///

Information Retrieval and Machine Learning

Massimo Melucci

University of Padua
Department of Information Engineering
massimo.melucci@unipd.it

CIMI School in Machine Learning 2015

Hands-on Session

- Retrieval function optimization
- Experimenting retrieval function optimization

Information Retrieval and Machine Learning

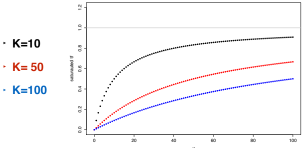
Massimo Melucci

University of Padua
Department of Information Engineering
massimo.melucci@unipd.it

CIMI School in Machine Learning 2015

Retrieval function optimization[1]

- Retrieval functions have parameters.
- Retrieval effectiveness depends on parameters other than on model.
- Problem: To find the optimal parameters (i.e. those maximising a measure of effectiveness or minimising a measure of risk).
    - For example: Best Match N. 25 (BM25) has three parameters (b, k_1, k_3) and the default values are often suboptimal for most collections.
- Suppose a new retrieval function is to be evaluated.
    - For example: another form of saturation term of BM25 $\frac{\mathrm{tf}}{K+\mathrm{tf}}$ is introduced and we wonder whether and the degree to which it improves effectiveness.
    - We may use many different values of K.



[1]Thanks to Emanuele Di Buccio

Why optimization?

- The comparison between the “new” retrieval function and the baseline must be *fair*, that is, both must be evaluated at its best condition.
- The optimal parameters may differ depending on collections, queries, search tasks, users, language, etc., in general, on *context*.

Methodology

Step	Action	Example
1	To define an hypothesis	Saturation term increases effectiveness
2	To select a baseline retrieval function	VSM, TFIDF
3	To select test collections	TIPSTER, some TREC topic set
4	To select effectiveness measure	MAP or NDCG
5	To optimize the retrieval function w.r.t. the measure using the test collection	Find the K that maximises MAP using TIPSTER
6	To test the hypothesis	Wilcoxon's test using the APs computed for the baseline and the retrieval function equipped with the saturation term

Experimental data preparation (step 5)

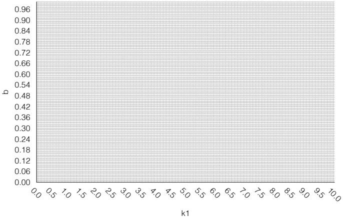
- Per-query overfitting.
  - For each query, find the optimal parameters.
  - Used to optimize for the "few" frequent queries (remember frequent query word distribution).
  - Very computationally expensive yet very highly effective.
- Query set overfitting.
  - For each query *set*, find the optimal parameters.
  - Used to optimize for sets of "few" frequent queries.
  - Computationally expensive yet highly effective.

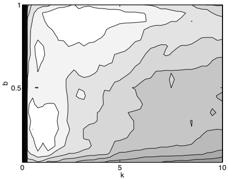
Experimental data preparation (step 5)

- Train/test split.
  - Split the query set in training set and test set.
  - Optimize using the training set and compare using the test set.
  - Less computationally expensive yet still quite effective.
  - For example:
    - TREC 2004 robust track: 250 queries = 150 training queries + 100 test queries.
    - TREC traditional ad-hoc tracks: $n + 50$ queries = n past TREC queries + 50 current year queries.
- Cross validation.
  - When the query set is small.
  - To use the same measure and the same document set.
- Cross collection (a.k.a. transfer learning).
  - To train using one test collection.
  - To test using another test collection.

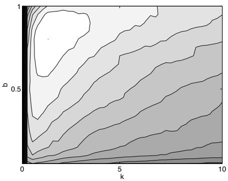
Grid search

- Prepare a d-dimensional grid for d parameters and with $n_1 \times \cdots \times n_d$ cells.
- Compute the effectiveness measure for each cell.
- BM25 example:
  - $d = 2$ (b, k_1)
  - First dimension: $b \in \{0, 0.01, 0.02, \ldots, 1.00\}$
  - Second dimension: $k_1 \in \{0, 0.1, \ldots, 1, 1.1, \ldots, 9.8, 9.9, 10\}$
  - 100 runs for $b\times$ 100 for k_1 $=$ 10000 runs amouns to 8 hours (3 seconds $\times$ per run using CACM).





NDCG (Train)

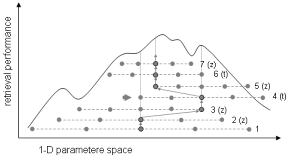


NDCG (Test)

[3]

Robust line search

One parameter ($d = 1$). See [2]



Two parameters ($d = 2$).

1a Search b' (k_1 fixed).

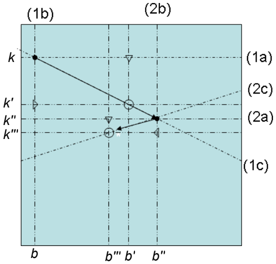
1b Search k_1' (b fixed).

1c Search (b'', k_1'') along the line passing through (b, k_1) and (b', k_1').

2a Search b''' (k_1'' fixed).

2b Search k_1''' (b'' fixed).

2c Search (b'''', k_1'''') along the line between (b'', k_1'') and (b''', k_1''').



