# Parameter Estimation for a Simple Hierarchical Generative Model for XML Retrieval

Paul Ogilvie and Jamie Callan

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
pto@lti.cs.cmu.edu, callan@lti.cs.cmu.edu

**Abstract.** This paper explores the possibility of using a modified Expectation-Maximization algorithm to estimate parameters for a simple hierarchical generative model for XML retrieval. The generative model for an XML component is estimated by linearly interpolating statistical language models estimated from the text of the component itself, the parent component, the document component, and its children. We modify EM to allow the incorporation of negative examples, then attempt to maximize the likelihood of the relevant components while minimizing the likelihood of non-relevant components found in training data. This provides an effective algorithm to estimate the parameters in the linear combination mentioned above. Some experiments are presented on the CO.Thorough task that support these claims.

## 1 Introduction

In previous work [1][2][3], we proposed using hierarchical language models for ranking XML document components for retrieval. However, we left the problem of estimating parameters as future work. In this work, we present a parameter estimate method for a simplified version of the hierarchical language models.

Similar to the language models we estimated in the past, we construct a language model for each component in the document. What is different from previous work is that we do not recursively smooth the language models. Instead, we linearly interpolate the parent's unsmoothed language model, each child's unsmoothed language model, the document's unsmoothed language model, and the collection language model. This simplification allows us to formulate the parameter estimation problem simply so that we can apply the Generalized Expectation Maximization algorithm.

However, we observed that this approach places the most weight on the document language model, which results in very poor retrieval performance. We modify the likelihood we wish to maximize by including negative examples. These negative examples are non-relevant components that come from documents that contain relevant components. To include these negative examples in the likelihood we raise one minus the probability of the unsmoothed language model generating the query term to a very large power so that it is on a similar

scale to that of the language models for relevant components. While this is an ad-hoc method for including the negative examples, we have found it to work well in practice.

The next section describes the model in detail, and Section 3 presents the Generalized Expectation Maximization (GEM) algorithm for the model. Section 4 presents our adaptation of the GEM algorithm to include negative examples. We present experimental methodology and describe our system in Section 5. Section 6 contains our experiments with using the GEM algorithm on CO.Thorough task, and conclusions and discussion is contained in Section 7.

## 2 Model

We rank components by estimating the probability that the a language model estimated for the component generated the query. We use simple unigram language models, which are multinomial probability distributions over words in the vocabulary. That is, a language model $\mu$ specifies $P(w|\mu)$. Document components (or elements) are then ordered by $P(Q|\mu_e) = \prod_{i=1}^{|Q|} P(q_i|\mu_e)$ where $\mu_e$ is the language model estimated for a particular element $e$.

In order to estimate the language model $\mu_e$, we note that we would like to incorporate evidence from the document, its parent, and its children. With that in mind, we estimate $\mu_e$ as a linear combination of several language models:

$$\begin{aligned} P(w|\mu_e) = & \lambda_P P\left(w\middle|\theta_{P(e)}\right) \\ & +\lambda_D P\left(w\middle|\theta_{d(e)}\right) \\ & +\lambda_C P(w|\theta_C) \\ & +\lambda_O \frac{|s(e)|}{|s(e)|+\sum_{j'\in c(i)} \alpha_{t(j')}|j'|} P\left(w\middle|\lambda_{s(e)}\right) \\ & +\lambda_O \sum_{j\in c(e)} \frac{\alpha_{t(j')}|j|}{|s(e)|+\sum_{j'\in c(i)} \alpha_{t(j')}|j'|} P(w|\lambda_j) \end{aligned} \tag{1}$$

where $\theta_x$ refers to a language model estimated for $x$, $P(x)$ refers to the parent of $x$, $d(x)$ refers to the document containing $x$, $s(x)$ refers to the component $x$ (self), $c(x)$ returns a list containing the children of $x$, $t(x)$ refers to the type of the element $x$, and $C$ refers to the entire collection. We choose to set the $\lambda$ parameters in the interpolation to be constant across all elements in the collection to reduce the number of parameters we must estimate. The $\alpha$ parameters allow us to provide additional weight to the children of components, where the weight is dependent on the type of the child component. Note that we also multiply alpha by the length of the component, which results in an assumption that the extra value of a child component is dependent on both the type and length of the child.

