A Tree-Position Kernel for Document Compression

Hal Daumé III
University of Southern California
Department of Computer Science
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
hdaume@isi.edu

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

Abstract

We describe our entry into the DUC 2004 automatic document summarization competition. We competed only in the single document, headline generation task. Our system is based on a novel kernel dubbed the tree position kernel, combined with two other well-known kernels. Our system performs well on white-box evaluations, but does very poorly in the overall DUC evaluation. However, the latter results are offset by the fact that baseline systems consistently outperform well engineered systems.

1 Introduction

We participated in only one task for DUC 2004: summarization of single documents down to a very short summary of 75 bytes or less (aka a headline). In order to maintain brevity, the interested reader is referred to other papers in this collection (Hahn and Harman, 2004) or to (Mani, 2001) for motivation and standard methodologies used for this task.

We treat the single document summarization task (headline generation task) as one of compression: we select words and phrases from the original document as our summary. This is a reasonably common view, and has typically been performed using some sort of noisy-channel model (Knight and Marcu, 2002; Daumé III and Marcu, 2002; Banko et al., 2000; R. Schwartz and Door, 2002). In such a model, one needs to build two complementary components: a source (language) model for modeling fluency of a hypothesized summary, and a channel (compression) model for modeling the faith of a hypothesized summary to the original document. These two models compete via Bayes' rule in a search process performed by a decoder in order to find a good summary.

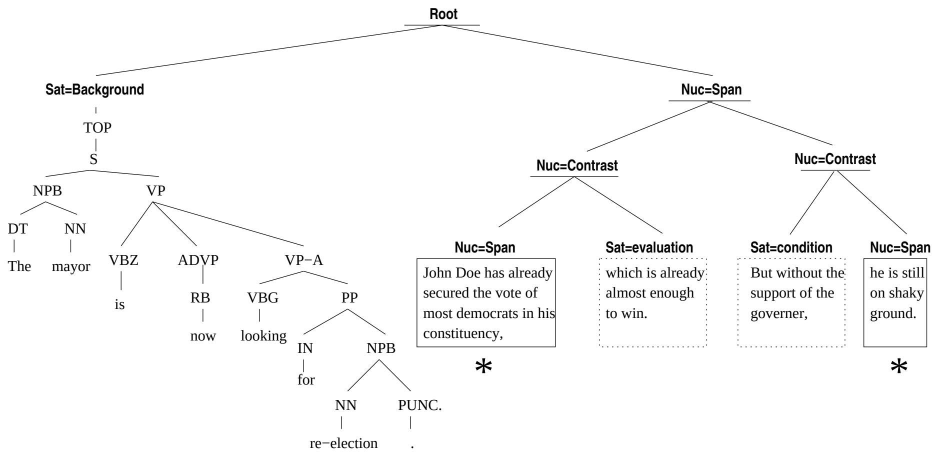
In our work, we expand on the document compression model of Daumé III and Marcu (2002), which follows the noisy-channel paradigm. In Section 2, we will briefly describe this model, the data it is trained on, and discuss its immediate weaknesses (beyond the fact that it is a purely compression-based model). In Section 3, we will briefly review kernel-based learning methods, since our augmentation to the compression model relies on them. In Section 4, we will describe a new model for computing channel probabilities based on a new kernel we introduce to solve the compression problem. The language model will be described briefly in Section 5 and we will conclude with white-box and black-box experiments using our system in Section 6.

2 Our Document Compression Model

In the document compression model described in (Daumé III and Marcu, 2002), one first builds a discourse tree over the source document (Marcu, 2000), merging leaves (EDUs) appropriately, so that each leaf corresponds to exactly one sentence. These sentences are then syntactically parsed, in that case using Collin's parser (1997), and the textual leaves in the discourse tree are replaced by the corresponding syntax trees. Thus, the entire document has been converted into a large tree, made up both of discourse nodes and syntactic constituents (which we call a "DS tree"). We show an example of a such a DS tree (with only one of the syntactic constituents expanded to save space) in Figure 1.

