2010 Sixth IEEE International Conference on e–Science

SubSift web services and workflows for profiling and comparing scientists and their published works

Simon Price*†, Peter A. Flach†, Sebastian Spiegler†, Christopher Bailey* and Nikki Rogers*
*Institute for Learning and Research Technology, and †Intelligent Systems Laboratory
University of Bristol
Bristol, UK
Email: simon.price@bristol.ac.uk

***Abstract*—Scientific researchers, laboratories and organisations can be profiled and compared by analysing their published works, including documents ranging from academic papers to web sites, blog posts and Twitter feeds. This paper describes how the vector space model from information retrieval, more normally associated with full text search, has been employed in the open source SubSift software to support workflows to profile and compare such collections of documents. SubSift was originally designed to match submitted conference or journal papers to potential peer reviewers based on the similarity between the paper's abstract and the reviewer's publications as found in online bibliographic databases. The software is implemented as a family of RESTful web services that, composed into a re-usable workflow, have already been used to support several major data mining conferences. Alternative workflows and service compositions are now enabling other interesting applications.**

I. Introduction

SubSift is a family of RESTful web services for profiling and matching text. Such services have numerous applications within e-Science and e-Research, both in terms of supporting administration of the scientific process itself and as a flexible processing component within data mining, machine learning and pattern analysis research. SubSift's URI design and built-in RDF support automatically connect its shared data to the Linked Data graph and wider Semantic Web.

SubSift exploits the intuitive idea that the published works of scientific researchers, laboratories and organisations, in some sense, describe their specific research interests and expertise. Furthermore, by analysing these published works in relation to the body as a whole, discriminating profiles may be produced that effectively characterise heterogeneous documents ranging from traditional academic papers to web sites, blog posts and Twitter feeds. Such profiles have applications in their own right but can also be used to compare one body of documents to another, ranking arbitrary combinations of documents and, by proxy, individuals or groups by their similarity to each other.

SubSift, short for *submission sifting*, was originally designed to match submitted conference or journal papers to potential peer reviewers based on the similarity between the paper's abstract and the reviewer's publications as found in online bibliographic databases. In this setting the software has already been used to support several major data mining conferences. The origin of the SubSift tools to support the research paper review process of the *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2009* (SIGKDD'09) and its subsequent evolution into RESTful web services[1] is documented elsewhere [1], [2]. In this paper we present this submission sifting use case as just one amongst a number of other re-usable workflows.

SubSift workflows have all been derived from the submission sifting design pattern shown in Figure 1. Workflows in this paper extend this pattern at the input, e.g. harvesting documents from the web, or at the output, e.g. transforming similarity data into reports and graphs.

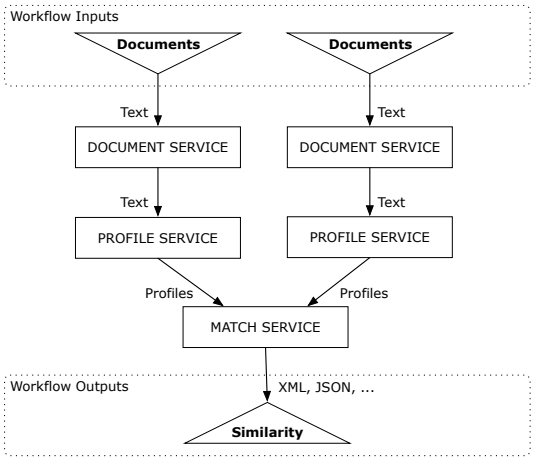

Figure 1. Design pattern of original SubSift workflow.

SubSift is a freely available online application hosted by the University of Bristol. The software has also been released as open source on Google Code. Further details can be found on the SubSift website[2] along with extensive documentation and a growing number of demonstrations.

[1]Funded by the UK Joint Information Services Council (JISC) under the Rapid Innovation strand of the Information Environment Programme 2009-11

[2]SubSift website – http://subsift.ilrt.bris.ac.uk

978-0-7695-4290-4/10 $26.00 © 2010 IEEE
DOI 10.1109/eScience.2010.29

The remainder of this paper begins with a review of related work, followed by a recap of relevant background topics, an overview of the SubSift REST API, and a detailed look at a number of re-usable workflows, before rounding up with future work and conclusions.

