Using a Partially Annotated Corpus to Build a Dependency Parser for Japanese

Manabu Sassano

Fujitsu Laboratories, Ltd., 4-1-1, Kamikodanaka, Nakahara-ku,
Kawasaki 211-8588, Japan
sassano@jp.fujitsu.com

Abstract. We explore the use of a partially annotated corpus to build a dependency parser for Japanese. We examine two types of partially annotated corpora. It is found that a parser trained with a corpus that does not have any grammatical tags for words can demonstrate an accuracy of 87.38%, which is comparable to the current state-of-the-art accuracy on the Kyoto University Corpus. In contrast, a parser trained with a corpus that has only dependency annotations for each two adjacent *bunsetsus* (chunks) shows moderate performance. Nonetheless, it is notable that features based on character n-grams are found very useful for a dependency parser for Japanese.

1 Introduction

Corpus-based supervised learning is now a standard approach to build a system which shows high performance for a given task in NLP. However, the weakness of such approach is to need an annotated corpus. Corpus annotation is labor intensive and very expensive. To reduce or avoid the cost of annotation, various approaches are proposed, which include unsupervised learning, minimally supervised learning (e.g., [1]), and active learning (e.g., [2,3]).

To discuss clearly the cost of corpus annotation, we here consider a simple model of the cost:

$$\text{annotation cost} \propto \sum_t c(t)n(t)$$

where t is a type of annotation such as POS tagging, chunk tagging, etc., $c(t)$ is a cost per type t annotation, and $n(t)$ is the number of type t annotation.

Previous work to tackle the problem of annotation cost has mainly focused on reducing $n(t)$. For example, in active learning, useful examples to be annotated are selected based on some criteria, and then the number of examples to be annotated is considerably reduced. In contrast, we here focus on reducing $c(t)$ instead of $n(t)$. Obviously, if some portion of annotations are not given, the performance of a NLP system will deteriorate. The question here is how much the performance deteriorates. Is there a good trade-off between saving the cost and losing the performance?

Minimizing portions of annotations is also very important from the point of view of engineering. Suppose that we want to build an annotated corpus to make a parser for some real-world application. The design and strategy of corpus annotation is crucial in order to get a good parser while saving the cost. Furthermore, we have to keep in mind the maintenance cost of both the corpus and the parser. For example, we may find some errors in the annotations and the design of linguistic categories. In this situation fewer annotations lead to saving the cost because the corpus is more stable and less prone to errors.

The main purpose of this study is to explore the use of a partially annotated corpus to build a dependency parser for Japanese. In this paper, we describe experiments to investigate the feasibility of a partially annotated corpus. In addition, we propose features for parsing which are based on character n-grams. Even if grammatical tags are not given, a parser with these features demonstrates better performance than does the maximum entropy parser [4] with full grammatical features. Similarly, we have conducted experiments on *bunsetsu* (described in Sect. 2.1) chunking trained with a corpus which does not have grammatical tags. After that, we have tested a parser trained with a corpus which is partially annotated for dependency structures.

2 Parsing Japanese

2.1 Syntactic Properties of Japanese

The Japanese language is basically an SOV language. Word order is relatively free. In English the syntactic function of each word is represented with word order, while in Japanese postpositions represent the syntactic function of each word. For example, one or more postpositions following a noun play a similar role to declension of nouns in German, which indicates a grammatical case.

Based on such properties, the concept of *bunsetsus*[1] was devised and has been used to describe the structure of a sentence in Japanese. A *bunsetsu* consists of one or more content words followed by zero or more function words. By defining a bunsetsu like that, we can analyze a sentence in a similar way that is used when analyzing the grammatical role of words in inflecting languages like German.

Thus, strictly speaking, bunsetsu order rather than word order is free except the bunsetsu that contains the main verb of a sentence. Such bunsetsu must be placed at the end of the sentence. For example, the following two sentences have an identical meaning: (1) Ken-ga kanojo-ni hon-wo age-ta. (2) Ken-ga hon-wo kanojo-ni age-ta. (-ga: subject marker, -ni: dative case particle, -wo: accusative case particle. English translation: Ken gave a book to her.) Note that the rightmost bunsetsu 'age-ta,' which is composed of a verb stem and a past tense marker, has to be placed at the end of the sentence.