In this work, we will take $\theta_x$ to be the Maximum Likelihood Estimate from the text contained in $x$, which is given by:

$$P(w|\theta_x) = \frac{\text{count of } w \text{ in text of } x}{\text{length in words of text of } x} \tag{2}$$

Note that this is different than our previous work. In our previous work, we excluded the text of the child's components when performing hierarchical smoothing. In this model we include that text. This allows a more clear and consistent parameter estimation scheme. The $\alpha_t$ parameters represent the *additional* value of a word in components of type $t$. Additionally, we do not recursively smooth the components. This is a limiting factor in current work that simplifies the parameter estimation process.

Unfortunately, due to a bug in our system we did not rank components by $P(Q|\mu_e)$. In our official submissions, we ranked by

$$\begin{aligned} & P\left(Q\left|\theta'_{P(e)}\right.\right)^{\lambda_P} \times P\left(Q\left|\theta'_{d(e)}\right.\right)^{\lambda_D} \\ \times & P\left(Q\left|\theta'_{s(e)}\right.\right)^{\lambda_O \frac{|s(i)|}{|s(i)|+\sum_{j' \in c(i)} \alpha_{t(j')}|j'|}} \\ \times & \prod_{j \in c(e)} P\left(Q\left|\theta'_j\right.\right)^{\lambda_O \sum_{j \in c(e)} \frac{\alpha_{t(j')}|j|}{|s(e)|+\sum_{j' \in c(i)} \alpha_{t(j')}|j'|}} \end{aligned} \tag{3}$$

where

$$P(w|\theta'_x) = (1-\lambda_C)P(w|\theta_x) + \lambda_C P(w|\theta_C) \tag{4}$$

This model does allow relative weighting of the different structural components of messages in the thread. However, it does not have the intended effect of combining evidence at the word level; it only combines query level evidence. This model corresponds to the linear weighted combination of log probabilities, which we investigated in [4]. We will refer to ranking by $P(Q|\theta_e)$ as the mixture method and Equation 3 as the post query combination approach.

Rather than discuss our official submissions in Section 6, we will present experiments using the corrected $P(Q|\theta_e)$. We also apply a linear length prior [5] to our rankings. That is, we multiply $P(Q|\theta_e)$ by $length(e)$ to obtain the retrieval status values used in our rankings.

## 3 Parameter Estimation Using EM

This section describes how we estimate parameters for ranking results by $P(Q|\theta_e)$.

Suppose there are $M$ language models in the collection, which we will denote

$$\theta_1, \theta_2, \ldots, \theta_M.$$

Suppose that we are given some queries and rankable components that are relevant to these queries. We will treat words in these queries as observations from the relevant components:

$$\mathbf{x} = (x_1, x_2, \ldots, x_N),$$

where we denote the relevant components as

$$\mu_1, \mu_2, \ldots, \mu_N.$$

Note that there may be repeated query terms and components in these lists; this is not an issue in the estimation process.

Let us now assume that the $\mu$ components are linear interpolations of the components, giving:

$$P(x|\mu_i) = \sum_{j=1}^{M} \lambda_{ij} P(x|\theta_j). \tag{5}$$

This results in a model where we do not know the $\Lambda = (\lambda_{11}, \ldots, \lambda_{NM})$ parameters.

We would like to maximize the probability of $P(\mathbf{x}|\mu)$. In order to reduce the number of parameters we must estimate in this model, we will assume that each $\mu_i$ is estimated from using a small number of components we will call the $family$ of $i$. In relation to the model presented before, the $family$ of $i$ will be child components, the collection component, the parent component, the document component and the component itself:

$$family(i) = \left(\theta_1, \theta_{document(i)}, \theta_{parent(i)}, \theta_{self(i)}\right) \cup_{k \in children(i)} (\theta_k)$$

or using the first letter as an abbreviation for the *document*, *parent*, $self$ and *children* functions:

$$family(i) = \left(\theta_1, \theta_{d(i)}, \theta_{p(i)}, \theta_{s(i)}\right) \cup_{k \in c(i)} (\theta_k). \tag{6}$$

where $\theta_1$ is the special collection model used for smoothing. Given the $family$ of component $i$, we can rewrite Equation 5 as

$$P(x|\mu_i) = \sum_{j \in f(i)} \lambda_{ij} P(x|\theta_j), \tag{7}$$

greatly reducing the number of parameters we must estimate. Note that we also place the constraints

$$\lambda_{ij} \geq 0, \quad \sum_{j \in f(i)} \lambda_{ij} = 1 \tag{8}$$

upon the $\Lambda$ parameters.

