Language Models and Structured Document Retrieval

Paul Ogilvie, Jamie Callan
Carnegie Mellon University
Pittsburgh, PA USA
{pto,callan}@cs.cmu.edu

ABSTRACT

We discuss possibilities for the use of language models in structured document retrieval. We use a tree-based generative language model for ranking documents and components. Nodes in the tree correspond to document components such as titles, sections, and paragraphs. At each node in the document tree, there is a language model. The language model for a leaf node is estimated directly from the text present in the document component associated with the node. Inner nodes in the tree are estimated using a linear interpolation among the children nodes. This paper also describes how some common structural queries would be satisfied within this model.

1. INTRODUCTION

With the growth of XML, there has been increasing interest in studying structured document retrieval. XML provides a standard for structured-document markup, and is increasingly being used. With the spread in the availability of structured documents, it is increasingly unclear whether the standard information retrieval algorithms are appropriate for retrieval on structured documents.

In this paper, we discuss how the generative language model approach to information retrieval could be extended to model and support queries on structured documents. We propose a tree-based language model to represent a structured document and its components. This structure is similar to many previous models for structured document retrieval [4][5][6][8][9][11], but differs in that language modeling provides some guidance in combining information from nodes in the tree and estimating term weights. The approach presented in this paper allows for structured queries and allows ranking of document components. It also matches some of our intuitions about coverage, which we discuss in Section 4.3.

The rest of the paper is structured as follows. Section 2 provides background in language modeling in information retrieval. In Section 3 we present our approach to modeling structured documents. Section 4 describes querying the tree-based language models presented in the previous section. In Section 5 we briefly discuss parameter training. We discuss relationships to other approaches to structured document retrieval in Section 6, and Section 7 concludes the paper.

2. BACKGROUND IN LANGUAGE MODELS FOR DOCUMENT RETRIEVAL

Language modeling was developed by the speech recognition community as a means of estimating the probability of a word sequence (such as a sentence) given a sequence of phonemes recognized from an audio signal. The speech recognition community has developed sophisticated methods for estimating these probabilities. Their most important contributions to the use of language models in information retrieval are smoothing and methods for combining language models.

In information retrieval, documents and sometimes queries are represented using language models. These are typically unigram language models, which are much like bags-of-words, where word order is ignored. The unigram language model specifically estimates the probability of a word given a chunk of text. It is a "unigram" language model because it ignores word order. Document ranking is done one of two ways: by measuring how much a query language model diverges from document language models [10][12], or by estimating the probability that each document generated the query string [13][7][14][15].

2.1 Kullback-Leibler Divergence

The first method ranks by the negative of the Kullback-Leibler (KL) divergence of the query from each document [10]:

$$-KL(\theta_Q \| \theta_D) = -\sum_w P(w|\theta_Q) \log \frac{P(w|\theta_Q)}{P(w|\theta_D)}$$

$$\propto \sum_w P(w|\theta_Q) \log P(w|\theta_D)$$

where θ_D is the language model estimated from the document, θ_Q is the language model estimated from the query, and $P(w|\theta)$ estimates the probability of the word w given the language model θ. The $P(w|\theta_Q)$ within the log can be dropped in ranking because it is a constant with respect to the query. Documents

where the query's model diverges less from the document's model are ranked higher.

2.2 The Generative Language Model

The generative method ranks documents by directly estimating the probability of the query using the documents' language models [13][7][14][15]:

$$P(Q|\theta_D) = \prod_{w \in (q_1,q_2,\ldots,q_n)} P(w|\theta_D)$$

$$\propto \sum_{w \in (q_1,q_2,\ldots,q_n)} \log P(w|\theta_D)$$

where $Q = (q_1,q_2,\ldots,q_n)$ is the query string. Documents more likely to have produced the query are ranked higher. Under the assumptions that query terms are generated independently and that the query language model used in KL-divergence is the maximum-likelihood estimate, the generative model and KL divergence produce the same rankings [12].

2.3 The Maximum-Likelihood Estimate of a Language Model

The most direct way to estimate a language model given some observed text is to use the maximum-likelihood estimate, assuming an underlying multinomial model. In this case, the maximum-likelihood estimate is also the empirical distribution or the histogram distribution. An advantage of this estimate is that it is easy to compute. It is very good at estimating the probability distribution for the language model when the size of the observed text is very large. It is given by:

$$P(w|\theta_T) = \frac{count(w;T)}{|T|}$$

where T is the observed text, $count(w;T)$ is the number of times the word w occurs in T, and $|T|$ is the length of the text. The maximum likelihood estimate is not good at estimating low frequency terms for short texts, as it will assign zero probability to those words. This creates a serious problem for estimating document language models in both KL divergence and generative language model approaches to ranking documents, as the log of zero is negative infinity. The solution to this problem is smoothing.

