Statistics-Based Summarization — Step One: Sentence Compression

Kevin Knight and Daniel Marcu
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
{knight,marcu}@isi.edu

Abstract

When humans produce summaries of documents, they do not simply extract sentences and concatenate them. Rather, they create new sentences that are grammatical, that cohere with one another, and that capture the most salient pieces of information in the original document. Given that large collections of text/abstract pairs are available online, it is now possible to envision algorithms that are trained to mimic this process. In this paper, we focus on sentence compression, a simpler version of this larger challenge. We aim to achieve two goals simultaneously: our compressions should be grammatical, and they should retain the most important pieces of information. These two goals can conflict. We devise both noisy-channel and decision-tree approaches to the problem, and we evaluate results against manual compressions and a simple baseline.

Introduction

Most of the research in automatic summarization has focused on extraction, i.e., on identifying the most important clauses/sentences/paragraphs in texts (see (Mani & Maybury 1999) for a representative collection of papers). However, determining the most important textual segments is only half of what a summarization system needs to do because, in most cases, the simple catenation of textual segments does not yield coherent outputs. Recently, a number of researchers have started to address the problem of generating coherent summaries: McKeown et al. (1999), Barzilay et al. (1999), and Jing and McKeown (1999) in the context of multidocument summarization; Mani et al. (1999) in the context of revising single document extracts; and Witbrock and Mittal (1999) in the context of headline generation.

The approach proposed by Witbrock and Mittal (1999) is the only one that applies a probabilistic model trained directly on ⟨*Headline, Document*⟩ pairs. However, this model has yet to scale up to generating multiple-sentence abstracts as well as well-formed, grammatical sentences. All other approaches employ sets of manually written or semi-automatically derived rules for deleting information that is redundant, compressing long sentences into shorter ones, aggregating sentences, repairing reference links, etc.

Our goal is also to generate coherent abstracts. However, in contrast with the above work, we intend to eventually use ⟨*Abstract, Text*⟩ tuples, which are widely available, in order to automatically learn how to rewrite *Texts* as coherent *Abstracts*. In the spirit of the work in the statistical MT community, which is focused on sentence-to-sentence translations, we also decided to focus first on a simpler problem, that of *sentence compression*. We chose this problem for two reasons:

- First, the problem is complex enough to require the development of sophisticated compression models: Determining what is important in a sentence and determining how to convey the important information grammatically, using only a few words, is just a scaled down version of the text summarization problem. Yet, the problem is simple enough, since we do not have to worry yet about discourse related issues, such as coherence, anaphors, etc.
- Second, an adequate solution to this problem has an immediate impact on several applications. For example, due to time and space constraints, the generation of TV captions often requires only the most important parts of sentences to be shown on a screen (Linke-Ellis 1999; Robert-Ribes *et al.* 1999). A good sentence compression module would therefore have an impact on the task of automatic caption generation. A sentence compression module can also be used to provide audio scanning services for the blind (Grefenstette 1998). In general, since all systems aimed at producing coherent abstracts implement manually written sets of sentence compression rules (McKeown *et al.* 1999; Mani, Gates, & Bloedorn 1999; Barzilay, McKeown, & Elhadad 1999), it is likely that a good sentence compression module would impact the overall quality of these systems as well. This becomes particularly important for text genres that use long sentences.

In this paper, we present two approaches to the *sentence compression* problem. Both take as input a sequence of words $W = w_1, w_2, \ldots, w_n$ (one sentence). An algorithm may drop any subset of these words. The words that remain (order unchanged) form a compression. There are 2^n compressions to choose from—some are reasonable, most are not. Our first approach develops a probabilistic noisy-channel model for sentence compression. The second approach develops a decision-based, deterministic model.

A noisy-channel model for sentence compression

This section describes a probabilistic approach to the compression problem. In particular, we adopt the *noisy channel* framework that has been relatively successful in a number of other NLP applications, including speech recognition (Jelinek 1997), machine translation (Brown *et al.* 1993), part-of-speech tagging (Church 1988), transliteration (Knight & Graehl 1998), and information retrieval (Berger & Lafferty 1999).