However, there are still many cases where we must estimate $\lambda$ parameters for texts and we have no training data, as the $\mathbf{x}$ vector is very small in comparison to the total number of rankable texts in the corpus. We must make further assumptions to reduce the parameter space. Given our understanding of the XML retrieval domain, we will assume constant parameters across all models for the combination with the *collection*, *document* and *parent* components. For the *children* components, we will assume that the weight placed should be a simple function of the $t = type$ of the child component and its length. Under

these assumptions:

$$\lambda_{ij} = \begin{cases} \lambda_C & \text{if } j = 1, \\ \lambda_D & \text{if } j = d(i), \\ \lambda_P & \text{if } j = p(i), \\ \lambda_O \frac{|j|}{|s(i)| + \sum_{j' c(i)} e^{\beta_{t(j')}} |j'|} & \text{if } j = s(i), \\ \lambda_O \frac{e^{\beta_{t(j)}} |j|}{|s(i)| + \sum_{j' c(i)} e^{\beta_{t(j')}} |j'|} & \text{if } j \in c(i), \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

where the *type* function returns a value in $(1, 2, \ldots, T)$. This now greatly reduces the number of parameters we must estimate to $T + 4$. In addition to the constraints in Equation 8, we place this additional constraint:

$$\lambda_C + \lambda_D + \lambda_P + \lambda_O = 1 \tag{10}$$

Note that we reparameterized $\alpha_k$ as $e^{\beta_k}$ as this will ensure that $\alpha_k$ is positive. Given Equation 9, we can rewrite Equation 7 using the parameters we must estimate:

$$\begin{aligned} P(x|\mu_i) = \lambda_C P(x|\theta_1) + \lambda_D P\left(x\middle|\theta_{d(i)}\right) + \lambda_P P\left(x\middle|\theta_{p(i)}\right) \\ + \lambda_O \left( \begin{array}{l} \frac{|i|}{|i| + \sum_{j \in c(i)} e^{\beta_{t(j)}} |j|} P\left(x\middle|\theta_{s(i)}\right) + \\ \sum_{j \in c(i)} \frac{e^{\beta_{t(j)}} |j|}{|i| + \sum_{j \in c(i)} e^{\beta_{t(j)}} |j|} P(x|\theta_j) \end{array} \right) \end{aligned} \tag{11}$$

We would like to maximize the likelihood of the observed data, which is

$$\mathcal{L}(\Lambda|\mathcal{X}) = P(\mathbf{x}|\mu) = \prod_{i=1}^{N} P(x_i|\mu_i) = \prod_{i=1}^{N} \sum_{j=1}^{M} \lambda_{ij} P(x_i|\theta_j) \tag{12}$$

Unfortunately, the summation within the product makes it difficult to differentiate, so we must use an alternative approach to maximizing the likelihood. We choose to use the Expectation-Maximization method to optimizing the likelihood.

Suppose we were given additional information $\mathcal{Y} = (y_1, \ldots, y_N)$ which specify that the $\theta_{y_i}$ distribution generated the $x_i$ query term. Given knowledge of $\mathbf{y}$, the likelihood becomes

$$\mathcal{L}(\Lambda|\mathcal{X}, \mathcal{Y}) = \prod_{i=1}^{N} \lambda_{iy_i} P(x_i|\theta_{y_i}) \tag{13}$$

and the log-likelihood of the data is then

$$\log \mathcal{L}\left(\Lambda \left| \mathcal{X}, \mathcal{Y} \right.\right) = \sum_{i=1}^{N} \log\left(\lambda_{iy_i} P\left(x_i \left| \theta_{y_i} \right.\right)\right) \tag{14}$$

The problem is now that we do not know the values of $\mathcal{Y}$. However, we may treat it as a random vector and apply Expectation-Maximization.

Suppose we have a guess at the $\Lambda$ parameters we shall call $\Lambda^g$. Using $\Lambda^g$ we can compute $P\left(x_i \left| \mu_j^g \right.\right)$. Applying Bayes rule, we calculate

$$P\left(y_i \left| x_i, \Lambda^g \right.\right) = \frac{\lambda_{iy_i}^g P\left(x_i \left| \theta_{y_i} \right.\right)}{P\left(x_i \left| \Lambda^g \right.\right)} = \frac{\lambda_{iy_i}^g P\left(x_i \left| \theta_{y_i} \right.\right)}{\sum_{j=1}^{M} \lambda_{ij}^g P\left(x_i \left| \theta_j \right.\right)} = \frac{\lambda_{iy_i}^g P\left(x_i \left| \theta_{y_i} \right.\right)}{\sum_{j \in family(i)} \lambda_{ij}^g P\left(x_i \left| \theta_j \right.\right)} \tag{15}$$

and

$$P\left(\mathbf{y} \left| \mathcal{X}, \Lambda^g \right.\right) = \prod_{i=1}^{N} P\left(y_i \left| x_i, \Lambda^g \right.\right) \tag{16}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is independently drawn value of the random vector.