2.4 Smoothing

Smoothing is the re-estimation of the probabilities in a language model. Smoothing is motivated by the fact that many of the language models we estimate are based on a small sample of the "true" probability distribution. Smoothing improves the estimates by leveraging known patterns of word usage in language and other language models based on larger samples. In information retrieval smoothing is very important [15], because the language models tend to be constructed from very small amounts of text. How we estimate low probability words can have large effects on the document scores. In both approaches to ranking documents, the document score is a sum of logarithms of the probability of a word given the document's model. In addition to the problem of zero probabilities mentioned for maximum-likelihood estimates, much care is required if this probability is close to zero. Small changes in the probability will have large effects on the logarithm of the probability, in turn having large effects on the document scores.

The smoothing technique most commonly used is linear interpolation. Linear interpolation is a simple approach to combining estimates from different language models:

$$P(w|\theta) = \sum_{i=1}^{k} \lambda_i P(w|\theta_i)$$

where k is the number of language models we are combining, and λ_i is the weight on the model θ_i. To ensure that this is a valid probability distribution, we must place these constraints on the lambdas:

$$\sum_{i=1}^{k} \lambda_i = 1 \quad \text{and for } 1 \le i \le k,\ \lambda_i \ge 0$$

One use of linear interpolation is to smooth a document's language model with a collection language model. This new model would then be used as the smoothed document language model in either the generative or KL-divergence ranking approach. A specific form of linearly interpolating a document and a collection language model is called Bayesian smoothing using Dirichlet priors [15]. The document is modeled using maximum likelihood estimate. θ_1 is the document language model, θ_2 is the collection language model, and the linear interpolation parameters are:

$$\lambda_1 = \frac{|D|}{|D| + \mu} \qquad \lambda_2 = \frac{\mu}{|D| + \mu}$$

where the parameter μ is set according to the collection and is typically close to the average document length. This smoothing technique has been found effective for ad-hoc document retrieval on several collections [12] [14][15].

3. MODELING STRUCTURED DOCUMENTS

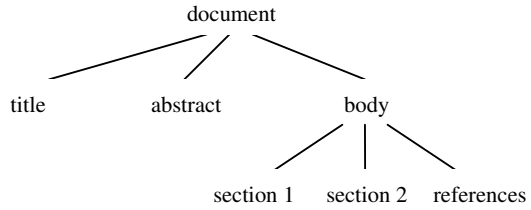
The previous section described how language modeling is used in unstructured document retrieval. With structured documents such as XML or HTML, we believe that the information contained in the structure of the document can be used to improve

document retrieval. In order to leverage this information, we need to model document structure in the language models.

The method we propose borrows from natural language processing. Probabilistic context free grammars (PCFGs) [1] are used to estimate the probability of parse trees of sentences. A PCFG is a context free grammar that has a probability associated with each rule. The probability of a specific parse tree is the product of the probabilities of all rules applied in creating the tree. The analogy we draw from PCFGs to structured documents is that the structure contained in the document can be represented as a context free grammar. The parse tree for the document is given by the structure. For example, if an XML schema specifies that a document is a title, abstract, and body text, then a corresponding rule in the grammar would be:

document → title abstract body

Similarly, a partial tree for a document might look like:



Certain nodes, such as title and abstract, would be designated leaf nodes. In a traditional context-free grammar, a leaf node would be a word. In this model of documents, a leaf node would be a unit of text that does not have additional structure embedded in it. A language model for the leaf node would be estimated from the text.

An important distinction of the document tree language model from PCFGs used for parsing sentences is that we know the tree of the document. This is given directly by the document structure. Since we know the structure, it does not make sense to estimate the probability of a rule. Instead, we feel that we should view the rule as stating that the language model for the parent node consists of the language models of the children nodes.

However, in cases where the document structure is not known, the PCFG analogy is useful. Given a component recognizer and some training data, one could estimate a tree for documents. For example, one could use the existing INEX documents and corresponding flat text versions of the documents as a training set for an automatic tagger for computer science articles.