We here list the constraints of Japanese dependency including ones mentioned above.

C1. Each bunsetsu has only one head except the rightmost one.
C2. Each head bunsetsu is always placed at the right hand side of its modifier.
C3. Dependencies do not cross one another.

These properties are basically shared also with Korean and Mongolian.

[1] The word 'bunsetsu' in Japanese is composed of two Chinese characters, i.e., 'bun' and 'setsu.' 'Bun' means a sentence and 'setsu' means a segment. A 'bunsetsu' is considered to be a small syntactic segment in a sentence. A *eojeol* in Korean [5] is almost the same concept as a bunsetsu. Chunks defined in [6] for English are also very similar to bunsetsus.

2.2 Typical Steps of Parsing Japanese

Because Japanese has the properties above, the following steps are very common in parsing Japanese:

S1. Break a sentence into morphemes (i.e. morphological analysis).
S2. Chunk them into bunsetsus.
S3. Analyze dependencies between these bunsetsus.
S4. Label each dependency with a semantic role such as agent, object, location, etc.

Note that since Japanese does not have explicit word delimiters like white spaces, we first have to tokenize a sentence into morphemes and at the same time give a POS tag to each morpheme (S1). Therefore, when building an annotated corpus of Japanese, we have to decide boundaries of each word (morpheme) and POS tags of all the words.

3 Experimental Setup

3.1 Parsing Algorithm

We employ the Stack Dependency Analysis (SDA) algorithm [7] to analyze the dependency structure of a sentence in Japanese. This algorithm, which takes advantage of C1, C2, and C3 in Sect. 2.1, is very simple and easy to implement. Sassano [7] has proved its efficiency in terms of time complexity and reported the best accuracy on the Kyoto University Corpus [8]. The SDA algorithm as well as Cascaded Chunking Model [9] is a shift-reduce type algorithm.

The pseudo code of SDA is shown in Fig. 1. This algorithm is used with any estimator that decides whether a bunsetsu modifies another bunsetsu. A trainable classifier, such as an SVM, a decision tree, etc., is a typical choice for the estimator.

3.2 Corpus

To facilitate comparison with previous results, we used the Kyoto University Corpus Version 2 [8]. Parsers used in experiments were trained on the articles on January 1st through 8th (7,958 sentences) and tested on the articles on January 9th (1,246 sentences). The articles on January 10th were used for development. The usage of these articles is the same as in [4,10,9,7].

3.3 Choice for Classifiers

We use SVMs [11] for estimating dependencies between two bunsetsus because they have excellent properties. One of them is that combinations of features in an example are automatically considered with polynomial kernels. Excellent performance has been reported for many NLP tasks including Japanese dependency parsing, e.g., [9]. Please see [11] for formal descriptions of SVMs.

3.4 SVM Setting

Polynomial kernels with the degree of 3 are used and the misclassification cost is set to 1.

```
//
// Input: N: the number of bunsetsus in a sentence.
//     w[]: an array that keeps a sequence of bunsetsus in the sentence.
//
// Output: outdep[]: an integer array that stores an analysis result,
//     i.e., dependencies between the bunsetsus. For example, the
//     head of w[j] is outdep[j].
//
// stack: a stack that holds IDs of modifier bunsetsus
//     in the sentence. If it is empty, the pop method
//     returns EMPTY (−1).
//
// function estimate_dependency(j, i, w[]):
//     a function that returns non-zero when the j-th
//     bunsetsu should modify the i-th bunsetsu.
//     Otherwise returns zero.
//
procedure analyze(w[], N, outdep[])
// Push 0 on the stack.
stack.push(0);
// Variable i for a head and j for a modifier.
for (int i = 1; i < N; i++) {
    // Pop a value off the stack.
    int j = stack.pop();
    while (j != EMPTY && (i == N − 1 || estimate_dependency(j, i, w))) {
        // The j-th bunsetsu modifies the i-th bunsetsu.
        outdep[j] = i;
        // Pop a value off the stack to update j.
        j = stack.pop();
    }
    if (j != EMPTY)
        stack.push(j);
    stack.push(i);
}
```

Fig. 1. Pseudo code of the Stack Dependency Analysis algorithm. Note that "i == N − 1" means the i-th bunsetsu is the rightmost one in the sentence. Any classifiers can be used in estimate_dependency().

