MarcXimiL : near duplicates detection

KRAUSE, Jan Brice, BOREL, Alain

Abstract
MarcXimiL is an open source tool which works on MARCXML records and calculates similarity indices between these records. After a short theoretical introduction, the tutorial will focus on how to install, parametrize and use the tool. This tool can be implemented in order to:

- prevent creation of duplicates (similar records are shown during the validation process)
- identify duplicates into batch files before ingest
- find duplicates inside a collection
- suggest to users similar records to the one found after a request
- match related documents eg. preprints and articles

Reference

Available at:
http://archive-ouverte.unige.ch/unige:23173

Disclaimer: layout of this document may differ from the published version.
MarcXimiL – near-duplicates detection

• flexible, open-source, multi-platform software supporting
• implementation of multiple strategies for record comparisons
• modularity: each task is performed by a function, which may be selected among functions sets that share a common programming interface (in fact Python modules)
• initial double blind test of algorithms (precision, recall)
• lately, focus on speed optimization
• detailed information: http://infoscience.epfl.ch/record/141894
Bibliographic similarity – applications

• merging collections (matching records)
• relevance feedback (more like this / similar records)
• classification (e.g. regroup records for FRBR)
• plagiarism detection (N.B. full-text oriented, but similar)
• collection structure analysis

• near duplicates detection (de-duplication)
Bibliographic records – MARCXML

- **MarcXimiL** has its own internal records representations.
- But uses currently only MARCXML as input.
- **MARCXML** is an XML of the classic MARC format.
- Described in an XML Schema hosted by the **U.S. Library of Congress**.
- Most library catalogues are in MARC formats (or compatible).
- Possible to **generalize bibliographic similarity analysis by the comparison of MARC records**.
- MARC is sufficient now, but other loading functions may be added easily.
Bibliographic records – MARCXML

<collection>
  <record>
    <controlfield tag="001">
      <subfield code="a"/>
    </controlfield>
    <controlfield tag="005">
      <subfield code="z"/>
    </controlfield>
    <datafield tag="123" ind1="7" ind2="8">
      <subfield code="a"/>
    </datafield>
    <datafield tag="456" ind1="9" ind2=""/>
  </record>
  <record>
    <controlfield tag="001">
      <subfield code="a"/>
    </controlfield>
    <controlfield tag="005">
      <subfield code="z"/>
    </controlfield>
    <datafield tag="123" ind1="7" ind2="8">
      <subfield code="a"/>
    </datafield>
    <datafield tag="456" ind1="9" ind2=""/>
  </record>
</collection>
MarcXimiL MARCXML parsing functions:

- **controlfields**:
  - parse_controlfield (usually for system record ids, ex: 001)

- **datafields**:
  - non repeatable fields: parse_nonrep (ex: abstract = 520$a)
  - repeatable: parse_multi (ex: authors = 100$a + 700$a)
  - concatenation: parse_concat (ex: titles+subs = 245$a+245$b)
  - conditional: parse_conditional (ex: year = 260$c or if absent 269$c)
Monte Carlo study of forward $\pi^0$ production spectra to be measured by the LHCf experiment for the purpose of benchmarking hadron interaction models at 185+173 eV.

Subfield code: a) LHCf: Calibration of hadron interaction models for high energy cosmic-ray physics at the LHC energy.

Subfield code: b) Adriani, O.

Subfield code: c) Castellini, G.

Subfield code: d) Bongi, M.

Subfield code: e) Bonechi, L.

Subfield code: f) Faus, A.

Subfield code: g) Itoh, Y.

Subfield code: h) Menjo, H.
Different strategies for different applications

• Duplicate detection: search for nearly exact matches (high precision), including many different fields – absolute detection threshold important.

• FRBRization: exact matches only, but on a very limited selection of fields?

• Relevance feedback/”More like this”: fuzzier search acceptable, relative ranking matters more
General description of field comparison functions

- Compare fields or sets of fields in two records
- Return a score between 0.0 and 1.0 (or None)
- Divided into several families depending on how we process the field content
RAW field comparison functions

- Similarity score calculated based on the fields in two records
- The rest of the collection doesn't play any role
years_comp__raw()

- Simple decreasing function of the difference in publication year

\[ 1 - 0.1 \times |\text{year1} - \text{year2}| \]

1) \(<\text{datafield tag="260" ind1="" ind2=""}>\)
\(<\text{subfield code="c">2011</subfield>\)</datafield>

2) \(<\text{datafield tag="260" ind1="" ind2=""}>\)
\(<\text{subfield code="c">2010</subfield>\)</datafield>