2.1 The Channel Model

In the channel model for the document compression model, one assumes that a tree corresponding to a summary has been given and calculates $p(d|s)$, the probability of the document tree given that summary tree. In the original model, this was calculated through a generative story, in which context free rules were applied. For instance, if the summary tree contained a syntactic node `NP -> DT NN` and the corresponding node in the discourse tree was `NP -> DT JJ NN PP`, then, for that



Root
Sat=Background
TOP
S
NPB
VP
DT
NN
The
mayor
VBZ
is
ADVP
RB
now
VP−A
VBG
looking
PP
IN
for
NPB
NN
re−election
PUNC.
.
Nuc=Span
Nuc=Contrast
Nuc=Contrast
Nuc=Span
John Doe has already secured the vote of most democrats in his constituency,
Sat=evaluation
which is already almost enough to win.
Sat=condition
But without the support of the governer,
Nuc=Span
he is still on shaky ground.
*
*

Figure 1: An example discourse (full)/syntax (partial) tree.

particular node, the probability of *expansion* would be $p($NP -> DD JJ NN PP$|$NP -> DT NN$)$. Rules at the discourse level had the same form, but included nuclearity information as well as relation information.

2.2 The Training Data

We use two separate sources of data for training the compression model (namely, estimating probabilities of the form $p($NP -> DD JJ NN PP$|$NP -> DT NN$)$). One source, the Ziff-Davis corpus, is used for training probabilities for the syntactic CFG rules; the other, the RST corpus (Carlson et al., 2003), is used for training probabilities for the discourse CFG rules. More information on the former can be found in (Knight and Marcu, 2002), while more information on the latter can be found in (Daumé III and Marcu, 2002).

Briefly, in order to estimate the channel probabilities for the syntax nodes, we take a collection of documents and their corresponding human-written abstracts. For each abstract sentence "$w_1 w_2 w_3 \dots w_n$", we search in the corresponding summary for a sentence of the form "$.*w_1.*w_2.*w_3.*\dots *w_n.*$", where "$.*$" can be anything. That is, we look for pairs of an abstract sentence and a document sentence, for which the abstract sentence *is* a compression of the document sentence. We then syntactically parsed the document sentence and considered any node in the syntax tree that dominated a word that was selected in the abstract sentence to be "good." Based on these data, we calculated maximum likelihood estimates (i.e., counted relative frequencies) of the context free rules.

For the discourse probabilities, the task was somewhat simpler. In the RST corpus, there are 150 documents that are hand-parsed and have a paired extractive summary at the EDU level. For instance, for the discourse tree in Figure 1, it might be that the two starred EDUs were extracted in such a summary. As in the syntax case, these selections are propagated up the tree so that any node dominating them is also selected. In the case in this Figure, the four counts that would be increased are:

```
Nuc=Contrast -> Nuc=Span Sat=eval |
   Nuc=Contrast -> Nuc=Span
Nuc=Contrast -> Sat=condition Nuc=Span |
   Nuc=Contrast -> Nuc=Span
Nuc=Span -> Nuc=Contrast Nuc=Contrast |
   Nuc=Span -> Nuc=Contrast Nuc=Contrast
Root -> Sat=Background Nuc=Span |
   Root -> Nuc=Span
```

2.3 The Decoder

The decoder has the job of combining the channel model probability $p(d|s)$ with the language model probability $p(s)$ (we used a bigram language model) to find $\text{argmax}_s\, p(s|d) = \text{argmax}_s\, p(s)p(d|s)$. In the original work and in the work described in this paper, we use a packed-forest representation for the problem and the generic decoder build by Langkilde (2000) for performing the argmax operation.

2.4 Difficulties with the Model

There are several problems with the model described in this section. Of course, the fact that it only performs compressions and cannot change words or reorder constituents is a large flaw, but it is not a flaw we seek to fix in this paper. Constraining ourselves to the compression task, perhaps the largest difficulty with this model is that we are required to estimate an enormous set of parameters from very little data.

For the syntactic probabilities, out of the Ziff-Davis corpus, a collection of roughly $4,000$ document/abstract pairs, with abstracts averaging a length of 15 sentences, only just over $1,050$ compression pairs were extracted, a recall of 1.75%. In a subsequent study of downloading document/abstract pairs off of news sites, we found that out of roughly $20,000$ downloaded abstract sentences, only 6 were compressions of document sentences according to the automatic extraction method described earlier. Human review of these found that two were identical and three were incorrect, leaving essentially two out of twenty thousand as good examples.

