Incremental Text Structuring with Online Hierarchical Ranking

Erdong Chen, Benjamin Snyder and Regina Barzilay
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{edc,bsnyder,regina}@csail.mit.edu

Abstract

Many emerging applications require documents to be repeatedly updated. Such documents include newsfeeds, webpages, and shared community resources such as Wikipedia. In this paper we address the task of inserting new information into existing texts. In particular, we wish to determine the best location in a text for a given piece of new information. For this process to succeed, the insertion algorithm should be informed by the existing document structure. Lengthy real-world texts are often hierarchically organized into chapters, sections, and paragraphs. We present an online ranking model which exploits this hierarchical structure – representationally in its features and algorithmically in its learning procedure. When tested on a corpus of Wikipedia articles, our hierarchically informed model predicts the correct insertion paragraph more accurately than baseline methods.

1 Introduction

Many emerging applications require documents to be repeatedly updated. For instance, newsfeed articles are continuously revised by editors as new information emerges, and personal webpages are modified as the status of the individual changes. This revision strategy has become even more prevalent with the advent of community edited web resources, the most notable example being Wikipedia. At present this process involves massive human effort. For instance, the English language version of Wikipedia averaged over 3 million edits[1] per month in 2006. Even so, many articles quickly become outdated. A system that performs such updates automatically could drastically decrease maintenance efforts and potentially improve document quality.

Currently there is no effective way to automatically update documents as new information becomes available. The closest relevant text structuring technique is the work on sentence ordering, in which a complete reordering of the text is undertaken. Predictably these methods are suboptimal for this new task because they cannot take advantage of existing text structure.

We introduce an alternative vision of text structuring as a process unfolding over time. Instead of ordering sentences all at once, we start with a well-formed draft and add new information at each stage, while preserving document coherence. The basic operation of incremental text structuring is the insertion of new information. To automate this process, we develop a method for determining the best location in a text for a given piece of new information.

The main challenge is to maintain the continuity and coherence of the original text. These properties may be maintained by examining sentences adjacent to each potential insertion point. However, a local sentence comparison method such as this may fail to account for global document coherence (e.g. by allowing the mention of some fact in an inappropriate section). This problem is especially acute in the case of lengthy, real-world texts such as books, technical reports, and web pages. These documents

[1] http://stats.wikimedia.org/EN/TablesWikipediaEN.htm

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 83–91, Prague, June 2007.

are commonly organized hierarchically into sections and paragraphs to aid reader comprehension. For documents where hierarchical information is not explicitly provided, such as automatic speech transcripts, we can use automatic segmentation methods to induce such a structure (Hearst, 1994). Rather than ignoring the inherent hierarchical structure of these texts, we desire to directly model such hierarchies and use them to our advantage – both representationally in our features and algorithmically in our learning procedure.

To achieve this goal, we introduce a novel method for sentence insertion that operates over a hierarchical structure. Our document representation includes features for each layer of the hierarchy. For example, the word overlap between the inserted sentence and a section header would be included as an upper-level section feature, whereas a comparison of the sentence with all the words in a paragraph would be a lower-level paragraph feature. We propose a linear model which simultaneously considers the features of every layer when making insertion decisions. We develop a novel update mechanism in the online learning framework which exploits the hierarchical decomposition of features. This mechanism limits model updates to those features found at the highest incorrectly predicted layer, without unnecessarily disturbing the parameter values for the lower reaches of the tree. This conservative update approach maintains as much knowledge as possible from previously encountered training examples.

We evaluate our method using real-world data where multiple authors have revised preexisting documents over time. We obtain such a corpus from Wikipedia articles,[2] which are continuously updated by multiple authors. Logs of these updates are publicly available, and are used for training and testing of our algorithm. Figure 1 shows an example of a Wikipedia insertion. We believe this data will more closely mirror potential applications than synthetic collections used in previous work on text structuring.

Our hierarchical training method yields significant improvement when compared to a similar non-hierarchical model which instead uses the standard perceptron update of Collins (2002). We also report human performance on the insertion task in order to provide a reasonable upper-bound on machine performance. An analysis of these results shows that our method closes the gap between machine and human performance substantially.