\(\text{score} = 1.0 - (2011-2010) \times 0.1 = 0.9\)
items_comp__raw()

- Fields are treated as sets of values
  \[ \frac{N(\text{set1} \&\& \text{set2})}{N(\text{set1} \mid\mid \text{set2})} \]
- Items in set1 and set2 compared using strict text equality

1) <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Menjo, Hitomi</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Adriano, Ottavio</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bonechi, Luigi</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bongi, Marco</subfield></datafield>

2) <datafield tag="100" ind1=" " ind2=" "><subfield code="a">Menjo, H</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Adriani, O</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bonechi, L</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bongi, M</subfield></datafield>
<datafield tag="700" ind1=" " ind2=" "><subfield code="a">Castellini, G</subfield></datafield>

No exact match, the score is zero!
authors_comp__raw()

N(authors1 && authors2)/N(authors1 || authors2)

• Individual names parsed as “Lastname, F” or “Firstname Lastname”

1) <datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Menjo, Hitomi</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Adriano, Ottavio</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Bonechi, Luigi</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Bongi, Marco</subfield>
</datafield>

2) <datafield tag="100" ind1=" " ind2=" ">
   <subfield code="a">Menjo, H</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Adriani, O</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Bonechi, L</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Bongi, M</subfield>
</datafield>
<datafield tag="700" ind1=" " ind2=" ">
   <subfield code="a">Castellini, G</subfield>
</datafield>

Score = 3 / (3 + 3) = 0.5
Levenshtein distance $L = \text{minimal number of character modifications to transform string1 into string2}$

$0 < \exp(-L/L_{\text{par}}) \leq 1$

$L_{\text{par}} = 10.0$ by default

1) <datafield tag="245" ind1=" " ind2=" ">
<subfield code="a">Monte Carlo study of forward $\pi^0$ production spectra to be measured by the LHCf experiment for the purpose of benchmarking hadron interaction models at $10^{17}$ eV</subfield></datafield>

2) <datafield tag="245" ind1=" " ind2=" ">
<subfield code="a">Monte Carlo study of forward $\pi^0$ production spectra to be measured by the LHCf experiment for the purpose of benchmarking hadron interaction models at $10^{17}$ eV</subfield></datafield>

$L = 10$, Score = $\exp(-10/10) = 0.3679$
identifiers_comp__raw(), multiidentifiers_comp__raw()

• Compares two identifiers (or sets of identifiers) and looks for at least 1 shared value.
• Since some identifier conventions are case-insensitive, we convert everything to lowercase.
• Exclude a prefix of 5 chars or less, followed by a colon: it could be just an indication of the identifier type (such as DOI:, ArXiv:, etc), it's easier to neglect it.
isbn_comp__raw()

• Compare sets of ISBNs (10 or 13 digits) looking for at least one shared item
• Return either 1.0 (match found) or 0.0
identifier_synonyms_comp__raw()

- Returns pre-defined scores based on identifiers that we assume to be synonymous or nearly so. Example: ambiguous document types

```python
globalvars.doctype={'08doctype':{'ARTICLE':{'CONF':0.9}, 'ARTICLE':{'REVIEW':1.0}}}
```
WC field comparison functions

- Similarity score calculated based on indexed fields
- Require indexing the whole collection

We invest some preprocessing time to (hopefully) save more during the comparisons proper.
ntfidf_vectorcosine_wc() et al.

- Vector model of information retrieval: represents documents as vectors over a space of terms (words).
- Component in the dimension of term $t$: term frequency $tf$ (# of occurrences of term $t$) * inverse document frequency $idf$ (1 over # of documents containing $t$); ntf implies further normalization.
- Similarity expressed as a dot product of two vectors (cosine) or other combinations (Dice, Jaccard).
ntfidf_vectordice_wc() \[
\text{Dice} = \frac{2 \sum_{t=1}^{T} w_{qt} w_{dt}}{\sum_{t=1}^{T} w_{qt}^2 + \sum_{t=1}^{T} w_{dt}^2}
\]

More efficient than rigorous cosine (would involve a square root)

\[w_{dt} = \frac{tf_{d,t}}{\max_d (tf_{d,t})} \times \ln \left( \frac{N}{df_t} \right)\]

= 0 if df = N, i.e. term found in all documents;
let us cheat and use \(N \geq e\)

1) <datafield tag="245" ind1=" " ind2=" "><subfield code="a">Monte Carlo study of forward \(\pi^0\) production spectra to be measured by the LHCf experiment for the purpose of benchmarking hadron interaction models at 10^{17} \text{eV}</subfield></datafield>

