Discriminative Sentence Compression with Soft Syntactic Evidence

Ryan McDonald
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
ryantm@cis.upenn.edu

Abstract

We present a model for sentence compression that uses a discriminative large-margin learning framework coupled with a novel feature set defined on compressed bigrams as well as deep syntactic representations provided by auxiliary dependency and phrase-structure parsers. The parsers are trained out-of-domain and contain a significant amount of noise. We argue that the discriminative nature of the learning algorithm allows the model to learn weights relative to any noise in the feature set to optimize compression accuracy directly. This differs from current state-of-the-art models (Knight and Marcu, 2000) that treat noisy parse trees, for both compressed and uncompressed sentences, as gold standard when calculating model parameters.

1 Introduction

The ability to compress sentences grammatically with minimal information loss is an important problem in text summarization. Most summarization systems are evaluated on the amount of relevant information retained as well as their compression rate. Thus, returning highly compressed, yet informative, sentences allows summarization systems to return larger sets of sentences and increase the overall amount of information extracted.

We focus on the particular instantiation of sentence compression when the goal is to produce the compressed version solely by removing words or phrases from the original, which is the most common setting in the literature (Knight and Marcu, 2000; Riezler et al., 2003; Turner and Charniak, 2005). In this framework, the goal is to find the shortest substring of the original sentence that conveys the most important aspects of the meaning. We will work in a supervised learning setting and assume as input a training set $\mathcal{T}=(\boldsymbol{x}_t, \boldsymbol{y}_t)_{t=1}^{|\mathcal{T}|}$ of original sentences $\boldsymbol{x}_t$ and their compressions $\boldsymbol{y}_t$. We use the Ziff-Davis corpus, which is a set of 1087 pairs of sentence/compression pairs. Furthermore, we use the same 32 testing examples from Knight and Marcu (2000) and the rest for training, except that we hold out 20 sentences for the purpose of development. A handful of sentences occur twice but with different compressions. We randomly select a single compression for each unique sentence in order to create an unambiguous training set. Examples from this data set are given in Figure 1.

Formally, sentence compression aims to shorten a sentence $\boldsymbol{x} = x_1 \dots x_n$ into a substring $\boldsymbol{y} = y_1 \dots y_m$, where $y_i \in \{x_1, \dots, x_n\}$. We define the function $I(y_i) \in \{1, \dots, n\}$ that maps word y_i in the compression to the index of the word in the original sentence. Finally we include the constraint $I(y_i) < I(y_{i+1})$, which forces each word in $\boldsymbol{x}$ to occur at most once in the compression $\boldsymbol{y}$. Compressions are evaluated on three criteria,

1. Grammaticality: Compressed sentences should be grammatical.
2. Importance: How much of the important information is retained from the original.
3. Compression rate: How much compression took place. A compression rate of 65% means the compressed sentence is 65% the length of the original.

Typically grammaticality and importance are traded off with compression rate. The longer our

The Reverse Engineer Tool is priced from $8,000 for a single user to $90,000 for a multiuser project site .

The Reverse Engineer Tool is available now and is priced on a site-licensing basis , ranging from $8,000 for a single user to $90,000 for a multiuser project site .

Design recovery tools read existing code and translate it into defi nitions and structured diagrams .

Essentially , design recovery tools read existing code and translate it into the language in which CASE is conversant – defi nitions and structured diagrams .

Figure 1: Two examples of compressed sentences from the Ziff-Davis corpus. The compressed version and the original sentence are given.

compressions, the less likely we are to remove important words or phrases crucial to maintaining grammaticality and the intended meaning.

The paper is organized as follows: Section 2 discusses previous approaches to sentence compression. In particular, we discuss the advantages and disadvantages of the models of Knight and Marcu (2000). In Section 3 we present our discriminative large-margin model for sentence compression, including the learning framework and an efficient decoding algorithm for searching the space of compressions. We also show how to extract a rich feature set that includes surface-level bigram features of the compressed sentence, dropped words and phrases from the original sentence, and features over noisy dependency and phrase-structure trees for the original sentence. We argue that this rich feature set allows the model to learn which words and phrases should be dropped and which should remain in the compression. Section 4 presents an experimental evaluation of our model compared to the models of Knight and Marcu (2000) and finally Section 5 discusses some areas of future work.