In the following section, we provide an overview of existing work on text structuring and hierarchical learning. Then, we define the insertion task and introduce our hierarchical ranking approach to sentence insertion. Next, we present our experimental framework and data. We conclude the paper by presenting and discussing our results.

2 Related Work

Text Structuring The insertion task is closely related to the extensively studied problem of sentence ordering.[3] Most of the existing algorithms represent text structure as a linear sequence and are driven by local coherence constraints (Lapata, 2003; Karamanis et al., 2004; Okazaki et al., 2004; Barzilay and Lapata, 2005; Bollegala et al., 2006; Elsner and Charniak, 2007). These methods induce a total ordering based on pairwise relations between sentences. Researchers have shown that identifying precedence relations does not require deep semantic interpretation of input sentences: shallow distributional features are sufficient for accurate prediction. Our approach employs similar features to represent nodes at the lowest level of the hierarchy.

The key departure of our work from previous research is the incorporation of hierarchical structure into a corpus-based approach to ordering. While in symbolic generation and discourse analysis a text is typically analyzed as a tree-like structure (Reiter and Dale, 1990), a linear view is prevalent in data-driven methods to text structuring.[4] Moving beyond a linear representation enables us to handle longer texts where a local view of coherence does not suffice. At the same time, our approach does not require any manual rules for handling tree insertions, in contrast to symbolic text planners.

[2]Data and code used in this paper are available at `http://people.csail.mit.edu/edc/emnlp07/`

[3]Independently and simultaneously with our work, Elsner and Charniak (2007) have studied the sentence insertion task in a different setting.

[4]Though statistical methods have been used to induce such trees (Soricut and Marcu, 2003), they are not used for ordering and other text-structuring tasks.

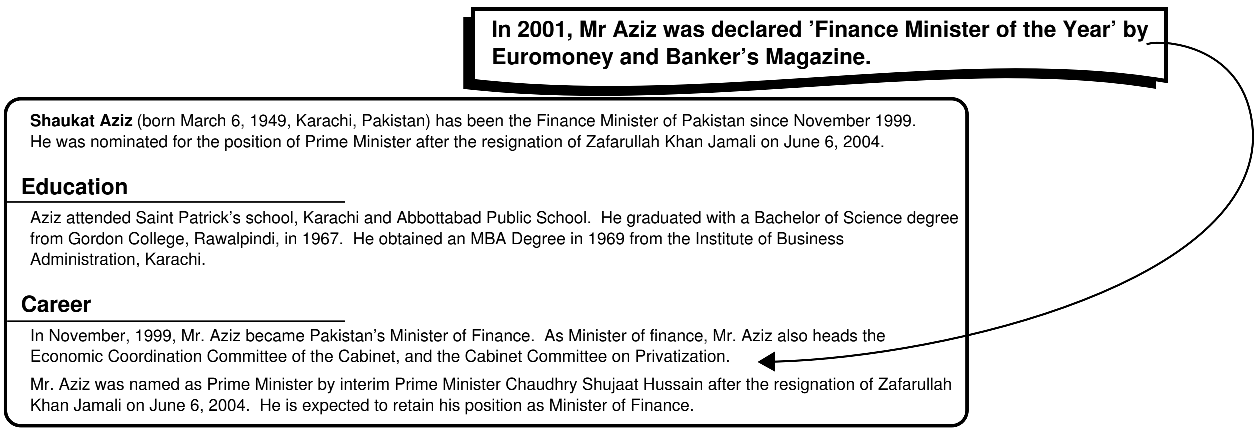

Figure 1: An example of Wikipedia insertion.

Hierarchical Learning There has been much recent research on multiclass hierarchical classification. In this line of work, the set of possible labels is organized hierarchically, and each input must be assigned a node in the resulting tree. A prototype weight vector is learned for each node, and classification decisions are based on all the weights along the path from node to root. The essence of this scheme is that the more ancestors two nodes have in common, the more parameters they are forced to share. Many learning methods have been proposed, including SVM-style optimization (Cai and Hofmann, 2004), incremental least squares estimation (Cesa-Bianchi et al., 2006b), and perceptron (Dekel et al., 2004).