In this framework, we look at a long string and imagine that (1) it was originally a short string, and then (2) someone added some additional, optional text to it. Compression is a matter of identifying the original short string. It is not critical whether or not the "original" string is real or hypothetical. For example, in statistical machine translation, we look at a French string and say, "This was originally English, but someone added 'noise' to it." The French may or may not have been translated from English originally, but by removing the noise, we can hypothesize an English source—and thereby translate the string. In the case of compression, the noise consists of optional text material that pads out the core signal. For the larger case of text summarization, it may be useful to imagine a scenario in which a news editor composes a short document, hands it to a reporter, and tells the reporter to "flesh it out" ... which results in the article we read in the newspaper. As summarizers, we may not have access to the editor's original version (which may or may not exist), but we can guess at it—which is where probabilities come in.

As in any noisy channel application, we must solve three problems:

- Source model. We must assign to every string s a probability $\mathrm{P}(s)$, which gives the chance that s is generated as an "original short string" in the above hypothetical process. For example, we may want $\mathrm{P}(s)$ to be very low if s is ungrammatical.
- Channel model. We assign to every pair of strings $\langle s, t \rangle$ a probability $\mathrm{P}(t \mid s)$, which gives the chance that when the short string s is expanded, the result is the long string t. For example, if t is the same as s except for the extra word "not," then we may want $\mathrm{P}(t \mid s)$ to be very low. The word "not" is not optional, additional material.
- Decoder. When we observe a long string t, we search for the short string s that maximizes $\mathrm{P}(s \mid t)$. This is equivalent to searching for the s that maximizes $\mathrm{P}(s) \cdot P(t \mid s)$.

It is advantageous to break the problem down this way, as it decouples the somewhat independent goals of creating a short text that (1) looks grammatical, and (2) preserves important information. It is easier to build a channel model that focuses exclusively on the latter, without having to worry about the former. That is, we can specify that a certain substring may represent unimportant information, but we do not need to worry that deleting it will result in an ungrammatical structure. We leave that to the source model, which worries exclusively about well-formedness. In fact, we can make use of extensive prior work in source language modeling for speech recognition, machine translation, and natural language generation. The same goes for actual compression ("decoding" in noisy-channel jargon)—we can re-use generic software packages to solve problems in all these application domains.

Statistical Models

In the experiments we report here, we build very simple source and channel models. In a departure from the above discussion and from previous work on statistical channel models, we assign probabilities $\mathrm{P}_{tree}(s)$ and $\mathrm{P}_{expand\_tree}(t \mid s)$ to trees rather than strings. In decoding a new string, we first parse it into a large tree t (using Collins' parser (1997)), and we then hypothesize and rank various small trees.

Good source strings are ones that have both (1) a normal-looking parse tree, and (2) normal-looking word pairs. $\mathrm{P}_{tree}(s)$ is a combination of a standard probabilistic context-free grammar (PCFG) score, which is computed over the grammar rules that yielded the tree s, and a standard word-bigram score, which is computed over the leaves of the tree. For example, the tree s =(S (NP John) (VP (VB saw) (NP Mary))) is assigned a score based on these factors:

$$\begin{aligned}
&\mathrm{P}_{tree}(s) = \mathrm{P}(\mathrm{TOP} \rightarrow \mathrm{S} \mid \mathrm{TOP}) \cdot \\
&\mathrm{P}(\mathrm{S} \rightarrow \mathrm{NP\ VP} \mid \mathrm{S}) \cdot \mathrm{P}(\mathrm{NP} \rightarrow \mathrm{John} \mid \mathrm{NP}) \cdot \\
&\mathrm{P}(\mathrm{VP} \rightarrow \mathrm{VB\ NP} \mid \mathrm{VP}) \cdot \mathrm{P}(\mathrm{VP} \rightarrow \mathrm{saw} \mid \mathrm{VB}) \cdot \\
&\mathrm{P}(\mathrm{NP} \rightarrow \mathrm{Mary} \mid \mathrm{NP}) \cdot \\
&\mathrm{P}(\mathrm{John} \mid \mathrm{EOS}) \cdot \mathrm{P}(\mathrm{saw} \mid \mathrm{John}) \cdot \\
&\mathrm{P}(\mathrm{Mary} \mid \mathrm{saw}) \cdot \mathrm{P}(\mathrm{EOS} \mid \mathrm{Mary})
\end{aligned}$$