2 Previous Work

Knight and Marcu (2000) first tackled this problem by presenting a generative noisy-channel model and a discriminative tree-to-tree decision tree model. The noisy-channel model defines the problem as finding the compressed sentence with maximum conditional probability

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) = \arg\max_{\boldsymbol{y}} P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y})$$

$P(\boldsymbol{y})$ is the source model, which is a PCFG plus bigram language model. $P(\boldsymbol{x}|\boldsymbol{y})$ is the channel model, the probability that the long sentence is an expansion of the compressed sentence. To calculate the channel model, both the original and compressed versions of every sentence in the training set are assigned a phrase-structure tree. Given a tree for a long sentence $\boldsymbol{x}$ and compressed sentence $\boldsymbol{y}$, the channel probability is the product of the probability for each transformation required if the tree for $\boldsymbol{y}$ is to expand to the tree for $\boldsymbol{x}$.

The tree-to-tree decision tree model looks to rewrite the tree for $\boldsymbol{x}$ into a tree for $\boldsymbol{y}$. The model uses a shift-reduce-drop parsing algorithm that starts with the sequence of words in $\boldsymbol{x}$ and the corresponding tree. The algorithm then either shifts (considers new words and subtrees for $\boldsymbol{x}$), reduces (combines subtrees from $\boldsymbol{x}$ into possibly new tree constructions) or drops (drops words and subtrees from $\boldsymbol{x}$) on each step of the algorithm. A decision tree model is trained on a set of indicative features for each type of action in the parser. These models are then combined in a greedy global search algorithm to find a single compression.

Though both models of Knight and Marcu perform quite well, they do have their shortcomings. The noisy-channel model uses a source model that is trained on uncompressed sentences, even though the source model is meant to represent the probability of compressed sentences. The channel model requires aligned parse trees for both compressed and uncompressed sentences in the training set in order to calculate probability estimates. These parses are provided from a parsing model trained on out-of-domain data (the WSJ), which can result in parse trees with many mistakes for both the original and compressed versions. This makes alignment difficult and the channel probability estimates unreliable as a result. On the other hand, the decision tree model does not rely on the trees to align and instead simply learns a tree-to-tree transformation model to compress sentences. The primary problem with this model is that most of the model features encode properties related to including or dropping constituents from the tree with no encoding of bigram or trigram surface features to promote grammaticality. As a result, the model will sometimes return very short and ungrammatical compressions.

Both models rely heavily on the output of a noisy parser to calculate probability estimates for the compression. We argue in the next section that

ideally, parse trees should be treated solely as a source of evidence when making compression decisions to be balanced with other evidence such as that provided by the words themselves.

Recently Turner and Charniak (2005) presented supervised and semi-supervised versions of the Knight and Marcu noisy-channel model. The resulting systems typically return informative and grammatical sentences, however, they do so at the cost of compression rate. Riezler et al. (2003) present a discriminative sentence compressor over the output of an LFG parser that is a packed representation of possible compressions. This work and the work of Riezler et al. (2003) are highly related. The primary difference in this work is that our inference and learning algorithms consider the space of all possible compressions of a sentence through a substring search, and not just those deemed viable by an auxilary parser.

3 Discriminative Sentence Compression

For the rest of the paper we use $\boldsymbol{x} = x_1 \dots x_n$ to indicate an uncompressed sentence and $\boldsymbol{y} = y_1 \dots y_m$ a compressed version of $\boldsymbol{x}$, i.e., each y_j indicates the position in $\boldsymbol{x}$ of the j^{th} word in the compression. We always pad the sentence with dummy start and end words, x_1 = -START- and x_n = -END-, which are always included in the compressed version (i.e. $y_1 = x_1$ and $y_m = x_n$).

