SPEC - Sentence Pattern Extraction and Analysis Architecture

Michal Ptaszynski † Rafal Rzepka ‡ Kenji Araki ‡ Yoshio Momouchi §

† JSPS Research Fellow / High-Tech Research Center, Hokkai-Gakuen University
ptaszynski@hgu.jp

‡ Graduate School of Information Science and Technology, Hokkaido University
{kabura,araki}@media.eng.hokudai.ac.jp

§ Department of Electronics and Information Engineering,
Faculty of Engineering, Hokkai-Gakuen University
momouchi@eli.hokkai-s-u.ac.jp

Abstract

A "sentence pattern" in modern Natural Language Processing and Computational Linguistics is often considered as a subsequent string of words (n-grams). However, in many branches of linguistics, like Pragmatics or Corpus Linguistics, it has been noticed that simple n-gram patterns are not sufficient to reveal the whole sophistication of grammar patterns. We present a language independent architecture for extracting from sentences more sophisticated patterns than n-grams. In this architecture a "sentence pattern" is considered as n-element ordered combination of sentence elements (words). This paper presents general description of the architecture.

1 Introduction

Automated text analysis and classification is a usual task in Natural Language Processing (NLP). Some of the approaches to text (or document) classification include Bag Of Words (BOW) or n-gram. In the BOW model, a text or document is perceived as an unordered set of words. BOW thus disregards grammar and word order. An approach in which word order is retained is called the n-gram approach. This approach perceives a given sentence as a set of n-long ordered sub-sequences of words. This allows for matching the words while retaining the sentence word order. However, the n-gram approach allows only for a simple sequence matching, while disregarding the grammar structure of the sentence. Although instead of words one could represent a sentence in parts of speech (POS), or dependency structure, the matching using n-grams still does nto allow for matching more sophisticated patterns than the subsequent string of items. An example of a pattern more sophisticated than n-gram is presented in top part of Figure 1. A sentence in Japanese "*Kyō wa nante kimochi ii hi nanda !*" (What a pleasant day it is today!) contains a well known syntactic pattern "*nante * nanda !*"[1]. However, it is not possible to discover this subtle pattern using n-gram approach. Methods trying to go around this problem, include a set of machine learning (ML) techniques, such as Neural Networks (NN) or Support Vector Machines (SVM). Machine learning have proved its usefulness for NLP in text classification for different domains [3, 4]. However, there are several problems with the ML approach. Firstly, since machine learning is a self-organizing method, it disregards any linguistic analysis of data, which often makes detailed error analysis difficult. Moreover, the statistical analysis performed within ML is still based on words (although represented as vectors), which hinders dealing with word inflection and more sophisticated patterns such as the one mentioned above. Although there are attempts to deal with this problem, such as the string kernel method [5], in ML one always needs to know the initial feature set to feed the algorithm. Finally, methods for text classification are usually inapplicable in other tasks, such as language understanding and generation.

In our research we aimed to cerate an architecture capable to deal or help dealing with the above problems. The system presented in this paper, SPEC, is meant to extract from sentences patterns more sophisticated than n-grams, while preserving the word order. SPEC can work with one or more corpora written in any language, as long as the corpora are preprocessed (spaced, POS tagging, etc.). It extracts all frequent patterns, such as the one mentioned above. The patterns could be further used in direct document classification and as features for machine learning algorithms. Moreover, the patterns could be used as sentence templates in language generation tasks. This paper presents general description of the system, proposes some of the methods to evaluate SPEC performance and mentions several possible applications.

2 System Description

This section contains detailed description of SPEC, or *Sentence Pattern Extraction and analysis arChitecturte*. In the sections below we describe the system sub-

[1]equivalent of wh-exclamatives in English; see [2] for details.

procedures. This includes corpus preprocessing, generation of all possible patterns, extraction of frequent patterns, post-processing and further analysis. By ”corpus” we consider any collection of sentences or instances. It can be very large, containing hundreds of thousands of sentences (or more), or it can be rather small consisting of only several or several dozen sentences. In any case SPEC will extract sentence patterns distinguishable for the corpus. In the assumption, the larger and the more coherent the original corpus is, the more frequent patterns will be extracted.

2.1 Corpus Preprocessing

SPEC is capable to deal with any not preprocessed raw corpora, as long as the lexical form of the language consists of smaller distinguishable parts, like letters, or characters. This makes SPEC capable to deal with a corpus written in any type of language, including analytic languages (like English or Mandarin Chinese), agglutinative languages (like Japanese, Korean or Turkish), or even polysynthetic languages like Ainu, in both spaced and non-spaced form. However, in the Pattern Generation sub-procedure, SPEC creates a very large number of temporary patterns (all possible ordered combinations of sentence elements). Therefore, considering the time of processing, as a default we will assume that the corpus should be at least spaced. Other relevant, optional preprocessing might include part-of-speech (POS) and dependency relation tagging (or any other additional information as long as there exist a sufficient tool). Three examples of sentence preprocessing with and without POS tagging are presented in Table 1 for a sentence in Japanese[2]. The sentence in the example was spaced and tagged using MeCab [1], a standard POS tagger for Japanese.