Our stochastic channel model performs minimal operations on a small tree s to create a larger tree t. For each internal node in s, we probabilistically choose an *expansion template* based on the labels of the node and its children. For example, when processing the S node in the tree above, we may wish to add a prepositional phrase as a third child. We do this with probability $\mathrm{P}(\mathrm{S} \rightarrow \mathrm{NP\ VP\ PP} \mid \mathrm{S} \rightarrow \mathrm{NP\ VP})$. Or we may choose to leave it alone, with probability $\mathrm{P}(\mathrm{S} \rightarrow \mathrm{NP\ VP} \mid \mathrm{S} \rightarrow \mathrm{NP\ VP})$. After we choose an expansion template, then for each new child node introduced (if any), we grow a new subtree rooted at that node—for example (PP (P in) (NP Pittsburgh)). Any particular subtree is grown

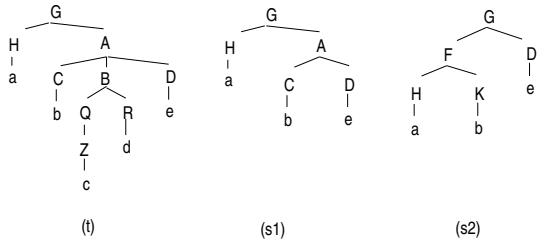

Figure 1: Examples of parse trees.

with probability given by its PCFG factorization, as above (no bigrams).

Example

In this section, we show how to tell whether one potential compression is more likely than another, according to the statistical models described above. Suppose we observe the tree t in Figure 1, which spans the string `abcde`. Consider the compression $s1$, which is shown in the same figure.

We compute the factors $\mathrm{P}_{tree}(s1)$ and $\mathrm{P}_{expand\_tree}(t \mid s1)$. Breaking this down further, the source PCFG and word-bigram factors, which describe $\mathrm{P}_{tree}(s1)$, are:

P(TOP → G | TOP)	P(H → a | H)
P(G → H A | G)	P(C → b | C)
P(A → C D | A)	P(D → e | D)
P(a | EOS)	**P(e | b)**
P(b | a)	P(EOS | e)

The channel expansion-template factors and the channel PCFG (new tree growth) factors, which describe $\mathrm{P}_{expand\_tree}(t \mid s1)$, are:

P(G → H A | G → H A)	
P(A → C B D | A → C D)	
P(B → Q R | B)	P(Z → c | Z)
P(Q → Z | Q)	P(R → d | R)

A different compression will be scored with a different set of factors. For example, consider a compression of t that leaves t completely untouched. In that case, the source costs $\mathrm{P}_{tree}(t)$ are:

P(TOP → G | TOP)	P(H → a | H)	P(a | EOS)
P(G → H A | G)	P(C → b | C)	**P(b | a)**
P(A → C D | A)	P(Z → c | Z)	**P(c | b)**
P(B → Q R | B)	P(R → d | R)	**P(d | c)**
P(Q → Z | Q)	P(D → e | D)	P(e | d)
		P(EOS | e)

The channel costs $\mathrm{P}_{expand\_tree}(t \mid t)$ are:

The documentation is typical of Epson quality: excellent.
Documentation is excellent.

All of our design goals were achieved and the delivered performance matches the speed of the underlying device.
All design goals were achieved.

Reach's E-mail product, MailMan, is a message- management system designed initially for VINES LANs that will eventually be operating system-independent.
MailMan will eventually be operating system-independent.

Although the modules themselves may be physically and/or electrically incompatible, the cable-specific jacks on them provide industry-standard connections.
Cable-specific jacks provide industry-standard connections.

Ingres/Star prices start at $2,100.
Ingres/Star prices start at $2,100.

Figure 2: Examples from our parallel corpus.