In this section we described a discriminative online learning approach to sentence compression, the core of which is a decoding algorithm that searches the entire space of compressions. Let the score of a compression $\boldsymbol{y}$ for a sentence $\boldsymbol{x}$ as

$$s(\boldsymbol{x}, \boldsymbol{y})$$

In particular, we are going to factor this score using a first-order Markov assumption on the words in the *compressed* sentence

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=2}^{|\boldsymbol{y}|} s(\boldsymbol{x}, I(y_{j-1}), I(y_j))$$

Finally, we define the score function to be the dot product between a high dimensional feature representation and a corresponding weight vector

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=2}^{|\boldsymbol{y}|} \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, I(y_{j-1}), I(y_j))$$

Note that this factorization will allow us to define features over two adjacent words in the compression as well as the words in-between that were dropped from the original sentence to create the compression. We will show in Section 3.2 how this factorization also allows us to include features on dropped phrases and subtrees from both a dependency and a phrase-structure parse of the original sentence. Note that these features are meant to capture the same information in both the source and channel models of Knight and Marcu (2000). However, here they are merely treated as evidence for the discriminative learner, which will set the weight of each feature relative to the other (possibly overlapping) features to optimize the models accuracy on the observed data.

3.1 Decoding

We define a dynamic programming table $C[i]$ which represents the highest score for any compression that ends at word x_i for sentence $\boldsymbol{x}$. We define a recurrence as follows

$$C[1] = 0.0$$
$$C[i] = \max_{j<i} C[j] + s(\boldsymbol{x}, j, i) \text{ for } i > 1$$

It is easy to show that $C[n]$ represents the score of the best compression for sentence $\boldsymbol{x}$ (whose length is n) under the first-order score factorization we made. We can show this by induction. If we assume that $C[j]$ is the highest scoring compression that ends at word x_j, for all $j < i$, then $C[i]$ must also be the highest scoring compression ending at word x_i since it represents the max combination over all high scoring shorter compressions plus the score of extending the compression to the current word. Thus, since x_n is by definition in every compressed version of $\boldsymbol{x}$ (see above), then it must be the case that $C[n]$ stores the score of the best compression. This table can be filled in $O(n^2)$.

This algorithm is really an extension of Viterbi to the case when scores factor over dynamic substrings of the text (Sarawagi and Cohen, 2004; McDonald et al., 2005a). As such, we can use back-pointers to reconstruct the highest scoring compression as well as k-best decoding algorithms.

This decoding algorithm is dynamic with respect to compression rate. That is, the algorithm will return the highest scoring compression regardless of length. This may seem problematic since longer compressions might contribute more to the score (since they contain more bigrams) and

thus be preferred. However, in Section 3.2 we define a rich feature set, including features on words dropped from the compression that will help disfavor compressions that drop very few words since this is rarely seen in the training data. In fact, it turns out that our learned compressions have a compression rate very similar to the gold standard.

That said, there are some instances when a static compression rate is preferred. A user may specifically want a 25% compression rate for all sentences. This is not a problem for our decoding algorithm. We simply augment the dynamic programming table and calculate $C[i][r]$, which is the score of the best compression of length r that ends at word x_i. This table can be filled in as follows

$C[1][1] = 0.0$
$C[1][r] = -\infty$ for $r > 1$
$C[i][r] = \max_{j<i} C[j][r-1] + s(\boldsymbol{x}, j, i)$ for $i > 1$

Thus, if we require a specific compression rate, we simple determine the number of words r that satisfy this rate and calculate $C[n][r]$. The new complexity is $O(n^2 r)$.

3.2 Features

So far we have defined the score of a compression as well as a decoding algorithm that searches the entire space of compressions to find the one with highest score. This all relies on a score factorization over adjacent words in the compression, $s(\boldsymbol{x}, I(y_{j-1}), I(y_j)) = \mathbf{w} \cdot \mathbf{f}(\boldsymbol{x}, I(y_{j-1}), I(y_j))$. In Section 3.3 we describe an online large-margin method for learning $\mathbf{w}$. Here we present the feature representation $\mathbf{f}(\boldsymbol{x}, I(y_{j-1}), I(y_j))$ for a pair of adjacent words in the compression. These features were tuned on a development data set.

3.2.1 Word/POS Features

The first set of features are over adjacent words y_{j-1} and y_j in the compression. These include the part-of-speech (POS) bigrams for the pair, the POS of each word individually, and the POS context (bigram and trigram) of the most recent word being added to the compression, y_j. These features are meant to indicate likely words to include in the compression as well as some level of grammaticality, e.g., the adjacent POS features "JJ&VB" would get a low weight since we rarely see an adjective followed by a verb. We also add a feature indicating if y_{j-1} and y_j were actually adjacent in the original sentence or not and we conjoin this feature with the above POS features. Note that we have not included any lexical features. We found during experiments on the development data that lexical information was too sparse and led to overfitting, so we rarely include such features. Instead we rely on the accuracy of POS tags to provide enough evidence.