II. Related Work

SubSift's functionality is achieved through the application of techniques from information retrieval that are more normally associated with *full text search* rather than profiling or matching entire collections of documents [3]. Established full text search tools such as Apache Lucene[3], recent versions of PostgreSQL[4] and Oracle UltraSearch[5] all support text matching on large-scale document collections. Although these systems were all designed to compare a single text query against a document collection, there is no reason why a full Cartesian product of one document collection against another document collection cannot be performed to return pairwise document similarity in the same way that SubSift does. Indeed, recently a paper duplicate detection system using Lucene [4] has been developed to do just that. However, implementing SubSift as a bespoke application has a number of advantages over full text search tools, including the following.

- Detailed metadata about factors contributing to term importance and similarity are available throughout.
- SubSift is, in part, research and so the ability to easily incorporate and instrument new algorithms is vital.
- Full control over the core algorithm and storage model is important for migration to an HPC setting.
- The open source release has fewer licensing and software dependencies by avoiding a text search engine.

As far as we are aware, the submission sifting use case for SubSift is a novel application of document similarity methods. By contrast, the other use cases covered by the workflows in this paper have been the subject of prior work. Clustering data using similarity and distances is an active research topic in information retrieval and data mining [5]–[7] and has been applied before in e-Science and e-Research – for instance, a recent system automatically clusters search results according to the similarity of document content [8]. However, SubSift contributes by exposing its functionality through web services rather than embedding it within a single application or experimental framework.

SubSift's RESTful services and the workflows presented in this paper are compatible with workflow management systems such as Kepler, Taverna and YAWL [9]–[11]. Compatibility would be further increased by the creation of a WSDL 2.0 description of SubSift's REST API which would add SOAP support to simplify integration into existing workflow design and enactment tools [12], [13].

[3]Lucene – http://lucene.apache.org

[4]PostgreSQL – http://www.postgresql.org

[5]UltraSearch – http://www.oracle.com/technology/products/ultrasearch/

III. Background

The theoretical basis for profiling and matching functionality of SubSift is the well known *vector space model* from information retrieval [3]. SubSift software also draws on the *representational state transfer* (REST) design pattern for web services [14]. In this section we give a brief overview of each of these topics. Readers already familiar with them can safely skip to the next section.

A. Vector Space Model

The canonical task in information retrieval is, given a query in the form of a list of words (terms), rank a set of text documents D in order of their similarity to the query. The vector space model is a common approach to solving this problem. Each document $d \in D$ is represented as the multiset of terms (bag-of-words) occurring in that document. The set of distinct terms in D, vocabulary V, defines a vector space with dimensionality $|V|$ and thus each document d is represented as a vector $\vec{d}$ in this space. The query q can also be represented as a vector $\vec{q}$ in this space, assuming it shares vocabulary V. The query and a document are considered similar if the angle θ between their vectors is small. The angle can be conveniently captured by its cosine, which is equal to the dot product of the vectors scaled to unit length, giving rise to the *cosine similarity*, s.

$$s(\vec{q}, \vec{d}) = \cos(\theta) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||}$$

However, if raw term counts are used in vectors $\vec{q}$ and $\vec{d}$ then similarity will be biased in favour of long documents and will treat all terms as equally important. The *term frequency – inverse document frequency* (tf-idf) weighting scheme compensates for this by normalising term counts within a document by the total number of terms in that document, and by penalising terms which occur in many documents. More formally, *term frequency* tf_{ij} of term t_i in the document d_j, and *inverse document frequency* idf_i of term t_i are defined as

$$\mathrm{tf}_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}, \quad \mathrm{idf}_i = \log_2 \left(\frac{|D|}{\mathrm{df}_i} \right), \quad \text{tf-idf}_{ij} = \mathrm{tf}_{ij} \times \mathrm{idf}_j$$

where *term count* n_{ij} is the number of times term t_i occurs in the document d_j, and *document frequency* df_i of term t_i is the number of documents in D in which term t_i occurs.

In SubSift, instead of comparing a single query against a set of documents, we pairwise compare every document in one collection D_1 (e.g. abstracts) with every document in another collection D_2 (e.g. reviewer bibliographies) to produce a ranked list for each document. To capture the overall importance of each term across the combined collections, df_i, and hence tf-idf_i, values are calculated over the union of both collections, $D_1 \cup D_2$.

B. RESTful Web Services

REST is an easily understood design pattern for web services, based around the ubiquitous HTTP protocol and its familiar vocabulary of URIs, media types, requests and responses [14]. Recently, REST web services have become popular as the web API behind numerous Web 2.0 sites, including Twitter, Flickr and Facebook and are also widely used in e-Science and e-Research. Like conventional websites, *RESTful* services offer a stateless, cacheable, layered and uniform client-server interface. However, unlike conventional sites, which are designed to render human-readable data as HTML pages to be viewed in a browser, RESTful sites serve data in formats such as XML, JSON and RDF, that may be readily consumed by arbitrary applications. Furthermore, in the same way that HTML forms on conventional web pages can be used to submit data from the client to the server for storage and processing, arbitrary RESTful applications can use exactly the same protocols to achieve the same end. Also, the usual HTTP authentication and authorisation mechanisms can be used to control access to specific services and resources.