We may now estimate the expectation of $\Lambda$ given $\Lambda^g$:

$$\begin{aligned} Q\left(\Lambda, \Lambda^g\right) &= \textstyle\sum_{\mathbf{y} \in \Upsilon} \log\left(\mathcal{L}\left(\Lambda \left| \mathcal{X}, \mathbf{y} \right.\right)\right) P\left(\mathbf{y} \left| \mathcal{X}, \Lambda^g \right.\right) \\ &= \textstyle\sum_{l=1}^{M} \sum_{i=1}^{N} \log\left(\lambda_{il} P\left(x_i \left| \theta_l \right.\right)\right) P\left(l \left| x_i, \Lambda^g \right.\right) \end{aligned} \tag{17}$$

At this point we observe that to maximize this equation, we must take the partial derivative of $Q\left(\Lambda, \Lambda^g\right)$ with respect to each of the $\Lambda$ parameters.

To maximize $\lambda_C$, we must introduce the Lagrange multiplier $\phi$ with the constraint that $\lambda_C + \lambda_D + \lambda_P + \lambda_O = 1$ and solve the following equation:

$$\frac{\partial}{\partial \lambda_C} \left[ \sum_{l=1}^{M} \sum_{i=1}^{N} \log\left(\lambda_{il} P\left(x_i \left| \theta_l \right.\right)\right) P\left(l \left| x_i, \Lambda^g \right.\right) + \phi\left(\lambda_C + \lambda_D + \lambda_P + \lambda_O - 1\right) \right] = 0$$

$$\frac{\partial}{\partial \lambda_C} \begin{bmatrix} \sum_{i=1}^{N} \log\left(\lambda_C\right) P\left(y = 1 \left| x_i, \Lambda^g \right.\right) + \phi \lambda_C \\ \\ +\text{some constants with respect to } \lambda_C \end{bmatrix} = 0$$

$$\frac{1}{\lambda_C} \sum_{i=1}^{N} P\left(y = 1 \left| x_i, \Lambda^g \right.\right) + \phi = 0 \tag{18}$$

Similarly, to maximize $\lambda_D$, $\lambda_P$, and $\lambda_O$, we use

$$\begin{gathered} \frac{1}{\lambda_D} \sum_{i=1}^{N} P\left(y = d\left(i\right) \left| x_i, \Lambda^g \right.\right) + \phi = 0 \\ \\ \frac{1}{\lambda_P} \sum_{i=1}^{N} P\left(y = p\left(i\right) \left| x_i, \Lambda^g \right.\right) + \phi = 0 \\ \\ \frac{1}{\lambda_O} \sum_{i=1}^{N} \sum_{j \in (s(i)) \cup c(i)} P\left(y = j \left| x_i, \Lambda^g \right.\right) + \phi = 0 \end{gathered} \tag{19}$$

By summing these equations we get $\phi = -N$. We can then obtain the following update rules:

$$\begin{aligned}
\lambda_C^{[t]} &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} P\left(y = 1 \left| x_i, \Lambda^{[t-1]}\right.\right) \\
\lambda_D^{[t]} &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} P\left(y = d\left(i\right) \left| x_i, \Lambda^{[t-1]}\right.\right) \\
\lambda_P^{[t]} &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} P\left(y = p\left(i\right) \left| x_i, \Lambda^{[t-1]}\right.\right) \\
\lambda_O^{[t]} &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \sum_{j\in(s(i))\cup c(i)} P\left(y = j \left| x_i, \Lambda^{[t-1]}\right.\right)
\end{aligned} \tag{20}$$

Let us continue to the $\beta_k$ parameters. Let

$$\begin{aligned}
a_{ik} &= \textstyle\sum_{j'\in c(i), t(j')=k} \left|j'\right| \\
b_{ik} &= \left|s\left(i\right)\right| + \textstyle\sum_{j'\in c(i), t(j')\neq k} \beta_{t(j')} \left|j'\right| \\
f_{ik} &= P\left(y = s\left(i\right) \left| x_i, \Lambda^g\right.\right) + \textstyle\sum_{j\in c(i), t(j)\neq k} P\left(y = j \left| x_i, \Lambda^g\right.\right) \\
h_{ik} &= \textstyle\sum_{j\in c(i), t(j)=k} P\left(y = j \left| x_i, \Lambda^g\right.\right)
\end{aligned} \tag{21}$$