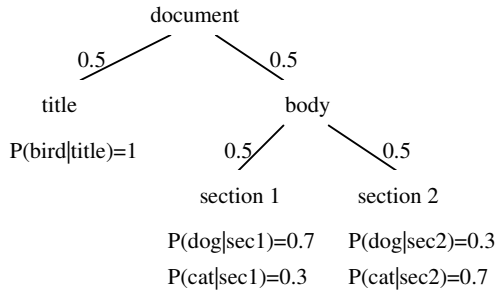
The example rule given above states that a document language model consists of a title, an abstract, and a body language model. We next must specify how to combine the children language models. We suggest that linear interpolation is an appropriate method of combining the children language models. We believe that the optimal parameters for the linear interpolations in the rules depend on the task at hand and on the corpus. Training these parameters is a difficult problem which we will discuss more in Section 5.

This model as described assumes that all leaf nodes contain textual data only. However, it is common to have non-text data present in a document, such as dates, numbers, and pictures. As a language model is a probability distribution over a vocabulary, there really isn't anything stopping us from modeling non-text data in a language model. Appropriate smoothing methods for dates and numbers may be different than for text. For example, we may assume that a number may be normally distributed and taking the mean to be the observed value, using some reasonable estimate of variance. Images may also be modeled in this setting, though the approach may be more complex. Westerveld [13] proposes a method modeling images using a Gaussian Mixture Model, which he argues provides a framework for combining image retrieval with text-based language modeling. Combining the language models of mixed field types as prescribed by a rule may seem a little odd. Here, it may make sense to think of the interpolation weights as measures of relative importance. Additionally, we do not have to explicitly flatten the tree to a single language model; we can preserve the structure in our system and traverse the tree at query time.

The resulting tree for a given document would have a language model associated with every node and weight on the tree branches given by linear interpolation parameters specified in the rules. This provides a rich description of the document, which may be used for comparison to queries. The following section will discuss methods for querying.

4. RANKING THE TREE MODELS

In a retrieval environment for structured documents, it is desirable to provide support for both structured queries and unstructured, free-text queries. It is easier to adapt the generative language model to structured documents, so we only consider that model in this paper. We will sometimes refer to the following toy document model:



In this diagram, we specified the linear interpolation parameters on the edges. To keep things simple, we use equal parameters for the interpolation. We also specified the language models for the leaf nodes. It is simpler to support unstructured queries, so we will describe retrieval for them first.

4.1 Unstructured Queries

To rank document components for unstructured queries, we can use either traditional language modeling approach for IR described in Section 2. For full document retrieval, we need only compute the probability that the document language model generated the query. If we wish to return arbitrary document components, we need to compute the probability that each component generated the query.

We would probably wish to remove document components in the ranking where a parent or child component is present higher in the ranking. This would prevent returning the same component multiple times. Other strategies for filtering the ranking have been proposed. An empirical study comparing techniques for filtering rankings is needed.

4.2 Structured Queries

Processing structure queries requires some adaptation of the language model retrieval approaches, as they do not currently allow for structural constraints. We will work with the generative language model here, as it is easier to adapt to structured queries. Following [7], Boolean style operators can be incorporated as follows:

a AND b: Multiply $P(a|\theta)$ and $P(b|\theta)$. This is the default operator in the generative language model.

a OR b: Add $P(a|\theta)$ and $P(b|\theta)$. This is interpreted as the probability that the language model θ generated either a or b (or both). This assumes independence of a and b. Allowing this only on individual query terms would fit within the unigram assumptions in the model. Alternatives here would be $P(a|\theta) + P(b|\theta) - P(a \text{ and } b|\theta)$, and $(1 - (1 - P(a|\theta))(1 - P(b|\theta)))$.

NOT a: Take $1 - P(a|\theta)$. This is the probability that the model θ did not generate a.

Note that these Boolean operators enforce exact matches only when the MLE is used and no smoothing is applied to the leaf nodes. When smoothing the leaf nodes, the Boolean operators are soft matches.

There are many structural constraints that could be supported within this model, but we will only discuss how we would support a few constraints. A more thorough and complete description would be needed to implement a real system. Some constraints could be modeled as described below.

A simple constraint on which document components could be returned would be interpreted literally. For instance, if a query specifies the user wishes titles only to be returned, the system would only rank document titles.

