FuzzyPhoto Project WP5 Record Matching

David Croft

February 2, 2015

Contents

Contact

This documentation is intended to contain all the information necessary to understand the fuzzymatch process created for the FuzzyPhoto project from beginning to end (see section 4). It is, however, difficult to conceive of every circumstance especially when covering topics which seem intuitive to me because I wrote the process. If this documentation does not address your specific query, please contact me at david@somewebsite.co.uk, this e-mail address will remain valid indefinitely.

Dr David Croft.

Chapter 1

Match creation

The fuzzyphoto project is a 2 year Arts & Humanities Research Council (AHRC) funded research project (AH/J004367/1) that attempts to identify co-referent records across records from multiple collections of historic photographs. Due to the nature of the records, the disparate sources of the records, a lack of standardisation for metadata within GLAM institutions and other factors, even when two records are referring to the same photograph the chance of those records being identical is very low. In order to identify the co-referent matches this fuzzymatch process attempts to identify commonalities between records and the information they contain despite the differences in the precise information held. In order to identify matches, four pieces of information from each record are used. These are the title of the photograph, the name of the photographer, the photographic process used to create the photograph and the dates associated with the photograph (most typically the date taken).

1.1 Title

The similarity metric used for processing the title fields is a modified version of the LSS metric published previously. The metric is effectively just a cosine similarity metric. However, instead of operating on the raw term vectors representing the title fields being compared, the metric modifies these term vectors to account for semantic similarities between the individual terms within the vectors. In this respect the metric is similar to LSA or STASIS. The metric we use is, however, significantly faster than either of those. The term vectors are modified to account for semantic similarities between the terms using similarity values produced using WordNet.

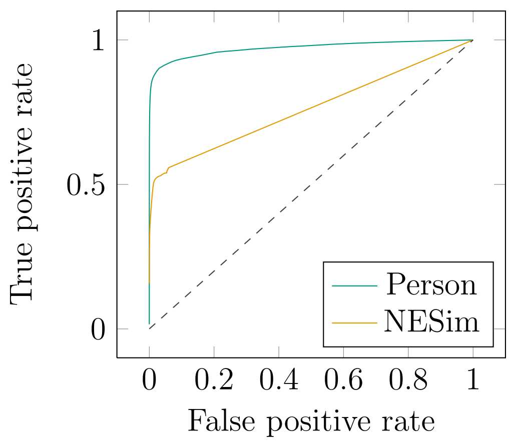

Figure 1.1: ROC curve comparison of NESim and person metric performance.

Full details on the Lightweight Semantic Similarity (LSS) metric can be found in the previously published paper[1] on that subject, the only modification is the inclusion of a minimum word similarity threshold (set at 0.33). Any individual term similarity values that are less than or equal to the threshold should be lowered to 0.0.

1.2 Person

Multiple name similarity algorithms exist, the earliest of which date back more than a century. Unfortunately for this project the format of the names held in the records is unknown from one record to another. Tests with existing algorithms able to handle names held in an unknown format (e.g. Named Entity Similarity (NESim)) were unsatisfactory. This field is therefore handled by a custom similarity metric which uses a combination of approximate string comparison to compare the individual name elements and a series of heuristic rules to identify probable matches between those elements despite the elements being (potentially) recorded in different orders. While the precision rate of the algorithm is still lower than desired, the recall rate is excellent and the overall algorithm has produced dramatically better results than NESim (see fig. 1.1).

[1]D. Croft, S. Coupland, J. Shell, and S. Brown, A fast and efficient semantic short text similarity metric, in Computational Intelligence (UKCI), 2013 13th UK Workshop on, Sept 2013, pp. 221227.

1.2.1 Tokenisation/filtering

The first stage is tokenisation and filtering of the raw data. The raw text is converted to lower case and split into separate elements at the word boundaries. Word boundaries are considered to be anywhere a punctuation character[2] is found and non alphabetic characters are removed. For the worked example this produces the two vectors seen below.

- A = [‘benjamin’, ‘frances’, ‘johnston’]
- B = [‘b’, ‘francis’, ‘johnston’, ‘miss’]

As our research has, so far, focused entirely on collections from Western Europe, North America and other English speaking counties, our approach has only been designed to work with the Latin characwter set. This would restrict our approach the Germanic (e.g. English and German) and Romance languages (e.g. French, Spanish etc). At present the C++ Jaro-Winkler implementation we use only supports American Standard Code for Information Interchange (ASCII) coded strings, however Unicode supporting implementations do exist in other programming languages. We are, therefore, hopeful that our approach can be expanded to handle other encodings in the near future, for the moment non-ASCII characters should be converted to their base forms (e.g. ò ó ô õ ō ȯ ö ŏ ǒ ọ → o) before being processed by our metric and it does not work for languages such as Russian, Japanese or Arabic where no ASCII compatible base form of the character set exists.

1.2.2 Element similarity

The second stage is the generation of a complete similarity matrix for elements of the two vectors being compared. This has the potential to be computationally expensive for vectors with a large number of elements, however the average number of elements is only 0.34[3]. The resulting matrix for the worked example can be seen in table 1.1.

Jaro-Winkler was used as the individual elements comparison method over other techniques (specifically Jaro) as it applies additional significance to the start of the terms being compared. This efficiently addresses the problem of name comparisons between full names, initials and alternate short forms of full names in most circumstances. Obviously initials will be based on the first letter of a full name but short forms are also predominately based on the

[2]I.e. commas, colons, semi-colons and spaces.