4 Dropping POS Tags

First we conducted experiments on dropping POS tags. In corpus building for a parser, disambiguating POS tags is one of time consuming tasks. In addition, it takes much time to prepare guidelines for POS tagging. Furthermore, in the case of a Japanese corpus, we will need more time because we have to deal with word boundaries as well as POS tags. Therefore, it would be desirable to avoid or reduce POS annotations while minimizing the loss of performance of the parser.

4.1 Features

To examine the effect of dropping POS tags, we built the following four sets of features and measured parsing performance with these feature sets.

Standard Features. By the "standard features" here we mean the feature set commonly used in [4,10,12,9,7]. We employ the features below for each bunsetsu:

1. Rightmost Content Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
2. Rightmost Function Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
3. Punctuation (periods, and commas)
4. Open parentheses and close parentheses
5. Location - at the beginning of the sentence or at the end of the sentence.

In addition, features as to the gap between two bunsetsus are also used. They include: distance, particles, parentheses, and punctuation.

Words-Only Features. If POS tags are not available, we have to use only tokens (words) as features. In addition, we cannot identify easily content words and function words in a bunsetsu. Therefore, we here chose the simplest form of feature sets. We constructed a bag of words in each bunsetsu and then used them as features. For example, we assume that there are three words in a bunsetsu: *keisan* (computational), *gengogaku* (linguistics), *no* (of). In this case we get $\{$*keisan*, *gengogaku*, *no*$\}$ as features.

Character N-Gram Features. Next we constructed a feature set without word boundaries or POS tags. In this feature set, we can use only the character string of a bunsetsu. At first glance, such a feature set is silly and it seems that a corpus without POS tags cannot yield a good parser. It is because no explicit syntactic information is given.

Can we extract good features from a string? We found useful ideas in Sato and Kawase's papers [13,14]. They define a similarity score between two sentences in Japanese and use it for ranking translation examples. Their similarity score is based on character subsequence matching. Just raw character strings are used and neither morphological analysis, POS tagging, nor parsing is applied. Although no advanced analysis was applied, they had good results enough for translation-aid. In [13], DP matching based scores are investigated, and in [14], the number of common 2-grams and 3-grams of characters between two sentences is incorporated into a similarity score.

In our experiments we use blended n-grams which are both 1-grams and 2-grams. All the 1-grams and 2-grams from the character string of a bunsetsu are extracted as features. For example, suppose we have a bunsetsu the string of which is a sequence of three characters: *kano-jo-no* where '-' represents a boundary between Japanese characters and this string is actually written with three characters in Japanese. The following features are extracted from the string: *kano*, *jo*, *no*, \$-*kano*, *kano-jo*, *jo-no*, *no*-\$, where '\$' represents a bunsetsu boundary.

Combination of "Standard Features" and Character N-grams. The fourth feature set that we have investigated is a combination of "standard features" and character n-grams, which are described in the previous subsection.

4.2 Results and Discussion

Performance of parsers trained with these feature sets on the development set and the test set is shown in Table 1. For comparison to previous work we use the standard measures for the Kyoto University Corpus: dependency accuracy and sentence accuracy. The dependency accuracy is the percentage of correct dependencies and the sentence accuracy is the percentage of sentences, all the dependencies in which are correctly analyzed.

Table 1. Performance on Development Set and Test Set

	Dev. Set		Test Set	
Feature Set	Dep. Acc.	Sent. Acc.	Dep. Acc.	Sent. Acc.
"Standard"	88.97	46.18	88.72	45.28
Bag of Words (Words Only)	85.22	35.02	84.43	34.95
Character N-Grams	87.79	42.66	87.38	40.84
"Standard" + Character N-Grams	89.72	47.04	89.07	46.89

To our surprise, the parser with the feature set based on character n-grams achieved an accuracy of 87.38%, which is very good. Although this is worse than that of "standard feature set," the performance is still surprising. We considered POS tags were essential for parsing. Why so successful?