This previous work rests on the assumption that a predetermined set of atomic labels with a fixed hierarchy is given. In our task, however, the set of possible insertion points – along with their hierarchical organization – is unique to each input document. Furthermore, nodes exhibit rich internal feature structure and cannot be identified across documents, except insofar as their features overlap. As is commonly done in NLP tasks, we make use of a feature function which produces one feature vector for each possible insertion point. We then choose among these feature vectors using a single weight vector (casting the task as a *structured ranking* problem rather than a *classification* problem). In this framework, an explicit hierarchical view is no longer necessary to achieve parameter tying. In fact, each parameter will be shared by exactly those insertion points which exhibit the corresponding feature, both across documents and within a single document. Higher level parameters will thus naturally be shared by all paragraphs within a single section.

In fact, when the perceptron update rule of (Dekel et al., 2004) – which modifies the weights of every divergent node along the predicted and true paths – is used in the ranking framework, it becomes virtually identical with the standard, flat, ranking perceptron of Collins (2002).[5] In contrast, our approach shares the idea of (Cesa-Bianchi et al., 2006a) that "if a parent class has been predicted wrongly, then errors in the children should not be taken into account." We also view this as one of the key ideas of the incremental perceptron algorithm of (Collins and Roark, 2004), which searches through a complex decision space step-by-step and is immediately updated at the first wrong move.

Our work fuses this idea of selective hierarchical updates with the simplicity of the perceptron algorithm and the flexibility of arbitrary feature sharing inherent in the ranking framework.

3 The Algorithm

In this section, we present our sentence insertion model and a method for parameter estimation. Given a hierarchically structured text composed of sections and paragraphs, the sentence insertion model determines the best paragraph within

[5] The main remaining difference is that Dekel et al. (2004) use a passive-aggressive update rule (Crammer et al., 2006) and in doing so enforce a margin based on tree distance.

which to place the new sentence. To identify the exact location of the sentence within the chosen paragraph, local ordering methods such as (Lapata, 2003) could be used. We formalize the insertion task as a structured ranking problem, and our model is trained using an online algorithm. The distinguishing feature of the algorithm is a selective correction mechanism that focuses the model update on the relevant layer of the document's feature hierarchy.

The algorithm described below can be applied to any hierarchical ranking problem. For concreteness, we use the terminology of the sentence insertion task, where a hierarchy corresponds to a document with sections and paragraphs.

3.1 Problem Formulation

In a sentence insertion problem, we are given a training sequence of instances $(s^1, \mathcal{T}^1, \ell^1), \ldots, (s^m, \mathcal{T}^m, \ell^m)$. Each instance contains a sentence s, a hierarchically structured document $\mathcal{T}$, and a node ℓ representing the correct insertion point of s into $\mathcal{T}$. Although ℓ can generally be any node in the tree, in our problem we need only consider leaf nodes. We cast this problem in the ranking framework, where a feature vector is associated with each sentence-node pair. For example, the feature vector of an internal, section-level node may consider the word overlap between the inserted sentence and the section title. At the leaf level, features may include an analysis of the overlap between the corresponding text and sentence. In practice, we use disjoint feature sets for different layers of the hierarchy, though in theory they could be shared.

Our goal then is to choose a leaf node by taking into account its feature vector as well as feature vectors of all its ancestors in the tree.

More formally, for each sentence s and hierarchically structured document $\mathcal{T}$, we are given a set of feature vectors, with one for each node: $\{\phi(s,n) : n \in \mathcal{T}\}$. We denote the set of leaf nodes by $\mathcal{L}(\mathcal{T})$ and the path from the root of the tree to a node n by $\mathcal{P}(n)$. Our model must choose one leaf node among the set $\mathcal{L}(\mathcal{T})$ by examining its feature vector $\phi(s,\ell)$ as well as all the feature vectors along its path: $\{\phi(s,n) : n \in \mathcal{P}(\ell)\}$.