The intuition behind REST is that URIs are used to represent resources and that HTTP request methods are used to specify fundamental operations on those resources. The most widely used HTTP request method, `GET`, is invoked every time a web browser requests a URI from a web server. HTTP `GET` is only one of several HTTP request methods; the five that are most significant for REST web services are summarised in Table I. These correspond to the *CRUD(E)* (Create, Replace, Update, Delete and Exists) operations from Web 2.0. Additional operations are specified by adding verbs into the URIs themselves. Pairs of attribute-value parameters can be supplied with the HTTP request to modify the behaviour of operations.

The following is an example of a REST request, using `curl`[6] as a convenient command line client, to retrieve a (usually XML) representation of a book by its ISBN.

```
curl -X GET http://www.foo.com/books/0471941522
```

A similar request to the same URI using the `DELETE` method will delete that same representation.

```
curl -X DELETE http://www.foo.com/books/0471941522
```

As the second example makes clear, authentication and authorisation are sometimes required for REST operations. All the usual HTTP techniques can be used for this purpose.

IV. SubSift Web Services

The decision to package and host the SubSift software as web services was easily made. Apart from the advantage of prospective users being able to use the software without having to install it, the web services model has a number of other benefits, including the ability for users to: make their own application-specific customisations using the services with mashup tools like Yahoo Pipes; select other data sources, such as Citeseer, Google Scholar, eprints, news and blogs; and, importantly, integrate the software with their own research tools without having to modify code.

Table I
Interpretation of HTTP Request Methods in REST

Method	Usage in REST	Changes Resource
`GET`	*show* and *list* operations	no
`HEAD`	*exists* operations (does resource exist)	no
`POST`	*create* and *compute* operations	yes
`PUT`	*update* and *recompute* operations	yes
`DELETE`	*destroy* operations	yes

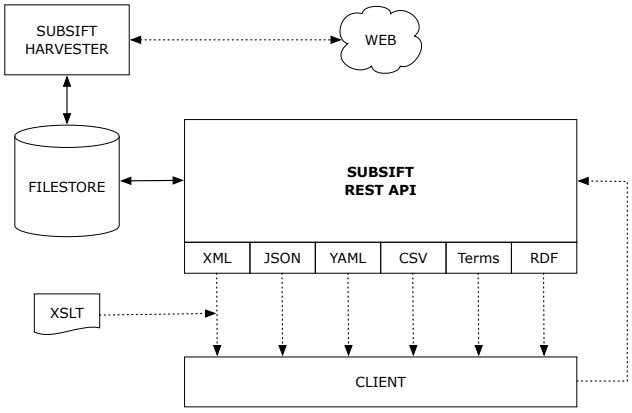

Figure 2. SubSift System Architecture. Dotted lines represent HTTP. Solid lines represent file access. XML, JSON, YAML, CSV, Terms and RDF are response formats. Terms are Prolog terms. XSLT stylesheets are invoked via optional inclusion of XSL processing command in XML returned by SubSift.

The choice of RESTful web services over SOAP-based web services was more difficult. Popular e-Science workflow tools, such myExperiment [15] and Taverna [10] support both REST and SOAP to varying degrees. However, SOAP is well established in e-Science, as evidenced by the BioCatalogue[7] life sciences web services directory which, at the time of writing, has 1634 SOAP services as compared to only 61 REST services. Even so, the compelling argument for implementing SubSift as lightweight RESTful services was our strong use cases for Web 2.0 applications and mashups.

Figure 2 shows the high-level design of the SubSift system, with its REST API and supporting web harvester robot. Details of the API itself are given in the next section and further details appear in the workflow descriptions below.

