An extractive supervised two-stage method for sentence compression

Dimitrios Galanis[*] and **Ion Androutsopoulos**[*+]
[*]Department of Informatics, Athens University of Economics and Business, Greece
[+]Digital Curation Unit – IMIS, Research Centre "Athena", Greece

Abstract

We present a new method that compresses sentences by removing words. In a first stage, it generates candidate compressions by removing branches from the source sentence's dependency tree using a Maximum Entropy classifier. In a second stage, it chooses the best among the candidate compressions using a Support Vector Machine Regression model. Experimental results show that our method achieves state-of-the-art performance without requiring any manually written rules.

1 Introduction

Sentence compression is the task of producing a shorter form of a single given sentence, so that the new form is grammatical and retains the most important information of the original one (Jing, 2000). Sentence compression is valuable in many applications, for example when displaying texts on small screens (Corston-Oliver, 2001), in subtitle generation (Vandeghinste and Pan, 2004), and in text summarization (Madnani et al., 2007).

People use various methods to shorten sentences, including word or phrase removal, using shorter paraphrases, and common sense knowledge. However, reasonable machine-generated sentence compressions can often be obtained by only removing words. We use the term *extractive* to refer to methods that compress sentences by only removing words, as opposed to *abstractive* methods, where more elaborate transformations are also allowed. Most of the existing compression methods are extractive (Jing, 2000; Knight and Marcu, 2002; McDonald, 2006; Clarke and Lapata, 2008; Cohn and Lapata, 2009). Although abstractive methods have also been proposed (Cohn and Lapata, 2008), and they may shed more light on how people compress sentences, they do not always manage to outperform extractive methods (Nomoto, 2009). Hence, from an engineering perspective, it is still important to investigate how extractive methods can be improved.

In this paper, we present a new extractive sentence compression method that relies on supervised machine learning.[1] In a first stage, the method generates candidate compressions by removing branches from the source sentence's dependency tree using a Maximum Entropy classifier (Berger et al., 2006). In a second stage, it chooses the best among the candidate compressions using a Support Vector Machine Regression (SVR) model (Chang and Lin, 2001). We show experimentally that our method compares favorably to a state-of-the-art extractive compression method (Cohn and Lapata, 2007; Cohn and Lapata, 2009), without requiring any manually written rules, unlike other recent work (Clarke and Lapata, 2008; Nomoto, 2009). In essence, our method is a two-tier over-generate and select (or rerank) approach to sentence compression; similar approaches have been adopted in natural language generation and parsing (Paiva and Evans, 2005; Collins and Koo, 2005).

2 Related work

Knight and Marcu (2002) presented a noisy channel sentence compression method that uses a language model $P(y)$ and a channel model $P(x|y)$, where x

[1] An implementation of our method will be freely available from http://nlp.cs.aueb.gr/software.html

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 885–893,
Los Angeles, California, June 2010.

is the source sentence and y the compressed one. $P(x|y)$ is calculated as the product of the probabilities of the parse tree tranformations required to expand y to x. The best compression of x is the one that maximizes $P(x|y) \cdot P(y)$, and it is found using a noisy channel decoder. In a second, alternative method Knight and Marcu (2002) use a tree-to-tree transformation algorithm that tries to rewrite directly x to the best y. This second method uses C4.5 (Quinlan, 1993) to learn when to perform tree rewriting actions (e.g., dropping subtrees, combining subtrees) that transform larger trees to smaller trees. Both methods were trained and tested on data from the Ziff-Davis corpus (Knight and Marcu, 2002), and they achieved very similar grammaticality and meaning preservation scores, with no statistically significant difference. However, their compression rates (counted in words) were very different: 70.37% for the noisy-channel method and 57.19% for the C4.5-based one.

McDonald (2006) ranks each candidate compression using a function based on the dot product of a vector of weights with a vector of features extracted from the candidate's n-grams, POS tags, and dependency tree. The weights were learnt from the Ziff-Davis corpus. The best compression is found using a Viterbi-like algorithm that looks for the best sequence of source words that maximizes the scoring function. The method outperformed Knight and Marcu's (2002) tree-to-tree method in grammaticality and meaning preservation on data from the Ziff-Davis corpus, with a similar compression rate.