Then we can rewrite the above as

$$\frac{\partial}{\partial\beta_k}\left[\begin{array}{c}
\sum_{i:j\in c(i),t(j)=k}\left[\begin{array}{c}
\log\left(\frac{|s(i)|}{b_{ik}+e^{\beta_k}a_{ik}}\right) P\left(y = s\left(i\right) \left| x_i, \Lambda^g\right.\right) \\
+\sum_{j\in c(i),t(j)=k} \log\left(\frac{e^{\beta_k}|j|}{b_{ik}+e^{\beta_k}a_{ik}}\right) P\left(y = j \left| x_i, \Lambda^g\right.\right) \\
+\sum_{j\in c(i),t(j)\neq k} \log\left(\frac{e^{\beta_{t(j)}}|j|}{b_{ik}+e^{\beta_k}a_{ik}}\right) P\left(y = j \left| x_i, \Lambda^g\right.\right)
\end{array}\right] \\
+\text{some constants with respect to } \beta_k
\end{array}\right] = 0 \tag{22}$$

We first take the chain rule, resulting in the multiplier $\beta_k$, then take the partial derivative of the summation with respect to $\beta_k$

$$e^{\beta_k}\textstyle\sum_{i:j\in c(i),t(j)=k}\left[\begin{array}{c}
\frac{-a_{ik}}{b_{ik}+\beta_k a_{ik}} P\left(y = s\left(i\right) \left| x_i, \Lambda^g\right.\right) \\
+\sum_{j\in c(i),t(j)=k} \frac{b_{ik}}{e^{\beta_k}\left(b_{ik}+e^{\beta_k}a_{ik}\right)} P\left(y = j \left| x_i, \Lambda^g\right.\right) \\
+\sum_{j\in c(i),t(j)\neq k} \frac{-a_{ik}}{b_{ik}+e^{\beta_k}a_{ik}} P\left(y = j \left| x_i, \Lambda^g\right.\right)
\end{array}\right] = 0 \tag{23}$$

$$\beta_k \sum_{i:j\in c(i),t(j)=k} \frac{-a_{ik}f_{ik} + \frac{b_{ik}h_{ik}}{e^{\beta_k}}}{b_{ik}+e^{\beta_k}a_{ik}} = 0 \tag{24}$$

Since we cannot solve directly solve this equation for $\beta_k$, we will use a linear approximation around the point $\beta_k^g$:

$$\frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right) \approx \frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g} + \left(\beta_k - \beta_k^g\right) \frac{\partial^2}{\partial \beta_k^2} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g} \tag{25}$$

Since we set $\frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right) = 0$,

$$\beta_k \approx \beta_k^g - \frac{\frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g}}{\frac{\partial^2}{\partial \beta_k^2} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g}} \tag{26}$$

where

$$\frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g} = e^{\beta_k^g} \sum_{i:j \in c(i), t(j)=k} \frac{-a_{ik} f_{ik} + \frac{b_{ik} h_{ik}}{e^{\beta_k^g}}}{b_{ik} + e^{\beta_k^g} a_{ik}} \tag{27}$$

and

$$\frac{\partial^2}{\partial \beta_k^2} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g} = e^{\beta_k^g} \left( \begin{array}{l} \frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^g} + \\ \\ e^{\beta_k^g} \sum_{i:j \in c(i), t(j)=k} \frac{a_{ik}^2 f_{ik} - \frac{b_{ik}^2 h_{ik}}{e^{\beta_k^g}}}{\left(b_{ik} + e^{\beta_k^g} a_{ik}\right)^2} \end{array} \right) \tag{28}$$

Thus, we will have the following update rule for our $\beta_k$ parameter estimates:

$$\beta_k^{[t]} = \beta_k^{[t-1]} - \frac{\frac{\partial}{\partial \beta_k} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^{[t-1]}}}{\frac{\partial^2}{\partial \beta_k^2} Q\left(\Lambda, \Lambda^g\right)_{\beta_k=\beta_k^{[t-1]}}} \tag{29}$$

## 4 Incorporating Negative Examples

While the above presentation of EM to learn parameters attempts to maximize the likelihood of training examples, doing so using only relevant components results in very poor parameter estimation. This is a direct result of the fact that optimizing the likelihood of relevant components may also increase the likelihood of components that are not relevant. In our own experiments, using only relevant components during training will result in most of the weight being placed in $\lambda_D$. We feel this may be a side effect of the bias-variance problem in estimation. The document language model has more bias than the language models estimated from the components, but the variance is lower as the sample sizes are larger for documents than for components. When combining the language models during smoothing, the document language models tend to have a higher likelihood of generating the query terms due to this lower variance.