Using these thousand training sentences from the Ziff-Davis corpus, a total of $1,132$ different context free rules were extracted, 708 of which were observed only once. Given these skewed numbers, it is unlikely that maximum likelihood techniques would be able to estimate parameters in a reasonable way.

For the discourse probabilities, the same story holds. From the 150 extracts in the RST corpus, $1,146$ unique rules were extracted, 629 of which occurred only once. Again, based on this data, it is unlikely that simple maximum likelihood estimates will adequately model the conditional probability distribution we seek.

Conversely, the models we are estimating are completely context free. That is, the probability distributions are conditioned on exactly one context-free rule: when making a decision what to cut and what to keep, the model only has access to the internal tag (syntax tag or discourse tag) and the labels of each of its children. This makes for a clean model, but unfortunately does not account for features that have been found to be incredibly useful in the literature: for instance, position in the original document.

We deal with both of these issues in the model presented in this paper. However, before moving on to how we describe our new channel model, we will briefly review kernel-based learning methods, since they form the back-bone of the new channel.

3 Kernel-Based Learning Methods

In the simplest binary classification setting, given an input vector $\vec{x}$ and target class $y \in \{-1, 1\}$ one wishes to find a weight vector $\vec{w}$ and a bias term b such that $f(\vec{x}) = \vec{w} \cdot \vec{x} + b$ is positive whenever $y = 1$ and is negative whenever $y = -1$. A solution of the form $(\hat{\vec{w}}, \hat{b})$ is a linear separator, since the decision boundary described by this solution is a “flat” hyperplane.

In many learning algorithms, however, all computations involving the input vectors $\vec{x}_i$ take the form of dot products: $\vec{x}_i \cdot \vec{x}_j$. One can then simply replace all occurrences of vectors $\vec{x}_i$ with the image of that vector under some (possibly non-linear) transformation Φ. Then, the dot products all have the form $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$.

Since we are only ever taking dot products and never actually evaluating $\Phi(\vec{x}_i)$ explicitly, there is nothing from preventing us from using Φ to map $\vec{x}_i$ into a very high-dimensional or possibly infinite-dimensional space. Once $\vec{x}_i$ has been mapped into this high-dimensional space, learning can proceed as normal, where the linear separator now becomes a linear separator in this high-dimensional “feature space,” which may not (and typically does not) correspond to a linear separator in the original “input space.”

We can rewrite $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ as $k_\Phi(\vec{x}_i, \vec{x}_j)$ or more simply as $k_(\vec{x}_i, \vec{x}_j)$ when Φ is clear from context. It is clear that k is a symmetric function. When an arbitrary symmetric function k corresponds to a dot product in a feature space, it is called a *kernel*. A sufficient condition for a symmetric function k to be a kernel is that it is positive semi-definite; that is, for all reasonably smooth functions f, it holds that:

$$\int \int f(\vec{x}) k(\vec{x}, \vec{x}') f(x') \mathrm{d}\vec{x} \mathrm{d}\vec{x}' \geq 0 \tag{1}$$

Many popular learning techniques (in particular, support vector machines) make use of such kernels to find non-linear decision boundaries using, essentially, linear learning methods. For a good introduction to support vector machines, see the excellent tutorial by Burges (1998). For a tutorial-style introduction on the math behind kernel methods, see (Daumé III, 2004). Briefly, a soft-margin support vector machine performs the following optimization:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||\vec{w}||^2 + C \sum_{i=1}^{N} \xi_i & (2) \\ \text{subject to} \quad & y_i(\vec{w} \cdot \vec{x}_i + b) - 1 + \xi_i \geq 0 & (3) \\ & \xi_i \geq 0 & (4) \end{aligned}$$

The minimization described in Eq (2) describes a trade-off between good expected generalization (by minimizing the norm of $\vec{w}$) and training error (the sum of the ξ “slack” variables). The value C controls this trade-off: if C is too large, we risk overfitting the training data, while if C is too small, we might find too simple of a solution. Eq (3) ensures that the value predicted by the machine on all test points is at most ξ_i away from being correct. By representing this optimization problem in its “dual” form, we obtain an result that depends only on dot products between input vectors, thus enabling us to “kernelize” the algorithm.