{‘monte’: 1, ‘production’: 1, ‘at’: 1, ‘lhcf’: 1, ‘ev’: 1, ‘carlo’: 1, ‘for’: 1, ‘to’: 1, ‘experiment’: 1, ‘forward’: 1, ‘be’: 1, ‘\$\pi^0\$’: 1, ‘models’: 1, ‘10^{17}’: 1, ‘purpose’: 1, ‘measured’: 1, ‘by’: 1, ‘hadron’: 1, ‘interaction’: 1, ‘of’: 2, ‘study’: 1, ‘spectra’: 1, ‘the’: 2, ‘benchmarking’: 1}

2) <datafield tag="245" ind1=" " ind2=" "><subfield code="a">Monte Carlo study of forward \(\pi^0\) production spectra to be measured by the LHCf experiment for the purpose of benchmarking hadron interaction models at 10^{17} \text{eV}</subfield></datafield>

{‘monte’: 1, ‘production’: 1, ‘at’: 1, ‘lhcf’: 1, ‘ev’: 1, ‘carlo’: 1, ‘for’: 1, ‘to’: 1, ‘experiment’: 1, ‘forward’: 1, ‘be’: 1, ‘\$\pi^0\$’: 1, ‘models’: 1, ‘10^{17}’: 1, ‘purpose’: 1, ‘\pi^0’: 1, ‘measured’: 1, ‘by’: 1, ‘hadron’: 1, ‘interaction’: 1, ‘of’: 2, ‘study’: 1, ‘spectra’: 1, ‘the’: 2, ‘benchmarking’: 1}

Without cheating, most df’s would be 2, with only 2 documents, so \(w = 0\) unless the word appears only in 1 record => score = 0

Cheating: \(w = 1/2*\ln(e/2)=0.153\), score = 0.5686; would probably increase in a larger collection
okapibm25__wc()
INITIALS family: items_comp__initials()

- Like RAW, but only the initials of each word are used.

Could be extended to a generalized HASH family in future versions

1) <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Menjo, Hitomi</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Adriano, Ottavio</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bonechi, Luigi</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bongi, Marco</subfield></datafield>

2) <datafield tag="100" ind1=" " ind2=" "><subfield code="a">Menjo, H</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Adriani, O</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bonechi, L</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Bongi, M</subfield></datafield> <datafield tag="700" ind1=" " ind2=" "><subfield code="a">Castellini, G</subfield></datafield>

Score = \frac{3}{3+1} = 0.75
SHINGLES field comparison functions

- Like WC, but considering groups of $N$ (4 by default) adjacent words instead of isolated words
- Should be more precise than WC since the word order is taken into account – are you ready try?

1) {'lhcf experiment for the': 1, 'study of forward $\pi^0$': 1, 'spectra to be measured': 1, '$\pi^0$ production spectra to': 1, 'measured by the lhcf': 1, 'for the purpose of': 1, 'forward $\pi^0$ production spectra': 1, 'the purpose of benchmarking': 1, 'hadron interaction models at': 1, 'production spectra to be': 1, 'the lhcf experiment for': 1, 'of forward $\pi^0$ production': 1, 'purpose of benchmarking hadron': 1, 'be measured by the': 1, 'models at 10$^{17}$ ev': 1, 'benchmarking hadron interaction models': 1, 'interaction models at 10$^{17}$': 1, 'by the lhcf experiment': 1, 'of benchmarking hadron interaction': 1, 'carlo study of forward': 1, 'to be measured by': 1, 'monte carlo study of': 1, 'experiment for the purpose': 1}
Further families...

- `ntfnidf_vectordice_comp__soundex()`: same as `ntfnidf_vectordice_comp__wc()` but words are reduced to an approximate spelling (or a code number for one) using the Soundex algorithm => more words considered the same

Score = 0.7672 => higher than for `ntfnidf_vectordice_comp__wc()`
Similarity strategies – Example

record_rules = geometric_mean
records_comp = records_comp_2collections_luhn
globalvars.luhn_comparisons_field = 'title'
globalvars.output_threshold = 0.75 # use -1 to output everything
record_structure = { 
  'recids' : { 'marc' : '001', 
  'weight' : 0, 
  'parse-func': parse_controlfield, 
  'comp-func' : fields_concat__raw },
  'year' : { 'marc' : '260  c', 
  'weight' : 1, 
  'parse-func': parse_nonrep, 
  'comp-func' : years_comp__raw },
  'title' : { 'marc' : ['245  a', '245  b'], 
  'weight' : 2, 
  'parse-func': parse_concat, 
  'comp-func' : ntfidf_vectordice_comp__wc },
  'authors' : { 'marc' : ['100  a', '700  a', '9001 a'], 
  'weight' : 1, 
  'parse-func': parse_multi, 
  'comp-func' : authors_comp__raw },
  'source' : { 'marc' : ['773  t', '7112 a'], 
  'weight' : 1, 
  'parse-func': parse_concat, 
  'comp-func' : items_comp__initials } }
Similarity strategies – Python scripts

• Strategies:
  – allow to customize all key aspects of the analysis through a combination of functions (and parameters).