[3]Based on an analysis of 342 797 records from 7 Gallery, Library, Archive and Museum (GLAM) collections, the same records produced a maximum size of 20.

start of a full name rather than the middle and end. E.g. Dave from David, Matt from Matthew. Exceptions do exist e.g. Beth from Elizabeth, Dick from Richard. As handling these forms would likely require a white list of known nicknames which would increase the complexity and processing time of our approach these rare exceptions are ignored.

	benjamin	frances	johnston
b	0.71	0.00	0.00
francis	0.49	0.94	0.51
johnston	0.47	0.51	0.00
miss	0.00	0.00	1.00

Table 1.1: Jaro-Winkler similarity matrix.

1.2.3 Element ordering

Stage 3 is the ordering of the the Jaro-Winkler similarity values. For each element of A the similarity values against B are ordered from highest to lowest, see table 1.2.

benjamin		frances		johnston	
b	0.71	francis	0.94	johnston	1.00
francis	0.49	johnston	0.51	francis	0.51
johnston	0.47	miss	0.00	miss	0.00
miss	0.00	b	0.00	b	0.00

Table 1.2: Ordered Jaro-Winkler similarity matrix.

The best match between each element of A to an element from B is then selected, in the case of the worked example the best matches are 'benjamin' $\leftrightarrow$ 'b' = 0.71, 'frances' $\leftrightarrow$ 'francis' = 0.94 and 'johnston' $\leftrightarrow$ 'johnston' = 1.00.

Although in our earlier example every element of A matched against a different element of B in the order similarity matrix this will not always be the case[4]. Under our approach two or more elements in one vector are not allowed to match against the same element in the other. Our approach attempts to find the best overall (i.e. the configuration of non-overlapping

[4]It is, however, rare.

element matches that produces the highest combined Jaro-Winkler similarity value). Whilst an exhaustive search of all of the possible combinations (brute force) would guarantee that the optimum solution was found, this results in excessive computational requirements for the approach and is rarely necessary. Instead the combination of element pairs is selected heuristically, as follows.

If a collision is detected then at least one of the matches must be altered in order to point at a different element. The match with the lowest value should be changed. In cases where multiple matches have the same value, the match which will produce the smallest change should be chosen. If multiple matches will produce the same change, select the first one.

The following section demonstrates a collision situation and the match alternations required. For this example two field values in this case are A = [‘john’, ‘j’, ‘doe’] and B = [‘john’, ‘smith’, ‘doe’]. The resulting ordered Jaro-Winkler similarity matrix is shown in table 1.3. As that table shows, there is a collision between the ‘john’ and ‘j’ elements in A where both have matched to the ‘john’ element in B.

	john		j		doe
john	1.00	john	0.78	doe	1.00
doe	0.53	doe	0.00	john	0.53
smith	0.00	smith	0.00	smith	0.00

Table 1.3: Match collision example.

In this case the correct action is to change the ‘john’ $\leftrightarrow$ ‘j’ match instead of the ‘john’ $\leftrightarrow$ ‘john’ match as this has a similarity of 0.78 as opposed to 1.00. Unfortunately making said change produces a new collision and so the process must repeat again, the full list of changes can be seen in table 1.4.

For each element of A to match against a different element of B, $|A| \leq |B|$ must be true. This is easily achieved by simply assigning the shortest vector to be A, however in cases where $|A| = |B|$ then the element selection should be conducted twice with Jaro-Winkler similarity matrix transposed between iterations.

1.2.4 Match weighting

The Jaro-Winkler values for the element matches are then weighted according to the combined length of each pair of match elements as a proportion of combined length of all the elements. This stage means that matches between

	john		j		doe
john	1.00	john	0.78	doe	1.00
doe	0.53	doe	0.00	john	0.53
smith	0.00	smith	0.00	smith	0.00

	john		j		doe
john	1.00	doe	0.00	doe	1.00
doe	0.53	smith	0.00	john	0.53
smith	0.00			smith	0.00

	john		j		doe
john	1.00	smith	0.00	doe	1.00
doe	0.53			john	0.53
smith	0.00			smith	0.00

Table 1.4: Match collision example.

two initials or matches between an initial and a full name are considered by our process to be less important than matches between two full names. Although two initials could be identical it does not mean that the full name they represent is the same, our weighting approach allows the match between initials to contribute to the overall match value but also recognises its inherent uncertainty. This effect of this weighting is shown in table 1.5.

	benjamin	frances	johnston
	b	francis	johnston
Jaro-Winkler	0.71	0.94	1.00
Length	9	14	16
Weight	0.23	0.36	0.41
Combined	0.16	0.34	0.41
Result	0.91		

Table 1.5: Combining element pair values.

1.2.5 Overall weighting

Finally the overall similarity value produced so far is weighted according to the proportion of the combined elements from $A + B$ that were used. If, for example, we were to compare A against another vector $C =$ ['benjamin'], then under the approach described so far that would produce an overall similarity value of 1.0. Therefore in order to take into account the number of elements actually compared and so rank $A \leftrightarrow B < A \leftrightarrow C$, the similarity value is modified as shown in equation 1.1 where s is the unmodified similarity value.

$$0.5 \left(s + s \frac{2 \cdot (|A| \vee |B|)}{|A| + |B|} \right) \tag{1.1}$$

$$0.5 \left(0.91 + 0.91 \frac{2 \cdot (3 \vee 4)}{3 + 4} \right) = 0.845 \tag{1.2}$$

1.3 Process

The process metric attempts to match the text contained in the records to a list of known processes using a list of process keywords and an approximate string similarity method (Jaro-Winkler). Also taken into account is the difference between colour and monochrome images and between positive and negative images. These factors had been taken into account in two ways, in an implicit manner where it is known that certain photographic processes can only produce monochome/colour or positive/negative images but the metric also looks for keywords associated with colour/monochome and positive/negative and can use these to override assumptions that would have to be made based on the overall process type.