The next constraint is of the form "return components of type *x* where it has component *y* that contains the query term *w*." We first consider the constraint where *y* is a direct descendent of *x*. An example is "return *documents* where the *title* is contains the word *bird*." This constraint can be viewed as measuring the probability that the document language model would generate the word *bird* from its title model. We observe that the linear interpolation weights can be viewed as probabilities. These correspond to the probability that the model was selected to produce a query term during generation. Formally, this constraint is given by $P(w|y) \cdot P(y)$, where $P(y)$ is the linear interpolation weight for the document component *y*. For our example document and query, this would be

$$P(\text{bird}|\text{title}) \cdot P(\text{title}) = 1 \cdot 0.5 = 0.5.$$

Constraints that are nested more than one level deep can be modeled in a similar manner. However, instead of including only the linear interpolation weight for the constraint component, we include each weight in the path of the query constraint. Consider ranking the query "return *documents* where the *body's first section* contains the word *dog*" on our example document. This query would be ranked according to

$$P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \cdot P(\text{body}) = 0.7 \cdot 0.5 \cdot 0.5 = 0.175.$$

We now have the mechanism to remove the constraint on which component to return in the previous examples. For the example query "return *components* where *section 1* contains the word *dog*." A system would rank each component in the document that had section 1 component somewhere

in its tree. A decision would need to be made whether a section 1 component could be returned for the query. In our example document, both the document and body components would be ranked (and possibly the section 1 component). For the document component, the score would be

$$P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \cdot P(\text{body}),$$

and the body component would have a score of

$$P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}).$$

The body component's score will be greater than or equal to the document component's score. It may seem odd to have a query of this form, but when combined with other query components, then the document may be preferred. For instance, the document component would be preferred over the body component for the query such "*bird* and *section 1* contains *dog*."

A constraint that specifies a set of document components would treated as an OR operation. An example of this is "return *body components* where *any section* contains *dog*." For the example document, this would be evaluated as

$$\begin{aligned} &P(\text{dog}|\text{section 1}) \cdot P(\text{section 1}) \\ &+ P(\text{dog}|\text{section 2}) \cdot P(\text{section 2}) \\ &= 0.7 \cdot 0.5 + 0.3 \cdot 0.5 \\ &= 0.5. \end{aligned}$$

The multiplication of weights along the path to a node may seem like it places much more weight on nodes higher in the tree. This is only true under limited constraints. In general, as the model is multiplicative, the weights will factor out and be the same across documents. However, if there is an OR operation of two constraints, then this problem will happen. We do not expect this to be an issue for most queries.

This provides a sample of query operations that can be accommodated in the tree-based language model of documents. Any of the above operations can be combined into more complex queries, giving us the ability to represent and rank rather intricate queries.

4.3 Discussion

One nice benefit of the language modeling approach is that it implicitly deals with some of our intuitions about coverage. This is a result of how the language models estimate probabilities. To illustrate this, consider ranking the query Q = "dog cat" on our toy document. We will use the generative language model approach for this example. The probabilities for the leaf nodes are:

$$P(Q|\text{title}) = 0$$

$$\begin{aligned} &P(Q|\text{section 1}) = P(\text{dog}|\text{section 1}) \cdot P(\text{cat}|\text{section 1}) \\ &= 0.7 \cdot 0.3 \\ &= 0.21 \end{aligned}$$

$$\begin{aligned} &P(Q|\text{section 2}) = P(\text{dog}|\text{section 2}) \cdot P(\text{cat}|\text{section 2}) \\ &= 0.3 \cdot 0.7 \\ &= 0.21 \end{aligned}$$

The language model for the body node is a linear interpolation of the section 1 and section 2 nodes. Similarly, the language model for the document node is a linear interpolation of the body and title nodes. These probabilities associated with these language models are:

$$P(\text{dog}|\text{body}) = 0.5$$
$$P(\text{cat}|\text{body}) = 0.5$$

$$P(\text{dog}|\text{document}) = 0.25$$
$$P(\text{cat}|\text{document}) = 0.25$$
$$P(\text{bird}|\text{document}) = 0.5$$

Using these language models, we can now compute the probabilities that the body and the document generated the query:

$$\begin{aligned} &P(Q|\text{body}) = P(\text{dog}|\text{body}) \cdot P(\text{cat}|\text{body}) \\ &= 0.5 \cdot 0.5 \\ &= 0.25 \end{aligned}$$

$$\begin{aligned} &P(Q|\text{document}) \\ &= P(\text{dog}|\text{document}) \cdot P(\text{cat}|\text{document}) \\ &= 0.25 \cdot 0.25 \\ &= 0.125 \end{aligned}$$

We see that the highest ranking document component for the query is the body component. This follows our intuition that the body component is probably better than either of the section components alone. Another favorable benefit is that the body component is ranked above the document component, which includes extra unrelated information.