[6]curl – http://curl.haxx.se

[7]BioCatalogue website – http://www.biocatalogue.org/services

V. SUBSIFT REST API

The SubSift REST API is organised around a series of *folders* into which data *items* are stored. This organisation is modelled on the familiar filing system concept of folders and files. The three main folder types are documents, profiles and matches. The first request below would create a documents folder called `staff` and the second would list its items[8].

```
curl -X POST <uri>/<user_id>/documents/staff
curl -X GET <uri>/<user_id>/documents/staff/items
```

In SubSift, a document is a piece of text to be profiled and matched. A document will usually be the text from some external source such as the text of a web page or a conference paper abstract. A profile is a summary representation of the features of a single document, with respect to the other documents in the same documents folder. In the example fragment of a profile item below, the person's names are the most discriminating terms, occurring 77 times in the source DBLP author webpage[9] and having a tf-idf of 0.404 (last name) and 0.246 (first name) relative to the rest of the items in this profiles folder. The terms *logic* and *ILP* also rank highly.

```
profile:
  name: Peter Flach
  source: http://dblp.../Flach:Peter_A=.html
  term:
    - name: flach
      idf: 6.047
      n: 77
      tf: 0.067
      tfidf: 0.404
    - name: peter
      idf: 3.685
      n: 77
      tf: 0.067
      tfidf: 0.246
    - name: logic
      idf: 3.924
      n: 18
      tf: 0.016
      tfidf: 0.061
    - name: ilp
      idf: 4.685
      n: 14
      tf: 0.012
      tfidf: 0.057
```

One usage of profiles in SubSift is to obtain a list of distinguishing terms, or keywords, for a document – for example, automatically extracting keywords from abstracts of papers submitted to a conference. Another usage is for two profile folders to be compared against each other to produce a matches folder. A matches folder is created by analysing every pairing of profile items drawn from the two profiles folders. Each match item records the tf-idf cosine similarity, and various related statistics, of a single profile

[8]We omit the security token needed in the request header of `DELETE`, `POST` and `PUT` requests, and of all requests for folders marked as private.

[9]DBLP Computer Science Bibliography – http://dblp.uni-trier.de/

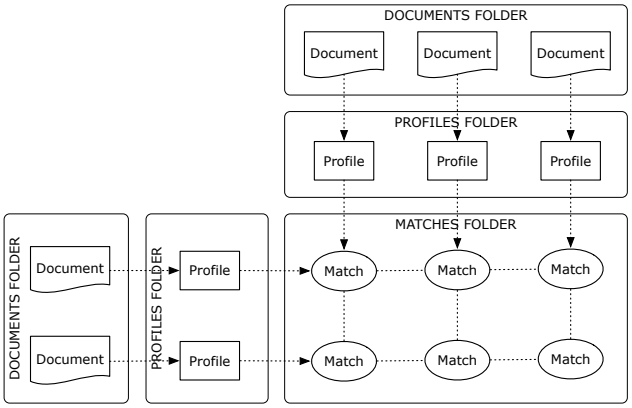

Figure 3. A SubSift REST API controlled sequence of transformations from a pair of document folders (e.g. abstracts versus reviewers) through to a folder of matching statistics. Original and computed data plus metadata can be obtained at each step in the process via API methods.

from the first profiles folder against every profile from the second profiles folder. A typical usage of such a comparison is to match submitted conference paper abstracts with the bibliography pages of Programme Committee (PC) members (i.e. reviewers) in order to rank potential reviewers for each paper and visa versa. This usage is depicted in Figure 3.

SubSift's API methods make intermediate data and metadata available for all folders and items. For example, the entire similarity matrix of a match folder can be exported or the relative contribution of each term towards a particular match item's similarity can be retrieved. In the match item fragment below, metadata about the similarity score of 0.076, between a person and a paper, shows that the terms *logic* and *ILP* made large contributions to the score relative to other terms.

```
match:
  name: Peter Flach
  source: http://dblp.../Flach:Peter_A=.html
  item:
    - description: paper title removed
      name: paper id removed
      score: 0.076
      source: text
      term:
        - name: logic
          contribution: 0.005
        - name: ilp
          contribution: 0.004
        - name: class
          contribution: 0.001
        - name: inductive
          contribution: 0.001
```

For ease of integration, there is flexibility in both input and output. Document text may be added per item or in bulk or by supplying a list of URLs to be fetched asynchronously by SubSift's harvester robot. The API methods can return data in the following representational formats: CSV, JSON, RDF, XML, YAML, and Prolog terms.

VI. Demonstration Workflows

In the following sections we present a number of SubSift workflows as use cases illustrated by data flow diagrams. This presentation avoids details of specific workflow runtime environments and at this level of abstraction, the workflows are highly re-usable.

A. Workflow 1 – Submission Sifting

Peer review of written works is an essential pillar of the academic process, providing the central quality control and feedback mechanism for submissions to conferences, journals and funding bodies across a wide range of disciplines. However, from the perspective of a busy conference PC chair, journal editor or funding manager, identifying the most appropriate reviewer for a given submission is a non-trivial and time-consuming task. Effective assignment, first and foremost, requires a good match to be made between the subject of the submission and the corresponding expertise of reviewers drawn from a, sometimes large, pool of potential reviewers. In the case of conferences, a recent trend transfers much of this allocation work to the reviewers themselves, giving them access to the full range of submissions and asking them to bid on submissions they would like to review. Their bids are then used to inform the allocation decisions of the Programme Committee chair.