Input : $(s^1, \mathcal{T}^1, \ell^1), \ldots, (s^m, \mathcal{T}^m, \ell^m)$.
Initialize : Set $\mathbf{w}^1 = 0$
Loop : For $t = 1, 2, ..., N$:

1. Get a new instance $s^t, \mathcal{T}^t$.
2. Predict $\hat{\ell}^t = \arg\max_{\ell \in \mathcal{L}(\mathcal{T})} \mathbf{w}^t \cdot \Phi(s^t, \ell)$.
3. Get the new label ℓ^t.
4. If $\hat{\ell}^t = \ell^t$:
   $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t$
   Else:
   $i^* \leftarrow \max\{i : \mathcal{P}(\ell^t)^i = \mathcal{P}(\hat{\ell}^t)^i\}$
   $a \leftarrow \mathcal{P}(\ell^t)^{i^*+1}$
   $b \leftarrow \mathcal{P}(\hat{\ell}^t)^{i^*+1}$
   $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \phi(s,a) - \phi(s,b)$

Output : $\mathbf{w}^{N+1}$.

Figure 2: Training algorithm for the hierarchical ranking model.

3.2 The Model

Our model consists of a weight vector $\mathbf{w}$, each weight corresponding to a single feature. The features of a leaf are aggregated with the features of all its ancestors in the tree. The leaf score is then computed by taking the inner product of this aggregate feature vector with the weights $\mathbf{w}$. The leaf with the highest score is then selected.

More specifically, we define the *aggregate feature vector* of a leaf ℓ to be the sum of all features found along the path to the root:

$$\Phi(s,\ell) = \sum_{n \in \mathcal{P}(\ell)} \phi(s,n) \tag{1}$$

This has the effect of *stacking together* features found in a single layer, and *adding* the values of features found at more than one layer.

Our model then outputs the leaf with the highest scoring aggregate feature vector:

$$\arg\max_{\ell \in \mathcal{L}(\mathcal{T})} \mathbf{w} \cdot \Phi(s,\ell) \tag{2}$$

Note that by using this criterion, our decoding method is equivalent to that of the standard linear ranking model. The novelty of our approach lies in our training algorithm which uses the hierarchical feature decomposition of Equation 1 to pinpoint its updates along the path in the tree.

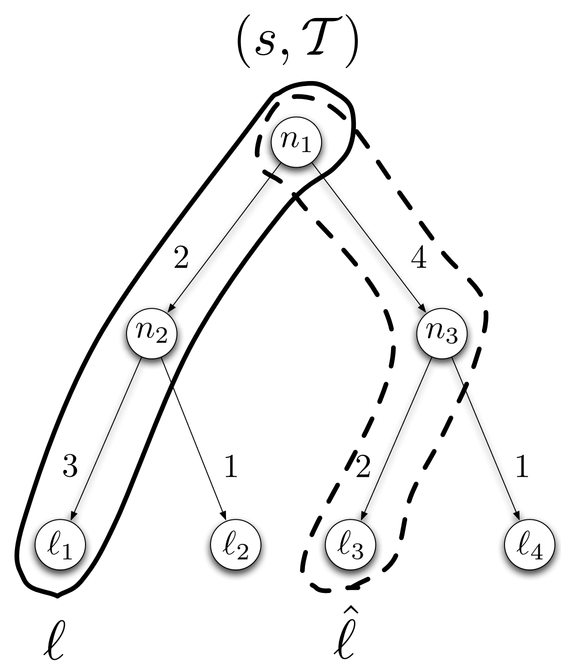

Figure 3: An example of a tree with the corresponding model scores. The path surrounded by solid lines leads to the correct node ℓ_1. The path surrounded by dotted lines leads to ℓ_3, the predicted output based on the current model.

