# A Web Service for Author Name Disambiguation in Scholarly Databases

Kunho Kim*, Athar Sefid*, Bruce A. Weinberg‡, and C. Lee Giles*†
*Computer Science and Engineering, †Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16801, USA
‡Department of Economics
Ohio State University, Columbus, OH 43210, USA
kunho@cse.psu.edu, azs5955@psu.edu, weinberg.27@osu.edu, giles@ist.psu.edu

***Abstract*—Author Name Disambiguation (AND) is the task of clustering unique author names from publication records in scholarly or related databases. Although AND has been extensively studied and has served as an important preprocessing step for several tasks (e.g. calculating bibliometrics and scientometrics for authors), there are few publicly available tools for disambiguation in large-scale scholarly databases. Furthermore, most of the disambiguated data is embedded within the search engines of the scholarly databases, and existing application programming interfaces (APIs) have limited features and are often unavailable for users for various reasons. This makes it difficult for researchers and developers to use the data for various applications (e.g. author search) or research. Here, we design a novel, web-based, RESTful API for searching disambiguated authors, using the PubMed database as a sample application. We offer two type of queries, attribute-based queries and record-based queries which serve different purposes. Attribute-based queries retrieve authors with the attributes available in the database. We study different search engines to find the most appropriate one for processing attribute-based queries. Record-based queries retrieve authors that are most likely to have written a query publication provided by a user. To accelerate record-based queries, we develop a novel algorithm that has a fast record-to-cluster match. We show that our algorithm can accelerate the query by a factor of 4.01 compared to a baseline naive approach.**

***Index Terms*—Web services, search, PubMed, author name disambiguation**

## I. Introduction

Scholarly databases usually consist of publication records from several data sources. For example, PubMed[1] and Web of Science[2] records are gathered from several publishers and venues. CiteSeerX[3] automatically gathers publicly available scientific papers from the web. Since data representation can vary across sources, unique identifiers are needed to identify the same entities. For example, an author named “Jane Doe” may appear with her full name “Jane Doe” in one publication, and with her first initial and last name “J. Doe” in another. For many reasons, identifying unique author entities is important for many problems, such as handling author-related queries and calculating bibliometric and scientometric measures for authors.

There are at least two different ways to identify unique authors. One approach is to generate a system that assigns an unique ID for each researcher and to encourage others to register and identify their publications. ORCID[4] is an example of this approach. The advantage of such an approach is that it can maintain high-quality disambiguated results if the author ID exists. The disadvantage is low completeness because researchers must enter their publications manually. Currently there are 4.3M ORCID IDs, but only 1.7M IDs have populated records (publications, affiliations, e-mail addresses, etc.). Another approach is to automatically identify unique authors among publication records, using machine learning classifiers to determine authorship. This approach is known as Author Name Disambiguation (AND), and many search engines of scholarly databases use it in their author search feature.

Although AND for large-scale scholarly databases has been extensively studied recently [7], [8], [10], [11], [13], [23], [24], only an early version of the CiteSeerX[5] and AMiner[6] disambiguation code is publicly available. Several scholarly databases provide a search module for disambiguated authors in their search engines, but few of them have publicly available APIs to allow direct queries without their search interface. Furthermore, existing databases allow only simple queries (e.g. querying by name), so the user can retrieve only limited information. For example, since PubMed does not provide author name disambiguation in their raw data, researchers use an outdated result of the Author-ity data, which was built around 2009 by Torvik and Smalheiser, with an update still in progress [23]. While the data are available, Author-ity Exporter[7] does not provide any APIs which would make it easier to acquire and use the disambiguated data.

For these reasons, access to disambiguated authors for all scholarly databases is limited. This makes it difficult for developers to use disambiguated author information in their products, and is equally challenging for researchers who wish to use disambiguated results in their research. To address this

[1] http://ncbi.nlm.nih.gov/pubmed
[2] http://webofknowledge.com
[3] http://citeseerx.ist.psu.edu
[4] https://orcid.org/
[5] https://github.com/SeerLabs/CiteSeerX
[6] https://github.com/askerlee/namedis
[7] http://abel.lis.illinois.edu/cgi-bin/exporter/search.pl

978-1-5386-7247-1/18/$31.00 ©2018 IEEE
DOI 10.1109/ICWS.2018.00041

IEEE computer society

problem, we propose a novel web service that provides a web-based, RESTful API for searching disambiguated authors.