Clarke and Lapata (2008) presented an unsupervised method that finds the best compression using Integer Linear Programming (ILP). The ILP obejctive function takes into account a language model that indicates which n-grams are more likely to be deleted, and a significance model that shows which words of the input sentence are important. Manually defined constraints (in effect, rules) that operate on dependency trees indicate which syntactic constituents can be deleted. This method outperformed McDonald's (2006) in grammaticality and meaning preservation on test sentences from Edinburgh's "written" and "spoken" corpora.[2] However, the compression rates of the two systems were different (72.0% vs. 63.7% for McDonald's method, both on the written corpus).

We compare our method against Cohn and Lapata's T3 system (Cohn and Lapata, 2007; Cohn and Lapata, 2009), a state-of-the-art extractive sentence compression system that learns parse tree transduction operators from a parallel extractive corpus of source-compressed trees. T3 uses a chart-based decoding algorithm and a Structured Support Vector Machine (Tsochantaridis et al., 2005) to learn to select the best compression among those licensed by the operators learnt.[3] T3 outperformed McDonald's (2006) system in grammaticality and meaning preservation on Edinburgh's "written" and "spoken" corpora, achieving comparable compression rates (Cohn and Lapata, 2009). Cohn and Lapata (2008) have also developed an abstractive version of T3, which was reported to outperform the original, extractive T3 in meaning preservation; there was no statistically significant difference in grammaticality.

Finally, Nomoto (2009) presented a two-stage extractive method. In the first stage, candidate compressions are generated by chopping the source sentence's dependency tree. Many ungrammatical compressions are avoided using hand-crafted drop-me-not rules for dependency subtrees. The candidate compressions are then ranked using a function that takes into account the inverse document frequencies of the words, and their depths in the source dependency tree. Nomoto's extractive method was reported to outperform Cohn and Lapata's abstractive version of T3 on a corpus collected via RSS feeds. Our method is similar to Nomoto's, in that it uses two stages, one that chops the source dependency tree generating candidate compressions, and one that ranks the candidates. However, we experimented with more elaborate ranking models, and our method does not employ any manually crafted rules.

3 Our method

As already mentioned, our method first generates candidate compressions, which are then ranked. The candidate compressions generator operates by removing branches from the dependency tree of the

[2] See http://homepages.inf.ed.ac.uk/s0460084/data/.

[3] T3 appears to be the only previous sentence compression method whose implementation is publicly available; see http://www.dcs.shef.ac.uk/∼tcohn/t3/.

input sentence (figure 1); this stage is discussed in section 3.1 below. We experimented with different ranking functions, discussed in section 3.2, which use features extracted from the source sentence s and the candidate compressions $c_1, \ldots, c_k$.

3.1 Generating candidate compressions

Our method requires a parallel training corpus consisting of sentence-compression pairs $\langle s, g \rangle$. The compressed sentences g must have been formed by only deleting words from the corresponding source sentences s. The $\langle s, g \rangle$ training pairs are used to estimate the propability that a dependency edge e of a dependency tree T_s of an input sentence s is retained or not in the dependency tree T_g of the compressed sentence g. More specifically, we want to estimate the probabilities $P(X_i|context(e_i))$ for every edge e_i of T_s, where X_i is a variable that can take one of the following three values: not_del, for not deleting e_i; del_u for deleting e_i along with its head; and del_l for deleting e along with its modifier. The head (respectively, modifier) of e_i is the node e_i originates from (points to) in the dependency tree. $context(e_i)$ is a set of features that represents e_i's local context in T_s, as well as the local context of the head and modifier of e_i in s.