In order to combat these effects, we also include negative examples in our training data. However, we do not wish to optimize the likelihood of the negative examples. We would prefer to maximize the likelihood that the language models

estimated for the non-relevant components *do not* generate the query terms. To model this one might include for each non-relevant component and query term an example where we use $(1 - P(x|\theta_j))$ in place of $P(x|\theta_j)$. Note that this is not quite the same as what we one might wish to optimize, as:

$$1 - P(Q|\mu_i) \neq \prod_{l=1}^{|Q|} (1 - P(q_l|\mu_i)) \tag{30}$$

However, this is a useful and effective approximation that requires only the above substitution for negative examples. A complication in learning using the inclusion of negative examples given above is that $P(x|\mu_i)$ tends to be very small in relation to $1 - P(x|\mu_i)$. That means that when when maximizing the log likelihood, a small improvement of a positive example may outweigh a large degradation in performance in a negative example.

To accommodate for that effect, we weight the negative probabilities by raising them to a large power. For a negative example, we replace

$$P(x|\theta_j) \tag{31}$$

with

$$(1 - P(x|\theta_j))^{\nu\delta} \tag{32}$$

where $\nu$ is a user chosen parameter that specifies how much emphasis the negative examples have relative to the positive examples and $\delta$ is chosen so that the average probability of a term given the relevant examples is equal to the average probability of a term given the non-relevant examples when $\nu = 1$:

$$\delta = \frac{\log\left(\frac{1}{|positive|}\sum_{positive} P(x_i|\mu_i)\right)}{\log\left(\frac{1}{|negative|}\sum_{negative} P(x_i|\mu_i)\right)} \tag{33}$$

This approach for the incorporation of non-relevant components is ad-hoc but effective, as we will see in the next section.

## 5 Experimental Methodology

We use a locally modified version of the Indri search engine of the Lemur toolkit [6] that supports the hierarchical shrinkage. The hierarchical shrinkage support will be made available in a December release. Release of the parameter estimation code is scheduled for a later release as the estimation methods are still in flux. We indexed the INEX collection using the InQuery stopword list and the Krovetz stemmer. To process queries we removed all quotes from the query (thus ignoring phrasal constraints) and all terms with a minus in front.

We will focus on the CO.Thorough task and present results using the strict and generalized quantizations for nxCG[10], nxCG[25], nxCG[50], and MAP of ep/gr to facilitate comparison to the official results presented at INEX.

## 6 Experiments

In this section we present experiments on the CO.Thorough task. We will disregard our official submissions as they were run with the desired model and they were not run on the entire corpus. We had some problems with using the system that prevented us from indexing the entire corpus which have since been resolved.

We trained our parameters using the INEX 1.8 corpus and CO topics 162-201 using one non-relevant document component as a negative example for each relevant component as a positive example. Components were considered relevant if and only if they were highly exhaustive and highly specific. The non-relevant examples were taken from the same documents as the relevant examples. Ten iterations were used for the EM algorithm. $\alpha_k$ values were updated only for cases where there were at least ten examples for type $k$ in the update rule.

Table 1 shows the a sample of the parameters the EM algorithm learned on the training topics. As $\nu$ increases, the weight on the collection language model ($\lambda_C$) decreases while the weight in the parent ($\lambda_P$) slightly increases and $\lambda_O$, the weight on the component and its children, noticeably increases.

With regards to the $\alpha$ parameters, the type specific length proportional weights on children, a few parameters start with relatively low values and increase rapidly as $\nu$ increases. Table 1 shows a few examples of this behavior. However, most parameters that are learned are very close to zero across all values of $\nu$.

There seems to be some undesirable variation in the parameters, as we can see with the $\alpha$ value for the p type. This may be a side effect of the algorithm being trained on relatively few examples for some types, but this should not be the case for the p tag. However, as it only really matters what the value is relative to the other tags at the same level, perhaps this variance is not an issue.