2.2 Pattern Generation

Generation of All Combinations from Sentence Elements In this sub-procedure, the system generates ordered non-repeated combinations from the elements of the sentence. In every n-element sentence there is k-number of combination groups, such as $1 \leq k \leq n$, where k represents all k-element combinations being a subset of n. The number of combinations generated for one k-element group of combinations is calculated as binomial coefficient, as represented in equation 1. Moreover, in this procedure we create all combinations for all values of k from the range of $\{1, ..., n\}$. Therefore the number of all combinations is equal to the sum of all combinations from all k-element groups of combinations, like in the equation 2.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

[2]Japanese is a non-spaced agglutinative language.

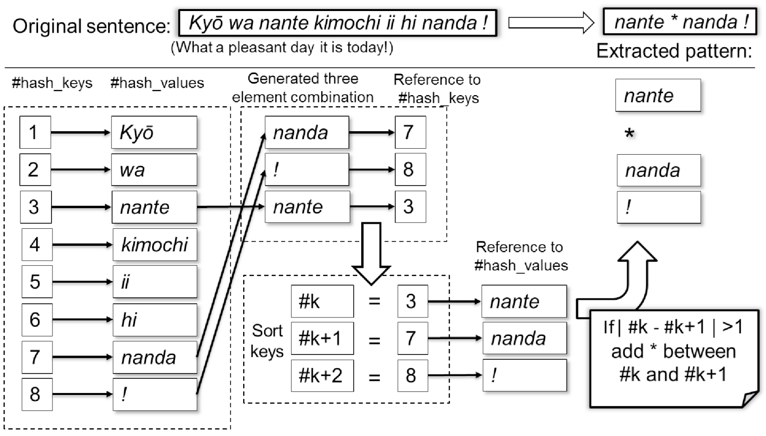
Table 1: Three examples of preprocessing of a sentence in Japanese with and without POS tagging; N = noun, TOP = topic marker, ADV = adverbial particle, ADJ = adjective, COP = copula, INT = interjection, EXCL = exclamat. mark.

Sentence:	今日はなんて気持ちいい日なんだ！
Transliteration:	*Kyōwanantekimochiiihinanda!*
Meaning:	Today TOP what pleasant day COP EXCL
Translation:	What a pleasant day it is today!
1. Words:	*Kyō wa nante kimochi ii hi nanda !*
2. POS:	N TOP ADV N ADJ N COP EXCL
3. Words+POS:	*Kyō*[N] *wa*[TOP] *nante*[ADV] *kimochi*[N] *ii*[ADJ] *hi*[N] *nanda*[COP] *!*[EXCL]

$$\sum_{k=1}^{n} \binom{n}{k} = \frac{n!}{1!(n-1)!} + \frac{n!}{2!(n-2)!} + ... + \frac{n!}{n!(n-n)!} \tag{2}$$

Ordering of Combinations In mathematics, combinations are groups of unordered elements. Therefore, using only the above formula, we would obtain patterns with randomized order of sentence elements, which would make further sentence querying impossible. To avoid randomization of sentence elements and retain the sentence order we needed to sort the elements after all combinations have been generated. To do that we used automatic generation of double hash maps. Firstly, all elements of the original sentence are assigned ordered numbers (1, 2, 3...). After a combination is generated, elements of this combination are re-assigned numbers corresponding to the numbers assigned to the original sentence elements. Then the new set of numbers is sorted. This provides the appropriate order of combination elements consistent with the order of elements in the original sentence.

Insertion of Wildcard On this stage the elements are sorted, however, to perform effective queries to a corpus we would also need to specify if the elements appear next to each other or whether they are separated by a distance. In practice, to solve this problem we need to place a wildcard between all non subsequent elements. We solved this using one simple heuristic rule. If absolute difference of hash keys assigned to the two subsequent elements of a combination is higher than 1 we add a wildcard between them. This way we obtain a set of ordered combinations of sentence elements with wildcards placed between non subsequent elements. Both parts of this procedure, the sorting of elements using automatically generated hash maps and the wildcard insertion, are represented in Figure 1. Finally, to prepare the set of all generated patterns to further processing, we perform a filtering of repeating patterns to leave only the original patterns.