Unfortunately, the model does not always behave as desired. Reconsider the query "dog cat." If there is a document node containing only "dog cat", then this leaf node will preferred over other nodes. This is undesirable, as there no context, resulting in an incoherent result. A way to deal with this issue is to rank by the probability of the document given the query. Using Bayes rule, this would allow us incorporate priors on the nodes. The prior for only the node being ranked would be used, and the system would multiply the probability that the node generated the query by the prior:

$$P(D|Q) = P(Q|\theta_D) P(D) / P(Q)$$

$$\propto P(Q|\theta_D) P(D)$$

This would result in ranking by the probability of the document given the query, rather than the other way around. An example prior may be some function of the number of words subsumed by that node in the tree.

5. TRAINING THE MODEL

Training the linear interpolation parameters in the grammar is a difficult problem. For a task where there are often many relevant documents for a query, such as ad-hoc retrieval, an Expectation-Maximization approach may work well. Given a training set of queries and relevance judgments, an EM approach to training the parameters would be:

1) Initialize the linear interpolation parameters for each rule to random values. These values must satisfy the constraints for correct linear interpolation.

2) For each rule, update the parameters using:

$$\lambda_j^{[t+1]} = \frac{1}{z} \sum_{(\mathrm{Q},\mathrm{D}) \in R} \sum_{w \in (q_1, q_2, \ldots, q_n)} \frac{\lambda_j^{[t]} \mathrm{P}\left(w \middle| \theta_{j,\mathrm{D}}\right)}{\sum_{i=1}^{k} \lambda_i^{[t]} \mathrm{P}\left(w \middle| \theta_{i,\mathrm{D}}\right)}$$

   where z is the normalizing constant that makes the new lambdas sum to one, the superscript t is used to denote values at the t^{th} iteration, and $(\mathrm{Q},\mathrm{D}) \in R$ represents the pairs of queries and documents marked relevant in the training set. For learning linear interpolation parameters, the expectation and the maximization steps can be combined.

3) Repeat step 2 until some convergence criterion is met or for a fixed number of iterations.

This strategy will not work for all tasks. For some tasks, such as named-page or known-item finding, there is only one relevant document per query. Using EM to maximize the relevant documents for the queries runs the risk of also maximizing the probability of other non-relevant documents. While it is true that this is also a risk for ad-hoc retrieval, the effects of this on the evaluation measures are more pronounced for named-page and known-item finding. This is in part due to the choice of evaluation measures commonly used for named-page finding (such as mean-reciprocal rank). Mean-reciprocal rank is very sensitive to changes in rank near the top of the ranking. For these other tasks, it is desirable to have a learning technique that allows the system to directly optimize the evaluation function. Algorithms that may be easily adapted to this without the calculation of difficult gradients include genetic algorithms [16] and simulated annealing.

The parameter training is not an intractable task, nor may it be as difficult as we have suggested. Simple techniques like hand-tuning the parameters may work well, and it is unclear just how sensitive the model is to different parameters. We have had some success with hand-chosen linear interpolation coefficients for a simpler model [3].

6. RELATED WORK

Fuhr and Großjohann proposed XIRQL [4], which is an extension of XQL. They model queries as events which are represented in a Boolean algebra. The queries are converted into Boolean expressions in disjunctive normal form. The queries are evaluated on documents using the inclusion-exclusion formula. The event probabilities are estimated using weights derived from the text. These event probabilities are different from those in the language models, as they do not have to sum to one across all terms. Augmentation weights are used to allow inclusion of the weights from children nodes. These weights are in the range [0:1], which down-weight the children nodes' influence as the weights are propagated upward. Augmentation is a generalization of linear interpolation, where the constraint that the weights sum to one is relaxed. Their model does not assume independence among events, while the model presented here does assume independence of query terms.

Kazai et al [8][9] represent documents as graphs. The document structure is represented using a tree, but horizontal links are allowed among neighbor nodes in the tree. They model nodes in the tree using vectors of term weights. They call combining information in the tree aggregation, and use ordered weighted averaging (OWA) to combine node vectors. OWA is essentially the same as linear interpolation. While our model does not explicitly model links among neighbor nodes, this effect could be achieved by smoothing a node's language model with those of its neighbors.

Grabs and Schek [5] compute term vectors dynamically and use idf values based on the node type. Similarly, we smooth the nodes using information from the nodes of the same type. Their method of creating the term vectors dynamically may prove useful when implementing our approach. Structural constraints in query terms are supported using augmentation weights similar to those used by Fuhr [4].