**Table 1.** Some parameters learned from training data. As $\nu$ increases, $\lambda_C$ decreases and $\lambda_O$ increases. Some $\alpha$ parameters seem fairly stable, such as that of the footnote type. Others increase greatly with larger $\nu$ while some seem somewhat erratic (e.g. p).

| | $\lambda$ | | | | $\alpha$ | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\nu$ | (C)ol | (D)oc | (P)ar | O-self | st | p | sub | footnote | ss1 |
| 1.0 | 0.475 | 0.222 | 0.035 | 0.268 | 0.38 | 0.23 | 0.00 | 0.28 | 0.50 |
| 2.0 | 0.385 | 0.212 | 0.037 | 0.365 | 1.07 | 0.00 | 0.22 | 2.49 | 0.37 |
| 3.0 | 0.342 | 0.210 | 0.040 | 0.408 | 22.75 | 9.77 | 7.77 | 2.22 | 1.75 |
| 4.0 | 0.321 | 0.210 | 0.041 | 0.428 | 189.28 | 0.00 | 9.42 | 1.83 | 6.01 |
| 5.0 | 0.309 | 0.213 | 0.043 | 0.435 | 48623.30 | 0.61 | 146.46 | 1.65 | 13289.10 |

Table 2 shows the effects of using the learned parameters for the CO task on the training topics 162-201. Note that we use the new INEX-2.2 corpus, so these results are not directly comparable to previous results on these topics. As there are many documents that in the INEX-2.2 corpus that were not available

for assessment for the topics, one should regard the evaluation numbers as a suboptimal estimate of performance. Nevertheless, we are mostly interested in the relative performance of the parameters learned for different values of $\nu$, and the values in Table 2 should be adequate for that purpose.

In Table 2 we see that setting $\nu = 1$ yields the most consistently good results for both quantizations. There also seems to be some variation in the columns that does not follow a nice curve. This is an undesirable property which could be a result of variance in the learning algorithm, a sign of instability in the evaluation metrics, or a symptom of too few topics to get a reliable point estimate given the topic variance of the system.

**Table 2.** Results of varying the negative weight $\nu$ on the CO task using training topics 162-201. Values in bold font indicate the largest value for a measure.

| | Strict | | | | Generalized | | | |
|---|---|---|---|---|---|---|---|---|
| | nxCG | | | MAP | nxCG | | | MAP |
| $\nu$ | 10 | 25 | 50 | ep/gr | 10 | 25 | 50 | ep/gr |
| 1.0 | 0.0704 | 0.0880 | **0.1307** | **0.0034** | **0.2946** | **0.2950** | **0.2944** | **0.0852** |
| 1.5 | 0.0593 | **0.0906** | 0.1266 | 0.0032 | 0.2938 | 0.2803 | 0.2710 | 0.0753 |
| 2.0 | 0.0593 | 0.0766 | 0.1237 | 0.0032 | 0.2899 | 0.2878 | 0.2816 | 0.0791 |
| 2.5 | 0.0704 | 0.0832 | 0.1226 | 0.0032 | 0.2922 | 0.2760 | 0.2637 | 0.0716 |
| 3.0 | 0.0704 | 0.0876 | 0.1210 | 0.0031 | 0.2911 | 0.2671 | 0.2536 | 0.0667 |
| 3.5 | 0.0704 | 0.0837 | 0.1218 | 0.0031 | 0.2920 | 0.2649 | 0.2490 | 0.0640 |
| 4.0 | 0.0593 | 0.0820 | 0.1197 | 0.0028 | 0.2903 | 0.2695 | 0.2447 | 0.0612 |
| 4.5 | **0.0741** | 0.0835 | 0.1219 | 0.0026 | 0.2857 | 0.2554 | 0.2383 | 0.0561 |
| 5.0 | 0.0630 | 0.0732 | 0.1087 | 0.0025 | 0.2791 | 0.2464 | 0.2256 | 0.0520 |

Table 3 shows the performance of the learned parameters on this year's CO.Thorough task. Performance for the generalized quantization peaks at $\nu = 2$ and around $\nu = 4$ for the strict quantization. This is quite a bit different from our observations on the training data. We would like to investigate this behavior in more detail. This could simply be the result of a training topic set that is too small or not representative enough. An alternative cause for difference is the change in the assessment methodology this year, which could result in assessors behaving giving different scores.

If we had submitted the system optimized to the training data ($\nu = 1$), then our results would have been in the top 10 official submissions for the strict quantization nxCG@50 metric and the generalized quantization MAP ep/gr metric. Supposing we had worked out our kinks in training (whether they be a result of the algorithm or the assessments) and we had selected the runs with $\nu = 2, 4$ for evaluation, then we would have had a run performing in the top 10 official submissions for the strict quantization nxCG@10,50 and MAP ep/gr metrics and for the generalized quantization nxCG@25,50 and MAP ep/gr metrics.