Next we added features over every dropped word in the original sentence between y_{j-1} and y_j, if there were any. These include the POS of each dropped word, the POS of the dropped words conjoined with the POS of y_{j-1} and y_j. If the dropped word is a verb, we add a feature indicating the actual verb (this is for common verbs like "is", which are typically in compressions). Finally we add the POS context (bigram and trigram) of each dropped word. These features represent common characteristics of words that can or should be dropped from the original sentence in the compressed version (e.g. adjectives and adverbs). We also add a feature indicating whether the dropped word is a negation (e.g., not, never, etc.).

We also have a set of features to represent brackets in the text, which are common in the data set. The first measures if all the dropped words between y_{j-1} and y_j have a mismatched or inconsistent bracketing. The second measures if the left and right-most dropped words are themselves both brackets. These features come in handy for examples like, *The Associated Press (AP) reported the story*, where the compressed version is *The Associated Press reported the story*. Information within brackets is often redundant.

3.2.2 Deep Syntactic Features

The previous set of features are meant to encode common POS contexts that are commonly retained or dropped from the original sentence during compression. However, they do so without a larger picture of the function of each word in the sentence. For instance, dropping verbs is not that uncommon - a relative clause for instance may be dropped during compression. However, dropping the main verb in the sentence is uncommon, since that verb and its arguments typically encode most of the information being conveyed.

An obvious solution to this problem is to include features over a deep syntactic analysis of the sentence. To do this we parse every sentence twice, once with a dependency parser (McDonald et al., 2005b) and once with a phrase-structure parser (Charniak, 2000). These parsers have been trained out-of-domain on the Penn WSJ Treebank and as a result contain noise. However, we are

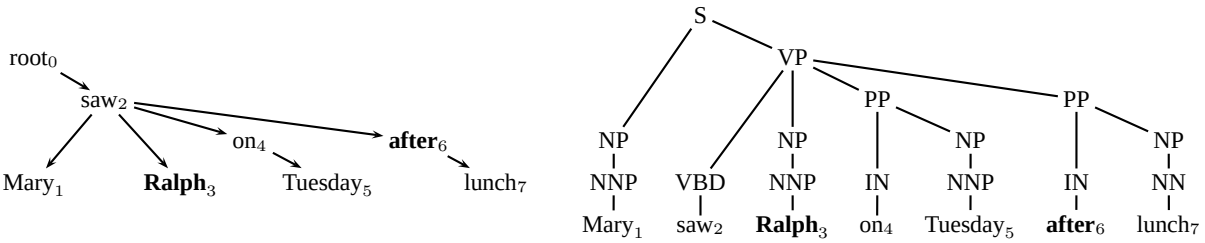

Figure 2: An example dependency tree from the McDonald et al. (2005b) parser and phrase structure tree from the Charniak (2000) parser. In this example we want to add features from the trees for the case when *Ralph* and *after* become adjacent in the compression, i.e., we are dropping the phrase *on Tuesday*.

merely going to use them as an additional source of features. We call this *soft syntactic evidence* since the deep trees are not used as a strict gold-standard in our model but just as more evidence for or against particular compressions. The learning algorithm will set the feature weight accordingly depending on each features discriminative power. It is not unique to use soft syntactic features in this way, as it has been done for many problems in language processing. However, we stress this aspect of our model due to the history of compression systems using syntax to provide hard structural constraints on the output.

Lets consider the sentence $\boldsymbol{x}$ = *Mary saw Ralph on Tuesday after lunch*, with corresponding parses given in Figure 2. In particular, lets consider the feature representation $\mathbf{f}(\boldsymbol{x},3,6)$. That is, the feature representation of making *Ralph* and *after* adjacent in the compression and dropping the prepositional phrase *on Tuesday*. The first set of features we consider are over dependency trees. For every dropped word we add a feature indicating the POS of the words parent in the tree. For example, if the dropped words parent is *root*, then it typically means it is the main verb of the sentence and unlikely to be dropped. We also add a conjunction feature of the POS tag of the word being dropped and the POS of its parent as well as a feature indicating for each word being dropped whether it is a leaf node in the tree. We also add the same features for the two adjacent words, but indicating that they are part of the compression.