It is an important trivial result that we will need later that for any two positive semi-definite symmetric functions (i.e., kernels) k and l, and for any real numbers α and β, the following expressions are all, always, also positive semi-definite:

$$\begin{aligned} m_1(\vec{x}, \vec{x}') &= k(\vec{x}, \vec{x}') + l(\vec{x}, \vec{x}') \\ m_2(\vec{x}, \vec{x}') &= \alpha k(\vec{x}, \vec{x}') \\ m_3(\vec{x}, \vec{x}') &= k(\vec{x}, \vec{x}') + \beta \end{aligned}$$

All three of these facts can be verified by plugging into Eq (1) under the assumptions of positive semi-definiteness of k and l. These results give us a straightforward way to combine multiple kernels, as described in the next section.

4 A Novel Kernel for the Channel

By viewing the compression problem as a binary classification problem (should I keep this node or not?), we are able to leverage the wealth of research on binary classifiers for this problem. From such a viewpoint, we can think of the problem as being given a complete DS tree and a position in that tree and then asking whether the node at that position should be kept. However, unlike the channel model described earlier, we can take any amount of context we want into account.

We use a combination of three kernels: a tree position kernel k_p, a tree content kernel k_c, and a global feature kernel k_f. We combine these three kernels with weighting factors α and β to given an overall kernel k of the form:

$$k(\vec{x}, \vec{x}') = \alpha k_p(\vec{x}, \vec{x}') + \beta k_c(\vec{x}, \vec{x}') + k_f(\vec{x}, \vec{x}')$$

We also learn α and β from the data, as described later.

4.1 Tree Position Kernel

We introduce a tree position kernel in this section. The motivation behind defining this kernel is that we wish to take into account global position information when making a decision about keeping a node. In the original compression model, the only position information we are given is the tag of the parent node. We extend this to take into account the tags of all ancestors, as well as their siblings.

Formally, let Σ bet the alphabet of all discourse and syntax tags. Then, a context free rule is of the form $(p, c) \in \Sigma \times \Sigma^+$; here, p is the tag of the parent node and c is the sequence of tags of the children. We define a *path rule* as an extension to a context free rule, which also has the form $(p, c, s) \in \Sigma \times \Sigma^+ \times \mathbb{N}$. Here, p is still the parent and c is the list of children, but s is the "selection" index of one of the c children.

We define a *path* as a sequence of path rules $\langle r_1, \dots, r_n \rangle$ such that the parent tag of r_1 is the root of the tree. Thus, a path defines a unique traversal down the tree beginning with the root.

Consider the `VP-A` node in the DS tree in Figure 1. The path to this node is:

```
Root -> Sat=background Nuc=Span, 0
Sat=background -> Top, 0
Top -> S, 0
S -> NPB VP, 1
VP -> VBZ ADVP VP-A, 2
```

Given a path q and a tree t, we say that q is a subpath of t exactly when it is possible to find q by walking down t beginning at the root. We do not require that there are no intervening nodes in t; simply that q can be found. For instance, if q is:

```
Root -> Sat=background Nuc=Span, 0
VP -> VBZ ADVP VP-A, 2
```

then q is a subpath of t, even though t contains extraneous nodes between the two elements of q. If q is a subpath of t, then we write $t_{[q]}$ to denote minimal subtree t' of t such that q is still a subpath of t', and write $|t_{[q]}|$ to refer to the length of the path q in t'. If q is not a subpath of t, we define $|t_{[q]}|$ to be $-\infty$.

Given this definition of a path, let $N \in \{1, 2, \dots\}$ and $\lambda \in (0, 1)$ be given, and for each path q of length N, define:

$$\Phi_q(t) = \lambda^{|t_{[q]}|} \qquad (5)$$

Now, let $\Phi = \{\Phi_q : |q| \leq N\}$ be the potentially infinite set of all such feature functions for paths up to length N. We now define our kernel k_p as the dot product in this feature space:

$$k_p(\vec{x}, \vec{x}') = \Phi(\vec{x}) \cdot \Phi(\vec{y}) \qquad (6)$$

Since we have explicitly enumerated the feature functions, it is clear that k_p does define a positive semi-definite function.