Summary

Test collection

- The following files can be downloaded from `http://www.dei.unipd.it/~melo/ml-school/`
- *Communications of the ACM* (CACM) test documents:
  - XML (one file for each document). `cacm.xml.zip`
  - Plain full text (one file for each document). `cacm.txt.zip`
  - List of triples Term Frequency (TF) (doc. id., word). `freq.docid.word.txt`
  - List of triples TF (doc. id., word stem). `freq.docid.stem.txt`
- Stop-list. `stoplist.txt`
- Test queries:
  - Plain full text. `queries.txt`
  - Plain full text in TREC format. `query.trec.txt`
  - Query word pairs (query id., query word). `query-keyword.txt`
  - Query word stem pairs (query id., query word). `query-stem.txt`
- Relevance assessments. `qrels-treceval.txt`

Programs

- Source code of the Text REtrieval Conference (TREC) evaluation program. `trec_eval.8.1.zip`
- One of these libraries:
  - Based on C/C++:
    - Lemur/Indri (`http://sourceforge.net/projects/lemur/?source=directory`).
  - Based on Java:
    - Lucene (`http://lucene.apache.org/`).
    - ElasticSearch (based on Lucene, `https://www.elastic.co/`).
    - Galago (based on Lemur/Indri, `http://sourceforge.net/p/lemur/wiki/Galago/`).
  - Based on Python:
    - PyLucene (wrapper for Lucene, `http://lucene.apache.org/pylucene/index.html`).

Document indexing

- Input: CACM document collection, stop-list.
- Output: index files (implemented depending on the software library).
- Parameters: stemming and others depending on the software library.

Document ranking

- Input: index files, queries.
- Output: run formatted according to the TREC guidelines:
  - One run is produced for each experiment and includes all the ranked document list produced for the queries. It is implemented as a file in which each row is formatted as follows:

    QueryId `Q0` DocId Rank Score RunLabel

    where QueryId is the query identifier, DocId is the document identifier, Rank is the rank of the document in the list of document retrieved against the query, Score is the score of the document in the list of document retrieved against the query, RunLabel is an alphanumeric string that identifies the run; for example:

...					
53	Q0	1234	1000	0.7886	testrun01
54	Q0	768	1	3.5677	testrun01
54	Q0	1205	2	3.5640	testrun01
54	Q0	13	3	2.3490	testrun01
...					

Ranking evaluation

- Input: run and relevance assessment file.
- Output: evaluation measures.
- Procedure: type `trec_eval` without parameters to get the help.

Install JCC

- Go to `https://lucene.apache.org/pylucene/install.html`
- Go to `https://lucene.apache.org/pylucene/jcc/install.html`
- `mkdir temp;cd temp`
- At prompt
  `svn co http://svn.apache.org/repos/asf/lucene/pylucene/trunk/jcc jcc`
- `cd jcc`
- `python setup.py build`
- `sudo python setup.py install` (sudo may correspond something different in windows)
- `cd ..`

Install PyLucene

- Download through `http://www.apache.org/dyn/closer.lua/lucene/pylucene/`
- Untar or unzip the package
- Open the package
- `cd` to the PyLucene source directory
- Edit Makefile and uncomment the lines of your system (check python version, OS version, Java version, etc.). Check also the paths (for example, where python is installed).
- `make`
- `make test` (recommended)
- `sudo make install` (you need admin or root password)

Indexing using pylucene

- Download Indexer, Searcher, Runner, Grid Searcher from the website
- Download
  `http://www.dei.unipd.it/~melo/ml-school/cacm.txt.zip`
- `mkdir docs;cd docs`
- `unzip ../cacm.txt.zip`
- `cd ..`
- Indexing:
  `python IndexFiles.py docs`
  and the `IndexFiles.index` directory will created

Searching using pylucene

- Searching:
  `python SearchFiles.py`
  and input a query

Running using pylucene

- Download
  `http://www.dei.unipd.it/~melo/ml-school/queries.txt`
- Batch running using the standard Lucene scoring:
  `python RunTFIDF.py runtag < queries.txt`
- For writing a run file:
  `python RunTFIDF.py runtag < queries.txt > runfile.txt`
- `runtag` and `runfile` should be changed when changing configuration (e.g. free parameters)
- Batch running using BM25:
  `python RunBM25.py 1.2 0.75 runtag < queries.txt` where $b = 1.2$ and $k_1 = 0.75$.

Grid searching using pylucene

- Download `http://www.dei.unipd.it/~melo/ml-school/qrels-treceval.txt`
- Check and adapt `GridSearchBM25.py`
- Grid searching:
  `python GridSearchBM25.py` and wait...
- For each runfile, run `trec_eval`:
  Using Unix, for example, from the prompt: `for b in {0..50..50}; do for k1 in {0..100..50}; do trec_eval qrels-treceval.txt BM25-b=$b-k1=$k1.txt; done; done`

[1] W. Croft and J. Lafferty, editors.
Language Modeling for Information Retrieval, volume 13 of *Kluwer International Series on Information Retrieval*.
Kluwer Academic Publishers, 2002.

[2] S. Robertson and H. Zaragoza.
The probabilistic relevance framework: BM25 and beyond.
Foundations and Trends in Information Retrieval, 3(4):333–389, 2009.

[3] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges.
Optimisation methods for ranking functions with multiple parameters.
In *Proceedings of CIKM*, pages 585–593, 2006.