For the phrase-structure features we find every node in the tree that subsumes a piece of dropped text and is not a child of a similar node. In this case the PP governing *on Tuesday*. We then add features indicating the context from which this node was dropped. For example we add a feature specifying that a PP was dropped which was the child of a VP. We also add a feature indicating that a PP was dropped which was the left sibling of another PP, etc. Ideally, for each production in the tree we would like to add a feature indicating every node that was dropped, e.g. "VP→VBD NP PP PP ⇒ VP→VBD NP PP". However, we cannot necessarily calculate this feature since the extent of the production might be well beyond the local context of first-order feature factorization. Furthermore, since the training set is so small, these features are likely to be observed very few times.

3.2.3 Feature Set Summary

In this section we have described a rich feature set over adjacent words in the compressed sentence, dropped words and phrases from the original sentence, and properties of deep syntactic trees of the original sentence. Note that these features in many ways mimic the information already present in the noisy-channel and decision-tree models of Knight and Marcu (2000). Our bigram features encode properties that indicate both good and bad words to be adjacent in the compressed sentence. This is similar in purpose to the source model from the noisy-channel system. However, in that system, the source model is trained on uncompressed sentences and thus is not as representative of likely bigram features for compressed sentences, which is really what we desire.

Our feature set also encodes dropped words and phrases through the properties of the words themselves and through properties of their syntactic relation to the rest of the sentence in a parse tree. These features represent likely phrases to be dropped in the compression and are thus similar in nature to the channel model in the noisy-channel system as well as the features in the tree-to-tree decision tree system. However, we use these syntactic constraints as *soft evidence* in our model. That is, they represent just another layer of evidence to be considered during training when setting parameters. Thus, if the parses have too much noise, the learning algorithm can lower the weight of the

parse features since they are unlikely to be useful discriminators on the training data. This differs from the models of Knight and Marcu (2000), which treat the noisy parses as gold-standard when calculating probability estimates.

An important distinction we should make is the notion of *supported* versus *unsupported* features (Sha and Pereira, 2003). Supported features are those that are on for the gold standard compressions in the training. For instance, the bigram feature "NN&VB" will be supported since there is most likely a compression that contains a adjacent noun and verb. However, the feature "JJ&VB" will not be supported since an adjacent adjective and verb most likely will not be observed in any valid compression. Our model includes all features, including those that are unsupported. The advantage of this is that the model can learn negative weights for features that are indicative of bad compressions. This is not difficult to do since most features are POS based and the feature set size even with all these features is only 78,923.

3.3 Learning

Having defined a feature encoding and decoding algorithm, the last step is to learn the feature weights $\mathbf{w}$. We do this using the Margin Infused Relaxed Algorithm (MIRA), which is a discriminative large-margin online learning technique shown in Figure 3 (Crammer and Singer, 2003). On each iteration, MIRA considers a single instance from the training set $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ and updates the weights so that the score of the correct compression, $\boldsymbol{y}_t$, is greater than the score of all other compressions by a margin proportional to their loss. Many weight vectors will satisfy these constraints so we pick the one with minimum change from the previous setting. We define the loss to be the number of words falsely retained or dropped in the incorrect compression relative to the correct one. For instance, if the correct compression of the sentence in Figure 2 is *Mary saw Ralph*, then the compression *Mary saw after lunch* would have a loss of 3 since it incorrectly left out one word and included two others.

Of course, for a sentence there are exponentially many possible compressions, which means that this optimization will have exponentially many constraints. We follow the method of McDonald et al. (2005b) and create constraints only on the k compressions that currently have the highest score, $\text{best}_k(\boldsymbol{x}; \mathbf{w})$. This can easily be calculated by extending the decoding algorithm with standard Viterbi k-best techniques. On the development data, we found that $k = 10$ provided the best performance, though varying k did not have a major impact overall. Furthermore we found that after only 3-5 training epochs performance on the development data was maximized.