The main contribution of this paper is to provide two types of author queries which serve different purposes: one is attribute-based and the other is record-based. Attribute-based queries use an internal resource (attribute) to query authors, which is supported by indexing records with attributes. An example is querying an author with the name ”Jane Doe”. For record-based queries, users provide their own resource to query authors. An example is to find author and publication records from a dissertation record that does not exist in the database. We discuss how to accelerate record-based queries using our proposed *record-to-cluster pairwise classification*.

Our service with proposed queries has several use cases. First, policy makers are eager to understand how scientific research impacts technological progress. By querying for a patent, the record-based query will allow people to determine whether a patent had a scientist as an inventor and view that scientists' scholarly research [1]. Second, universities are eager to understand how the training they provide translates into subsequent research. The record-based query will make it possible to enter a dissertation title and the identify the subsequent articles that a degree recipient has published. Third, the relationship between age and scientific productivity is a classic question in the science of science, one that is increasingly important as our scientific workforce ages. However, it has typically been studied using small-scale hand-curated datasets [6], [12], [21], [28]. Disambiguated data allows researchers to impute career age for population-scale data and estimate how productivity varies over a career at scale [17], [22]. Finally, the new UMETRICS project provides detailed data on the teams employed on research projects. The ability to link people to the articles they publish will allow researchers to determine how the size and composition of teams is related to the quantity and quality of the research that they produce [27].

In this paper, we particularly use the PubMed database to test and deploy our web service. We choose this database because the raw data is publicly available from the National Library of Medicine's website[8]. However, our service is not limited to the PubMed, and its architecture can be easily adapted to other scholarly databases. Our web service is publicly online[9]. We also made a PubMed author search engine named PubMedseer[10] as a sample application demonstrating our proposed web service.

The rest of the paper is organized as follows. Section II discusses related work on our author search web service. Section III discusses the API design of our web service. Section IV offers an overview of our architecture of the web service, highlighting each part in detail. Section V discusses how we processed the query requests. Section VI explains the experiments we performed to evaluate the web service. Section VII concludes and offers directions for future work.

## II. Related Work

AND has been studied in the context of various large-scale scholarly databases [5], including PubMed [13], [23], [24], CiteSeerX [7], [8], Web of Science [11], and USPTO patent databases [10]. Although several scholarly databases provide a search module for disambiguated authors with their associated search engines, there exist few services to directly query disambiguated authors without their search module. DBLP[11] and Semantic Scholar[12] offer an API to query authors, but users have to go through several steps to obtain the desired result. DBLP returns the author ID, but the user needs to perform the query again with the ID to obtain the list of publications and only through their search engine. Semantic Scholar offers an API to query with author ID, which the user needs to obtain from querying the specific papers. ArXiv[13] and PubMed offer API for querying publications records, but querying authors is not specifically provided and their author records are not disambiguated. For PubMed, Torvik and Smalheiser [23] developed a search tool, Author-ity Exporter, to query their disambiguated authors, but their results are from 2009 and are still being updated. AMiner and Microsoft Academic[14] have APIs to query authors with some attributes (e.g. name, affiliation, language, etc.). AMiner returns only the basic information on authors, and Microsoft Academic is a commercial API and provides only limited access to free users. Moreover, Google Scholar[15] does not provide any APIs from their service.

These existing services are generally intended for a specific scholarly database. Also the APIs are mostly provided by their backend search engine, so their use is not intuitive for users. They provide only basic query APIs, which feature querying with names and only a few attributes. In this work, we propose a web service that is highly modularized, and can be easily adopted to any scholarly database. Since each module is independent each other, one can easily modify and use more appropriate algorithm to adopt to support queries better for other scholarly database. For example, one can use another AND algorithm to disambiguate the authors, and then use the remaining part for supporting queries.

We provide two types of queries to satisfy different users' needs. One is an attribute-based query which allows users to query using attributes of each author record, similar to other services currently provided. Another is a record-based query, which allows users to query using a specific publication record from any database. The task is similar to the online disambiguation [8], [19], which assigns a new record to existing authors.