Computing k_p is very similar to computing so-called string kernels. One can opt for either a dynamic programming solution, in which case the computational complexity of calculating k_p is $\mathcal{O}(N|\vec{x}||\vec{x}'|)$ where $|\vec{x}|$ and $|\vec{x}'|$ are the lengths of the longest paths in $\vec{x}$ and $\vec{x}'$ respectively. Alternatively, one can use suffix trees and sufficient statistics to calculate the kernel in $\mathcal{O}(|\vec{x}| + |\vec{x}'|)$, independent of N, (Vishwanathan and Smola, 2002). We opt for this latter approach, since it allows us to use arbitrarily large N and, in test cases, takes roughly a quarter of the time to compute.

4.2 Tree Content Kernel

In addition to the full context of the ancestors in the tree above the location being considered, we also wish to take into account the nodes below it. That is, we wish to consider the subtree rooted at the node currently under consideration.

The kernel we use for this is the tree kernel, k_c, originally described by Collins and Duffy (2002). This kernel behaves very similarly to the string kernel and the tree position kernel, but instead of counting sub*sequences*, it counts sub*trees*. Collins and Duffy present a dynamic programming solution to the tree kernel problem. Again,

	Syntax	**Discourse**
# of training points	$19,702$	$8,735$
baseline accuracy	53.2	51.9
accuracy on training data	82.7	71.2
cross-validation accuracy	76.7	65.8

Table 1: Results on training data

we opt instead for a solution based on suffix trees (Vishwanathan and Smola, 2002), that is computationally significantly less expensive.

4.3 Global Feature Kernel

Finally, we use the global feature kernel k_f to define other features of the tree and subtree that have been found useful in other studies of summarization. These are all real-valued features and include:

- The total number of words in the document
- The number of words dominated by this node
- The minimum, maximum, sum and mean tf-idf scores of each word in this node
- The number of nodes dominated by this node
- The tag of the current node
- The nuclearity of the current node (discourse only)

We use a standard RBF kernel for combining the values of these features into our global feature kernel k_f.

4.4 Training the Model

We use the support vector framework for training the model. The implementation we use is our own[1], since we needed to implement the new kernel (as well as the tree kernel, which is not available in any other publicly available software package).

The model has four open parameters, α and β (the weighting factors for the different kernels), γ (the precision of the RBF kernel k_f) and C (the SVM slack parameter; see Eq (2)). Ideally, we would estimate these parameters through cross validation. Unfortunately, since our discourse data is disjoint from our syntax data, we cannot do so. Instead, we optimize α through 10-fold cross validation, and then optimize β through 10-fold cross validation. Once all have been optimized, the optimization repeats once to ensure good compatibility between the optimized values.

Since the set of data for which we have discourse-level judgments is disjoint from the set of data for which we have syntax-level judgments, we train each part separately, only on the data available to it. The results of training are shown in Table 1. The baseline accuracies are calculated by selecting the class with the highest prior probability. We can see that these models significantly outperform the baseline, and that the generalization ability (calculated using cross-validation) is not much worse than the accuracy when tested on the training data.

[1]SVMsequel is freely available for download online at `http://www.isi.edu/~hdaume/SVMsequel/`.

4.5 The Probabilistic Model

We need to combine the output of the SVM with the probabilities given by the language model, so we use the algorithm suggested by Lin et al. (2003) for converting the outputs of the SVM to probabilistic outputs (Lin's algorithm is a different form of the one originally given by Platt, but doesn't suffer from the same numerical problems that Platt's does). The result of this is that we now have a model that says, for any internal node n, the probability that that node will be kept, $p(keep|n)$.

This yields a straightforward calculation of $p(s|d)$. For example, we can calculate $p(\texttt{p -> y}|\texttt{p -> x y z})$ as $p(keep|\texttt{y})(1-p(keep|\texttt{x}))(1-p(keep|\texttt{z}))$. Note that this probability distribution, namely $p(s|d)$ is the reverse of what we want: $p(d|s)$. Unfortunately, calculating $p(d|s)$ directly in a discriminative fashion is difficult, and thus we will simply use $p(s|d)$ in its stead. This is, of course, probabilistically unfashionable; however, such a swap has been observed to have little effect in other natural language tasks (Och et al., 2003).

5 The Language Model

We use a trigram language model. The model is computed by taking a trigram language model computed over 300 million words of plain English and another trigram language model over 200 thousand words of headline data ($30,218$ headlines), collected over the past 6 months from various news sites. These two language models are linearly interpolated, and backed off using Knesey-Ney smoothing techniques.