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$
1. $\mathbf{w}_0 = 0;\ \mathbf{v} = 0;\ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. $\min \left\| \mathbf{w}^{(i+1)} - \mathbf{w}^{(i)} \right\|$
s.t. $s(\boldsymbol{x}_t, \boldsymbol{y}_t) - s(\boldsymbol{x}_t, \boldsymbol{y}') \geq L(\boldsymbol{y}_t, \boldsymbol{y}')$
where $\boldsymbol{y}' \in \text{best}_k(\boldsymbol{x}; \mathbf{w}^{(i)})$
5. $\mathbf{v} = \mathbf{v} + \mathbf{w}^{(i+1)}$
6. $i = i + 1$
7. $\mathbf{w} = \mathbf{v}/(N * T)$

Figure 3: MIRA learning algorithm as presented by McDonald et al. (2005b).

The final weight vector is the *average* of all weight vectors throughout training. Averaging has been shown to reduce overfitting (Collins, 2002) as well as reliance on the order of the examples during training. We found it to be particularly important for this data set.

4 Experiments

We use the same experimental methodology as Knight and Marcu (2000). We provide every compression to four judges and ask them to evaluate each one for grammaticality and importance on a scale from 1 to 5. For each of the 32 sentences in our test set we ask the judges to evaluate three systems: human annotated, the decision tree model of Knight and Marcu (2000) and our system. The judges were told all three compressions were automatically generated and the order in which they were presented was randomly chosen for each sentence. We compared our system to the decision tree model of Knight and Marcu instead of the noisy-channel model since both performed nearly as well in their evaluation, and the compression rate of the decision tree model is nearer to our system (around 57-58%). The noisy-channel model typically returned longer compressions.

Results are shown in Table 1. We present the average score over all judges as well as the standard deviation. The evaluation for the decision tree system of Knight and Marcu is strikingly similar to

the original evaluation in their work. This provides strong evidence that the evaluation criteria in both cases were very similar.

Table 1 shows that all models had similar compressions rates, with humans preferring to compress a little more aggressively. Not surprisingly, the human compressions are practically all grammatical. A quick scan of the evaluations shows that the few ungrammatical human compressions were for sentences that were not really grammatical in the first place. Of greater interest is that the compressions of our system are typically more grammatical than the decision tree model of Knight and Marcu.

When looking at importance, we see that our system actually does the best – even better than humans. The most likely reason for this is that our model returns longer sentences and is thus less likely to prune away important information. For example, consider the sentence

The chemical etching process used for glare protection is effective and will help if your office has the fluorescent-light overkill that's typical in offices

The human compression was *Glare protection is effective*, whereas our model compressed the sentence to *The chemical etching process used for glare protection is effective*.

A primary reason that our model does better than the decision tree model of Knight and Marcu is that on a handful of sentences, the decision tree compressions were a single word or noun-phrase. For such sentences the evaluators typically rated the compression a 1 for both grammaticality and importance. In contrast, our model never failed in such drastic ways and always output something reasonable. This is quantified in the standard deviation of the two systems.

Though these results are promising, more large scale experiments are required to really ascertain the significance of the performance increase. Ideally we could sample multiple training/testing splits and use all sentences in the data set to evaluate the systems. However, since these systems require human evaluation we did not have the time or the resources to conduct these experiments.

4.1 Some Examples

Here we aim to give the reader a flavor of some common outputs from the different models. Three examples are given in Table 4.1. The first shows two properties. First of all, the decision tree model completely breaks and just returns a single noun-phrase. Our system performs well, however it leaves out the complementizer of the relative clause. This actually occurred in a few examples and appears to be the most common problem of our model. A post-processing rule should eliminate this.

The second example displays a case in which our system and the human system are grammatical, but the removal of a prepositional phrase hurts the resulting meaning of the sentence. In fact, without the knowledge that the sentence is referring to *broadband*, the compressions are meaningless. This appears to be a harder problem – determining which prepositional phrases can be dropped and which cannot.

The final, and more interesting, example presents two very different compressions by the human and our automatic system. Here, the human kept the relative clause relating what languages the source code is available in, but dropped the main verb phrase of the sentence. Our model preferred to retain the main verb phrase and drop the relative clause. This is most likely due to the fact that dropping the main verb phrase of a sentence is much less likely in the training data than dropping a relative clause. Two out of four evaluators preferred the compression returned by our system and the other two rated them equal.