**Table 3.** Results of varying the negative weight $\nu$ on the CO.Thorough task using test topics 202-241. Values in bold font indicate the largest value for a measure.

| | Strict | | | | Generalized | | | |
|---|---|---|---|---|---|---|---|---|
| | nxCG | | | MAP | nxCG | | | MAP |
| $\nu$ | 10 | 25 | 50 | ep/gr | 10 | 25 | 50 | ep/gr |
| 1.0 | 0.0200 | 0.0639 | 0.1051 | 0.0021 | 0.2225 | 0.2298 | 0.2286 | 0.0854 |
| 1.5 | 0.0440 | 0.0623 | 0.0911 | 0.0022 | 0.2207 | 0.2218 | 0.2197 | 0.0801 |
| 2.0 | 0.0440 | 0.0639 | 0.1006 | 0.0022 | **0.2464** | **0.2421** | **0.2340** | **0.0882** |
| 2.5 | 0.0440 | 0.0655 | 0.1127 | 0.0026 | 0.2200 | 0.2215 | 0.2224 | 0.0813 |
| 3.0 | 0.0440 | 0.0712 | 0.1184 | 0.0027 | 0.2164 | 0.2221 | 0.2167 | 0.0771 |
| 3.5 | 0.0400 | 0.0744 | 0.1192 | 0.0022 | 0.2131 | 0.2189 | 0.2149 | 0.0717 |
| 4.0 | **0.0691** | **0.0747** | **0.1225** | 0.0028 | 0.2445 | 0.2248 | 0.2172 | 0.0751 |
| 4.5 | 0.0651 | 0.0715 | 0.1131 | **0.0029** | 0.2301 | 0.2144 | 0.2126 | 0.0701 |
| 5.0 | 0.0651 | 0.0731 | 0.1116 | **0.0029** | 0.2326 | 0.2183 | 0.2089 | 0.0682 |

## 7 Conclusions

We have derived a Generalized Expectation Maximization algorithm to learn the parameters of a simple hierarchical language modeling system for the ranking and retrieval of XML components. We showed a way to effectively incorporate non-relevant components during training.

We investigated the interaction of the relative weight on the negative training examples $\nu$ and retrieval effectiveness on the CO.Thorough task. Experimental evidence suggests that the optimal $\nu$ parameter may depend on the quantization function used in evaluation. However, we have not done a full investigation of the choice of positive and negative examples during training. In training, we relied only on components that were highly exhaustive and highly specific. This assumption is essentially the assumption of the strict quantization function. We have not done experiments where we use components deemed relevant by the generalized quantization function. While we leave this to future work, we recognize this may change the optimal choice of $\nu$ for optimizing performance for measures using the generalized quantization function.

Our incorporation of negative examples is ad-hoc. As future work, we plan to simulate replication of negative examples rather than directly modifying the probabilities of the language models we are combining. This is a minor change to the algorithm and will not change the maximum likelihood derivation presented in Section 3, but it will be more technically sound than the current incorporation of negative evidence presented in Section 4. We would also like to consider the possibility of performing the negative evidence at the query level, rather than negating probabilities at the level of query terms.

For these experiments, we worked with a simplified hierarchical model. Our previous work [1][2][3] presented a hierarchical model where components were smoothed recursively up and down the tree for a document. We would like to adapt the training algorithm to model recursive smoothing and learn parameters with that optimize the likelihood under that condition.

Up to this point we have discussed only flat text queries. We would like to adapt this approach to work with structured queries to learn approaches to weight components of the query. For example, we may learn that satisfaction of a phrasal constraint should receive higher weight than a constraint on the document structure.

## 8 Acknowledgments

This research was sponsored by National Science Foundation (NSF) grant no. CCR-0122581. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implicit, of the NSF or the US government.

## References

1. Ogilvie, P., Callan, J.: Language models and structured document retrieval. In: Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX). (2003)
2. Ogilvie, P., Callan, J.P.: Using language models for flat text queries in xml retrieval. In: Proc. of the Second Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX), Dagstuhl, Germany (2003)
3. Ogilvie, P., Callan, J.: Hierarchical language models for xml component retrieval. In: Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Springer-Verlag (2005) 224–237
4. Ogilvie, P., Callan, J.P.: Combining document representations for known-item search. In: Proc. of the 26th annual int. ACM SIGIR conf. on Research and development in informaion retrieval (SIGIR-03), New York, ACM Press (2003) 143–150
5. Kamps, J., de Rijke, M., Sigurbjörnsson, B.: Length normalization in xml retrieval. In: Proceedings of the Twenty-Seventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (2004) 80–87
6. http://lemurproject.org/: (The Lemur Toolkit for Language Modeling and Information Retrieval)