A SubSift demonstrator based on this workflow was implemented as a wizard-like series of web forms, taking the PC chair through the above process form by form. On the first form, a list of PC member names is entered. SubSift looks up these names on DBLP and suggests author pages which, after any required disambiguation, are used as documents to profile the PC members. Then the conference paper abstracts are uploaded as a CSV file and their text is used to profile the papers. After matching PC member profiles against paper profiles, SubSift produces reports with ranked lists of papers per reviewer, and ranked lists of reviewers per paper. Optionally, by manually specifying threshold similarity scores or by specifying absolute quantities, a CSV file can be downloaded with initial bid assignments for upload into the conference management tool.

The data flow details for this workflow are depicted in Figure 4. This is almost the generic workflow from Figure 1, apart from the inclusion of the Bookmarks Service to manage the list of URIs and SubSift's Harvester Robot to fetch the web pages. The HTML Generators use XSLT to transform XML into HTML reports.

This workflow was first used at SIGKDD'09 and quantitatively evaluated by comparing SubSift's initial bids against the revised final bids made by reviewers [1]. The results plus qualitative feedback from reviewers, were encouraging and demonstrated that there is considerable scope for automated support during the bidding process.

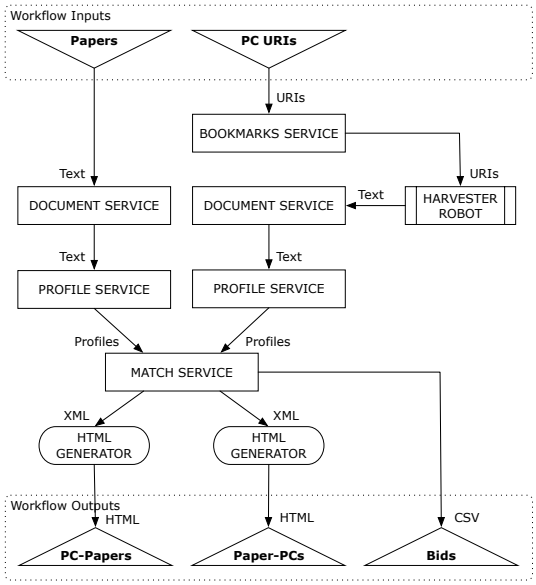

Figure 4. *Submission Sifting* workflow.

B. Workflow 2 – Finding an Expert

Finding an expert, within an organisation, on a given topic can be difficult if the person searching is not familiar with all members of that organisation or with the topic itself. This is often true for members of the media seeking an expert, within a university, on some newsworthy scientific report. It is also true for journal editors deciding which editorial board members to ask to review a particular paper.

In this workflow the user submits a fragment of text, typically an abstract or the full text of a paper, and SubSift compares this against the publications of a pre-defined list of researchers. The result is a list of researchers ranked by their similarity to the submitted fragment. Figure 5 shows *ILRT Matcher*, an entirely client-side, Web 2.0 JavaScript implementation of this workflow, comparing the submitted text against every ILRT staff members' homepage. The results are displayed as a bar chart. Optionally, the list of terms contributing to each researcher's similarity score can be viewed along with the percentage that each term contributed to the combined score, as shown in Figure 6.

This two-phase workflow is depicted in Figure 7. The first phase is preparatory and creates all the necessary folders and items for the second, interactive phase. The second phase is enacted each time a user submits text via the Web 2.0 form. In the first phase a null (empty) text fragment is used to form a singleton item in a documents folder to be profiled and compared against the staff web page profiles.

C. Workflow 3 – Visualising Similarity

Organisations usually have a formal hierarchical structure but there can also be hidden structures, both of which can be discovered by analysing the similarity of documents

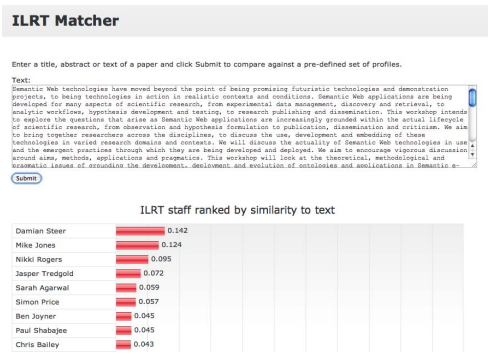

Figure 5. Similarity match between text fragment and ILRT staff.

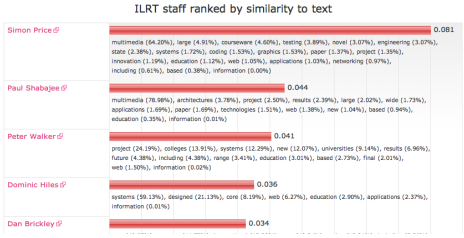

Figure 6. Term contributions for matches between text and ILRT staff.

associated with members of the organisation. The similarity between individuals can be represented as a directed labelled graph where the vertices are the individuals and the edges the similarity relation between two individuals. One strategy for making structure visible is to only add edges between nodes whose similarity score is above a threshold. Another is to only add the top n edges for each node, where n is a

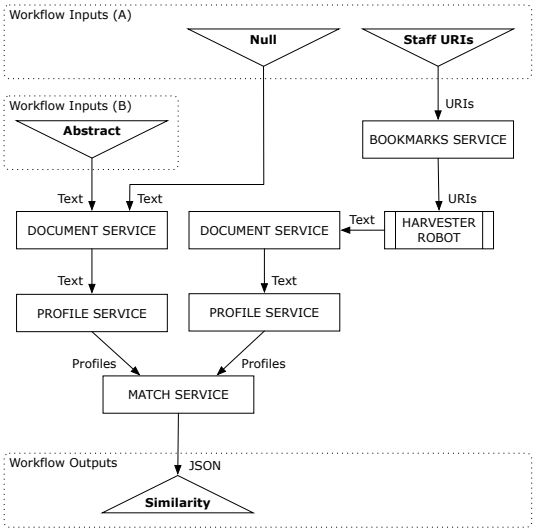

Figure 7. *Finding an Expert* workflow.

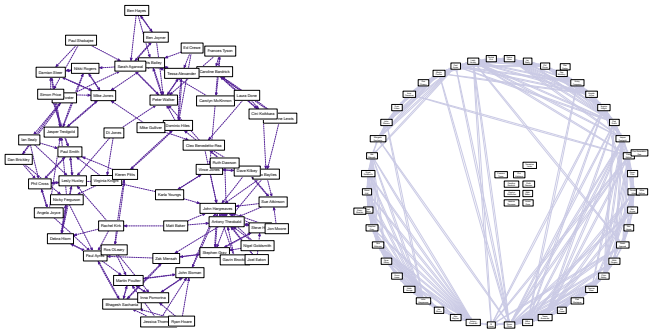

Figure 8. ILRT staff homepage similarity matches data output to DOT format and rendered in Graphviz (left: force-directed layout, right: circular).

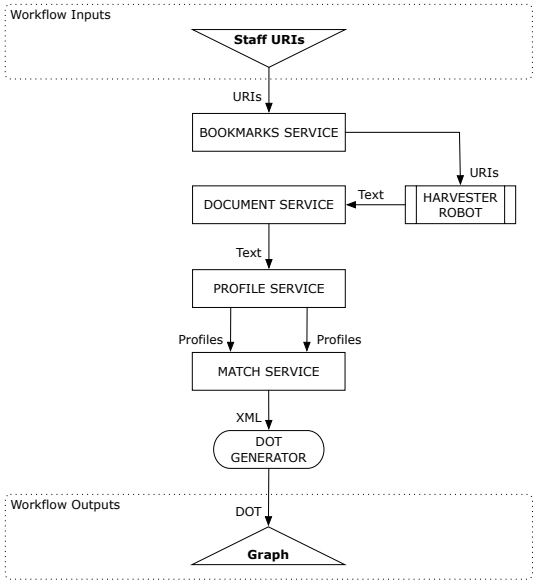

Figure 9. *Visualising Similarity* workflow.

small integer. Tools like Graphviz[10] can be used to visualise a textual representation (DOT format) of the graph as shown in Figure 8 for the homepages of ILRT staff members.

This workflow in Figure 9 compares a profiles folder against itself to rank all the documents profiled by their similarity to each other. Ignoring the reflexive matches (a profile is always identical to itself), the resultant similarity data returned by SubSift is then transformed by a SubSift XSLT file from XML into a textual DOT file suitable for rendering as an image in Graphviz.