5 Discussion

In this paper we have described a new system for sentence compression. This system uses discriminative large-margin learning techniques coupled with a decoding algorithm that searches the space of all compressions. In addition we defined a rich feature set of bigrams in the compression and dropped words and phrases from the original sentence. The model also incorporates soft syntactic evidence in the form of features over dependency and phrase-structure trees for each sentence.

This system has many advantages over previous approaches. First of all its discriminative nature allows us to use a rich dependent feature set and to optimize a function directly related to compression accuracy during training, both of which have been shown to be beneficial for other problems. Furthermore, the system does not rely on the syntactic parses of the sentences to calculate probability estimates. Instead, this information is incorporated as just another form of evidence to be consid-

	Compression Rate	Grammaticality	Importance
Human	53.3%	4.96 ± 0.2	3.91 ± 1.0
Decision-Tree (K&M2000)	57.2%	4.30 ± 1.4	3.60 ± 1.3
This work	58.1%	4.61 ± 0.8	4.03 ± 1.0

Table 1: Compression results.

Full Sentence	The first new product , ATF Protype , is a line of digital postscript typefaces that will be sold in packages of up to six fonts .
Human	ATF Protype is a line of digital postscript typefaces that will be sold in packages of up to six fonts .
Decision Tree	The first new product .
This work	ATF Protype is a line of digital postscript typefaces will be sold in packages of up to six fonts .
Full Sentence	Finally , another advantage of broadband is distance .
Human	Another advantage is distance .
Decision Tree	Another advantage of broadband is distance .
This work	Another advantage is distance .
Full Sentence	The source code , which is available for C , Fortran , ADA and VHDL , can be compiled and executed on the same system or ported to other target platforms .
Human	The source code is available for C , Fortran , ADA and VHDL .
Decision Tree	The source code is available for C .
This work	The source code can be compiled and executed on the same system or ported to other target platforms .

Table 2: Example compressions for the evaluation data.

ered during training. This is advantageous because these parses are trained on out-of-domain data and often contain a significant amount of noise.

A fundamental flaw with all sentence compression systems is that model parameters are set with the assumption that there is a single correct answer for each sentence. Of course, like most compression and translation tasks, this is not true, consider,

TapeWare , which supports DOS and NetWare 286 , is a value-added process that lets you directly connect the QA150-EXAT to a file server and issue a command from any workstation to back up the server

The human annotated compression is, *TapeWare supports DOS and NetWare 286*. However, another completely valid compression might be, *TapeWare lets you connect the QA150-EXAT to a file server*. These two compressions overlap by a single word.

Our learning algorithm may unnecessarily lower the score of some perfectly valid compressions just because they were not the exact compression chosen by the human annotator. A possible direction of research is to investigate multi-label learning techniques for structured data (McDonald et al., 2005a) that learn a scoring function separating a set of valid answers from all invalid answers. Thus if a sentence has multiple valid compressions we can learn to score each valid one higher than all invalid compressions during training to avoid this problem.

Acknowledgments

The author would like to thank Daniel Marcu for providing the data as well as the output of his and Kevin Knight's systems. Thanks also to Hal Daumé and Fernando Pereira for useful discussions. Finally, the author thanks the four reviewers for evaluating the compressed sentences. This work was supported by NSF ITR grants 0205448 and 0428193.

References

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL*.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*.

K. Knight and D. Marcu. 2000. Statistical-based summarization - step one: Sentence compression. In *Proc. AAAI 2000*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Flexible text segmentation with structured multilabel classification. In *Proc. HLT-EMNLP*.

R. McDonald, K. Crammer, and F. Pereira. 2005b. Online large-margin training of dependency parsers. In *Proc. ACL*.

S. Riezler, T. H. King, R. Crouch, and A. Zaenen. 2003. Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for lexical-functional grammar. In *Proc. HLT-NAACL*.

S. Sarawagi and W. Cohen. 2004. Semi-Markov conditional random fields for information extraction. In *Proc. NIPS*.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fi elds. In *Proc. HLT-NAACL*, pages 213–220.

J. Turner and E. Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proc. ACL*.