3.3 Training

Our training procedure is implemented in the online learning framework. The model receives each training instance, and predicts a leaf node according to its current parameters. If an incorrect leaf node is predicted, the weights are updated based on the divergence between the predicted path and the true path. We trace the paths down the tree, and only update the weights of the features found at the split point. Updates for shared nodes along the paths would of course cancel out. In contrast to the standard ranking perceptron as well as the hierarchical perceptron of (Dekel et al., 2004), no features further down the divergent paths are incorporated in the update. For example, if the model incorrectly predicts the section, then only the weights of the section features are updated whereas the paragraph feature weights remain untouched.

More formally, let $\hat{\ell}$ be the predicted leaf node and let $\ell \neq \hat{\ell}$ be the true leaf node. Denote by $\mathcal{P}(\ell)^i$ the i^{th} node on the path from the root to ℓ. Let i^* be the depth of the lowest common ancestor of ℓ and $\hat{\ell}$ (i.e., $i^* = \max\{i : \mathcal{P}(\ell)^i = \mathcal{P}(\hat{\ell})^i\}$). Then the update rule for this round is:

$$\mathbf{w} \leftarrow \mathbf{w} + \phi\left(s, \mathcal{P}(\ell)^{i^*+1}\right) - \phi\left(s, \mathcal{P}(\hat{\ell})^{i^*+1}\right) \quad (3)$$

Full pseudo-code for our hierarchical online training algorithm is shown in Figure 2.

We illustrate the selective update mechanism on the simple example shown on Figure 3. The correct prediction is the node ℓ_1 with an aggregate path score of 5, but ℓ_3 with the higher score of 6 is predicted. In this case, both the section and the paragraph are incorrectly predicted. In response to this mistake, the features associated with the correct section, n_2, are added to the weights, and the features of the incorrectly predicted section, n_3, are subtracted from the weights. An alternative update strategy would be to continue to update the feature weights of the leaf nodes, ℓ_1 and ℓ_3. However, by identifying the exact source of path divergence we preserve the previously learned balance between leaf node features.

4 Features

Features used in our experiments are inspired by previous work on corpus-based approaches for discourse analysis (Marcu and Echihabi, 2002; Lapata, 2003; Elsner et al., 2007). We consider three types of features: lexical, positional, and temporal. This section gives a general overview of these features (see code for further details.)

Lexical Features Lexical features have been shown to provide strong cues for sentence positioning. To preserve text cohesion, an inserted sentence has to be topically close to its surrounding sentences. At the paragraph level, we measure topical overlap using the TF*IDF weighted cosine similarity between an inserted sentence and a paragraph. We also use a more linguistically refined similarity measure that computes overlap considering only subjects and objects. Syntactic analysis is performed using the MINIPAR parser (Lin, 1998).

The overlap features are computed at the section level in a similar way. We also introduce an additional section-level overlap feature that computes the cosine similarity between an inserted sentence and the first sentence in a section. In our corpus, the opening sentence of a section is typically strongly indicative of its topic, thus providing valuable cues for section level insertions.

In addition to overlap, we use lexical features that capture word co-occurrence patterns in coherent texts. This measure was first introduced in the context of sentence ordering by Lapata (2003). Given a collection of documents in a specific domain, we compute the likelihood that a pair of words co-occur in adjacent sentences. From these counts, we induce the likelihood that two sentences are adjacent to each other. For a given paragraph and an inserted sentence, the highest adjacency probability between the inserted sentence and paragraph sentences is recorded. This feature is also computed at the section level.

Positional Features These features aim to capture user preferences when positioning new information into the body of a document. For instance, in the Wikipedia data, insertions are more likely to appear at the end of a document than at its beginning. We track positional information at the section and paragraph level. At the section level, we record whether a section is the first or last of the document. At the paragraph level, there are four positional features which indicate the paragraph's position (i.e., start or end) within its individual section and within the document as a whole.

Temporal Features The text organization may be influenced by temporal relations between underlying events. In temporally coherent text, events that happen in the same time frame are likely to be described in the same segment. Our computation of temporal features does not require full fledged temporal interpretation. Instead, we extract these features based on two categories of temporal cues: verb tense and date information. The verb tense feature captures whether a paragraph contains at least one sentence using the same tense as the inserted sentence. For instance, this feature would occur for the inserted sentence in Figure 1 since both the sentence and chosen paragraph employ the past tense.