The propabilities above can be estimated using the Maximum Entropy (ME) framework (Berger et al., 2006), a method for learning the distribution $P(X|V)$ from training data, where X is discrete-valued variable and $V = \langle V_1, \ldots, V_n \rangle$ is a real or discrete-valued vector. Here, $V = context(e_i)$ and $X = X_i$. We use the following features in V:

- The label of the dependency edge e_i, as well as the POS tag of the head and modifier of e_i.
- The entire head-label-modifier triple of e_i. This feature overlaps with the previous two features, but it is common in ME models to use feature combinations as additional features, since they may indicate a category more strongly than the individual initial features.[4]
- The POS tag of the father of e_i's head, and the label of the dependency that links the father to e_i's head.
- The POS tag of each one of the three previous and the three following words of e_i's head and modifier in s (12 features).
- The POS tag bi-grams of the previous two and the following two words of e_i's head and modifier in s (4 features).
- Binary features that show which of the possible labels occur (or not) among the labels of the edges that have the same head as e_i in T_s (one feature for each possible dependency label).
- Two binary features that show if the subtree rooted at the modifier of e_i or e_i's *uptree* (the rest of the tree, when e_i's subtree is removed) contain an important word. A word is considered important if it appears in the document s was drawn from significantly more often than in a background corpus. In summarization, such words are called signature terms and are thought to be descriptive of the input; they can be identified using the log-likelihood ratio λ of each word (Lin and Hovy, 2000; Gupta et al., 2007).

For each dependency edge e_i of a source training sentence s, we create a training vector V with the above features. If e_i is retained in the dependency tree of the corresponding compressed sentence g in the corpus, V is assigned the category not_del. If e_i is not retained, it is assigned the category del_l or del_u, depending on whether the head (as in the `ccomp` of "said" in Figure 1) or the modifier (as in the `dobj` of "attend") of e_i has also been removed. When the modifier of an edge is removed, the entire subtree rooted at the modifier is removed, and similarly for the uptree, when the head is removed. We do not create training vectors for the edges of the removed subtree of a modifier or the edges of the removed uptree of a head.

Given an input sentence s and its dependency tree T_s, the candidate compressions generator produces the candidate compressed sentences $c_1, \ldots, c_n$ by deleting branches of T_s and putting the remaining words of the dependency tree in the same order as in s. The candidates $c_1, \ldots, c_n$ correspond to possible assignments of values to the X_i variables (recall that $X_i = not\_del|del\_l|del\_u$) of the edges e_i of T_s.

[4] http://nlp.stanford.edu/pubs/maxent-tutorial-slides.pdf.

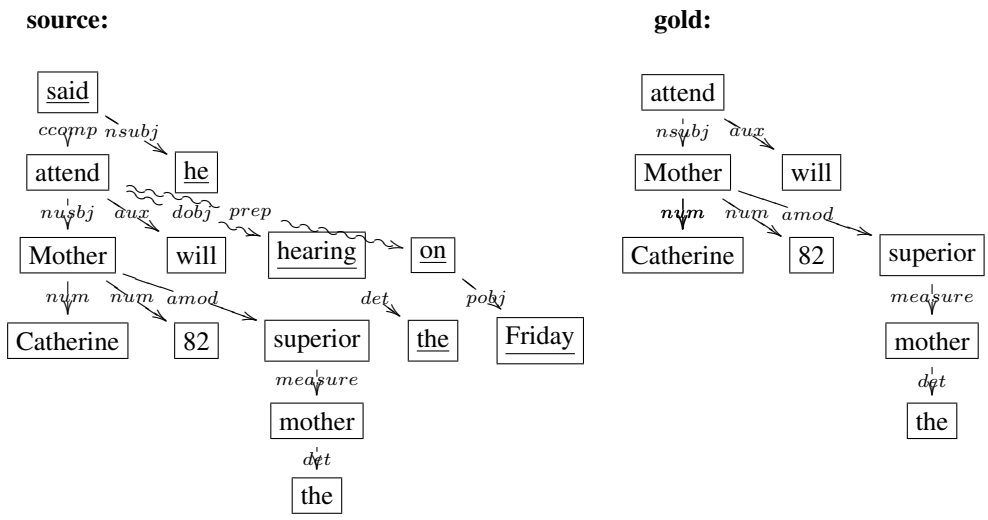

Figure 1: Dependency trees of a source sentence and its compression by a human (taken from Edinburgh's "written" corpus). The source sentence is: "Mother Catherine, 82, the mother superior, will attend the hearing on Friday, he said." The compressed one is: "Mother Catherine, 82, the mother superior, will attend." Deleted edges and words are shown curled and underlined, respectively.