Some web services and tools are available to handle specific tasks on scholarly databases. CiteSeerExtactor [25] provides a web service to extract metadata from headers and citations.

[8] https://www.nlm.nih.gov/databases/download/pubmed_medline.html

[9] http://heisenberg.ist.psu.edu:5000

[10] http://pubmedseer.ist.psu.edu:5000

[11] http://dblp.uni-trier.de/

[12] https://www.semanticscholar.org/

[13] https://arxiv.org/

[14] http://academic.research.microsoft.com/

[15] https://scholar.google.com/

TABLE I
LIST OF AVAILABLE RESTFUL APIS

| API Method | URL | Attributes | Description | Returns |
|---|---|---|---|---|
| POST | / | file=filename | Upload a PDF document or BibTeX to query | resource_ID |
| GET | /attribute/ | attribute_name=value | Query with attribute(s). If attribute_name is not specified, it queries with the author full name. Multiple pairs of attributes and values (concatenated with &) can be used for query. | Query results (JSON) |
| GET | /resource_ID/ | order=n | Query with the uploaded record. Must specify the author position to select which author to query. If no position is specified, the first author is queried. | Query results (JSON) |
| DELETE | /resource_ID | | Delete the resource specified. | Success / Error code |

Petinot et al. [18] discuss the web API services provided in CiteSeerX. Shen et al. [20] developed a tool to visualize similarities of ambiguous names so as to interactively disambiguate author names.

## III. API DESIGN

Recall that our API supports two different types of queries for the disambiguated authors: attribute-based and record-based. The key difference is what resource is used to query the disambiguated author data. An attribute-based query uses internal resources (indexed attributes), while a record-based query uses external resources (e.g. a publication record) provided by a user. Managing both internal and external resources is important in our system. In particular, external resources should be efficiently and securely stored during the query, and it is important to ensure that they are deleted properly after usage. For efficient resource management, we design our APIs as RESTful APIs. RESTful APIs are known to be light-weight, scalable and easily accessible [26]. RESTful APIs consist of four types of HTTP requests, GET, PUT, POST, and DELETE. GET is used for lookup requests, POST for resource creation, PUT for mutation, and DELETE for deletion.

Table I shows a list of web APIs provided with our web servers. Attribute-based queries use a single GET API with the URL /attribute to query the record and return the result in JSON format. They query with the full name of the author as default, and also can query with attribute(s) including title, name, coauthors, affiliation, venue, and MeSH terms. In contrast, record-based queries involve the creation, usage, and deletion of user-provided resources, and so have POST, GET, and DELETE APIs respectively. Users provide the publication record to query with a POST API, and the web service parses the user input to obtain associated author records, and return a resource ID to identify it. The Resource ID is generated in a secure manner with a randomized string to keep it secure from other users and applications. Currently the query data can be provided by users in either PDF format or BibTeX format. Additional formats can be supported with associated parsers. The GET API retrieves disambiguated authors associated with the query record in JSON format. Users need to specify which author in the query record to query; the default is to query the first author. The DELETE API removes the user-provided resource after its usage, and returns a success code to confirm the deletion with the user. It is used for explicitly removing the user resource, our web server also tracks all resources with the time to live (TTL) to ensure that all resources are deleted after certain amount of time.

TABLE II
AUTHOR RECORD EXAMPLE

| Attribute | Value |
|---|---|
| PMID | 11032038 |
| Name | C. Lee Giles |
| Affiliation | NEC Research Institute, Princeton, NJ, USA |
| Title | Learning chaotic attractors by neural networks. |
| Abstract | An algorithm is introduced that trains a neural network $\cdots$ |
| Venue | Neural Computation |
| Volume | 12 |
| Issue | 10 |
| Pages | 2355-83 |
| Year | 2000 |
| Coauthors | R Bakker, J C Shouten, F Takens, C M van den Bleek |
| MeSH | Algorithms, Artificial Intelligence, Neural Networks $\cdots$ |
| Chemicals | (empty) |
| Grants | (empty) |

## IV. ARCHITECTURE DESIGN

In this section, we discuss the architecture of the proposed service and briefly discuss each module. Figure 1 shows an overview of our proposed system. We first need to disambiguate all authors in the scholarly databases offline. We store the disambiguation results in a SQL database and then index authors in a search engine to handle attribute-based queries. The record-to-cluster matching module handles record-based queries, by finding the appropriate disambiguated author who most likely wrote the record queried by the user. Users and other applications use the proposed RESTful API in Table I to make a request, and the API web server distributes those requests to the modules appropriately. We discuss each module further in the following subsections.