D. Workflow 4 – Profiling Reading Lists

Many researchers share lists of bookmarks (URIs) to social bookmarking websites like Delicious[11]. As their interests change over time, so does the list of bookmarks. The bookmarks and the text of the web pages that they link to can

[10]Graphviz – http://www.graphviz.org
[11]Delicious social bookmarking – http://delicious.com

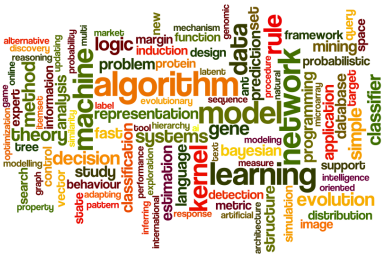

Figure 10. Tag cloud visualisation of researcher interests.

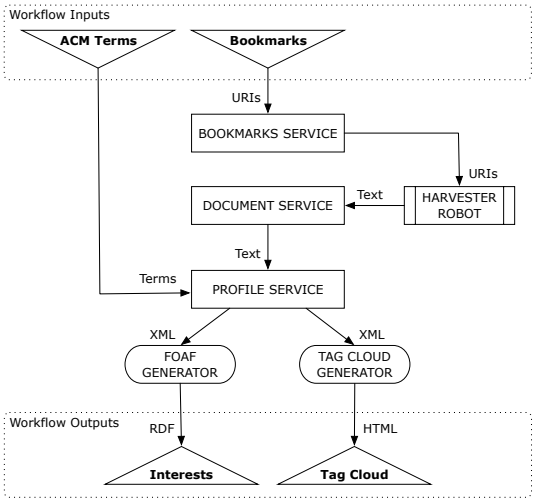

Figure 11. *Profiling Reading Lists* workflow.

be used to create a profile of a researcher's current interests. Such information is often displayed on homepages as a tag cloud, such as Figure 10 which was produced in Wordle[12] using SubSift profile terms. This information may also be published as Semantic Web data using formats like Friend of a Friend (FOAF).

A SubSift workflow to achieve this is shown in Figure 11. The bookmarks are harvested to create the researcher's web reading list profile. The researcher's bookmarks are not being compared to those of other researchers here and so the ranked list of terms produced will contain high scoring terms, such as the person's name, that are not relevant. To filter these out, the terms of the ACM Computing Classification System (CCS)[13] are used during profiling to ensure that all terms output from the Profile Service occur in the CCS list.

E. Workflow 5 – Ranking News Stories

A researcher's subscriptions to RSS blog and news feeds provide frequently updated textual information. However, reading RSS feeds can be time consuming and what may

[12]Wordle – http://wordle.net

[13]ACM CCS – http://www.acm.org/about/class/1998/

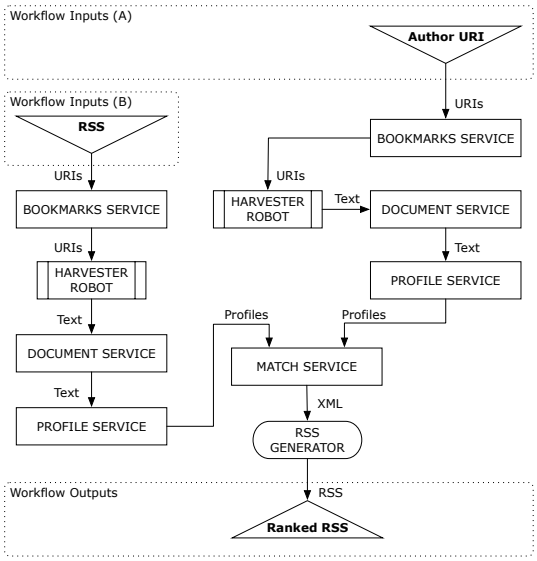

Figure 12. *Ranking News Stories* workflow.

initially appear relevant from the title may turn out not to be after following the associated link to the full text. The ability to rank stories in a feed by relevance to the research interests of the reader is a potential time saver.

Figure 12 shows a two-phase workflow based on an extension to the submission sifting workflow that can be used to reorder the stories in an RSS file according to their similarity to the researcher as defined by their own published works. In the first phase, the current homepage of the researcher is used to create their profile. This profile only needs recalculating periodically. In the second phase, an RSS feed is used as a list of URIs which are harvested and used to create a profile for each story. The story profiles are matched against the reviewer profile and the resultant ranking is used to reorder the original RSS file.

VII. Future Work

Use cases for existing SubSift workflows have emerged that require large-scale storage and High Performance Computing (HPC) capability – for instance, to profile and compare local document collections against all the documents in a sizeable online bibliography such as DBLP. As part of the University of Bristol's Exabyte Informatics research programme, parts of SubSift's functionality are being evaluated for migration to run on the University's BlueCrystal HPC facility and Petascale storage.