In [2], the authors present the ELIXER query language for XML document retrieval. They adapt XML-QL and WHIRL to allow for similarity matches on document components in the queries. The similarity scores are computed using the cosine similarity on tf·idf weighted vectors representing the query and the document component. Scores for multiple query components are combined by taking the product of the scores.

Myaeng et al [11] represent documents using Bayesian inference networks. The document components act as different document representations, and are combined in the network to produce a structure sensitive score for documents.

Only document scores are computed; document components are not ranked.

Hatano et al [6] match compute tf·idf vectors for each node in the tree. They compute similarities of text components using cosine similarity, and they use the *p*-norm function to combine the similarities of the children nodes. The document frequencies are not element specific, while our language model smoothing is element specific.

7. CLOSING REMARKS

We proposed a tree-based language model for the modeling of structured documents. We described methods of querying structured documents using the model we described, and gave examples of how this is accomplished.

One benefit of the model include guidance from language modeling on how to estimate the probabilities used in ranking. Another benefit is that the model captures some of our intuitions about selecting which components are most appropriate to return. The model also allows for including priors on components that can be used to model additional beliefs about coverage.

A disadvantage of the approach is that the linear interpolation parameters should be trained for best performance. These parameters may be corpus or task specific. However, we also present methods for training the parameters, such as EM or genetic algorithms.

The next steps for this work are to implement and test the model. Additionally, we will need to address concerns of efficiency and storage.

8. ACKNOWLEDGMENTS

We thank Yi Zhang and Victor Lavrenko for their insight and thoughts on structured documents and language modeling. This work was sponsored by the Advanced Research and Development Activity in Information Technology (ARDA) under its Statistical Language Modeling for Information Retrieval Research Program. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] Allen, J. *Natural Language Understanding* (1995), 2nd edition, Benjamin/Cummings Publishing.

[2] Chinenyanga, T.T. and N. Kushmerik. Expressive retrieval from XML documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 163-171.

[3] Collins-Thompson, K., P. Ogilvie, Y. Zhang, and J. Callan. Information filtering, novelty detection, and named-page finding. In *Proceedings of the Eleventh Text Retrieval Conference, TREC 2002,* notebook version, 338-349.

[4] Fuhr, N. and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 172-180.

[5] Grabs, T. and H.J. Schek. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Workshop on XML Information Retrieval* (2002), ACM.

[6] Hatanao, K., H. Kinutani, M. Yoshikawa, and S. Uemura. Information retrieval system for XML documents. In *Proceedings of Database and Expert Systems Applications* (DEXA 2002), Springer, 758-767.

[7] Hiemstra, D. *Using language models for information retrieval*, Ph.D. Thesis (2001), University of Twente.

[8] Kazai, G., M. Lalmas, and T. Rölleke. A model for the representation and focused retrieval of structured documents based on fuzzy aggregation. In *The 8th Symposium on String Processing and Information Retrieval* (SPIRE 2001), IEEE, 123-135.

[9] Kazai, G., M. Lalmas, and T. Rölleke. Focussed Structured Document Retrieval. In *Proceedings of the 9th Symposium on String Processing and Information Retrieval* (SPIRE 2002), Springer, 241-247.

[10] Lafferty, J., and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 111-119.

[11] Myaeng, S.H., D.H. Jang, M.S. Kim, and Z.C. Zhoo. A flexible model for retrieval of SGML documents. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 138-145.

[12] Ogilvie, P. and J. Callan. Experiments using the Lemur Toolkit. In *Proceedings of the Tenth Text Retrieval Conference, TREC 2001*, NIST Special publication 500-250 (2002), 103-108.

[13] Ponte, J., and W.B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM Press, 275-281.

[14] Westerveld, T., W. Kraaij, and D. Hiemstra. Retrieving web pages using content, links, URLs, and anchors. In *Proceedings of the Tenth Text Retrieval Conference, TREC 2001*, NIST Special publication 500-250 (2002), 663-672.

[15] Zhai, C. and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM Press, 334-342.

[16] Zhang, M., R. Song, C. Lin, L. Ma, Z. Jiang, Y. Jin, Y. Liu, L. Zhao, and S. Ma. THU at TREC 2002: novelty, web and filtering (draft). In *Proceedings of the Eleventh Text Retrieval Conference, TREC 2002,* notebook version, 29-42.