Another set of features takes into account the relation between the dates in a paragraph and those in an inserted sentence. We extract temporal expressions using the **TIMEX2** tagger (Mani and Wilson, 2000), and compute the time interval for a paragraph bounded by its earliest and latest dates. We record the degree of overlap between the paragraph time interval and insertion sentence time interval.

		Section	Paragraph	Tree Dist
T1	J1	0.575	0.5	1.85
	J2	0.7	0.525	1.55
T2	J3	0.675	0.55	1.55
	J4	0.725	0.55	1.45

Table 1: Accuracy of human insertions compared against gold standard from Wikipedia's update log. T1 is a subset of the data annotated by judges J1 and J2, while T2 is annotated by J3 and J4.

5 Experimental Set-Up

Corpus Our corpus consists of Wikipedia articles that belong to the category "Living People." We focus on this category because these articles are commonly updated: when new facts about a person are featured in the media, a corresponding entry in Wikipedia is likely to be modified. Unlike entries in a professionally edited encyclopedia, these articles are collaboratively written by multiple users, resulting in significant stylistic and content variations across texts in our corpus. This property distinguishes our corpus from more stylistically homogeneous collections of biographies used in text generation research (Duboue and McKeown, 2003).

We obtain data on insertions[6] from the update log that accompanies every Wikipedia entry. For each change in the article's history, the log records an article before and after the change. From this information, we can identify the location of every inserted sentence. In cases where multiple insertions occur over time to the same article, they are treated independently of each other. To eliminate spam, we place constraints on inserted sentences: (1) a sentence has at least 8 tokens and at most 120 tokens; (2) the MINIPAR parser (Lin, 1998) can identify a subject or an object in a sentence.

This process yields 4051 insertion/article pairs, from which 3240 pairs are used for training and 811 pairs for testing. These insertions are derived from 1503 Wikipedia articles. Relative to other corpora used in text structuring research (Barzilay and Lee, 2004; Lapata, 2003; Karamanis et al., 2004), texts in

[6] Insertion is only one type of recorded update, others include deletions and sentence rewriting.

our collection are long: an average article has 32.9 sentences, organized in 3.61 sections and 10.9 paragraphs. Our corpus only includes articles that have more than one section. When sentences are inserted between paragraphs, by convention we treat them as part of the previous paragraph.

Evaluation Measures We evaluate our model using *insertion accuracy* at the section and paragraph level. This measure computes the percentage of matches between the predicted location of the insertion and the true placement. We also report the *tree distance* between the predicted position and the true location of an inserted sentence. *Tree distance* is defined as the length of the path through the tree which connects the predicted and the true paragraph positions. This measure captures section level errors (which raise the connecting path higher up the tree) as well as paragraph level errors (which widen the path across the tree).

Baselines Our first three baselines correspond to naive insertion strategies. The RANDOMINS method randomly selects a paragraph for a new sentence, while FIRSTINS and LASTINS insert a sentence into the first and the last paragraph, respectively.

We also compare our HIERARCHICAL method against two competitive baselines, PIPELINE and FLAT. The PIPELINE method separately trains two rankers, one for section selection and one for paragraph selection. During decoding, the PIPELINE method first chooses the best section according to the section-layer ranker, and then selects the best paragraph within the chosen section according to the paragraph-layer ranker. The FLAT method uses the same decoding criterion as our model (Equation 2), thus making use of all the same features. However, FLAT is trained with the standard ranking perceptron update, without making use of the hierarchical decomposition of features in Equation 1.

Human Performance To estimate the difficulty of sentence insertion, we conducted experiments that evaluate human performance on the task. Four judges collectively processed 80 sentence/article pairs which were randomly extracted from the test set. Each insertion was processed by two annotators.