A by-product of this process similarity metric has been the creation of a photographic process hierarchy. The process metric hierarchy is unusual in comparison to other photographic process hierarchies already in existence. Instead of organising the processes according to date or technological progression, this hierarchy is organised based on the likelihood of misidentification between the processes. That is to say, processes which can be easily confused with one another (i.e. dagerrotype and tintype) are positioned close together in the hierarchy, whilst processes that would be difficult to confuse for one another (i.e. daguerreotype and mezzo tint) are positioned distantly from each other.

A visualisation of the full process ontology used by the fuzzymatch program can be seen in section 6.1.

1.3.1 Extraction

To compare the *process* fields we must first identify any known processes within the raw *process* text. As shown in section 6.1 a custom ontology of photographic processes and their identifying keywords was created for the FuzzyPhoto project.

The first stage of stage of the process identification is tokenization of the raw text. The same tokenization system used for the person metric is re-used here, however unlike the person metric no token filtering is required. The resulting process vector is compared against a list of the identifying keywords from the process ontology using the Jaro-Winkler algorithm. Elements from the process vector and the keyword list are counted as matches when the Jaro-Winkler value is greater than a pre-set threshold (0.9 for this project). With the matching keywords identified it is a simple matter to identify any processes in the ontology where the process vector has matched against all the corresponding keywords.

At the same time that the process vector is being matched against the process keyword list, the vector is also matched against secondary keyword lists containing colour/monochrome and positive/negative keywords. If the *process* matches against these then the relevant flags are set to indicate that this *process* describes a colour/negative/positive/negative image. Otherwise the default values from the ontology are used.

1.3.2 Metric

Process similarity is calculated based on the processes in the ontology that each *process* field matches against and not based on the raw *process* field text. As each *process* field can match to multiple processes, process distance is calculated by selecting the maximum value from an $n \cdot m$ comparison.

Three factors go into the *process* similarity, the first of these is the distance between the processes in the ontology. Similarities between individual pairs of processes starts with the assumption that identical processes have a similarity of 1 and that each edge between the pair elements halves the similarity. This distance similarity is then weighted to make up 0.5 of the overall similarity.

The other two factors consider the color/monochrome and positive/negative nature of the processes. Each factor makes up 0.25 of the overall similarity. If for example both *processes* are (and could only be) colour then that would count as a 0.25 increase in the overall similarity. If both *processes* are (and could only be) monochrome then that would also count as a 0.25 increase. If one *process* is colour and the other monochrome that would make a contribution of 0.0. If the colour/monochrome states of both/either *processes* is unclear but they could be a match then that makes a 0.125 contribution.

The same rules apply to the positive/negative contribution.

1.4 Date

The problems with the information contained in this field are two fold. Firstly there is the syntax independent nature of this field, i.e. date information can be represented in a number of different formats and the particular format used for each record is unknown. Secondly there is the imprecision of the date information held. These two problems are handled separately, The first is handled by a rule based system built around the python dateutil libraries and custom regexes which extracts the date information from the raw date fields. The task is significantly simplified by only extracting the year information of the date fields. Day and month information is discarded. The second problem is addressed using a custom similarity metric that produces date field similarity values based not just on how close two date fields are the same but also the precision of that date information. For example, two fields containing the text "19th century" and "19th century" are considered less similar that fields containing "1910" and "1915" despite the former being identical.

1.4.1 Extraction

Comparing *date* fields is treated as two separate problems. The first is extracting the date ranges from the *date* fields and the second is comparing the date ranges.

Extracting the date information is achieved using a combination of the python dateutil library[5] to identify well formatted date information (e.g. "01/05/1901", "1870-12-31" and "3rd June 1895"), a series of custom regexes to identify less structured information (e.g. "circa 19th century" and "1950-60s") and a rule based system to handle 'circa' modifiers. In order to simply the task, only the year information is extracted, however even with this simplifications some information is not successfully interpreted (e.g. "pre world war two").

1.4.2 Metric

Once extracted the date ranges are compared on the basis of three factors (see figure 1.2). Distance (D), measured as the number of years between the midpoints of the two date ranges. Span (S), the number of years between the earliest year in either range and the latest year in either range. Overlap (O), the number of years that both ranges cover.

[5]https://labix.org/python-dateutil

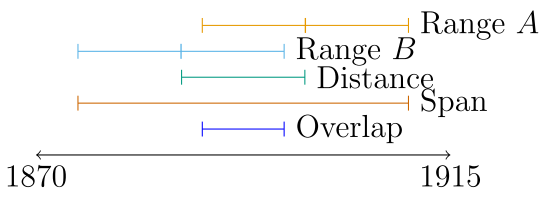

Figure 1.2: Example *date* ranges and gaps.

In addition to the D, S and O values the *date* metric takes an additional tuning parameter ω which controls the time span over which the dates are expected to appear. ω is set as 200 since this is approximately the number of years for which photography has existed. Equation 1.3 shows how these factors are combined.

As only year information is extracted from the *date* fields, the end date of all ranges are increased by 1 to account for the length of that year. This means that a *date* field has a span of at least one year but also that the similarity values produced are always < 1.0.

$$\begin{aligned}\sigma =& \frac{\omega}{\pi} \\ d =& \exp\left(-0.5\left(\frac{D}{\sigma}\right)^2\right) \\ s =& \exp\left(-0.5\left(\frac{S}{\sigma}\right)^2\right) \\ o =& \frac{0}{S} \\ datesim =& s\left(\frac{1}{2}(o+d)\right)\end{aligned} \tag{1.3}$$

1.5 Overall record

Due to the uncertainty and imprecision of the record information, combined with the uncertainty of the automated extraction of the field 'meaning' no single record field is sufficient to identify matching records. Matches are instead identified using a combination of the fields. FuzzyPhoto uses a Fuzzy Inference System (FIS) to combine the four values from the field similarity metrics and produce an overall record similarity. The FIS uses five rules, these are:

- **IF** bad_title **AND** bad_person **THEN** terrible

- **IF** bad_title **OR** bad_person **THEN** bad
- **IF** good_title **OR** good_person **THEN** good
- **IF** good_process **AND** good_date **THEN** good
- **IF** good_title **AND** good_person **THEN** excellent

As these rules show, the *title* and *person* metrics have a significantly greater affect on the overall record similarity than either *process* or *date*. The effectiveness of these rules was developed by a system of trial and error testing in conjunction with a survey of which fields GLAM professionals considered most important when searching manually.