The reason would be explained by the writing system of Japanese and its usage. In modern Japanese text mainly five different scripts are used: kanji, hiragana, katakana, Arabic numerals, and Latin letters. Usage of these scripts indicates implicitly the grammatical role of a word. For example, kanji is mainly used to represent nouns or stems of verbs and adjectives. It is never used for particles, which are always written in hiragana. Essential morphological and syntactic categories are also often indicated in hiragana. Conjugation forms of verbs and adjectives are represented with one or two hiragana characters. Syntactic roles of a bunsetsu are often indicated by the rightmost morpheme in it. Most of such morphemes are endings of verbs or adjectives, or particles. In other words, the rightmost characters in a bunsetsu are expected to indicate the syntactic role of a bunsetsu.

Bunsetsu Chunking. After we observed the results of the experiments on parsing, a new question arose to us. Can we chunk tokens to bunsetsus without POS tags, too? We carried out additional experiments on bunsetsu chunking. Following [15], we encode bunsetsu chunking as a tagging problem. In bunsetsu chunking, we use the chunk tag set {B, I} where B marks the first word of some bunsetsu and words marked I are inside a bunsetsu. In these experiments on bunsetsu chunking, we estimated the chunk tag of each word using a SVM from five words and their derived attributes. These five words are a word to be estimated and its two preceding/following words. Features are extracted from the followings for each word: word (token) itself, major POS, minor POS, conjugation type, conjugation form, the leftmost character, the character type of the leftmost character, the rightmost character, and the character type of the rightmost character. A character type has a value which indicates a script. It can be either kanji, hiragana, katakana, Arabic numerals, or Latin letters.

We conducted experiments with four sets of features. Performance on the development set and the test set is shown in Table 2. We used the same performance measures as in [16]. Precision (p) is defined as the percentage of words correctly marked `B` among all the words that the system marked `B`. Recall (r) is defined as the percentage of words correctly marked `B` among all the words that are marked `B` in the training set. F-measure is defined as: F-measure $= 2pq/(p+q)$.

Table 2. Bunsetsu Chunking Performance on Development Set and Test Set. Grammatical tags include POS tags and conjugation types/forms.

Feature Set	Dev. Set (F)	Test Set (F)
Surface Form + Grammatical Tags	99.58	99.57
Surface Form Only	97.65	97.02
Surface Form + Char. Features (No Grammatical Tags)	99.09	99.07
Mixed	99.64	99.64

The bunsetsu chunker with surface forms only yielded worse performance than did that with the grammatical tags including major/minor POS and conjugation type/form. However, the chunker with character features achieved good performance even if grammatical tags are not available. In addition, the feature set in which all the available features are used gives the best among the feature sets we tested. Again we found that features based on characters compensate performance deterioration caused by no grammatical tags.

We have found that both a practical parser and a practical bunsetsu chunker can be constructed from a corpus which does not have POS information. This means we can make a parser for Japanese which is less dependent on a morphological analyzer. It would be useful for improving the modularity of an analysis system for Japanese.

5 Dropping Longer Dependency Annotations

As previous work [4,17] reports, approximately 65% of bunsetsus modify the one on their immediate right hand side. From this observation, we simplify dependency annotations. For each bunsetsu we give either the `D` tag or `O` where bunsetsus marked `D` modify the one on their immediate right hand side and bunsetsus marked `O` do not.

	Ken-ga	*kanojo-ni*	*ano*	*hon-wo*	*age-ta.*
	Ken-subj	to her	that	book-acc	gave.
ID	0	1	2	3	4
Head	4	4	3	4	-
{`D`, `O`}	`O`	`O`	`D`	`D`	-

Fig. 2. Sample sentence with dependency annotations. Bunsetsus marked `D` modify the one on their immediate right hand side and bunsetsus marked `O` do not. An English translation is "Ken gave that book to her."

Figure 2 shows a sample sentence with dependency annotations. This encoding scheme represents some portion of the dependency structure of a sentence. Annotating under this scheme is easier than selecting the head of each bunsetsu. We examined usefulness of this type of partially annotated corpus following the encoding scheme above.

5.1 Using Partial Dependency Annotations

The SDA algorithm, which we employ for experiments, can work with a partially annotated corpus to parse a sentence in Japanese[2]. In training, first we construct a training set only from dependency annotations between two adjacent bunsetsus. We ignore relations between two bunsetsus which have a longer dependency. After that, we train a classifier for parsing from the training set. In testing, we use the classifier for both two adjacent bunsetsus and other pairs of bunsetsus.