Table 1 shows the insertion accuracy for each judge when compared against the Wikipedia gold standard. On average, the annotators achieve 66% accuracy in section placement and 53% accuracy in paragraph placement. We obtain similar results when we compare the agreement of the judges against each other: 65% of section inserts and 48% of paragraph inserts are identical between two annotators. The degree of variability observed in this experiment is consistent with human performance on other text structuring tasks such as sentence ordering (Barzilay et al., 2002; Lapata, 2003).

	Section	Paragraph	Tree Dist
RANDOMINS	0.318*	0.134*	3.10*
FIRSTINS	0.250*	0.136*	3.23*
LASTINS	0.305*	0.215*	2.96*
PIPELINE	0.579	0.314*	2.21*
FLAT	0.593	0.313*	2.19*
HIERARCHY	**0.598**	**0.383**	**2.04**

Table 2: Accuracy of automatic insertion methods compared against the gold standard from Wikipedia's update log. The third column gives tree distance, where a lower score corresponds to better performance. Diacritic * ($p < 0.01$) indicates whether differences in accuracy between the given model and the Hierarchical model is significant (using a Fisher Sign Test).

6 Results

Table 2 shows the insertion performance of our model and the baselines in terms of accuracy and tree distance error. The two evaluation measures are consistent in that they yield roughly identical rankings of the systems. Assessment of statistical significance is performed using a Fisher Sign Test. We apply this test to compare the accuracy of the HIERARCHICAL model against each of the baselines.

The results in Table 2 indicate that the naive insertion baselines (RANDOMINS, FIRSTINS, LASTINS) fall substantially behind the more sophisticated, trainable strategies (PIPELINE, FLAT, HIERARCHICAL). Within the latter group, our HIERARCHICAL model slightly outperforms the others based on the coarse measure of accuracy at the section level. However, in the final paragraph-level analysis, the performance gain of our model over its counterparts is quite significant. Moreover, according to tree distance error, which incorporates error at both the section and the paragraph level, the performance of the

HIERARCHICAL method is clearly superior. This result confirms the benefit of our selective update mechanism as well as the overall importance of joint learning.

Viewing human performance as an upper bound for machine performance highlights the gains of our algorithm. We observe that the gap between our method and human performance at the paragraph level is 32% smaller than that between the PIPELINE model and human performance, as well as the FLAT model and human performance.

Sentence-level Evaluation Until this point, we have evaluated the accuracy of insertions at the paragraph level, remaining agnostic as to the specific placement within the predicted paragraph. We perform one final evaluation to test whether the global hierarchical view of our algorithm helps in determining the *exact* insertion point. To make sentence-level insertion decisions, we use a local model in line with previous sentence-ordering work (Lapata, 2003; Bollegala et al., 2006). This model examines the two surrounding sentences of each possible insertion point and extracts a feature vector that includes lexical, positional, and temporal properties. The model weights are trained using the standard ranking perceptron (Collins, 2002).

We apply this local insertion model in two different scenarios. In the first, we ignore the global hierarchical structure of the document and apply the local insertion model to every possible sentence pair. Using this strategy, we recover 24% of correct insertion points. The second strategy takes advantage of global document structure by first applying our hierarchical paragraph selection method and only then applying the local insertion to pairs of sentences within the selected paragraph. This approach yields 35% of the correct insertion points. This statistically significant difference in performance indicates that purely local methods are insufficient when applied to complete real-world documents.

7 Conclusion and Future Work

We have introduced the problem of sentence insertion and presented a novel corpus-based method for this task. The main contribution of our work is the incorporation of a rich hierarchical text representation into a flexible learning approach for text structuring. Our learning approach makes key use of the hierarchy by selecting to update only the layer found responsible for the incorrect prediction. Empirical tests on a large collection of real-world insertion data confirm the advantage of this approach.

Sentence ordering algorithms too are likely to benefit from a hierarchical representation of text. However, accounting for long-range discourse dependencies in the unconstrained ordering framework is challenging since these dependencies only appear when a particular ordering (or partial ordering) is considered. An appealing future direction lies in simultaneously inducing hierarchical and linear structure on the input sentences. In such a model, tree structure could be a hidden variable that is influenced by the observed linear order.

