

[Main Page](#) | [Namespace List](#) | [Class Hierarchy](#) | [Class List](#) | [File List](#) | [Namespace Members](#) | [Class Members](#) | [File Members](#) | [Related Pages](#)

# Relevance Feedback Evaluation Application

This application runs retrieval experiments with relevance feedback. Different retrieval models can be used with different settings for the corresponding parameters. Although this program is designed for relevance feedback, it can be easily used for pseudo feedback -- you just need to set the parameter `feedbackDocuments` to a result file, i.e., interpreting a result file as if all the entries represent relevant documents.

Two important notes:

- All the feedback algorithms currently in **Lemur** assume that all entries in a judgment file are *relevant* documents, so you must remove all the entries of judged non-relevant documents. However, the judgment status is recorded in the internal representation of judgments, so that it is possible to distinguish judged relevant documents from judged non-relevant documents in a feedback algorithm.
- The format of the judgment file, when used for feedback, must be of three columns, i.e., with the second column removed so that each line has a query id, a document id, and a judgment value. This is to be consistent with the format of a result file. An alternative would be to use the original four-column format directly, but, then we would need to add a parameter to distinguish this four-column format from the three-column format of a result file.

Scoring is either done over a working set of documents (essentially re-ranking), or over the whole collection. This is indicated by the parameter `useWorkingSet`. When `useWorkingSet` has either a non-zero (integer) value or the value `true`, scoring will be on a working set specified in a file given by `workingSetFile`. The file should have three columns. The first is the query id; the second the document id; and the last a numerical value, which is ignored. The reason for having a third column of numerical values is so that any retrieval result of the simple format (i.e., non-trec format) generated by **Lemur** could be directly used as a `workingSetFile` for the purpose of re-ranking, which is convenient. Also, the third column could be used to provide a prior probability value for each document, which could be useful for some algorithms. By default, scoring is on the whole collection.

It currently supports five different models:

1. The popular TFIDF retrieval model
2. The Okapi BM25 retrieval function
3. The KL-divergence language model based retrieval method
4. The InQuery (CORI) retrieval model
5. Cosine similarity model

The parameter to select the model is `retModel`. Valid values are:

- `tfidf` or 0 for TFIDF
- `okapi` or 1 for Okapi
- `kl` or 2 for Simple KL

- `inquery` or 3 for INQUERY
- `cori_cs` or 4 for CORI\_CS
- `cos` or 5 for cosine similarity

It is suspected that there is a bug in the implementation of the feedback for Okapi BM25 retrieval function, because the performance is not as expected.

Other common parameters (for all retrieval methods) are:

1. `index`: The complete name of the index table-of-content file for the database index.
2. `textQuerySet`: the query text stream
3. `resultFile`: the result file
4. `resultFormat`: whether the result format should be of the TREC format (i.e., six-column) or just a simple three-column format `<queryID, docID, score>`. String value, either `trec` for TREC format or `3col` for three column format. The integer values, zero for non-TREC format, and non-zero for TREC format used in previous versions of `lemur` are accepted. Default: TREC format.
5. `resultCount`: the number of documents to return as result for each query
6. `feedbackDocuments` : the file of feedback documents to be used for feedback. In the case of pseudo feedback, this can be a result file generated from an initial retrieval process. In the case of relevance feedback, this is usually a 3-column relevance judgment file. Note that this means you can *NOT* use a TREC-style judgment file directly; you must remove the second column to convert it to three-column.
7. `feedbackDocCount`: the number of docs to use for feedback (negative value means using all judged documents for feedback). The documents in the `feedbackDocuments` are sorted in decreasing order according to the numerical value in the third column, and then the top documents are used for feedback.
8. `feedbackTermCount`: the number of terms to add to a query when doing feedback. Note that in the KL-div. approach, the actual number of terms is also affected by two other parameters.(See below.)