6 Evaluation and Conclusions

Five randomly selected outputs of our system from the DUC 2004 data are shown in Table 2. Many of these appear reasonable (though judgments are difficult without seeing the original document. Clearly there are grammaticality issues, especially in the first and third examples. Furthermore, without more sophisticated linguistic processing (for instance, replacing words with shorter synonyms), it is difficult to squeeze much information into such a short 75 byte span.

The evaluation carried out by DUC for this task was only automatic, using the ROUGE scoring software. This metric essentially computes n-gram recall rates against

Document ID	**Output**
APW19981027.0491	Cambodia's ruling party responded criticized a resolution passed
APW19981001.0312	Rebels attacked and killed six civilians said occurred overnight Wednesday
APW19981108.0837	To adopt further reforms in their effort Among the necessary steps is
APW19981221.1044	President Suleyman Demirel appeared to persuade bickering political leaders
NYT19981007.0302	The Internet offered a new deal in computing, a fresh opportunity.

Table 2: Five randomly selected outputs.

"gold standard" summaries. It is our opinion that the lack of human evaluations makes this whole evaluation more or less meaningless, since results regarding the effectiveness of such automatic metrics for single-document summarization to date are not entirely convincing.

Nevertheless, on the ROUGE metrics, our system consistently performed in the bottom 10 systems (out of approximately 75). This is quite disappointing, due to the relatively good performance our system showed on the white-box evaluation, depicted in Table 1. There are several possible direct reasons for this. First, all our training data is from definitively different genres than the DUC evaluation data. Secondly, our training data is aimed at compressions of about 20%, rather than the very short summaries the system is evaluated on. Lastly, since no human evaluation was performed, we cannot know how correlated the ROUGE scores would be to judgments a human might make.

Acknowledgments

This work was partially supported by DARPA-ITO grant N66001-00-1-9814, NSF grant IIS-0097846, and a USC Dean Fellowship to Hal Daumé III. Special thanks to Eric Horvitz and Kevin Knight for enlightening discussions related to the project.

References

Michele Banko, Vibhu Mittal, and Michael Witbrock. 2000. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL–2000)*, pages 318–325, Hong Kong, October 1–8.

Christopher J. C. Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurowski. 2003. Building a discourse-tagged corpus in the framework of rhetorical structure theory. *Current Directions in Discourse and Dialogue*.

Michael Collins and Nigel Duffy. 2002. Convolution kernels for natural language. In *NIPS 14*.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL–97)*, pages 16–23, Madrid, Spain, July 7-12.

Hal Daumé III and Daniel Marcu. 2002. A noisy-channel model for document compression. In *Proceedings of the Conference of the Association of Computational Linguistics (ACL 2002)*.

Hal Daumé III. 2004. From zero to reproducing kernel hilbert spaces in twelve pages or less. Available from `http://www.isi.edu/~hdaume/docs/`.

Udo Hahn and Donna Harman, editors. 2004. *Fourth Document Understanding Conference (DUC-2004)*, Boston, MA, May.

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

Irene Langkilde. 2000. Forest-based statistical sentence generation. In *Proceedings of NAACL*, pages 170–177, Seattle, Washington, April 30–May 3.

H.T. Lin, C.J. Lin, and R.C. Weng. 2003. A note on Platt's probabilistic outputs for support vector machines.

Inderjeet Mani. 2001. *Automatic Summarization*, volume 3 of *Natural Language Processing*. John Benjamins Publishing Company, Amsterdam/Philadelphia.

Daniel Marcu. 2000. *The Theory and Practice of Discourse Parsing and Summarization*. The MIT Press, Cambridge, Massachusetts.

Franz Josef Och, Richard Zens, and Hermann Ney. 2003. Efficient search for interactive statistical machine translation. In *Proceedings of EACL*, pages 287–293, Budapest, Hungary, April.

D. Zajic R. Schwartz and B. Door. 2002. Automatic headline generation for newspaper stories. In *Proceedings of the Document Understanding Conference (DUC)*.

S.V.N. Vishwanathan and A.J. Smola. 2002. Fast kernels on strings and trees. In *NIPS 14*.