Hence, there are at most 3^{m-1} candidate compressions, where m is the number of words in s. This is a large number of candidates, even for modestly long input sentences. In practice, the candidates are fewer, because del_l removes an entire subtree and del_u an entire uptree, and we do not need to make decisions X_i about the edges of the deleted subtrees and uptrees. To reduce the number of candidates further, we ignore possible assignments that contain decisions $X_i = x$ to which the ME model assigns probabilities below a threshold t; i.e., the ME model is used to prune the space of possible assignments.

When generating the possible assignments to the X_i variables, we examine the edges e_i of T_s in a top-down breadth-first manner. In the source tree of Figure 1, for example, we first consider the edges of "said"; the left-to-right order is random, but let us assume that we consider first the `ccomp` edge. There are three possible actions: retain the edge (not_del), remove it along with the head "said" (del_u), or remove it along with the modifier "attend" and its subtree (del_l). If the ME model assigns a low probability to one of the three actions, that action is ignored. For each one of the (remaining) actions, we obtain a new form of T_s, and we continue to consider its (other) edges. We process the edges in a top-down fashion, because the ME model allows del_l actions much more often than del_u actions, and when del_l actions are performed near the root of T_s, they prune large parts of the space of possible assignments to the X_i variables. Some of the candidate compressions that were generated for an input sentence by setting $t = 0.2$ are shown in Table 1, along with the gold (human-authored) compression.

3.2 Ranking candidate compressions

Given that we now have a method that generates candidate compressions $c_1, \ldots, c_k$ for a sentence s, we need a function $F(c_i|s)$ that will rank the candidate compressions. Many of them are ungrammatical and/or do not convey the most important information of s. $F(c_i|s)$ should help us select a short candidate that is grammatical and retains the most important information of s.

3.2.1 Grammaticality and importance rate

A simple way to rank the candidate compressions is to assign to each one a score intended to measure its *grammaticality* and *importance rate*. By grammaticality, $Gramm(c_i)$, we mean how grammatically well-formed candidate c_i is. A common way to obtain such a measure is to use an n-gram lan-

s: Then last week a second note, in the same handwriting, informed Mrs Allan that the search was on the wrong side of the bridge.
g: Last week a second note informed Mrs Allan the search was on the wrong side of the bridge.
c_1: Last week a second note informed Mrs Allan that the search was on the side. c_2: Last week a second note informed Mrs Allan that the search was. c_3: Last week a second note informed Mrs Allan the search was on the wrong side of the bridge. c_4: Last week in the same handwriting informed Mrs Allan the search was on the wrong side of the bridge.

Table 1: A source sentence s, its gold (human authored) compression g, and candidate compressions $c_1, \ldots, c_4$.

guage model trained on a large background corpus. However, language models tend to assign smaller probabilities to longer sentences; therefore they favor short sentences, but not necessarily the most appropriate compressions. To overcome this problem, we follow Cordeiro et al. (2009) and normalize the score of a trigram language model as shown below, where $w_1, \ldots, w_m$ are the words of candidate c_i.

$$Gramm(c_i) = \log P_{LM}(c_i)^{1/m} = (1/m) \cdot \log(\prod_{j=1}^{m} P(w_j|w_{j-1}, w_{j-2})) \quad (1)$$

The importance rate $ImpRate(c_i|s)$, defined below, estimates how much information of the original sentence s is retained in candidate c_i. $tf(w_i)$ is the term frequency of w_i in the document that contained ξ ($\xi = c_i, s$), and $idf(w_i)$ is the inverse document frequency of w_i in a background corpus. We actually compute $idf(w_i)$ only for nouns and verbs, and set $idf(w_i) = 0$ for other words.

$$ImpRate(c_i|s) = Imp(c_i)/Imp(s) \quad (2)$$

$$Imp(\xi) = \sum_{w_i \in \xi} tf(w_i) \cdot idf(w_i) \quad (3)$$

The ranking $F(c|s)$ is then defined as a linear combination of grammaticality and importance rate:

$$F(c_i|s) = \lambda \cdot Gramm(c_i) + (1 - \lambda) \cdot \cdot ImpRate(c_i|s) - \alpha \cdot CR(c_i|s) \quad (4)$$

A compression rate penalty factor $CR(c_i|s) = |c|/|s|$ is included, to bias our method towards generating shorter or longer compressions; $|\cdot|$ denotes word length in words (punctuation is ignored). We explain how the weigths λ, α are tuned in following sections. We call LM-IMP the configuration of our method that uses the ranking function of equation 4.

3.2.2 Support Vector Regression

A more sophisticated way to select the best compression is to train a Support Vector Machines Regression (SVR) model to assign scores to feature vectors, with each vector representing a candidate compression. SVR models (Chang and Lin, 2001) are trained using l training vectors $(x_1, y_1), \ldots, (x_l, y_l)$, where $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, and learn a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that generalizes the training data. In our case, x_i is a feature vector representing a candidate compression c_i, and y_i is a score indicating how good a compression c_i is. We use 98 features:

- $Gramm(c_i)$ and $ImpRate(c_i|s)$, as above.
- 2 features indicating the ratio of important and unimportant words of s, identified as in section 3.1, that were deleted.
- 2 features that indicate the average depth of the deleted and not deleted words in the dependency tree of s.
- 92 features that indicate which POS tags appear in s and how many of them were deleted in c_i. For every POS tag label, we use two features, one that shows how many POS tags of that label are contained in s and one that shows how many of these POS tags were deleted in c_i.

To assign a regression score y_i to each training vector x_i, we experimented with the following functions that measure how similar c_i is to the gold compression g, and how grammatical c_i is.

- Grammatical relations overlap: In this case, y_i is the F_1-score of the dependencies of c_i against those of the gold compression g. This measure has been shown to correlate well with human judgements (Clarke and Lapata, 2006). As in

the ranking function of section 3.2.1, we add a compression rate penalty factor.

$$y_i = F_1(d(c_i)), d(g)) - \alpha \cdot CR(c_i|s) \quad (5)$$

$d(\cdot)$ denotes the set of dependencies. We call SVR-F1 the configuration of our system that uses equation 5 to rank the candidates.

- Tokens accuracy and grammaticality: Tokens accuracy, $TokAcc(c_i|s, g)$, is the percentage of tokens of s that were correctly retained or removed in c_i; a token was correctly retained or removed, if it was also retained (or removed) in the gold compression g. To calculate $TokAcc(c_i|s, g)$, we need the word-to-word alignment of s to g, and s to c_i. These alignments were obtained as a by-product of computing the corresponding (word) edit distances. We also want the regression model to favor grammatical compressions. Hence, we use a linear combination of tokens accuracy and grammaticality of c_i:

$$y_i = \lambda \cdot TokAcc(c_i|s, g) + (1 - \lambda) \cdot Gramm(c_i) - \alpha \cdot CR(c_i|s) \quad (6)$$

Again, we add a compression rate penalty, to be able to generate shorter or longer compressions. We call SVR-TOKACC-LM the configuration of our system that uses equation 6.

4 Baseline and T3

As a baseline, we use a simple algorithm based on the ME classifier of section 3.1. The baseline produces a single compression c for every source sentence s by considering sequentially the edges e_i of s's dependency tree in a random order, and performing at each e_i the single action (not_del, del_u, or del_l) that the ME model considers more probable; the words of the chopped dependency tree are then put in the same order as in s. We call this system Greedy-Baseline.

We compare our method against the extractive version of T3 (Cohn and Lapata, 2007; Cohn and Lapata, 2009), a state-of-the-art sentence compression system that applies sequences of transduction operators to the syntax trees of the source sentences. The available tranduction operators are learnt from the syntax trees of a set of source-gold pairs. Every operator transforms a subtree α to a subtree γ, rooted at symbols X and Y, respectively.

To find the best sequence of transduction operators that can be applied to a source syntax tree, a chart-based dynamic programming decoder is used, which finds the best scoring sequence q^*:

$$q^* = \arg\max_q score(q; w) \quad (7)$$

where $score(q; w)$ is the dot product $\langle \Psi(q), w \rangle$. $\Psi(q)$ is a vector-valued feature function, and w is a vector of weights learnt using a Structured Support Vector Machine (Tsochantaridis et al., 2005).

$\Psi(q)$ consists of: (i) the log-probability of the resulting candidate, as returned by a tri-gram language model; and (ii) features that describe how the operators of q are applied, for example the number of the terminals in each operator's α and γ subtrees, the POS tags of the X and Y roots of α and γ etc.

5 Experiments

We used Stanford's parser (de Marneffe et al., 2006) and ME classifier (Manning et al., 2003).[5] For the (trigram) language model, we used SRILM with modified Kneser-Ney smoothing (Stolcke, 2002).[6] The language model was trained on approximately 4.5 million sentences of the TIPSTER corpus. To obtain $idf(w_i)$ values, we used approximately 19.5 million verbs and nouns from the TIPSTER corpus.

T3 requires the syntax trees of the source-gold pairs in Penn Treebank format, as well as a trigram language model. We obtained T3's trees using Stanford's parser, as in our system, unlike Cohn and Lapata (2009) that use Bikel's (2002) parser. The language models in T3 and our system are trained on the same data and with the same options used by Cohn and Lapata (2009). T3 also needs a word-to-word alignment of the source-gold pairs, which was obtained by computing the edit distance, as in Cohn and Lapata (2009) and SVR-TOKACC-LM.

We used Edinburgh's "written" sentence compression corpus (section 2), which consists of source-gold pairs (one gold compression per source

[5] Both available from http://nlp.stanford.edu/.

[6] See http://www.speech.sri.com/projects/srilm/.

sentence). The gold compressions were created by deleting words. We split the corpus in 3 parts: 1024 training, 324 development, and 291 testing pairs.

5.1 Best configuration of our method

We first evaluated experimentally the three configurations of our method (LM-IMP, SVR-F1, SVR-TOKACC-LM), using the $F1$-measure of the dependencies of the machine-generated compressions against those of the gold compressions as an automatic evaluation measure. This measure has been shown to correlate well with human judgements (Clarke and Lapata, 2006).

In all three configurations, we trained the ME model of section 3.1 on the dependency trees of the source-gold pairs of the training part of the corpus. We then used the trained ME classifier to generate the candidate compressions of each source sentence of the training part. We set $t = 0.2$, which led to at most 10,000 candidates for almost every source sentence. We kept up to 1.000 candidates for each source sentence, and we selected randonly approximately 10% of them, obtaining 18,385 candidates, which were used to train the two SVR configurations; LM-IMP requires no training.

To tune the λ parameters of LM-IMP and SVR-TOKACC-LM in equations 4 and 6, we initially set $\alpha = 0$ and we experimented with different values of λ. For each one of the two configurations and for every different λ value, we computed the average compression rate of the machine-generated compressions on the development set. In the rest of the experiments, we set λ to the value that gave an average compression rate approximatelly equal to that of the gold compressions of the training part.

We then experimented with different values of α in all three configurations, in equations 4–6, to produce smaller or longer compression rates. The α parameter provides a uniform mechanism to fine-tune the compression rate in all three configurations, even in SVR-F1 that has no λ. The results on the development part are shown in Figure 2, along with the baseline's results. The baseline has no parameters to tune; hence, its results are shown as a single point. Both SVR models outperform LM-IMP, which in turn outperforms the baseline. Also, SVR-TOKACC-LM performs better or as well as SVR-F1 for all compression rates. Note, also, that the perfomance of the two SVR configurations might be improved further by using more training examples, whereas LM-IMP contains no learning component.

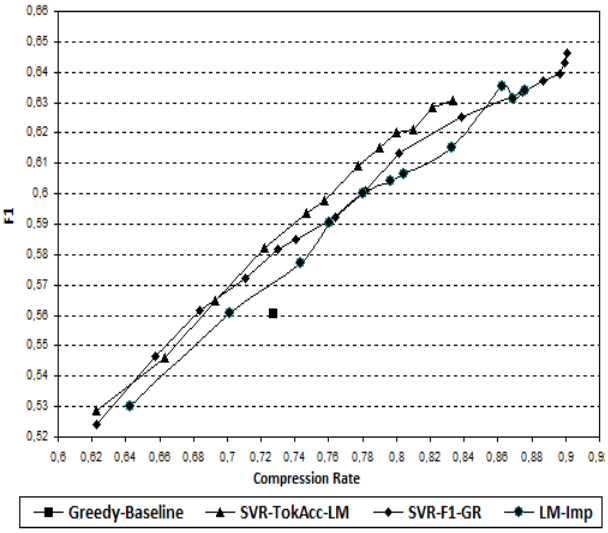

Figure 2: Evaluation results on the development set.

5.2 Our method against T3

We then evaluated the best configuration of our method (SVR-TOKACC-LM) against T3, both automatically ($F1$-measure) and with human judges. We trained both systems on the training set of the corpus. In our system, we used the same λ value that we had obtained from the experiments of the previous section. We then varied the values of our system's α parameter to obtain approximately the same compression rate as T3.

For the evaluation with the human judges, we selected randomly 80 sentences from the test part. For each source sentence s, we formed three pairs, containing s, the gold compression, the compression of SVR-TOKACC-LM, and the compression of T3, repsectively, 240 pairs in total. Four judges (graduate students) were used. Each judge was given 60 pairs in a random sequence; they did not know how the compressed sentenes were obtained and no judge saw more than one compression of the same source sentence. The judges were told to rate (in a scale from 1 to 5) the compressed sentences in terms of grammaticality, meaning preservation, and overall quality. Their average judgements are shown in Table 2, where the $F1$-scores are also included. Cohn and Lapata (2009) have reported very similar scores

for T3 on a different split of the corpus (F1: 49.48%, CR: 61.09%).

system	G	M	Ov	F1 (%)	CR (%)
T3	3.83	3.28	3.23	47.34	59.16
SVR	4.20	3.43	3.57	52.09	59.85
gold	4.73	4.27	4.43	100.00	78.80

Table 2: Results on 80 test sentences. G: grammaticality, M: meaning preservation, Ov: overall score, CR: compression rate, SVR: SVR-TOKACC-LM.

Our system outperforms T3 in all evaluation measures. We used Analysis of Variance (ANOVA) followed by post-hoc Tukey tests to check whether the judge ratings differ significantly ($p < 0.1$); all judge ratings of gold compressions are significantly different from T3's and those of our system; also, our system differs significantly from T3 in grammaticality, but not in meaning preservation and overall score. We also performed Wilcoxon tests, which showed that the difference in the $F1$ scores of the two systems is statistically significant ($p < 0.1$) on the 80 test sentences. Table 3 shows the $F1$ scores and the average compression rates for all 291 test sentences. Both systems have comparable compression rates, but again our system outperforms T3 in $F1$, with a statistically significant difference ($p < 0.001$).

system	$F1$	CR
SVR-TOKACC-LM	53.75	63.72
T3	47.52	64.16

Table 3: $F1$ scores on the entire test set.

Finally, we computed the Pearson correlation r of the overall (Ov) scores that the judges assigned to the machine-generated compressions with the corresponding $F1$ scores. The two measures were found to corellate reliably ($r = 0.526$). Similar results have been reported (Clarke and Lapata, 2006) for Edinburgh's "spoken" corpus ($r = 0.532$) and the Ziff-Davis corpus ($r = 0.575$).

6 Conclusions and future work

We presented a new two-stage extractive method for sentence compression. The first stage generates candidate compressions by removing or not edges from the source sentence's dependency tree; an ME model is used to prune unlikely edge deletion or non-deletions. The second stage ranks the candidate compressions; we experimented with three ranking models, achieving the best results with an SVR model trained with an objective function that combines token accuracy and a language model. We showed experimentally, via automatic evaluation and with human judges, that our method compares favorably to a state-of-the-art extractive system. Unlike other recent approaches, our system uses no hand-crafted rules. In future work, we plan to support more complex tranformations, instead of only removing words and experiment with different sizes of training data.

The work reported in this paper was carried out in the context of project INDIGO, where an autonomous robotic guide for museum collections is being developed. The guide engages the museum's visitors in spoken dialogues, and it describes the exhibits that the visitors select by generating textual descriptions, which are passed on to a speech synthesizer. The texts are generated from logical facts stored in an ontology (Galanis et al., 2009) and from canned texts; the latter are used when the corresponding information is difficult to encode in symbolic form (e.g., to store short stories about the exhibits). The descriptions of the exhibits are tailored depending on the type of the visitor (e.g., child vs. adult), and an important tailoring aspect is the generation of shorter or longer descriptions. The parts of the descriptions that are generated from logical facts can be easily made shorter or longer, by conveying fewer or more facts. The methods of this paper are used to automatically shorten the parts of the descriptions that are canned texts, instead of requiring multiple (shorter and longer) hand-written versions of the canned texts.

Acknowledgements

This work was carried out in INDIGO, an FP6 IST project funded by the European Union, with additional funding provided by the Greek General Secretariat of Research and Technology.[7]

[7]Consult `http://www.ics.forth.gr/indigo/`.

References

A.L. Berger, S.A. Della Pietra, and V.J. Della Pietra. 2006. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

D. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of the 2nd International Conference on Human Language Technology Research*, pages 24–27.

C.C Chang and C.J Lin. 2001. LIBSVM: a library for Support Vector Machines. Technical report. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

J. Clarke and M. Lapata. 2006. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *Proceedings of COLING*, pages 377–384.

J. Clarke and M. Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Artificial Intelligence Research*, 1(31):399–429.

T. Cohn and M. Lapata. 2007. Large margin synchronous generation and its application to sentence compression. In *Proceedings of EMNLP-CoNLL*, pages 73–82.

T. Cohn and M. Lapata. 2008. Sentence compression beyond word deletion. In *Proceedings of COLING*, pages 137–144.

T. Cohn and M. Lapata. 2009. Sentence compression as tree to tree tranduction. *Artificial Intelligence Research*, 34:637–674.

M. Collins and T. Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69.

J. Cordeiro, G. Dias, and P. Brazdil. 2009. Unsupervised induction of sentence compression rules. In *Proceedings of the ACL Workshop on Language Generation and Summarisation*, pages 391–399.

S. Corston-Oliver. 2001. Text compaction for display on very small screens. In *Proceedings of the NAACL Workshop on Automatic Summarization*, pages 89–98.

M.C. de Marneffe, B. MacCartney, and C. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 449–454.

Dimitrios Galanis, George Karakatsiotis, Gerasimos Lampouras, and Ion Androutsopoulos. 2009. An open-source natural language generator for OWL ontologies and its use in protege and second life. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 17–20, Athens, Greece, April. Association for Computational Linguistics.

S. Gupta, A. Nenkova, and D. Jurafsky. 2007. Measuring importance and query relevance in topic-focused multi-document summarization. In *Proceedings of ACL*, pages 193–196.

H. Jing. 2000. Sentence reduction for automatic text summarization. In *Proceedings of ANLP*, pages 310–315.

K. Knight and D. Marcu. 2002. Summarization beyond sentence extraction: A probalistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107.

C.W. Lin and E. Hovy. 2000. The automated acquisition of topic signatures for text summarization. In *Proceedings of ACL*, pages 495–501.

N. Madnani, D. Zajic, B. Dorr, N. F. Ayan, and J. Lin. 2007. Multiple alternative sentence compressions for automatic text summarization. In *Proceedings of DUC*.

C. D. Manning, D. Klein, and C. Manning. 2003. Optimization, maxent models, and conditional estimation without magic. In *tutorial notes of HLT-NAACL 2003 and ACL 2003*.

R. McDonald. 2006. Discriminative sentence compression with soft syntactic constraints. In *Proceedings of EACL*, pages 297–304.

T. Nomoto. 2009. A comparison of model free versus model intensive approaches to sentence compression. In *Proceedings of EMNLP*, pages 391–399.

D. Paiva and R. Evans. 2005. Empirically-based control of natural language generation. In *Proceedings of ACL*.

J. R. Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

A. Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2005. Support vector machine learning for independent and structured output spaces. *Machine Learning Research*, 6:1453–1484.

V. Vandeghinste and Y. Pan. 2004. Sentence compression for automated subtitling: A hybrid approach. In *Proceedings of the ACL Workshop "Text Summarization Branches Out"*, pages 89–95.