Original sentence:
Kyō wa nante kimochi ii hi nanda !
(What a pleasant day it is today!)
nante * nanda !
Extracted pattern:
#hash_keys
#hash_values
Generated three element combination
Reference to #hash_keys
Reference to #hash_values
Sort keys
#k = 3 nante
#k+1 = 7 nanda
#k+2 = 8 !
If| #k - #k+1 | >1 add * between #k and #k+1

Figure 1: The procedure of sorting of combination elements using automatically generated hash maps.

2.3 Pattern Extraction and Frequency Calculation

In this sub procedure SPEC uses all original patterns generated in the previous procedure to extract patterns that appear in a given corpus and calculate their frequency. This represents a similar approach to the usual TF (term frequency) approach. However, since in our research we do not use only single terms, but sentence patterns, it is necessary to propose a new nomenclature, namely PF (pattern frequency)[3]. PF is calculated for all patterns found in the matched corpus.

Although the pattern frequency calculation could be performed as a part of the previous procedure (pattern generation), we made this a step a separate sub-procedure for several reasons. Firstly, SPEC is assumed to deal with one corpus as well as several different corpora, depending on the task it is used in. This way we needed to retain the ability to generate patterns from one (given) corpus and match them to another (target) corpus. This ability could be useful in such tasks as lexicon expansion (when looking for certain phrases appearing with the patterns), or text classification (when classifying sentences from the target corpus using a set of patterns from the given corpus). Separating pattern generation and pattern extraction procedures was also needed for cases of dealing with two or more corpora, when both are given and target corpora. Making pattern extraction a separate procedure allows us to perform the extraction on all corpora concurrently, e.g., using a $fork$ or $thread$ function of parallel programing. Eventually, this significantly shortens the time of processing. Finally, we aimed in making a module based expandable system in which different procedures would work as separate agents. This way each separate procedure could be thoroughly evaluated and improved when needed.

[3]However, we do not recommend using PF-IDF, since the number of patterns extractable from sentences is much greater than single terms. Perhaps a division not by all possible patterns but by all the same k-element patterns might show some relevancies. When proposing an approach substituting TF-IDF, it should be also taken into an account that in the range $\{1...n\}$ the number of patterns increases in the middle and decreases on the ends of the range.

2.4 Post-processing

In the post-processing phase SPEC performs simple analysis of patterns extracted from the given corpus/corpora to provide pragmatic specifications of the corpus. We use the term pragmatic specifications in a similar way to Burkhanov, who generally includes here indications and examples of usage [6]. The post-processing is done differently for: 1) only one given corpus and 2) a set of two corpora. In the future we also consider the analysis of more than two corpora.

One Corpus Case The post-processing of one corpus is done as follows. Firstly all patterns that appeared only once are filtered out and deleted. This is done to eliminate quasi-patterns. A quasi-pattern is a pattern, which was created from one sentence in the process of pattern generation, but was not found elsewhere in the rest of the corpus. In practice, it means that it is not a frequently used sentence pattern and therefore keeping it would bias the results. The patterns that appeared more than once are grouped according to the pattern length (number of elements). Within one group the patterns are also sorted decreasingly. In this way the patterns can be used in further analysis. The general pragmatic rule which applies here says that the longer the pattern is, and the more often it appears in the corpus, the more it is specific and representative for the corpus. As an example of application of this post-processing procedure, we could mention such NLP tasks as topic detection/extraction or identifying individual user style characteristics (when the analyzed corpus consists of single user messages).

Two Corpora Case In many NLP tasks, especially those taking advantage of machine learning methods, it is often necessary to obtain lists of two distinctive sets of features [7]. Such tasks include all sorts of text classification, including spam filtering, sentiment and affect analysis [8], or cyber-bullying detection [4].

The post-processing of two corpora is done as follows. Firstly SPEC deletes only those quasi-patterns that ap-