1.5.1 Geometric defuzzification

The final stage in a FIS is the conversion of the resulting fuzzy set into a crisp value, a process called defuzzification. There are a number of different defuzzification methods available, the FuzzyPhoto system used the x co-ordinate of the centroid of the output fuzzy set as the defuzzified value. Calculation of the centroid was initially achieved using a discretization approach but this proved too slow.

FuzzyPhoto instead uses a geometric decomposition approach. For further details see our previously published paper on the subject[6].

1.6 Dendrogram

In order to produce the best possible chances of relevant co-referent matches being identified for each 'seed' record, the system returns not just those records with a high similarity to the seed record directly, but also those that have a high similarity via other nodes as seen in figure 1.3.

The similarity for each node in the dendrogram is simply the average of the similarity of the parent node and the similarity between the two nodes as produced by the FIS with an added edge penalty. Using the situation shown in fig.1.3 as an example, the final similarity values for the match to record C would be $0.4(\mathbf{sim}(A, B) + \mathbf{sim}(B, C))$. While the edge penalty means that the maximum depth of the dendrogram is limited, without this penalty the

[6] S. Coupland, D. Croft, and S. Brown, A fast geometric defuzzication operator for largescale information retrieval, in Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on, July 2014, pp. 11431149.

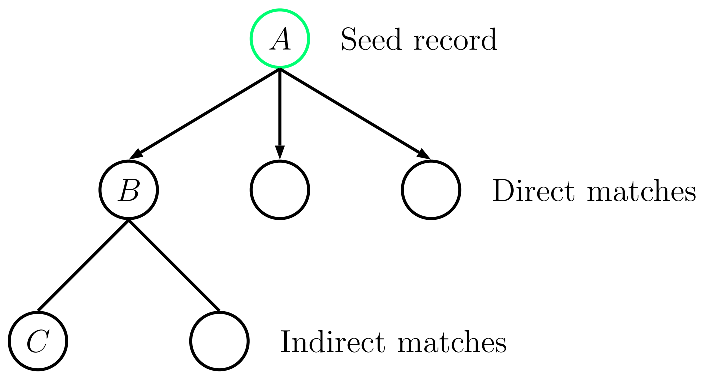

Figure 1.3: Relationships between and identification of seed, direct match and indirect match records.

matches display a topic drift[7] like effect where even though each node in the dendrogram shows a resemblance to its parent and child nodes, the seed bears no resemblance to the leaf nodes.

Of course every record will have some degree of similarity to any seed record. Records which produce low value matches are filtered using a simple threshold. Only matches above the threshold are accepted and then only to a maximum of 30 matches. For an indirect match to occur, (i.e. $A \rightarrow B \rightarrow C$ as show in fig.1.3) a direct match between those records (i.e. $A \rightarrow C$) will already have been attempted and rejected.

[7] A common problem in local feedback query expansion systems

Chapter 2

Directory structure

The main program in the match identification process is the fuzzymatch program. This program requires several support and configuration files. Although the locations of many of these support and configuration file locations can can be changed at runtime (see section 3.3), they have default expected locations. The file/directory hierarchy for a fuzzymatch program running in its default configuration is shown below:

```
/
├── words
│   ├── words.dat
│   └── words.xml
├── fuzzymatch
├── processconfig.xml
└── titlesimple.py
```

The complete file/directory hierarchy, including source files and secondary programs is shown below:

```
/
├── build
├── source
│   ├── datecompare.cpp
│   ├── datecompare.h
│   ├── fuzzyphoto.cpp
│   ├── fuzzyphoto.h
│   ├── jarowinkler.c
│   ├── jarowinkler.h
│   ├── main.cpp
│   ├── personcompare.cpp
│   ├── personcompare.h
│   ├── processcompare.cpp
│   ├── processcompare.h
│   ├── titlecompare.cpp
│   └── titlecompare.h
├── words
│   ├── words.dat
│   └── words.xml
├── createSingleFiles.py
├── dataConfig.py
├── datehandler.py
├── fuzzymatch
├── getRecordsFile.py
├── makefile
├── processconfig.xml
└── titlesimple.py
```

Chapter 3

Programs

3.1 Extracting records from database

The creation of the records file from the records held in the database is performed by getRecordsFile.py. This program connects to the specified database and schema and outputs a correctly formatted XML file (named ’records.xml’ by default).

3.1.1 Running

Program usage is:

```
python getRecordsFile.py [output filename]
```

For example:

```
python getRecordsFile.py records.xml
```

3.1.2 Dependencies

The getRecordsFile.py program was developed, tested and designed for use on a Debian based system. It should, however, work on any reasonably modern *nix platform. In order to work correctly the program has several dependencies beyond a base Python installation, specifically:

- Python - Python v2.*. Development was done using v2.7.2 but other versions may work. The code was not tested with v3.
- Apertium - Free translation package, used to create the `translation` tags for the ’orsay’ records. The necessary installation packages are, “apertium”, “apertium-en-es” and “apertium-fr-es”.