• Modularity:
  – each task is performed by a function, chosen among functions sets that share a common programming interface (Python modules)

• Python:
  – Python has a clean syntax to import modules
  – MarcXimiL is written in pure Python
    (multi-platform, popular, free, clean code, standard library is sufficient)
  – function names are used directly in strategies, new functions may be used at once.
Similarity strategies – Options

1) The way records are compared together [records_comp]:
   • 1 or 2 collections, Luhn, multi-processing ...

2) The description of the fields to be analysed and functions to compare them [record_structure]: years/dates, authors, text (title, abstract), identifiers...

3) The combination of fields similarities into record similarities [record_rules]: weighted means (arithm., geom., harmon.), and ad-hoc functions.

Others: output_threshold, luhn_comparisons_files...
Similarity strategies – record_comp

• Important level for analysis' speed optimization.

• For now, the best option are:
  – `records_comp_single_luhn` or `records_comp_2collections_luhn` on title fields.
  – Variations using authors names are available.
  – We are working on faster ways of doing.

• For a thorough analysis, use:
  – `records_comp_2collections_multiproc` or `records_comp_single_multiproc`
Similarity strategies – Luhn strategy

![Graph showing Luhn strategy and Full comparison]
Similarity strategies – record rules

• Globally, the most efficient is the \textit{geometric\_mean} function (weighted).
• Some ad-hoc functions also yield good results, like \textit{ubiquist2} (simplified):
  – Global similarity is set to zero.
  – If identifiers are the same, similarity becomes 1.0
    • and we jump to next record pair
  – Otherwise if author similarity is $\geq 0.85$ and year similarity is $\geq 0.9$
    • records similarity becomes $0.85 \times$ author's similarity.
  – Then abstract and title similarity are computed.
    • records similarity becomes the max of: current record similarity, title similarity, and abstract similarity.
Similarity strategies – deduplication example

record_rules = geometric_mean
records_comp = records_comp_2collections_luhn

globalvars.luhn_comparisons_field = 'title'
globalvars.output_threshold = 0.75 # use -1 to output everything
record_structure = { 
    'recids' : {'marc' : '001',
                'weight' : 0,
                'parse-func': parse_controlfield,
                'comp-func' : fields_concat__raw },
    'year' : {'marc' : '260 c',
              'weight' : 1,
              'parse-func': parse_nonrep,
              'comp-func' : years_comp__raw },
    'title' : {'marc' : ['245 a', '245 b'],
              'weight' : 2,
              'parse-func': parse_concat,
              'comp-func' : ntfnidf_vectorordice_comp__wc },
    'authors' : {'marc' : ['100 a', '700 a', '9001 a'],
                 'weight' : 1,
                 'parse-func': parse_multi,
                 'comp-func' : authors_comp__raw },
    'source' : {'marc' : ['773 t', '7112 a'],
                 'weight' : 1,
                 'parse-func': parse_concat,
                 'comp-func' : items_comp__initials } }
Similarity strategies – variations

You may take advantage of a [group of] field[s] twice using two different methods, for example in record_structure:

```python
[...]
'05authors1' : {'marc' : ['100 a', '700 a', '9001 a'],
               'weight' : 1,
               'parse-func': parse_concat,
               'comp-func' : items_comp__initials },

'05authors2' : {'marc' : ['100 a', '700 a', '9001 a'],
               'weight' : 1,
               'parse-func': parse_multi,
               'comp-func' : authors_comp__raw },

[...]```
Similarity strategies – combining results sets

- Depending on the configuration, MarcXimiL will store results in an SQL database (by default).
- In that case, you may combine/merge two or more results sets (for example obtained with two different strategies) using option -m “set1 set2 set3 ...“
- This will perform a geometric mean on the same results pairs.
Practical session & demo

- MarcXimiL version 0.3.6
  - Website: http://marcximil.sourceforge.net/
  - Download: http://sourceforge.net/projects/marcximil/files/marcximil-0.3.6.zip/download

- Requirement: Python 2.x (2.5, 2.6, 2.7)
  - Download: http://www.python.org/download/

- Howto: