Sentence Compression For Automatic Subtitling

Juhani Luotolahti and Filip Ginter
Department of Information Technology
University of Turku, Finland
mjluot,figint@utu.fi

Abstract

This paper investigates sentence compression for automatic subtitle generation using supervised machine learning. We present a method for sentence compression as well as discuss generation of training data from compressed Finnish sentences, and different approaches to the problem. The method we present outperforms state-of-the-art baseline in both automatic and human evaluation. On real data, 44.9% of the sentences produced by the compression algorithm have been judged to be useable as-is or after minor edits.

1 Introduction

Automated broadcast programme subtitling is an important task, especially with the recent introduction of legislation which mandates all programmes of the Finnish national broadcasting corporation to be subtitled, even in cases where the programme is in Finnish, and not in a foreign language. Providing such a subtitling is a resource-intensive task, requiring human editors to manually transcribe the programme and produce the subtitles. There is an obvious motivation to automate this process to increase the subtitling throughput and drive the costs down. While ultimately aiming at a fully automated pipeline from speech recognition to screen-ready subtitles, in this paper we focus specifically on the task of text compression for subtitling.

The need for this task arises from the fact that the whole spoken content of the programme cannot be displayed in the area of the screen devoted to subtitles while respecting the standards setting the maximum number of characters per line and the minimum amount of time the subtitles must be shown. The subtitling naturally also needs to remain in time synchronisation with the spoken programme. In practice, the subtitling editors thus need to compress the text of the subtitles, removing or abridging parts which are less critical for the understandability of the programme.

2 Sentence Compression

The goal of automatic sentence compression is to create a shorter version of the input sentence, in a way preserving its meaning. Sentence compression is most often extractive, formed by dropping words from a sentence that are not needed for the sentence to be grammatical and do not importantly contribute to the meaning of the sentence. Many sentence compression methods are based on supervised learning using parallel corpora as training material (Knight and Marcu, 2002; Turner and Charniak, 2005; McDonald, 2006; Cohn and Lapata, 2009). Some methods don't require parallel corpora, but are either based on rules (Gagnon and Da Sylva, 2005) or use language models or statistics gathered from non-parallel sources (Chiori and Furui, 2004; Filippova and Strube, 2008; Clarke and Lapata, 2006). While some systems prune the sentence based on the linear order of the words, others prune the parse trees or modified parse trees. Language models are commonly used to ensure grammatical output.

3 Data and its pre-processing

We draw our data from subtitles of the Finnish national broadcasting corporation television programs provided to us by Lingsoft Inc. From Lingsoft, we have obtained the texts both before and after the compression step of the subtitling process, extracted from the internal processing pipeline. As illustrated in Figure 1, each programme consists of the subtitle texts and the associated time-stamps which define the time period in which the subtitle is shown on the screen. The full, unabridged

Subtitle	Original
13 10:01:31,12 --> 10:01:35,24 Jaro has returned to Finland after a victorious racing trip.	11 00:01:48,000 --> 00:01:51,600 Jaro has returned to Finland after a victorious racing trip to Estonia.
14 10:01:36,01 --> 10:01:41,01 Champion Toni Gardemaister awaits him with a surprise.	12 00:01:52,400 --> 00:01:56,000 Champion Toni Gardemaister awaits him with a surprise.
15 10:01:42,00 --> 10:01:44,15 Oh, hello. -Hi there	13 00:01:57,700 --> 00:02:00,000 Oh, hello hello. Hi there. What's up? Nothing much.
16 10:01:44,17 --> 10:01:49,21 Let's go to the kart racing track and let's see how good you are.	15 00:02:00,600 --> 00:02:05,100 Let's, you know, go cruising a little to the kart racing track. And let's see how good you are at kart racing.

Figure 1: Example document excerpt from the data, translated to English.

Pertti	hei	,	mä	käyn	näyttämäs	näitä	dioja	yhelle	asiantuntijalle	.
Pertti	hey	,	I	will_go	show	these	slides	one	to_expert	.
*	*	*	Mä	käyn	näyttämässä	näitä	dioja	*	asiantuntijalle	.
-	-	-	I	will_go	show	these	slides	-	to_expert	.
D	D	D	+	-	=	-	-	D	-	-

*	Mites	tää	puhelin	,	onks	tää	toiminu	?
-	How_about	this	phone	,	has	it	worked	?
Onko	*	tää	puhelin	*	*	*	toiminu	?
Has	-	this	phone	-	-	-	worked	?
I	D	-	+	D	D	D	-	-

Figure 2: Example alignments

version of the programme is used for internal purposes of the company and is the result of manual correction of speech recognition output. The compressed version of the programme consists of completed subtitles, exactly as delivered and subsequently aired. The subtitles often include spoken language, with incomplete words and slang terms, making them different in style from the strictly grammatical text which would be ideal.

The first step in pre-processing the data is to obtain a good alignment of the texts so that the individual edits can be identified. The data was received as raw subtitle files. A subtitle file consists of text units to be shown on a screen at a particular moment and the time to show it. Because the unabridged version was a result of speech recognition, its timing didn't correspond with the abridged version's timing. The subtitles and the amount of sentences they include are also different in size, because in many cases whole sentences were removed or introduced in the abridging process.

To identify the edits made to the subtitles, especially tokens being removed, it was necessary to obtain token to token level alignments between the two versions of the subtitles. String alignment was created using a distance matrix generated by calculating Levenshtein distance between the two subtitles on a token level. The edits were extracted from the distance matrix the method generates. Tokens with only minimal modifications (eg. 'ohjelma', 'program' and 'ohjelmamme', 'our program') were aligned instead of counting as a substitution. Minimal modification of tokens was defined as being sufficiently close to each other, when calculated with a string matching algorithm.

Because of the edits made in the abridging pro-

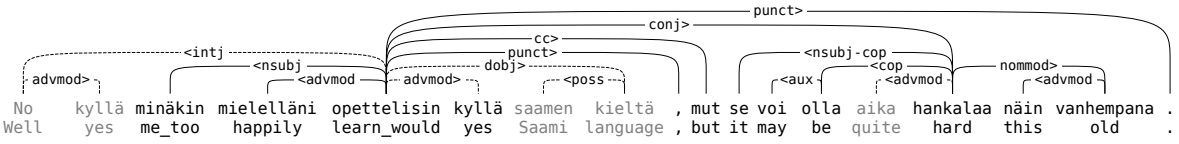

Figure 3: An example of the extended SD scheme.

cess, sometimes sentences were combined and sometimes cut in the abridged version. After we had the alignments the original subtitles with the abridged ones the subtitles were sentence split and the sentences were aligned with sentences of the abridged text.

Each sentence was parsed using the recently published Finnish statistical dependency parsing pipeline (Haverinen et al., 2014). The parser produces full dependency parse trees in a minor modification of the well-known Stanford Dependencies (SD) scheme (de Marneffe and Manning, 2008).

4 Methods

Since we have the appropriate training data at our disposal, we will approach the task as a supervised machine learning problem, whereby a classifier will predict which words will be removed from each sentence. We will test a few different approaches to the task, all based on supervised learning, but using different sentence reduction strategies. The feature set consists mainly of features based on the dependent token and features derived from the dependency tree, which we will describe later.

The first approach is to prune the dependency trees of the sentences. The goal of this approach is to produce more syntactically valid output than reducing the sentence based on its linear order. This, however, does not guarantee the syntactic validity of the sentence. Consider, for instance the subtree headed by *hankalaa (hard)* in Figure 3. Removing the word, together with its subtree will leave the conjunction *mut (but)* orphaned, resulting in an ungrammatical sentence. We will address the most common such cases with a set of straightforward post-processing rules. For dependency tree pruning we have the following strategies:

The first strategy is to let the classifier decide which dependencies (edges) to prune from the dependency tree and remove the complete subtree along with the removed dependency. For this we train an SVM classifier to recognize the dependencies to remove and clean the training data from dependencies which would be removed by a removal of a dependency higher up in the tree. The SVM implementation used is libSVM (Chang and Lin, 2011). In the results table, we refer to this approach as *mp svm*.

Another strategy is to remove a dependency only if all of the dependencies under it in the dependency tree have been removed as well. For this an SVM classifier is trained which for each dependency makes a prediction on whether to keep or remove it. Unlike previously, this time the training data contains all dependencies. The motivation behind this strategy is to conserve important subtrees of the tree. In the results table, we refer to this approach as *mk svm*.

The second approach for compressing the sentence is to let a classifier freely remove tokens from the sentence without limitations of the dependency tree structure. To gain advantage from the linear order of the sentence we cast the problem as a sequence classification problem and use Conditional Random Fields (CRF), as implemented in CRFSuite (Okazaki, 2007) with the same feature set as previously to predict which tokens to keep and which to drop. We refer to the CRF model as *crf*.

We also train another CRF model (referred to as *crf pos*) without the dependency tree features to see how well it fares against the full feature set and to judge the extent to which syntax contributes to the classification.

We will also implement a baseline system based on the work of Filippova and Strube (2008), which is based on finding an optimal abridged tree using Integer Linear Programming and unsupervised learning. This will be referred as *base ilp* in the evaluation tables.

The last system we build is a modification of the mentioned baseline system, described in detail in a later chapter. In this system we replace the statistical scores used by the baseline with those given by the *crf* model, such that basically the probabil-

ity of token being removed is decided by the CRF classifier and is used by the ILP process to make the final decisions. The appeal of this strategy is the use of ILP, and a direct comparison with the baseline since it uses a qualitatively different removal scheme than our systems. This is referred as *crf ilp* in the evaluation.

All of these systems allow their rate of removal to be fixed by altering the classifier threshold and in the case of the Integer Linear Programming based method by setting a fixed rate of removal. The score used to adjust the rate of removal for SVM is simply the classifier score and for CRF, the marginal probabilities for token removal. It is to be noted that for the CRF based methods altering the threshold is very important, since without threshold manipulation the classifier produces too low compression rates ($\sim$5% of tokens removed in the dev-set with full feature set).

As with any similar supervised machine learning method, feature engineering is an important part of the development. The first class of features we use is derived from the morphological analysis of the target word. The second class of features is based on the surrounding structure of the parse tree. These features model the syntactic context of the target word to capture its immediate syntactic neighbourhood. We also add combination features where appropriate, since the underlying classifier is linear. The third class of features includes those that encode information about the target word's position in the sentence and within the tree. We also employ features from the Semantic Role Labeling system of Kanerva et al. (2014). We list the exact features used below, and in Section 5 we will present a feature ablation study to demonstrate their relative merits.

Features

The exhaustive list of features is described in the following:

Features based on the token

- Morphological tags
- The morphological tags of the next and the previous token in the sentence
- Word and lemma of the token
- Next and previous word and lemma of the token
- Next and previous pos-tags of the token

Tree structure based Features

- Dependency type
- The dependency types of the next and the previous token in the sentence
- The types of dependencies that have the dependent token as a governor and also this feature combined with the dependency type of the dependent token
- Dependency type combined with the dependent token's morphological tags
- The dependency types of the tokens which are siblings of this token in the dependency tree with information about whether the tokens are to the left or to the right of the dependent token in the sentence. This is also combined with the dependency type of the token.
- The morphological tags of the governing token
- The dependency type of the governing token with and without the dependent token's dependency type combined
- How many tokens depend on the current token
- How large a subtree would be removed if this dependency was pruned
- Whether this token is a leaf in the tree
- Whether this token has incoming semantic edges

Location and sentence based Features

- Whether sentence length is over 5, 10, or 15 tokens with and without the dependency type
- How long a path in the dependency tree is to the root node from this node
- Whether the number of dependencies above this dependency in the path to the root node is in the first, second, third or fourth quarter of the longest path to the root node in the tree
- Whether the dependent token is located in the first, second, third or fourth quarter of the sentence
- The two above features combined into one

5 Evaluation

Problem setting

The data is divided into training, development and test sets, with the division carried out by sampling sentences randomly. The development and test sets were both 3247 sentences long (roughly 18%

of all sentences). In the training data, only sentences with removals and no other edits such as substitutions are used, to ensure the classifier does not learn to remove tokens that are in fact substituted for another token, possibly at some later point in the sentence. The development and test sets are, however, preserved exactly as-is, containing all sentences from the original data. The compression experiments are done with compression rate of 85% (i.e. 15% token removal) to be in line with the test data and also 70% to see how the systems fare with a higher rate of removal.

Baseline

As the baseline, we have re-implemented the method of Filippova and Strube (2008). Like our method, the baseline is based on the removal of dependency sub-trees, however, it is an unsupervised method, requiring only a dependency treebank for its training. This will allow us to study to what extent the availability of supervised data affects the overall performance on the compression task, as compared to an unsupervised baseline previously shown to have a good performance and based on the same principle of dependency subtree removal.

The baseline method has three steps: transforming the source tree by adding additional edges, compression, and tree linearisation. The method assigns a score for each edge of a dependency tree and tries to find optimal edges to remove, maximizing the score of the remaining edges and at the same time maintaining a correct tree structure. The edge scores are calculated from statistics derived from a treebank. The method uses integer linear programming to find a globally optimal solution.

In our experiment, statistics of the dependencies and tokens are calculated from an approximately 500 million token corpus obtained by extracting Finnish text from the CommonCrawl[1] Internet crawl data and automatically parsed using the Finnish dependency parsing pipeline of Haverinen et al. (2014).

The trees are first pre-processed by creating a dummy root node and adding an edge from the dummy root to each finite verb in the sentence, making the trees graphs. Then, auxiliary verbs, negation markers and function words are merged with their governors to ensure they are not removed separately. And finally, coordination structures are decomposed by propagating the governor of the first coordinated element to every other element in the coordination, preserving the dependency type.

Tree compression is then cast as an optimization problem and solved using integer linear programming. Each dependency from a governor word g to a dependent d with type l is assigned a binary variable:

$$x^l_{g,d} = \begin{cases} 1 & \text{if edge preserved;} \\ 0 & \text{if edge not preserved.} \end{cases}$$

The method then optimizes the objective function

$$f(X) = \sum_x x^l_{g,d} \cdot P(l|g) \cdot I(d) \qquad (1)$$

where $P(l|h)$ is the conditional probability of the dependency type l, given that g is the governor. $I(d)$ is an importance score defined as:

$$I(d_i) = \frac{l}{N} f_i \cdot \log(\frac{F_a}{F_i}) \qquad (2)$$

where l is the number of clause nodes above the dependency, N is the maximum level of embedding, f_i is the frequency of the word in current document, F_a is the sum of frequencies of topic words in the corpus and F_i is the frequency of the word in corpus.

Constraints are added into the integer linear programming problem in order to ensure that a correct structure is produced, making sure that each word has a governor. The maximal number of tokens in the resulting tree is also encoded as a constraint to the problem, allowing the control of the compression ratio. The pruned tree is then linearized in the original order of words in the sentence.

Test Set Evaluation

First, we compare the performance of our proposed methods and the baseline in terms of F-score on the test set. Precision is defined as the proportion of predicted removed tokens which were also removed in gold standard, and recall is conversely defined as the proportion of removed tokens in the gold standard, whose removal is also predicted by the system. The main results in Table 1 show that with essentially equal compression ratios, the feature-rich CRF (referred to as *crf*) results in the highest F-score in both rates of

[1] http://www.commoncrawl.org

	F	CR(tkn)	CR(chr)
gold	1.0	0.863	0.884
crf ilp85	0.2346	0.847	0.844
base ilp85	0.1983	0.845	0.819
crf85	**0.3809**	0.850	0.879
crf pos85	0.3511	0.850	0.884
mk svm85	0.3613	0.849	0.883
mp svm85	0.3258	0.849	0.872
crf ilp70	0.2611	0.715	0.712
base ilp70	0.2504	0.714	0.683
crf70	**0.3758**	0.700	0.761
crf pos70	0.3527	0.700	0.777
mk svm70	0.3640	0.700	0.759
mp svm70	0.3408	0.699	0.741

Table 1: F1-Scores for the test set. CR(tkn) is token level rate of compression and CR(chr) is character level compression rate of the system output.

removal, followed by the SVM classifier (which makes independent predictions, unlike the CRF sequence classifier). The baseline performs substantially worse. As a further insight into the methods, we present in Table 3 the ten most often removed dependency types for the best scoring *crf* model, the baseline, and in the test set. As expected, with few exceptions we see dependency types associated with modifiers and functional words rather than core semantic arguments. Most of these types will also tend to have a single, leaf dependent. We can also observe a rather wide overlap (7/10) of the commonly removed dependency types between the two methods, showing that the systems target similar points of compression.

Further, we perform a small-scale feature ablation study with a CRF classifier. The most important feature groups are those related to the token itself, such as its POS-tag and lemma. The features gathered from the syntactic trees, and related location group of features both contribute positively to the classification, even though the contribution is not major.

The F-score measure can be evaluated on as many runs as necessary, for instance in parameter selection, but it does not necessarily reflect the ability of the system to produce fluent and meaning-preserving compressions. The underlying problem is that any given sentence can have a number of valid compressions, but only one of them will be counted as a true positive, all others will be counted as false positives. To address these issues, we perform also a manual evaluation of the result, discussed in the following section.

Feature Set	Dev-set@85% F-score
Token	34.10
Token+location	35.96
Token+tree structure	36.31
All	37.16

Table 2: CRF feature ablation table on development set with 85% compression rate.

Gold		Base		crf	
advmod	26.0%	advmod	18.9%	advmod	36.8%
punct	17.9%	nommod	13.0%	punct	10.8%
nommod	7.3%	punct	9.0%	det	8.4%
det	5.6%	amod	6.7%	intj	6.1%
nsubj	5.4%	dobj	5.8%	cc	5.4%
intj	4.4%	det	5.5%	nommod	4.6%
cc	3.9%	poss	5.3%	nsubj	3.7%
amod	3.1%	cc	4.0%	complm	3.3%
conj	2.7%	conj	3.5%	amod	2.8%
dobj	2.6%	cop	3.4%	conj	2.3%

Table 3: Dependency types pruned in the test set

Manual evaluation

In this evaluation, we focus on the ability of the systems to produce fluent output and preserve important, content-bearing parts of the sentence (i.e. its "main message"). These two qualities are to some degree independent (although clearly not entirely so) and we thus evaluate them separately.

We selected 399 random sentences which had been compressed by the systems from the test section of the corpus, and for each sentence we produced four compressed versions: one using the baseline method, one using the *crf* model which got the highest F-score on the test set. In addition we test *crf pos* and *crf ilp*. The *crf pos* set is selected because of its relatively high F-score on the test set, even though it does not employ the syntax of the sentence. The *crf ilp* is selected to test both the integer linear programming method and to provide the baseline with a comparison using the same approach to sentence reduction. For the test, compression rates were aligned. In the end all systems had a rate of token removal of 75% for the sentences being tested.

The compressed versions of the sentences were subsequently evaluated by a native speaker in

	Fluency
3	Readable as is
2	Minor revisions needed
1	Major revisions needed
0	Incomprehensible
	Meaning
3	Message preserved perfectly
2	Important message preserved
1	Minor revisions needed
0	Important message lost

Table 4: Manual evaluation scales.

	Fluency	Meaning
crf	799 (66.7%)	609 (50.8%)
crf pos	796 (66.5%)	579 (49.9%)
crf ilp	795 (66.4%)	487 (40.7%)
Baseline	720 (60.2%)	383 (32.0%)

Table 5: Sum of the scores over all test sentences given by the evaluator. The percentages are given in terms of maximum possible value, which for all quantities is 399 sentences $\times$ maximum score of 3, i.e. 1197.

terms of fluency and in terms of content preservation using the scales shown in Table 4. The order in which the compressed versions were presented for evaluation was randomly changed for every sentence separately, i.e. it was not possible to distinguish the methods by the evaluator. Further, the evaluator was not involved in the development of the methods in any manner and was thus not primed to recognize features typical of compressions produced by any of these methods.

To gain an initial insight into the evaluation results, we show in Table 5 the sum of scores given across all sentences. We can see that the *crf* gains on top of the baseline in terms of both measures: 6.5pp (percent points) in terms of fluency and 18.8pp in terms of meaning. These correspond to 16.3% and 27.6% of relative error decrease over the baseline. Both differences are statistically significant with $p < 0.02$ (two-tailed paired t-test).

For practical deployment, the proportion of sentences which need no, or only minor corrections is a crucial measure. For the best performing CRF method, 75.4% and 44.9% (fluency and meaning) were assigned score of 2 or 3, i.e. usable as-is or with only minor corrections. For the baseline method, the corresponding proportions are 68.2% and 15.3%, reflecting a notable gain of the proposed method over the baseline.

When both fluency and meaning had to be assigned score of 2 or 3, 44.9% of the sentences produced by the *crf* method required only minor modifications for fluency or were readily usable. For the baseline method only 15.0% of the sentences were rated as readily usable or requiring only minor modifications for fluency. The 29.9pp gain of the proposed method corresponds to a 35.1% relative decrease in error, and ultimately manual effort saved by the proposed method. 74.2% of the *crf* produced sentences are usable as-is or require minor fixing in terms of fluency and/or meaning, when using a more relaxed criteria and requiring fluency to be scored 2 or 3 and meaning to be 1 (Minor revisions needed for maintaining meaning) or greater, while baseline produces such sentences 63.9% of the time. This difference of 10.3pp corresponds to a relative decrease in error rate of 28.5%. The difference of performance between *crf* and *crf pos* when it comes to fluency or meaning is not statistically significant, although *crf* is rated higher on both measures. Human evaluation would suggest the *crf pos* performs slightly worse in maintaining the meaning of the sentence (0.9pp) than *crf*, while the difference in fluency is very small (0.2pp). This would suggest the syntax of the sentence might help the system deciding which tokens are important for the meaning of the sentence.

The difference in fluency between *crf* and *crf ilp* is not statistically significant, but difference between meaning is statistically significant ($p < 0.02$ on two-tailed paired t-test). Because this system is identical in all respects but the scores used to prune the tree, to the baseline of which difference of fluency is statistically significant to the *crf*, this shows the CRF based scores do help with the fluency of the output. The comparison between *crf ilp* and the baseline is interesting, because they are essentially the same system except one is based on supervised learning and the other is based on statistics. The *crf ilp* outperforms the baseline on both metrics and the results are statistically significant. This speaks in favour of the supervised approach.

Human agreement

Earlier in the development process, we also performed another human evaluation to test both the

process of human evaluation and the performance of the system. We were especially interested in the subjectivity of the task and the human agreement. While in this earlier experiment the test data, system and rate of removal are different from the above final test setting, it still offers insight into the evaluation process and especially the level of agreement of the human evaluators. For this human evaluation round, two evaluators were used, and rate of token compression for both classifiers was set to 85%. The participants of this evaluation are *mp svm* against the ILP-based baseline. 200 sentences were selected and all judged independently by both evaluators.

The Kappa score of the inter-annotator agreement over all 800 annotation data points (2 tasks $\times$ 2 methods $\times$ 200 sentences) is 0.32. When measured per method and task, Kappa varies from 0.25 (baseline method, meaning task) to 0.36 (proposed method, meaning task). For the specific binary decision of whether the score of an individual sentence is $\geq$ 2 or not, the overall Kappa score is 0.39. The overall scores of 0.32 and 0.39 would be interpreted as "fair" using the criteria of Viera et al. (2005).

6 Discussion and conclusions

Comparison of the F-score evaluation between the supervised method and the unsupervised baseline shows that the supervised training gives a rather substantial benefit over the unsupervised baseline. Numerically, the F-scores remained very low, which, however, can be largely attributed to the rather arbitrary nature of the sentence compression task where any sentence of a reasonable length may have a number of alternative compressions. Of these, one was selected when producing the sub-titles and the alternatives count as errors in the F-score based evaluation. This cannot be addressed without a major data curation effort which, in our practical setting, is not possible.

The manual evaluation not only shows considerably more promising results in the numeric sense, but also shows correlation with the F-score based evaluation. This suggests that it is possible to use the F-score evaluation to develop and optimize the method, while a manual evaluation is clearly necessary to gain insight into the practical usability of the output compressions.

Interestingly, we find a clearly better performance of the CRF-based method also in terms of fluency, even though the baseline method uses linear programming to find a globally optimal solution and the statistics it relies on were gathered on a parsed corpus of a substantial size.

From a practical point of view, the manual evaluation shows that about one half of the compressed sentences are acceptable as-is or nearly as-is. If deployed in a setting where the necessary minor edits are technically easily carried out, it would seem feasible that the sentence compression would lead to a streamlining of the subtitling process and subsequent cost reduction.

The lack of training data is an often cited problem for sentence compression. We have shown that subtitling data is a good source for sentence compression method development, even though non-trivial pre-processing is necessary to align the textual corpora and produce suitable training data. With the increasing pressure on the availability of subtitling and textual transcriptions, this task represents an important use case for text compression and increases the chance that such data can become available through industry collaboration.

There are many future work opportunities. The method currently does not take into account context beyond a single sentence. We thus lose the opportunity to model whether a particular sentence element has been discussed in the preceding sentence and may be pruned with a higher likelihood. There is also room for improvement in ensuring the grammaticality of the output. Others have used for instance language models to improve the grammaticality and fluency of the output. Modelling subcategorization frames could also be applied for this purpose.

Studying the data, we have noticed that often long words are replaced with their shorter synonyms to compress the sentence without any loss of information. Finding shorter synonyms and learning to substitute words and phrases would be very helpful for the sentence compression task, possibly applying the recent advancements in vector space representations and modelling phrase compositionality.

Acknowledgments

This work was supported by the Kone Foundation. Computational resources were provided by CSC – IT Center for Science. We thank Simo Vihjanen from Lingsoft Inc. for the data and the overall problem setting.

References

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

HORI Chiori and Sadaoki Furui. 2004. Speech summarization: An approach through word extraction and a method for evaluation. *IEICE TRANSACTIONS on Information and Systems*, 87(1):15–25.

James Clarke and Mirella Lapata. 2006. Constraint-based sentence compression: An integer programming approach. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 144–151.

Trevor Cohn and Mirella Lapata. 2009. Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, 34:637–674.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*, pages 25–32.

Michel Gagnon and Lyne Da Sylva. 2005. Text summarization by sentence extraction and syntactic pruning. In *Proceedings of Computational Linguistics in the North East (CliNE05)*.

Katri Haverinen, Jenna Nyblom, Timo Viljanen, Veronika Laippala, Samuel Kohonen, Anna Missilä, Stina Ojala, Tapio Salakoski, and Filip Ginter. 2014. Building the essential resources for Finnish: The Turku Dependency Treebank. *Language Resources and Evaluation*, 48(3):493–531.

Jenna Kanerva, Juhani Luotolahti, and Filip Ginter. 2014. Turku: Broad-coverage semantic parsing with rich features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 678–682.

Kevin Knight and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107, July.

Ryan T McDonald. 2006. Discriminative sentence compression with soft syntactic evidence. In *Proceedings of the 11th conference of EACL*, pages 297–304.

Naoaki Okazaki. 2007. CRFsuite: A fast implementation of Conditional Random Fields (CRFs).

Jenine Turner and Eugene Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics(ACL)*, pages 290–297.

Anthony Viera and Joanne Garrett. 2005. Understanding interobserver agreement: The Kappa statistic. *Family Medicine*, 37(5):360–363.