5.2 Results and Discussion

Performance on the development set and the test set are shown in Table 3.

The parser trained with the partially annotated corpus yielded good performance. However, its accuracy is considerably worse than that of the parser with the fully annotated corpus. This tendency is clearer in terms of sentence accuracy. To examine differences in terms of quantity, we plot the learning curves with the two corpora. The curves are shown in Fig. 3.

Table 3. Performance of parsers trained with the fully annotated corpus and the partially annotated corpus

	# of Training	Dev. Set		Test Set	
Training Set	Examples	Dep. Acc.	Sent. Acc.	Dep. Acc.	Sent. Acc.
Full	98,689	88.97	46.18	88.72	45.28
Adjacent Annotations Only	61,899	85.65	38.00	85.50	38.58

How many sentences which are partially annotated do we need in order to achieve a given accuracy with some number of fully annotated sentences? It is found that we need 8 – 17 times the number of sentences when using the partially annotated corpus instead of the fully annotated one. If hiring linguistic experts for annotation is much more expensive than hiring non experts, or it is difficult to find a large enough number of experts, this type of partially annotated corpus could be useful.

The naive approach we examined was not so effective in the light of the number of sentences to be required. However, we should note that a partially annotated corpus is easier to maintain the consistency of annotations.

6 Related Work

In this section we briefly review related work from three points of view, i.e., parsing performance, the use of partially annotated corpora, and the use of character n-grams.

[2] Cascaded Chunking Model [9] also can be applicable to use a partially annotated corpus.

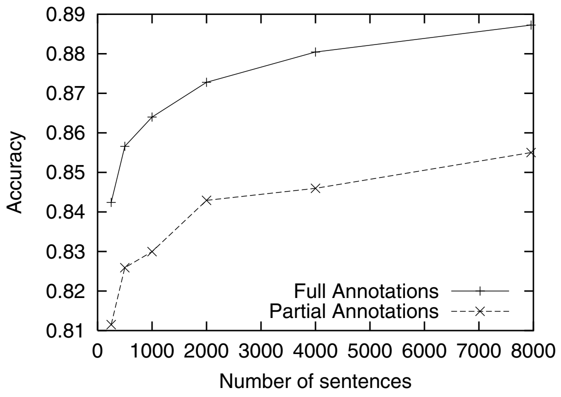

Fig. 3. Learning curves of parsers trained with the partially annotated corpus and the fully annotated corpus

Parsing Performance. Although improvement of the performance of a parser is not a primary concern in this paper, comparison with other results will indicate to us how practical the parser is. Table 4 summarizes comparison to related work on parsing accuracy. Our parsers demonstrated good performance although they did not outperform the best. It is notable that the parser which does not use any explicit grammatical tags outperforms one by [4], which employs a maximum entropy model with full grammatical features given by a morphological analyzer.

Table 4. Comparison to related work on parsing accuracy. KM02 = Kudo and Matsumoto 2002 [9], KM00 = Kudo and Matsumoto 2000 [12], USI99 = Uchimoto et al. 1999 [4], Seki00 = Sekine 2000 [18], and Sass04 = Sassano 2004 [7].

	Algorithm/Model/Features	Acc.(%)
This paper	Stack Dependency Analysis (cubic SVM) w/ char. n-grams	89.07
	Stack Dependency Analysis (cubic SVM) w/ char. n-grams, no POS	87.38
Sass04	Stack Dependency Analysis (cubic SVM) w/ various enriched features	89.56
KM02	Cascaded Chunking (cubic SVM) w/ dynamic features	89.29
KM00	Backward Beam Search (cubic SVM)	89.09
USI99	Backward Beam Search (MaxEnt)	87.14
Seki00	Deterministic Finite State Transducer	77.97

Use of Partially Annotated Corpora. Several papers address the use of partially annotated corpora. Pereira and Schabes [19] proposed an algorithm of inferring a stochastic context-free grammar from a partially bracketed corpus. Riezler et al. [20] presented a method of discriminative estimation of an exponential model on LFG parses from partially labeled data.

Our study differs in that we focus more on avoiding expensive types of annotations while minimizing the loss of performance of a parser.

Use of Character N-grams. Character n-grams are often used for POS tagging of unknown words, unsupervised POS tagging, and measures of string similarity. The number of common n-grams between two sentences is used for a similarity measure in [14]. This usage is essentially the same as in the spectrum kernel [21], which is one of string kernels [22].

7 Conclusion

We have explored the use of a partially annotated corpus for building a dependency parser for Japanese. We have examined two types of partially annotated corpora. It is found that a parser trained with a corpus that does not have any grammatical tags for words can demonstrate an accuracy of 87.38%, which is comparable to the current state-of-the-art accuracy. In contrast, a parser trained with a corpus that has only dependency annotations for each two adjacent bunsetsus shows moderate performance. Nonetheless, it is notable that features based on character n-grams are found very useful for a dependency parser for Japanese.

References

1. Yarowsky, D.: Unsupervised word sense disambiguation rivaling supervised methods. In: Proc. of ACL-1995. (1995) 189–196
2. Thompson, C.A., Califf, M.L., Mooney, R.J.: Active learning for natural language parsing and information extraction. In: Proc. of the Sixteenth International Conference on Machine Learning. (1999) 406–414
3. Tang, M., Luo, X., Roukos, S.: Active learning for statistical natural language parsing. In: Proc. of ACL-2002. (2002) 120–127
4. Uchimoto, K., Sekine, S., Isahara, H.: Japanese dependency structure analysis based on maximum entropy models. In: Proc. of EACL-99. (1999) 196–203
5. Yoon, J., Choi, K., Song, M.: Three types of chunking in Korean and dependency analysis based on lexical association. In: Proc. of the 18th Int. Conf. on Computer Processing of Oriental Languages. (1999) 59–65
6. Abney, S.P.: Parsing by chunks. In Berwick, R.C., Abney, S.P., Tenny, C., eds.: Principle-Based Parsing: Computation and Psycholinguistics. Kluwer Academic Publishers (1991) 257–278
7. Sassano, M.: Linear-time dependency analysis for Japanese. In: Proc. of COLING 2004. (2004) 8–14
8. Kurohashi, S., Nagao, M.: Building a Japanese parsed corpus while improving the parsing system. In: Proc. of the 1st LREC. (1998) 719–724
9. Kudo, T., Matsumoto, Y.: Japanese dependency analysis using cascaded chunking. In: Proc. of CoNLL-2002. (2002) 63–69
10. Sekine, S., Uchimoto, K., Isahara, H.: Backward beam search algorithm for dependency analysis of Japanese. In: Proc. of COLING-00. (2000) 754–760
11. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag (1995)
12. Kudo, T., Matsumoto, Y.: Japanese dependency structure analysis based on support vector machines. In: Proc. of EMNLP/VLC 2000. (2000) 18–25
13. Sato, S.: CTM: An example-based translation aid system. In: Proc. of COLING-92. (1992) 1259–1263

14. Sato, S., Kawase, T.: A high-speed best match retrieval method for Japanese text. Technical Report IS-RR-94-9I, Japan Advanced Institute of Science and Technology, Hokuriku (1994)
15. Ramshaw, L.A., Marcus, M.P.: Text chunking using transformation-based learning. In: Proc. of VLC 1995. (1995) 82–94
16. Murata, M., Uchimoto, K., Ma, Q., Isahara, H.: Bunsetsu identification using category-exclusive rules. In: Proc. of COLING-00. (2000) 565–571
17. Maruyama, H., Ogino, S.: A statistical property of Japanese phrase-to-phrase modifications. Mathematical Linguistics **18** (1992) 348–352
18. Sekine, S.: Japanese dependency analysis using a deterministic finite state transducer. In: Proc. of COLING-00. (2000) 761–767
19. Pereira, F., Schabes, Y.: Inside-outside reestimation from partially bracketed corpora. In: Proc. of ACL-92. (1992) 128–135
20. Riezler, S., King, T.H., Kaplan, R.M., Crouch, R., III, J.T.M., Johnson, M.: Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In: Proc. of ACL-2002. (2002) 271–278
21. Leslie, C., Eskin, E., Noble, W.S.: The spectrum kernel: A string kernel for SVM protein classification. In: Proc. of the 7th Pacific Symposium on Biocomputing. (2002) 564–575
22. Lodhi, H., Saunders, C., Shawe-Tayor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research **2** (2002) 419–444