However, the workflows presented in this paper demonstrate practical applications of SubSift on small data sets with modest hardware. We anticipate that these workflows will be used largely unchanged in the HPC setting; only the internal behaviour of SubSift will differ. The development of these workflows with the current small-scale implementation has elicited valuable use cases from researchers and research

managers; it has also served to refine the API requirements for different runtime settings, such as invocation from Web 2.0 pages or interaction with tools such as Matlab.

Both small-scale and HPC branches of SubSift will contribute towards the development of ExaMiner, a prototype application for mining and mapping a university's research landscape, beginning initially with the University of Bristol but with the ultimate aim of being able to map any research-centric institution. Specifically, the ExaMiner software will: generate automatic profiles of researchers and research groups from university web pages; allow visual navigation of these profiles, as well as navigation by free-text query; provide social networking components, including the ability to store a personal profile and a network of interesting contacts, events and connections; notify a user of new resources in their sphere of interest; and improve performance over time by responding to user feedback.

Possible users include academics, research managers, current and prospective students, and the media. It is hoped that a system based on this prototype would improve the discoverability of research within the institution, the community, the media, prospective students, and the government.

VIII. Conclusion

SubSift's RESTful web services show potential for integration into workflows far beyond the original peer review use case where SubSift has already proved to be a popular addition to the existing conference paper reviewing process. We suggest that the workflows presented in this paper are promising e-Science and e-Research use cases for profiling and comparing scientists and their published works. We also forward SubSift as a flexible tool for constructing, manipulating and publishing document-centric datasets for information retrieval, data mining and pattern analysis research. The publication of research data in this way supports the reproducibility of scientific experiments, while automatically connecting data through Linked Data and the Semantic Web.

Acknowledgment

We would like to thank conference PC chairs Bart Goethals (SDM'10), Qiang Yang (KDD'10), Mohammed Zaki (PAKDD'10) and Geoff Webb (ICDM'10) for trialing SubSift; other developers of the original software, Bruno Golénia from the Intelligent Systems Lab in Bristol, John Guiver, Ralf Herbrich and Thore Graepel from Microsoft Research in Cambridge, and Mohammed Zaki from Rensselaer Polytechnic Institute.

References

[1] P. A. Flach, S. Spiegler, B. Golénia, S. Price, J. G. Ralf, H. T. Graepel, and M. J. Zaki, "Novel tools to streamline the conference review process: Experiences from SIGKDD'09," *SIGKDD Explorations*, vol. 11, no. 2, pp. 63–67, December 2009.

[2] S. Price, P. A. Flach, and S. Spiegler, "SubSift: a novel application of the vector space model to support the academic peer review process," in *Workshop on Applications of Pattern Analysis (WAPA 2010)*. Windsor, UK, September 2010.

[3] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[4] Y. Ding, K. Yi, and R. Xiang, "Design of paper duplicate detection system based on lucene," *Wearable Computing Systems, Asia-Pacific Conference on*, pp. 36–39, 2010.

[5] M. E. J. Newman, "The structure of scientific collaboration networks," *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, January 2001.

[6] T. Gaertner, J. W. Lloyd, and P. A. Flach, "Kernels and distances for structured data," *Machine Learning*, vol. 57, no. 3, pp. 205–232, December 2004.

[7] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004.

[8] S. Ananiadou, P. Thompson, J. Thomas, T. Mu, S. Oliver, M. Rickinson, Y. Sasaki, D. Weissenbacher, and J. McNaught, "Supporting the education evidence portal via text mining," *Philosophical Transcations of the Royal Society A*, vol. 368, no. 1925, pp. 3829–3844, 2010.

[9] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on.*, 2004, pp. 423–424.

[10] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services." *Nucleic Acids Research*, vol. 34, no. Web Server issue, pp. 729–732, July 2006.

[11] W. van der Aalst and A. ter Hofstede, "YAWL: Yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, 2005.

[12] J. Lathem, K. Gomadam, and A. P. Sheth, "SA-REST and (S)mashups: Adding semantics to RESTful services," *International Conference on Semantic Computing*, pp. 469–476, 2007.

[13] A. P. Sheth, K. Gomadam, and J. Lathem, "SA-REST: Semantically interoperable and easier-to-use services and mashups," *IEEE Internet Computing*, vol. 11, pp. 91–94, 2007.

[14] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology*, vol. 2, pp. 115–150, May 2002.

[15] D. De Roure, C. Goble, and R. Stevens, "The design and realisation of the virtual research environment for social sharing of workflows," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 561–567, 2009.