- MySQLdb - Python interface to MySQL databases, used to query and extract the records from the records database. The necessary installation package is “python-mysqldb”
- Lxml - Python wrapper to the libxml2 libraries, used to handle XML file read/write. The necessary installation package is “python-lxml”
- Datehandler - Custom writing python library designed to handle uncertain date formats. Contained in the datehandler.py file.
- Dateutil - Extensions to the datetime module. The necessary installation package is “python-dateutil”.

On Debian based systems the necessary packages can be installed using the following command:

3.2 Setting seed records

The resulting file will not have any `seed` attributes configured. While it is possible to pass the file to the fuzzy match program as is, this will cause the program to generate matches for all of the records. At time of writing, only the erps, orsay, nmem and st_andrews collections were expected to have widget integration on their websites in the near future. As such, only those collections need matches generated for their records. The dataConfig.py program will read in a records XML file ('records.xml' by default) and output copy ('process.xml' by default) with only the required `seed` attributes set.

3.2.1 Running

Program usage is:

```
python dataConfig.py [input filename] [output filename]
```

For example:

```
python dataConfig.py records.xml process.xml
```

3.2.2 Dependencies

The dataConfig.py program was developed, tested and designed for use on a Debian based system. It should, however, work on any reasonably modern *nix platform. In order to work correctly the program has dependencies beyond a base Python installation, specifically:

- lxml - Python wrapper to the libxml2 libraries, used to handle XML file read/write. The necessary installation package is “python-lxml”

3.3 Match generation

3.3.1 Running

Program usage is:

```
./fuzzymatch xml_filename [dat filename]
```

Optional arguments

The fuzzymatch program is multi-threaded for additional performance, by default the program will run across all cores simultaneously. The program will maximise CPU usage on whatever cores it runs, this can cause problems for other time critical programs running on the same machine. The number of cores used can, therefore, be specified using the -t flag. Values supplied to this flag must be greater than zero.
For example:

```
./fuzzymatch process.xml output.dat
```

The names and relationships of the photographic processes used by the process metric are contained in the (by default) processconfig.xml file as seen in the file hierarchy shown in section 5.5.
The process configuration file used by the fuzzymatch program can be changed using the optional -p argument. For example:

```
./fuzzymatch -p processconfig.xml process.xml
```

3.3.2 Dependencies

The fuzzymatch program requires several external libraries in order to compile and run, specifically:

- Python - Python v2.*. Development was done using v2.7.2 but other versions may work. The code was not tested with v3.
- NLTK - Natural Language ToolKit. Specifically the python wrapper to it. The necessary installation package is “python-nltk”

- Tinyxml2 - On Debian based systems the installation packages available are via aptitude are flawed, tinyxml2 should therefore be installed from source, see section 3.3.2.
- Boost - Specifically the Regex libraries. The necessary installation package is "libboost-regex1.4.9-dev" but "libboost-all-dev" is recommended.

Tinyxml2 installation

Tinyxml2 can be installed using the following commands.

```
wget https://github.com/leethomason/tinyxml2/archive/master.zip
  -O tinyxml2-master.zip # download the source code
unzip tinyxml2-master.zip
mkdir tinyxml2-build
cd tinyxml2-build
cmake ../tinyxml2-master # run cmake
make -j4 2>&1 | tee ../Tinyxml2Make.log # make TinyXML2 and
  # save a log of the compilation
sudo make install 2>&1 | tee ../Tinyxml2MakeInstall.log # install
  # the compiled TinyXML2 and save a log of the installation
```

3.4 Single match file creation

3.4.1 Running

Program usage is:

```
python createSingleFiles.py recordfile datfile [source]
```

Optional arguments

The createSingleFiles program requires a record file and a match file. It is important that the same record and match files match up, i.e. the record file is the same one used to create the match file.
For example:

```
python createSingleFiles.py process.xml output.dat
```

By default the createSingleFiles program will produce a singlematch file for every record listed in the match file. However, if the matches for a specific

institution are required, then then those sources can be listed and the createSingleFiles program will only produce singlematch files for the specified sources.
For example:

```
python createSingleFiles.py process.xml output.dat erps loc
```

3.4.2 Dependencies

The dataConfig.py program was developed, tested and designed for use on a Debian based system. It should, however, work on any reasonably modern *nix platform. In order to work correctly the program has dependencies beyond a base Python installation, specifically:

- lxml - Python wrapper to the libxml2 libraries, used to handle XML file read/write. The necessary installation package is “python-lxml”

Chapter 4

Process flow

The complete series of actions necessary to go from the database containing collection records to a series of individual match files involves multiple files and programs which must be run in a specific sequence.
The complete sequence of programs along with how the output files of one process feed into the next process/es can be seen in figure 4.1. For simplicity's sake the default files names expected/produced by each process have been shown in figure 4.1. The filesnames/locations can be changed at runtime using as described in section 3.

4.1 Automation of the process

4.1.1 run.sh

In order to simply the creation of the matches and allow for program to be run automatically in the future, the run.sh batch file was created.
When executed, the run.sh file runs the complete sequence of programs required to go from records held in the LIDO database to the .json files which represent the record matches. The run.sh file also copies the .json files from the processing server onto the internet accessible webserver for use by the widgets on the various partner websites.

4.1.2 .json file transfer

Copying the .json files to the webserver is included as one of the run.sh actions and is achieved using the rsync file transfer program. For rsync to be successfully connect to the webserver without requiring manual interaction requires the use of ssh keys. By default the run.sh batch file will use 'fuzzy-webserver_rsa' private ssh key file located in the ssh_keys folder (see section

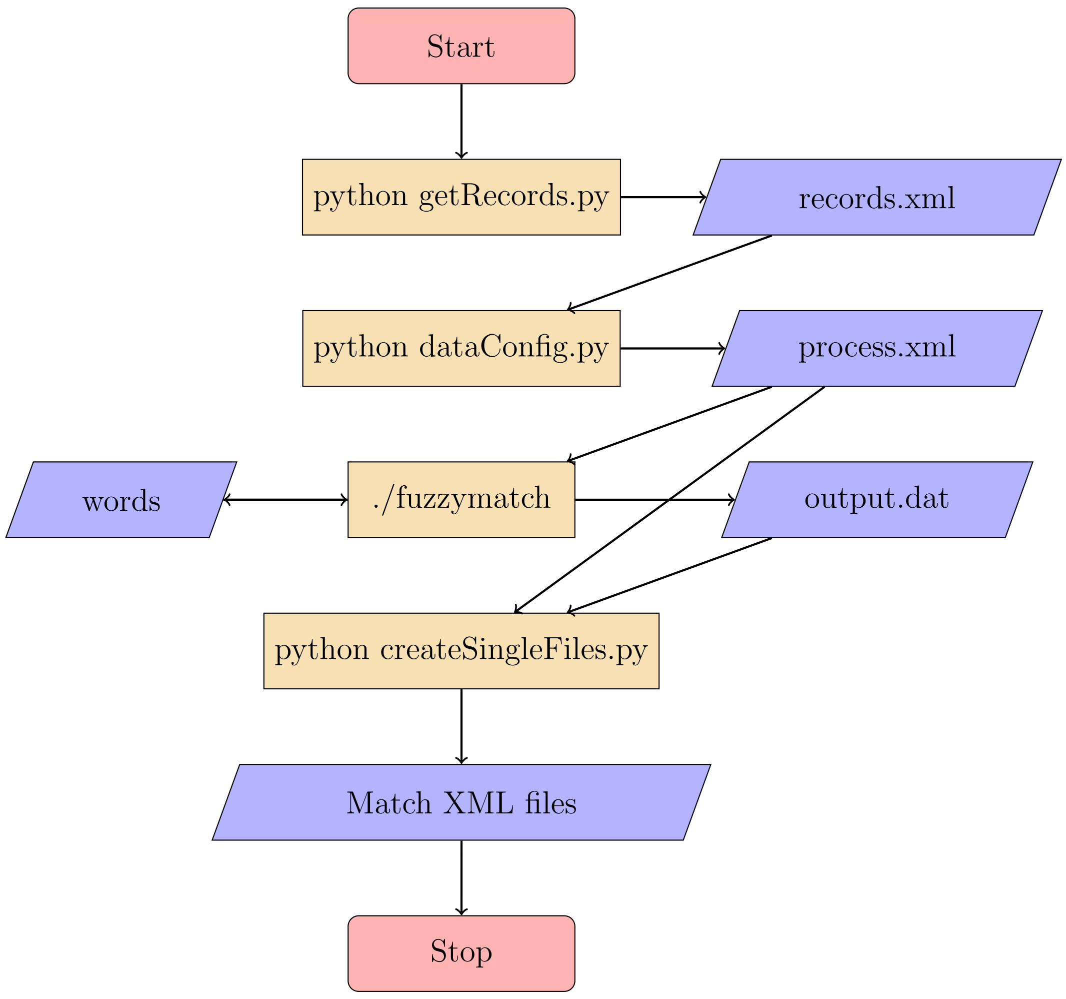

Figure 4.1: Process flow diagram.

2). For this to work the 'fuzzywebserver_rsa.pub' public ssh key file needs to have been added to the authorized_keys file located on the webserver in advance.

Chapter 5

File formats

5.1 Record file

Records from the various collections are stored in a MySQL database. During development the decision was made to keep the database and match portions of this project separate so as to ease development and prevent potential issues regarding network and/or database access. Therefore, the required information is extracted from the database before being passed to the match software as opposed to the match software having direct database integration. The match software ingests record information as XML files in the format detailed below. The format supplies not just the record information but also the configuration options of the fuzzy match software.
The basic file framework is as follows:

```
<records count="0" balancedthresh="0.7" personthresh="0.7"
                   titlethresh="0.5">
</records>
```

count is an optional attribute detailing the number of records contained in the balancedthresh, personthresh and titlethresh and optional attributes which control the minimum similarity thresholds for each of the match types performed by the fuzzy match program. Record pair matches with similarities lower than the relevant threshold will be ignored.
Individual records are described as follows:

```
<record id="12345" published_id="aaa" source="erps"
                   seed="true">
</record>
```

'id' is a required, numeric attribute. 'published_id' and 'source' are not required for the fuzzy match software but are required for later processes and

should, therefore, be included. The `seed` attribute controls which records will be processed by the fuzzymatch program and should have a value of `true` or `false`. If the `seed` attribute is missing, then a value of `false` is assumed.
The example above is for a blank record containing no field information, a more typical example would be.

```
<record id="12345" published_id="aaa" source="erps"
                   seed="true">
  <title>Example title 1</title>
  <person>John Doe</person>
  <process>Photograph</process>
  <startdate>1900</startdate>
  <enddate>1901</enddate>
  <link>http://www.example.com/12345</link>
</record>
```

`title`, `person` and `process` all accept strings. `startdate` and `enddate` should contain 4 digit integers only, representing the relevant years. `link` should contain a url or be left blank.

In the event that the `title` is in a language other that English, an additional `translation` tag should be used. This tag should contain the English translation of the contents of the `title` tag. For example:

```
<record id="54321" published_id="bbb" source="orsay" seed="true">
  <title>Le example title</title>
  <translation>The example title</translation>
</record>
```

At present the `translation` tag is only utilised with records from the Musée d'Orsay (source 'orsay').

```
<records count="2">
  <record id="12345" published_id="aaa" source="erps"
                     seed="true">
    <title>Example title 1</title>
    <person>John Doe</person>
    <process>Photograph</process>
    <startdate>1900</startdate>
    <enddate>1901</enddate>
    <link>http://www.example.com/12345</link>
  </record>
  <record id="54321" published_id="bbb" source="orsay"
                     seed="true">
    <title>Le example title</title>
    <translation>The example title</translation>
  </record>
</records>
```

5.2 Match file

This file is read in and outputted by the fuzzymatch program and read in by the createSingleFiles program. It contains the information regarding which records (as listed in a records file) have been processed. Similarity values from record comparisons and matches identifies. A single file contains all the information for the balanced, title and person matches. This file can also be used by the fuzzymatch program to resume processing at the point it previously stopped.

File structure is simple, consisting of lines of four fields. Each line has the following structure.

```
parent_id|type|child_id|value|
```

For example:

```
123|1|123|1.0|
123|1|456|0.5|
123|2|789|0.6|
456|1|456|1.0|
456|1|123|0.5|
```

The individual fields have the following meanings:

- parent_id - This is the id of the record that the match is from. The id value corresponds to the value of an `id` attribute in the records file.
- type - This is the type of match that the line records. Valid values are -3, -2, -1, 1, 2 and 3.

  Values 1, 2 and 3 are direct matches from one record to another. 1 is a balanced match, 2 a title match and 3 a person match

  Values -1, -2 and -3 are dendrogram matches from one record to another, potentially via other records. -1 is a balanced match, -2 a title match and -3 a person match.

- child_id - This is the id of the record that the match is to. The id value corresponds to the value of an `id` attribute in the records file.

- value - This is the similarity value associated with the match.

Some records will not produce matches to any other records. In order to record that these records have been processed but that they have not produced any matches, lines where both the parent and child ids are the same can be included. In these cases the record will be recorded as having been processed, but the value information will not actually be read in.

5.3 Single match file

Single match files are the final output of the fuzzymatch process. These files are subsequently read in by the partner widget code to be presented on the collection websites. The files are effectively trimmed down and rearranged versions of the record file and follow most of the same rules as described in section 5.1.

The differences are are follows:

- Each file now contains a single `record` tag although each `record` can contain multiple `match` tags.

- `match` tags follow the same structure as the `record` tags but additional attributes are included, these are the `institution` and `similarity` attributes.

- `institution` attributes contain a long form, human readable version of the collections specified in the `source` attribute.

- `similarity` attributes contain a 0.0 to 1.0 measure of the quality of that match.

- `match` tags are grouped inside `matches` tags depending on match type. Match type is specified using the `type` attribute, valid values are `balanced`, `title` and `person`.

- `match` tags are ordered from high to low according the values of the `similarity` attributes.

An example of a small, but fully formed and valid single match file is shown below.

```
<record id="123" published_id="aaa" source="erps"
        institution="Exhibitions of the Royal Photographic Society">
    <title>Example title</title>
    <person>Doe, J</person>
    <process>Picture</process>
    <url>http://www.somewebsite.co.uk/aaa</url>
    <startdate>1878</startdate>
    <enddate>1878</enddate>

  <matches type="balanced">
    <match id="456" published_id="bbb" source="erps"
           institution="Exhibitions of the Royal Photographic Society"
           similarity="1.0">
      <title>Example title</title>
      <person>Doe, J</person>
      <process>Picture</process>
      <url>http://www.somewebsite.co.uk/bbb</url>
      <startdate>1878</startdate>
      <enddate>1878</enddate>
    </match>
    <match id="789" published_id="ccc" source="orsay"
           institution="Musee d'Orsay" similarity="0.7">
      <title>Le Example Title</title>
      <person>Jane Doe</person>
      <process/>
      <url>http://www.culturegrid.org.uk/search/1553231.html</url>
      <startdate>1890</startdate>
      <enddate>1890</enddate>
      <translation>The Example Title</translation>
    </match>
  </matches>

  <matches type="title">
    <match id="456" published_id="bbb" source="erps"
           institution="Exhibitions of the Royal Photographic Society"
           similarity="1.0">
      <title>Example title</title>
      <person>Doe, J</person>
      <process>Picture</process>
      <url>http://www.somewebsite.co.uk/bbb</url>
      <startdate>1878</startdate>
      <enddate>1878</enddate>
    </match>
  </matches>

  <matches type="person">
    <match id="789" published_id="ccc" source="orsay"
           institution="Musee d'Orsay" similarity="0.7">
      <title>Le Example Title</title>
      <person>Jane Doe</person>
      <process/>
      <url>http://www.somewebsite.co.uk/ccc</url>
      <startdate>1890</startdate>
      <enddate>1890</enddate>
      <translation>The Example Title</translation>
    </match>
  </matches>
</record>
```

5.4 Word similarity files

The title similarity metric caches word similarity values to dramatically improve the performance of subsequent runs. This cached information is stored, between runs, in two files called words.xml and words.dat in the file hierarchy shown previously.
The fuzzymatch project will refuse to run without these two files being present and will update the files as necessary when new words are encountered. Producing the word similarity values is very time consuming, taking several days, as such it is very important that these files are backed up properly.

5.5 Processes configuration

The process configuration is an XML structured file describing a hierarchy of process relationships. Nodes in the hierarchy can have more than one parent (which would technically make it a graph and not a hierarchy) but this should be avoided when possible.

A simple example configuration file can be seen below.

```
<processes>
```

```
  <color>color</color>
  <color>colour</color>
  <monochrome>monochrome</monochrome>
  <positive>positive</positive>
  <negative>negative</negative>

  <process name="image"/>
  <process name="paper">
    <keywords>print</keywords>
    <keywords>paper</keywords>
    <parent>image</parent>
  </process>
  <process name="glass">
    <keywords>glass</keywords>
    <parent>image</parent>
  </process>
  <process name="salted paper" color="no" image="both">
    <keywords>salted</keywords>
    <keywords>silver chloride</keywords>
    <parent>paper</parent>
  </process>
</processes>
```

The process metric takes three factors into account, the process itself, whether the image is colour or monochrome and whether it is a positive or negative image. To aid this, identifying keywords associated with those factors can be specified using the `color`, `monochrome`, `positive` and `negative` tags.
Processes are defined using the `process` tag which has a required `name` attribute containing a unique name. Optional attributes are `color` and `image`. Certain photographic processes are always colour/monochrome or produce positive/negative images. The default assumption for each process can be set using these attributes. Valid values for both attributes are

Contained within `process` tags are `parent` tags containing the name of a process defined elsewhere in the configuration file and `keywords` tags.
`keywords` tags define the terms associated with that process, the same keywords can be used by multiple processes but this should be avoided as much as possible. If more than one term is supplied inside a set of `keywords` tags then those terms will only match if both terms are present, however the term order is not taken into account.
The process metric using Jaro-Winkler to perform approximate string match-

ing to the `keywords` terms. As such matches are still possible in the event of misspellings in the record data. However, in order to achieve the best results, common misspellings or regional variations (i.e. colour and color) can be included.

Chapter 6

Process hierarchy

6.1 Visualisation

In these diagrams rectangles represent keyword groupings associated with a process, while ellipses, octagons and hexagons represent processes. Ellipses are processes that produce positive images, octagons produce negative images and hexagons can produce both. Green shapes represent processes that produce colour images, red produce monochrome images and blue can produce both.

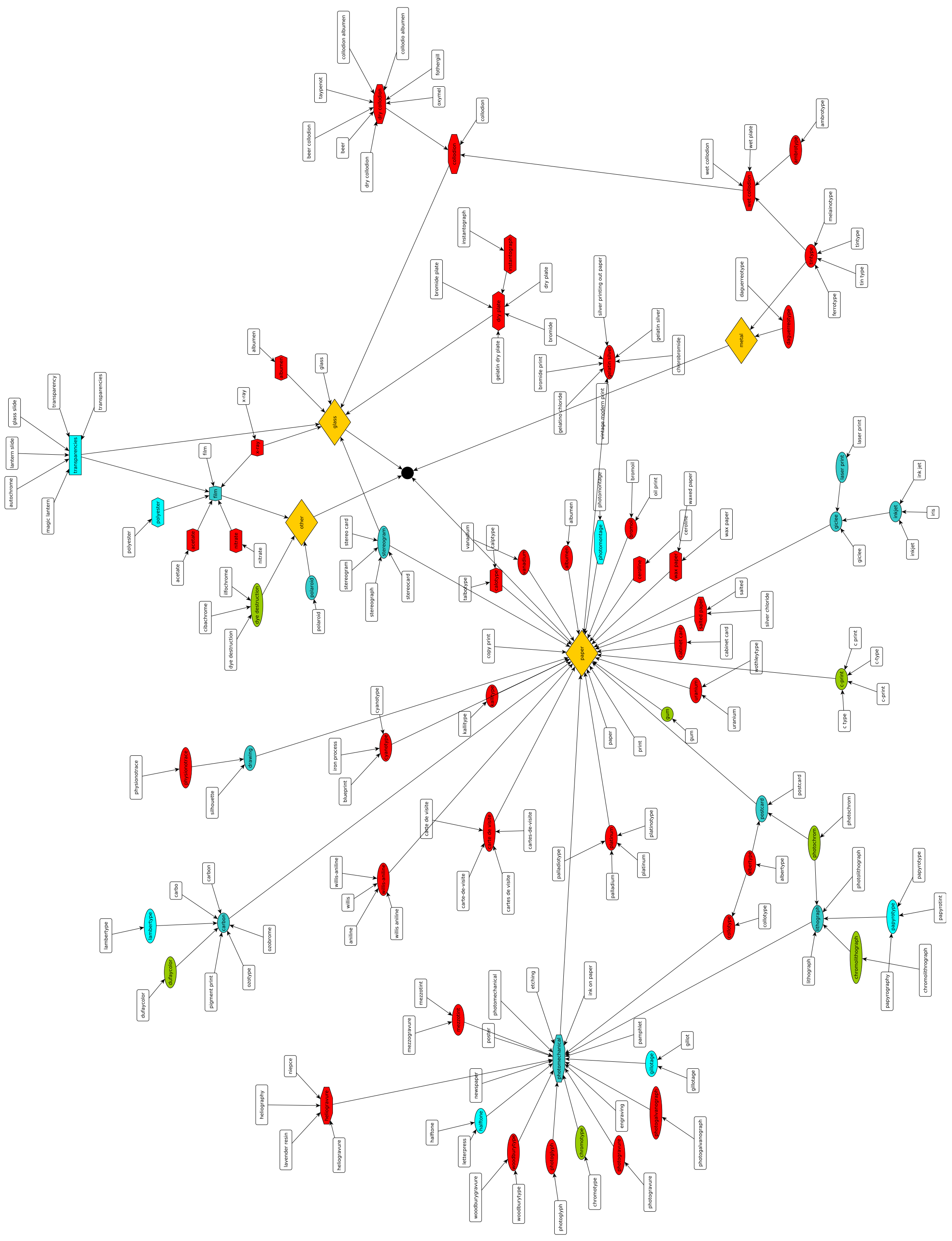

Figure 6.1: Complete process ontology.