P(G → H A | G → H A)
P(A → C B D | A → C B D)
P(B → Q R | B → Q R)
P(Q → Z | Q → Z)

Now we can simply compare $\mathrm{P}_{expand\_tree}(s1 \mid t) = P_{tree}(s1) \cdot P_{expand\_tree}(t \mid s1))/P_{tree}(t)$ versus $\mathrm{P}_{expand\_tree}(t \mid t) = P_{tree}(t) \cdot P_{expand\_tree}(t \mid t))/P_{tree}(t)$ and select the more likely one. Note that $P_{tree}(t)$ and all the PCFG factors can be canceled out, as they appear in any potential compression. Therefore, we need only compare compressions of the basis of the expansion-template probabilities and the word-bigram probabilities. The quantities that differ between the two proposed compressions are boxed above. Therefore, $s1$ will be preferred over t if and only if:

$$\begin{aligned}&\mathrm{P}(\mathrm{e} \mid \mathrm{b}) \cdot \mathrm{P}(\mathrm{A} \to \mathrm{C\ B\ D} \mid \mathrm{A} \to \mathrm{C\ D}) > \\ &\mathrm{P}(\mathrm{b} \mid \mathrm{a}) \cdot \mathrm{P}(\mathrm{c} \mid \mathrm{b}) \cdot \mathrm{P}(\mathrm{d} \mid \mathrm{c}) \cdot \\ &\mathrm{P}(\mathrm{A} \to \mathrm{C\ B\ D} \mid \mathrm{A} \to \mathrm{C\ B\ D}) \cdot \\ &\mathrm{P}(\mathrm{B} \to \mathrm{Q\ R} \mid \mathrm{B} \to \mathrm{Q\ R}) \cdot \mathrm{P}(\mathrm{Q} \to \mathrm{Z} \mid \mathrm{Q} \to \mathrm{Z})\end{aligned}$$

Training Corpus

In order to train our system, we used the Ziff-Davis corpus, a collection of newspaper articles announcing computer products. Many of the articles in the corpus are paired with human written abstracts. We automatically extracted from the corpus a set of 1067 sentence pairs. Each pair consisted of a sentence $t = t_1, t_2, \ldots, t_n$ that occurred in the article and a possibly compressed version of it $s = s_1, s_2, \ldots, s_m$, which occurred in the human written abstract. Figure 2 shows a few sentence pairs extracted from the corpus.

We decided to use such a corpus because it is consistent with two desiderata specific to summarization work: (i) the human-written Abstract sentences are

grammatical; (ii) the Abstract sentences represent in a compressed form the salient points of the original newspaper Sentences. We decided to keep in the corpus uncompressed sentences as well, since we want to learn not only *how* to compress a sentence, but also *when* to do it.

Learning Model Parameters

We collect expansion-template probabilities from our parallel corpus. We first parse both sides of the parallel corpus, and then we identify corresponding syntactic nodes. For example, the parse tree for one sentence may begin (S (NP ...) (VP ...) (PP ...)) while the parse tree for its compressed version may begin (S (NP ...) (VP ...)). If these two S nodes are deemed to correspond, then we chalk up one joint event (S $\rightarrow$ NP VP, S $\rightarrow$ NP VP PP); afterwards we normalize. Not all nodes have corresponding partners; some non-correspondences are due to incorrect parses, while others are due to legitimate reformulations that are beyond the scope of our simple channel model. We use standard methods to estimate word-bigram probabilities.

Decoding

There is a vast number of potential compressions of a large tree t, but we can pack them all efficiently into a shared-forest structure. For each node of t that has n children, we

- generate $2^n - 1$ new nodes, one for each non-empty subset of the children, and
- pack those nodes so that they are referred to as a whole.

For example, consider the large tree t above. All compressions can be represented with the following forest:

G $\rightarrow$ H A	B $\rightarrow$ R	A $\rightarrow$ B C	H $\rightarrow$ a
G $\rightarrow$ H	Q $\rightarrow$ Z	A $\rightarrow$ C	C $\rightarrow$ b
G $\rightarrow$ A	A $\rightarrow$ C B D	A $\rightarrow$ B	Z $\rightarrow$ c
B $\rightarrow$ Q R	A $\rightarrow$ C B	A $\rightarrow$ D	R $\rightarrow$ d
B $\rightarrow$ Q	A $\rightarrow$ C D		D $\rightarrow$ e

We can also assign an expansion-template probability to each node in the forest. For example, to the B $\rightarrow$ Q node, we can assign P(B $\rightarrow$ Q R | B $\rightarrow$ Q). If the observed probability from the parallel corpus is zero, then we assign a small floor value of 10^{-6}. In reality, we produce forests that are much slimmer, as we only consider compressing a node in ways that are locally grammatical according to the Penn Treebank—if a rule of the type A $\rightarrow$ C B has never been observed, then it will not appear in the forest.

At this point, we want to extract a set of high-scoring trees from the forest, taking into account both expansion-template probabilities and word-bigram probabilities. Fortunately, we have such a generic extractor on hand (Langkilde 2000). This extractor was designed for a hybrid symbolic-statistical natural language generation system called Nitrogen. In that application, a rule-based component converts an abstract semantic representation into a vast number of potential English renderings. These renderings are packed into a forest, from which the most promising sentences are extracted using statistical scoring.

For our purposes, the extractor selects the trees with the best combination of word-bigram and expansion-template scores. It returns a list of such trees, one for each possible compression length. For example, for the sentence *Beyond that basic level, the operations of the three products vary*, we obtain the following "best" compressions, with negative log-probabilities shown in parentheses (smaller = more likely):

Beyond that basic level, the operations of the three products vary widely (1514588)
Beyond that level, the operations of the three products vary widely (1430374)
Beyond that basic level, the operations of the three products vary (1333437)
Beyond that level, the operations of the three products vary (1249223)
Beyond that basic level, the operations of the products vary (1181377)
The operations of the three products vary widely (939912)
The operations of the products vary widely (872066)
The operations of the products vary (748761)
The operations of products vary (690915)
Operations of products vary (809158)
The operations vary (522402)
Operations vary (662642)

Length Selection

It is useful to have multiple answers to choose from, as one user may seek a 20% compression, while another seeks a 60% compression. However, for purposes of evaluation, we want our system to be able to select a single compression. If we rely on the log-probabilities as shown above, we will almost always choose the shortest compression. (Note above, however, how the three-word compression scores better than the two-word compression, as the models are not entirely happy removing the article "the"). To create a more fair competition, we divide the log-probability by the length of the compression, rewarding longer strings. This is commonly done in speech recognition.

If we plot this normalized score against compression length, we usually observe a (bumpy) U-shaped curve, as illustrated in Figure 3. In a typical more difficult case, a 25-word sentence may be optimally compressed by a 17-word version. Of course, if a user requires a shorter compression than that, she may select another region of the curve and look for a local minimum.

A decision-based model for sentence compression

In this section, we describe a decision-based, history model of sentence compression. As in the noisy-channel approach, we again assume that we are given as input

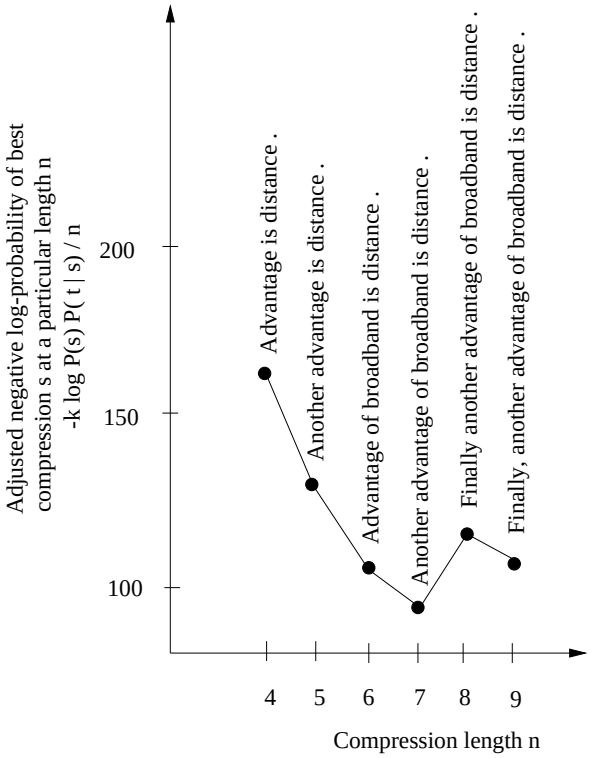

Figure 3: Adjusted log-probabilities for top-scoring compressions at various lengths (lower is better).

a parse tree t. Our goal is to "rewrite" t into a smaller tree s, which corresponds to a compressed version of the original sentence subsumed by t. Suppose we observe in our corpus the trees t and $s2$ in Figure 1. In this model, we ask ourselves how we may go about rewriting t into $s2$. One possible solution is to decompose the rewriting operation into a sequence of shift-reduce-drop actions that are specific to an extended shift-reduce parsing paradigm.

In the model we propose, the rewriting process starts with an empty Stack and an Input List that contains the sequence of words subsumed by the large tree t. Each word in the input list is labeled with the name of all syntactic constituents in t that start with it (see Figure 4). At each step, the rewriting module applies an operation that is aimed at reconstructing the smaller tree $s2$. In the context of our sentence-compression module, we need four types of operations:

- SHIFT operations transfer the first word from the input list into the stack;
- REDUCE operations pop the k syntactic trees located at the top of the stack; combine them into a new tree; and push the new tree on the top of the stack. Reduce operations are used to derive the structure of the syntactic tree of the short sentence.
- DROP operations are used to delete from the input list subsequences of words that correspond to syntactic constituents. A DROP X operations deletes from the input list all words that are spanned by constituent X in t.
- ASSIGNTYPE operations are used to change the label of trees at the top of the stack. These actions assign POS tags to the words in the compressed sentence, which may be different from the POS tags in the original sentence.

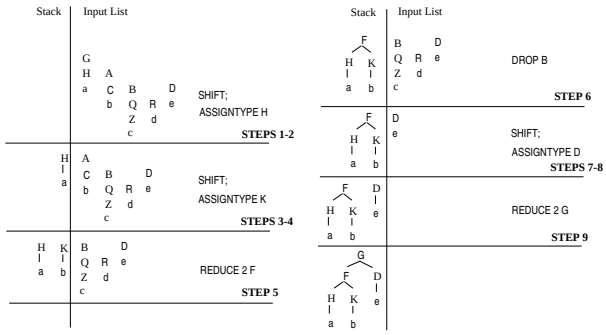

Figure 4: Example of incremental tree compression.

The decision-based model is more flexible than the channel model because it enables the derivation of trees whose skeleton can differ quite drastically from that of the tree given as input. For example, using the channel model, we are unable to obtain tree $s2$ from t. However, the four operations listed above enable us to rewrite a tree t into *any* tree s, as long as an in-order traversal of the leaves of s produces a sequence of words that occur in the same order as the words in the tree t. For example, the tree $s2$ can be obtained from tree t by following this sequence of actions, whose effects are shown in Figure 4: SHIFT; ASSIGNTYPE H; SHIFT; ASSIGNTYPE K; REDUCE 2 F; DROP B; SHIFT; ASSIGNTYPE D; REDUCE 2 G.

To save space, we show SHIFT and ASSIGNTYPE operations on the same line; however, the reader should understand that they correspond to two distinct actions. As one can see, the ASSIGNTYPE K operation rewrites the POS tag of the word `b`; the REDUCE operations modify the skeleton of the tree given as input. To increase readability, the input list is shown in a format that resembles as closely as possible the graphical representation of the trees in figure 1.

Learning the parameters of the decision-based model

We associate with each configuration of our shift-reduce-drop, rewriting model a learning case. The cases are generated automatically by a program that derives sequences of actions that map each of the large trees in our corpus into smaller trees. The rewriting procedure simulates a bottom-up reconstruction of the smaller trees.

Overall, the 1067 pairs of long and short sentences yielded 46383 learning cases. Each case was labeled

with one action name from a set of 210 possible actions: There are 37 distinct ASSIGNTYPE actions, one for each POS tag. There are 63 distinct DROP actions, one for each type of syntactic constituent that can be deleted during compression. There are 109 distinct REDUCE actions, one for each type of reduce operation that is applied during the reconstruction of the compressed sentence. And there is one SHIFT operation. Given a tree t and an arbitrary configuration of the stack and input list, the purpose of the decision-based classifier is to learn what action to choose from the set of 210 possible actions.

To each learning example, we associated a set of 99 features from the following two classes:

Operational features reflect the number of trees in the stack, the input list, and the types of the last five operations. They also encode information that denote the syntactic category of the root nodes of the partial trees built up to a certain time. Examples of such features are: numberTreesInStack, wasPreviousOperationShift, syntacticLabelOfTreeAtTheTopOfStack, etc.

Original-tree-specific features denote the syntactic constituents that start with the first unit in the input list. Examples of such features are: inputListStartsWithA_CC, inputListStartsWithA_PP, etc.

The decision-based compression module uses the C4.5 program (Quinlan 1993) in order to learn decision trees that specify how large syntactic trees can be compressed into shorter trees. A ten-fold cross-validation evaluation of the classifier yielded an accuracy of 87.16% ($\pm$ 0.14). A majority baseline classifier that chooses the action SHIFT has an accuracy of 28.72%.

Employing the decision-based model

To compress sentences, we apply the shift-reduce-drop model in a deterministic fashion. We parse the sentence to be compressed (Collins 1997) and we initialize the input list with the words in the sentence and the syntactic constituents that "begin" at each word, as shown in Figure 4. We then incrementally inquire the learned classifier what action to perform, and we simulate the execution of that action. The procedure ends when the input list is empty and when the stack contains only one tree. An inorder traversal of the leaves of this tree produces the compressed version of the sentence given as input.

Since the model is deterministic, it produces only one output. The advantage is that the compression is very fast: it takes only a few milliseconds per sentence. The disadvantage is that it does not produce a range of compressions, from which another system may subsequently choose. It is straightforward though to extend the model within a probabilistic framework by applying, for example, the techniques used by Magerman (1995).

Evaluation

To evaluate our compression algorithms, we randomly selected 32 sentence pairs from our parallel corpus, which we will refer to as the *Test Corpus*. We used the other 1035 sentence pairs for training. Figure 5 shows three sentences from the Test Corpus, together with the compressions produced by humans, our compression algorithms, and a baseline algorithm that produces compressions with highest word-bigram scores. The examples are chosen so as to reflect good, average, and bad performance cases. The first sentence is compressed in the same manner by humans and our algorithms (the baseline algorithm chooses though not to compress this sentence). For the second example, the output of the Decision-based algorithm is grammatical, but the semantics is negatively affected. The noisy-channel algorithm deletes only the word "break", which affects the correctness of the output less. In the last example, the noisy-channel model is again more conservative and decides not to drop any constituents. In constrast, the decision-based algorithm compresses the input substantially, but it fails to produce a grammatical output.

We presented each original sentence in the *Test Corpus* to four judges, together with four compressions of it: the human generated compression, the outputs of the noisy-channel and decision-based algorithms, and the output of the baseline algorithm. The judges were told that all outputs were generated automatically. The order of the outputs was scrambled randomly across test cases.

To avoid confounding, the judges participated in two experiments. In the first experiment, they were asked to determine on a scale from 1 to 5 how well the systems did with respect to selecting the most important words in the original sentence. In the second experiment, they were asked to determine on a scale from 1 to 5 how grammatical the outputs were.

We also investigated how sensitive our algorithms are with respect to the training data by carrying out the same experiments on sentences of a different genre, the scientific one. To this end, we took the first sentence of the first 26 articles made available in 1999 on the *cmplg* archive. We created a second parallel corpus, which we will refer to as the *Cmplg Corpus*, by generating by ourselves compressed grammatical versions of these sentences. Since some of the sentences in this corpus were extremely long, the baseline algorithm could not produce compressed versions in reasonable time.

The results in Table 1 show compression rates, and mean and standard deviation results across all judges, for each algorithm and corpus. The results show that the decision-based algorithm is the most aggressive: on average, it compresses sentences to about half of their original size. The compressed sentences produced by both algorithms are more "grammatical" and contain more important words than the sentences produced by the baseline. T-test experiments showed these differences to be statistically significant at $p < 0.01$ both for individual judges and for average scores across

Original:	Beyond the basic level, the operations of the three products vary widely.
Baseline:	Beyond the basic level, the operations of the three products vary widely.
Noisy-channel:	The operations of the three products vary widely.
Decision-based:	The operations of the three products vary widely.
Humans:	The operations of the three products vary widely.

Original:	Arborscan is reliable and worked accurately in testing, but it produces very large dxf files.
Baseline:	Arborscan and worked in, but it very large dxf.
Noisy-channel:	Arborscan is reliable and worked accurately in testing, but it produces very large dxf files.
Decision-based:	Arborscan is reliable and worked accurately in testing very large dxf files.
Humans:	Arborscan produces very large dxf files.

Original:	Many debugging features, including user-defined break points and variable-watching and message-watching windows, have been added.
Baseline:	Debugging, user-defined and variable-watching and message-watching, have been.
Noisy-channel:	Many debugging features, including user-defined points and variable-watching and message-watching windows, have been added.
Decision-based:	Many debugging features.
Humans:	Many debugging features have been added .

Figure 5: Compression examples

Corpus	Avg. orig. sent. length		Baseline	Noisy-channel	Decision-based	Humans
Test	21 words	Compression	63.70%	70.37%	57.19%	53.33%
		Grammaticality	1.78±1.19	4.34±1.02	4.30±1.33	4.92±0.18
		Importance	2.17±0.89	3.38±0.67	3.54±1.00	4.24 ±0.52
Cmplg	26 words	Compression	–	65.68%	54.25%	65.68%
		Grammaticality	–	4.22±0.99	3.72±1.53	4.97±0.08
		Importance	–	3.42±0.97	3.24±0.68	4.32±0.54

Table 1: Experimental results

all judges. T-tests showed no significant statistical differences between the two algorithms. As Table 1 shows, the performance of the compression algorithms is much closer to human performance than baseline performance; yet, humans perform statistically better than our algorithms at $p < 0.01$.

When applied to sentences of a different genre, the performance of the noisy-channel compression algorithm degrades smoothly, while the performance of the decision-based algorithm drops sharply. This is due to a few sentences in the *Cmplg Corpus* that the decision-based algorithm over-compressed to only two or three words. We suspect that this problem can be fixed if the decision-based compression module is extended in the style of Magerman (1995), by computing probabilities across the sequences of decisions that correspond to a compressed sentence. Likewise, there are substantial gains to be had in noisy-channel modeling—we see clearly in the data many statistical dependencies and processes that are not captured in our simple initial models. More grammatical output will come from taking account of subcategory and head-modifier statistics (in addition to simple word-bigrams), and an expanded channel model will allow for more tree manipulation possibilities. Work on extending the algorithms presented in this paper to compressing multiple sentences is currently underway.

References

Barzilay, R.; McKeown, K.; and Elhadad, M. 1999. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL–99)*, 550–557.

Berger, A., and Lafferty, J. 1999. Information retrieval as statistical translation. In *Proceedings of the 22nd Conference on Research and Development in Information Retrieval (SIGIR–99)*, 222–229.

Brown, P.; Della Pietra, S.; Della Pietra, V.; and Mercer, R. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* 19(2):263–311.

Church, K. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, 136–143.

Collins, M. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL–97)*, 16–23.

Grefenstette, G. 1998. Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *Working Notes of the AAAI*

Spring Symposium on Intelligent Text Summarization, 111–118.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition*. The MIT Press.

Jing, H., and McKeown, K. 1999. The decomposition of human-written summary sentences. In *Proceedings of the 22nd Conference on Research and Development in Information Retrieval (SIGIR–99)*.

Knight, K., and Graehl, J. 1998. Machine transliteration. *Computational Linguistics* 24(4):599–612.

Langkilde, I. 2000. Forest-based statistical sentence generation. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*.

Linke-Ellis, N. 1999. Closed captioning in America: Looking beyond compliance. In *Proceedings of the TAO Workshop on TV Closed Captions for the hearing impaired people*, 43–59.

Magerman, D. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283.

Mani, I., and Maybury, M., eds. 1999. *Advances in Automatic Text Summarization*. The MIT Press.

Mani, I.; Gates, B.; and Bloedorn, E. 1999. Improving summaries by revising them. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 558–565.

McKeown, K.; Klavans, J.; Hatzivassiloglou, V.; Barzilay, R.; and Eskin, E. 1999. Towards multidocument summarization by reformulation: Progress and prospects. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI–99)*.

Quinlan, J. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Robert-Ribes, J.; Pfeiffer, S.; Ellison, R.; and Burnham, D. 1999. Semi-automatic captioning of TV programs, an Australian perspective. In *Proceedings of the TAO Workshop on TV Closed Captions for the hearing impaired people*, 87–100.

Witbrock, M., and Mittal, V. 1999. Ultra-summarization: A statistical approach to generating highly condensed non-extractive summaries. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGIR'99), Poster Session*, 315–316.