 is repeated until the list of pattern elements is exhausted.

This algorithm has a complexity close to linear along the number of pattern elements (step 1). On current hardware, response is immediate (less than one second) on corpora up to 50Mb of text.

3.3 Examples

We have developed part of an ontology for the domain related to ABP pensions. This ontology is mainly a taxonomy consisting of 450 concepts, and more than 150 synonyms and abbreviations for these concepts. The concepts in this taxonomy can all be expressed as a single word or a compound term. As mentioned above this taxonomy was used to compute similarity between documents.

If we are searching for inconsistencies we are not only interested in terms representing a concept but also in all other passages referring to that concept (GPC). Moreover, we are interested in concepts that are more complex in the sense that they are expressed in terms of several simple concepts.

Take for example the concept *Betaalplichtige* (person who is responsible for payment). In the *materieboek* we find a table in which this term occurs. In a letter, on the contrary, we might find something like the following:

... bij uw werkgever of bij de instantie waarvan u uw uitkering krijgt. Die houdt de premie in op uw salaris of uitkering. (... to your employer or to the agency you get your benefit from. He/she deducts the contribution from your salary or benefit.)

In order to find fragments like this, we have defined a number of patterns for each GPC. For *Betaalplichtige* the following patterns have been defined:

(houden) . . . [Premie] in op . . . [Inkomen]		[Betaalplichtige]
[Premie] wordt ingehouden . . . [Inkomen]		[Inhouding] op [Inkomen]
[Premie] . . . komt voor rekening van . . . [Persoon]		[Inhouding] van [Premie]
{(u) [Rechtspersoon]} . . . betaalt . . . #		
{(u) [Persoon]} . . . (betalen) . . . [Premie]		
(betalen) . . . {(u) [Persoon]} . . . [Premie]		

The first pattern of this set matches the underlined part in the example above.

4 Related work

At first glance the techniques we have used to detect possible inconsistencies are very similar to information extraction (IE).

In traditional IE a common pattern element is the so-called named entity, such IE systems would define a class called company which would match all company names known to the system. In tOKo, a concept in an ontology can be used as a pattern element and this makes it possible to use inference rules to match concepts to the corpus and it also provides an opportunity to re-use available ontologies in RDF or OWL.

Kaiser and Miksch [2] define information extraction as *a technique used to detect relevant information in larger documents and present it in a structured format*. Thus, IE, consists of two tasks: detecting and locating specific pieces of information and interpreting and extracting the information. The first part (locating the information) is part of our approach as well. The second stage is not part of our approach, since we believe that fully automatic detection of inconsistencies in unstructured text is beyond the present possibilities.

As a first consequence our task of finding the right fragments is much easier than IE in general. Consider again the example discussed above. For IE we would have to analyze the internal structure of the match (the underlined part in the example) in order to find the correct relations between the mentioned entities. Given this match that would be very problematic. The object of **houdt . . . in** (deducts) is **salaris of uitkering** (salary or benefit), but only the first conjunct is matched. For IE this might result in an incorrect result. The subject of the sentence, **Die** (that one), is not matched at all, and even if it would match, we would need to do anaphora resolution in order to find out who is meant. Since we are only searching for statements about a certain concept, these shortcomings do not constitute a problem.

The second consequence is that the intermediate result of IE, the location of the relevant fragments is much more important. If a certain fact is mentioned three times in a text, for most IE tasks we will be happy if two of them are found and interpreted correctly. In our case, on the contrary we need every reference to a certain GPC to be able to establish that all statements are consistent.

5 Conclusions

As may have become clear from the discussion above we are interested in very high recall, while precision is less important. In our pilot we defined patterns for six GPCs. For these concepts we could indeed achieve a very high recall. As expected the precision was much less high since some patterns might match irrelevant sentences.

However, we expect that a number of concepts is much more problematic to match compared to the six we have implemented. Another problem is that our corpus consists of texts that all were produced in the course of one major change in the pension regulations at ABP. We did not yet check, how good the patterns are if we would consider a larger range of texts.

While tOKo is a general tool that allows researchers to experiment, it is not suited to demonstrate to end-users the possibilities a technique might offer. For this reason we have implemented a simple user interface that allows browsing the content and calls tOKo for classifying texts and searching patterns in the relevant class. The tool presents the text with almost original lay-out and all GPCs found in that text. If the user selects a GPC a list of text from the same class in which this GPC occurs as well is given. Now the user can compare the texts in which the phrases matching the selected GPC are highlighted. Given that the task of finding inconsistencies between documents is currently carried out manually by “reading” the documents, ABP feels that the technique, though still not perfect, represents a significant improvement.

Acknowledgements

This work was carried out within the *Grip op Content* project, which is financed by the Algemeen Burgerlijk Pensioenfonds (ABP) and the Telematica Instituut. We would like to thank Joop Knippenbergh and Wim Steskens from ABP and Robert Slagter and Wijnand Derks from the Telematica Instituut.

References

1. Anjo Anjewierden et al. tOKo and Sigmund: text analysis support for ontology development and social research. <http://www.toko-sigmund.org>, 2006.
2. Katharina Kaiser and Silvia Miksch. Information extraction - a survey. Technical Report Asgaard-TR-2005-6, Vienna University of Technology, Vienna, 2005.
3. Ares Kralendonk. Analyse van inconsistentie van contentproducten rondom het partnerplus-pensioen. Technical report, University of Twente, Enschede, 2005.
4. Farid Meziane and Yacine Rezgui. A document management methodology based on similarity contents. *Inf. Sci.*, 158:15–36, 2004.