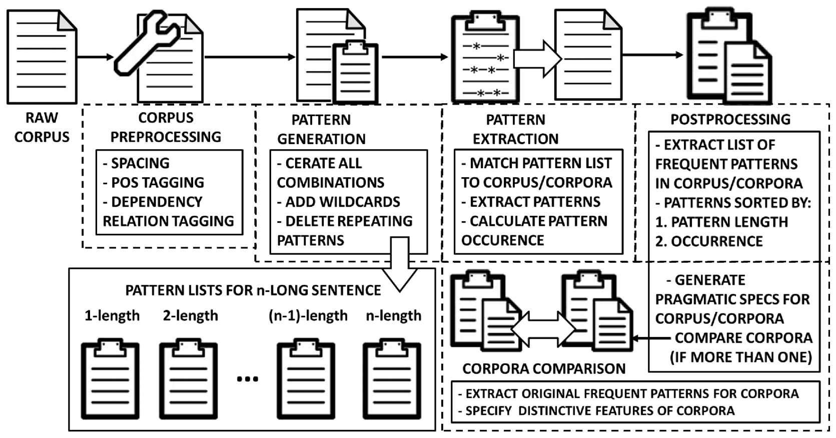

RAW CORPUS
CORPUS PREPROCESSING
- SPACING
- POS TAGGING
- DEPENDENCY RELATION TAGGING
PATTERN GENERATION
- CERATE ALL COMBINATIONS
- ADD WILDCARDS
- DELETE REPEATING PATTERNS
PATTERN EXTRACTION
- MATCH PATTERN LIST TO CORPUS/CORPORA
- EXTRACT PATTERNS
- CALCULATE PATTERN OCCURENCE
POSTPROCESSING
- EXTRACT LIST OF FREQUENT PATTERNS IN CORPUS/CORPORA
- PATTERNS SORTED BY:
1. PATTERN LENGTH
2. OCCURRENCE
- GENERATE PRAGMATIC SPECS FOR CORPUS/CORPORA
COMPARE CORPORA (IF MORE THAN ONE)
PATTERN LISTS FOR n-LONG SENTENCE
1-length
2-length
(n-1)-length
n-length
...
CORPORA COMPARISON
- EXTRACT ORIGINAL FREQUENT PATTERNS FOR CORPORA
- SPECIFY DISTINCTIVE FEATURES OF CORPORA

Figure 2: Flow chart of the SPEC system.

peared only once in both corpora. For all patterns which, appeared more then once in at least one corpus, SPEC calculates for which of the two corpora the they are more distinctive. Firstly a weight w_p of a pattern in k-long group of patterns is calculated by dividing pattern frequency (PF) by number of all patterns left in the group after deletion of quasi-patterns, like in the equation 3. The confrontation of two assigned weights shows for which corpus the pattern is more distinctive.

$$w_p = \frac{PF}{all - quasi} \tag{3}$$

3 Conclusions and Future Work

In this paper we presented a description of SPEC, or *Sentence Pattern Extraction and analysis arChitecturte*. The presented system, is meant to extract sentence patterns from corpora. The extracted patterns are more sophisticated than the ones obtained in a usual n-gram approach, as SPEC does not assume that two subsequent elements of a pattern appear subsequently also in the sentence. SPEC firstly generates all combinations of possible patterns and calculates their pattern frequency (PF). SPEC is capable of processing corpora written in any language, as long as they are minimally preprocessed (spacing, POS tagging, dependency structure, etc.). The patterns could be further applied in different NLP tasks, including document classification, user detection, or sentiment analysis. Frequent patterns could be also used as sentence templates in language generation tasks.

In the near future we plan to verify several aspects of SPEC. Firstly, we will confront SPEC with the usual n-gram approach. We plan at least two types of evaluation. In Quantitative Evaluation we will compare the number of extracted n-grams to the number of extracted SPEC (non n-gram) patterns (excluding quasi-patterns and quasi n-grams). In Qualitative Evaluation we will compare pattern frequency (PF) with n-gram frequency to specify the effectiveness of our method, or how many valuable patterns are lost in the n-gram approach.

Another method for evaluation will be applying SPEC to other tasks, such as spam classification, sentiment analysis [8], or cyber-bullying detection [4]. We also plan to use SPEC in lexicon expansion, which has been noticed necessary in tasks like affect analysis [9].

Acknowledgments

This research was supported by (JSPS) KAKENHI Grant-in-Aid for JSPS Fellows (22-00358).

References

[1] Taku Kudo. MeCab: Yet Another Part-of-Speech and Morphological Analyzer, 2001. http://mecab.sourceforge.net/

[2] Kaori Sasai, The Structure of Modern Japanese Exclamatory Sentences: On the Structure of the Nanto-Type Sentence. *Studies in the Japanese Language*, Vol, 2, No. 1, pp. 16-31, 2006

[3] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1), pp. 1-47, 2002.

[4] Michal Ptaszynski, Pawel Dybala, Tatsuaki Matsuba, Fumito Masui, Rafal Rzepka and Kenji Araki. Machine Learning and Affect Analysis Against Cyber-Bullying, In *Proceedings of AISB2010*, LaCATODA Symposium, pp. 7-16.

[5] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels, *The Journal of Machine Learning Research*, 2, pp. 419-444, 2002.

[6] Igor Burkhanov. Pragmatic specifications: Usage indications, labels, examples; dictionaries of style, dictionaries of collocations, In Piet van Sterkenburg (Ed.). *A practical guide to lexicography*, John Benjamins Publishing Company, 2003.

[7] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3 pp. 1289-1305, 2003.

[8] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical Methods in Natural Language Processing (EMNLP '02)*, pp. 79-86.

[9] Ptaszynski, M., Dybala, P., Rzepka, R. and Araki, K., Affecting Corpora: Experiments with Automatic Affect Annotation System - A Case Study of the 2channel Forum, *Proceedings PACLING-09*, pp. 223-228, 2009.