We are also interested in further developing our system for automatic update of Wikipedia pages. Currently, our system is trained on insertions in which the sentences of the original text are not modified. However, in some cases additional text revisions are required to guarantee coherence of the generated text. Further research is required to automatically identify and handle such complex insertions.

Acknowledgments

The authors acknowledge the support of the National Science Foundation (CAREER grant IIS-0448168 and grant IIS-0415865) and the Microsoft Research Faculty Fellowship. Any opinions, findings, and conclusions or recommendations expressed above are those of the authors and do not necessarily reflect the views of the NSF. Thanks to S.R.K. Branavan, Eugene Charniak, Michael Collins, Micha Elsner, Jacob Eisenstein, Dina Katabi, Igor Malioutov, Christina Sauper, Luke Zettlemoyer, and the anonymous reviewers for helpful comments and suggestions. Data used in this work was collected and processed by Christina Sauper.

References

Regina Barzilay and Mirella Lapata. 2005. Modeling local coherence: An entity-based approach. In *Proceedings of the ACL*, pages 141–148.

Regina Barzilay and Lillian Lee. 2004. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *Proceedings of HLT-NAACL*, pages 113–120.

Regina Barzilay, Noemie Elhadad, and Kathleen McKeown. 2002. Inferring strategies for sentence ordering in multidocument news summarization. *JAIR*, 17:35–55.

Danushka Bollegala, Naoaki Okazaki, and Mitsuru Ishizuka. 2006. A bottom-up approach to sentence ordering for multi-document summarization. In *Proceedings of the COLING/ACL*, pages 385–392.

Lijuan Cai and Thomas Hofmann. 2004. Hierarchical document categorization with support vector machines. In *Proceedings of the CIKM*, pages 78–87.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. 2006a. Hierarchical classification: Combining bayes with SVM. In *Proceedings of the ICML*, pages 177–184.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. 2006b. Incremental algorithms for hierarchical classification. *JMLR*, 7:31–54.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the ACL*, pages 111–118.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the EMNLP*, pages 1–8.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *JMLR*, 7:551–585.

Ofer Dekel, Joseph Keshet, and Yoram Singer. 2004. Large margin hierarchical classification. In *Proceedings of the ICML*, pages 209–216.

Pablo Duboue and Kathleen McKeown. 2003. Statistical acquisition of content selection rules for natural language generation. In *Proceedings of the EMNLP*, pages 121–128.

Micha Elsner and Eugene Charniak. 2007. A generative discourse-new model for text coherence. Technical Report CS-07-04, Brown University.

Micha Elsner, Joseph Austerweil, and Eugene Charniak. 2007. A unified local and global model for discourse coherence. In *Proceedings of the HLT-NAACL*, pages 436–443.

Marti Hearst. 1994. Multi-paragraph segmentation of expository text. In *Proceedings of the ACL*, pages 9–16.

Nikiforos Karamanis, Massimo Poesio, Chris Mellish, and Jon Oberlander. 2004. Evaluating centering-based metrics of coherence for text structuring using a reliably annotated corpus. In *Proceedings of the ACL*, pages 391–398.

Mirella Lapata. 2003. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the ACL*, pages 545–552.

Dekang Lin. 1998. Dependency-based evaluation of minipar. In *Proceedings of the Workshop on the Evaluation of Parsing Systems, LREC*, pages 48–56.

Inderjeet Mani and George Wilson. 2000. Robust temporal processing of news. In *Proceedings of the ACL*, pages 69–76.

Daniel Marcu and Abdessamad Echihabi. 2002. An unsupervised approach to recognizing discourse relations. In *Proceedings of the ACL*, pages 368–375.

Naoaki Okazaki, Yutaka Matsuo, and Mitsuru Ishizuka. 2004. Improving chronological sentence ordering by precedence relation. In *Proceedings of the COLING*, pages 750–756.

Ehud Reiter and Robert Dale. 1990. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge.

Radu Soricut and Daniel Marcu. 2003. Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the HLT-NAACL*, pages 149–156.