Model-specific parameters are:

- For TFIDF:
  1. `feedbackPosCoeff`: the coefficient for positive terms in (positive) Rocchio feedback. We only implemented the positive part and non-relevant documents are ignored.
  2. `doc.tfMethod`: document term TF weighting method: 0 for RawTF, 1 for log-TF, and 2 for BM25TF
  3. `doc.bm25K1`: BM25 k1 for doc term TF
  4. `doc.bm25B` : BM25 b for doc term TF
  5. `query.tfMethod`: query term TF weighting method: 0 for RawTF, 1 for log-TF, and 2 for BM25TF
  6. `query.bm25K1`: BM25 k1 for query term TF. `bm25B` is set to zero for query terms

- For Okapi:
  1. BM25K1 : BM25 K1
  2. BM25B : BM25 B
  3. BM25K3: BM25 K3
  4. BM25QTF: The TF for expanded terms in feedback (the original paper about the Okapi system is not clear about how this is set, so it's implemented as a parameter.)
- For INQUERY:
  1. TF\_factor
  2. TF\_baseline
  3. collCounts - Use value "USE\_INDEX\_COUNTS" to use counts from the index if no separate collection counts file is available. For collection selection indexes built from collSell application, that file is auto generated.
- For COS:
  1. feedbackPosCoeff: the coefficient for positive terms in (positive) Rocchio feedback as for the TFIDF model.
  2. L2File - File containing precomputed L2 Norms (generated with GenL2Norm).
- For KL-divergence:
  1. smoothSupportFile: The name of the smoothing support file (e.g., one generated by GenerateSmoothSupport).
  2. smoothMethod: One of the four:
    - jelinekmercer or jm or 0 for Jelinek-Mercer
    - dirichletprior or dir or 1 for Dirichlet prior
    - absolutediscount or ad or 2 for Absolute discounting
    - twostage or 2s or 3 for two stage.
  3. smoothStrategy: Either interpolate or 0 for interpolate or backoff or 1 for backoff.
  4. adjustedScoreMethod: Which type of score to output, one of:
    - "querylikelihood" or "ql" for query likelihood.
    - "crossentropy" or "ce" for cross entropy.
    - "negativekld" or "-d" for negative KL divergence.
  5. JelinekMercerLambda: The collection model weight in the JM interpolation method. Default: 0.5
  6. DirichletPrior: The prior parameter in the Dirichlet prior smoothing method. Default: 1000
  7. discountDelta: The delta (discounting constant) in the absolute discounting method. Default 0.7.
  8. queryUpdateMethod: feedback method, one of:
    - mixture or mix or 0 for mixture.
    - divmin or div or 1 for div min
    - markovchain or mc or 2 for markov chain
    - relevancemodel1 or rm1 or 3 for relevance model 1.
    - relevancemodel2 or rm2 or 4 for relevance model 2.
- Method-specific feedback parameters:

For all interpolation-based approaches (i.e., the new query model is an interpolation of the original model with a (feedback) model computed based on the feedback documents), the following four parameters apply:

1. `feedbackCoefficient`: the coefficient of the feedback model for interpolation. The value is in  $[0,1]$ , with 0 meaning using only the original model (thus no updating/feedback) and 1 meaning using only the feedback model (thus ignoring the original model).
2. `feedbackTermCount`: Truncate the feedback model to no more than a given number of words/terms.
3. `feedbackProbThresh`: Truncate the feedback model to include only words with a probability higher than this threshold. Default value: 0.001.
4. `feedbackProbSumThresh`: Truncate the feedback model until the sum of the probability of the included words reaches this threshold. Default value: 1.


Parameters `feedbackTermCount`, `feedbackProbThresh`, and `feedbackProbSumThresh` work conjunctively to control the truncation, i.e., the truncated model must satisfy all the three constraints.

All the three feedback methods also recognize the parameter `feedbackMixtureNoise` (default value :0.5), but with *different* interpretations.

- For the collection mixture model method, `feedbackMixtureNoise` is the collection model selection probability in the mixture model. That is, with this probability, a word is picked according to the collection language model, when a feedback document is "generated".
- For the divergence minimization method, `feedbackMixtureNoise` means the weight of the divergence from the collection language model. (The higher it is, the farther the estimated model is from the collection model.)
- For the Markov chain method, `feedbackMixtureNoise` is the probability of *not* stopping, i.e.,  $1 - \alpha$ , where  $\alpha$  is the stopping probability while walking through the chain.

In addition, the collection mixture model also recognizes the parameter `emIterations`, which is the maximum number of iterations the EM algorithm will run. Default: 50. (The EM algorithm can terminate earlier if the log-likelihood converges quickly, where convergence is measured by some hard-coded criterion. See the source code in [SimpleKLRetMethod.cpp](#) for details. )

---

Generated on Tue Jun 15 11:02:58 2010 for Lemur by  1.3.4