### A. Author Name Disambiguation

AND identifies unique authors from all author records in the scholarly databases [5]. Table II shows an example of an author record in PubMed. The record has several basic pieces of information on the author, such as name and affiliation, and also publication information. The goal of AND is to cluster all author records $R = \{r_1.r_2, \cdots, r_n\}$ and generate a list of

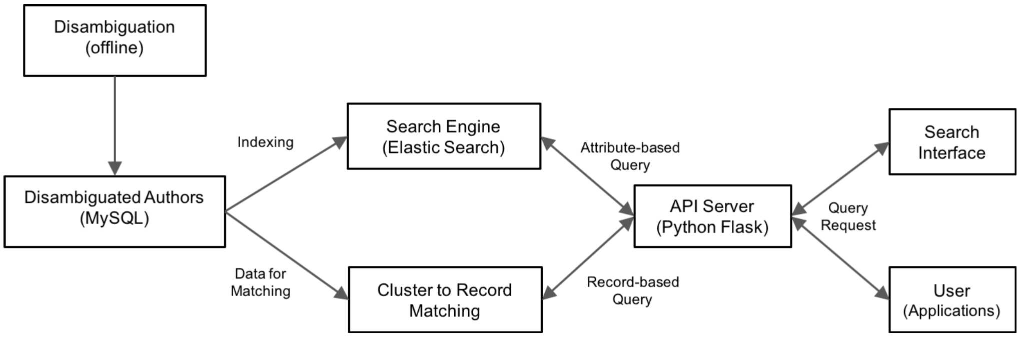


Fig. 1. Architecture Overview

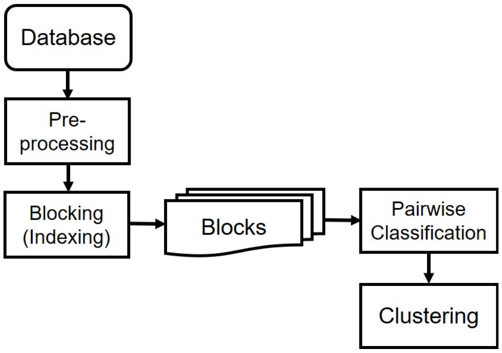


Fig. 2. Author Name Disambiguation Pipeline

TABLE III
FEATURES USED FOR PAIRWISE CLASSIFICATION FOR DISAMBIGUATION

| Attribute | Features |
|---|---|
| Title | Cosine (Bag-of-Words), Cosine (Word Embedding) |
| Abstract | Cosine (Bag-of-Words), Cosine (Word Embedding) |
| Afilliation | Cosine (Bag-of-Words) |
| Venue | Cosine (Bag-of-Words) |
| Grant | Cosine (Bag-of-Words) |
| Chemical | Cosine (Bag-of-Chemical Words) |
| MeSH | Cosine (Bag-of-MeSH Terms) |
| Coauthor | Cosine (Bag-of-Names (First Name Initial + Last Name)) |
| Year | Absolute Year Difference |

unique authors $\{A_1, A_2, \cdots, A_m\}$ and their publication lists $L_1, L_2, \cdots, L_m$ where $\bigcup_{i=1}^{m} L_i = R$.

Figure 2 shows the general pipeline of AND. Because processing AND is not the main contribution of the paper, we briefly explain our method to disambiguate PubMed. For all author records, first preprocessing is done to unify some attribute representations (e.g. removing punctuation, and converting diacritics to the English alphabet). Then, blocking is applied to distribute the data into small chunks and cluster within each of them for efficiency. Next, for each block, pairwise classification is done for each pair of data, which classifies whether each pair of records is from the same person or not. Based on this result, finally we cluster the records to identify unique authors.

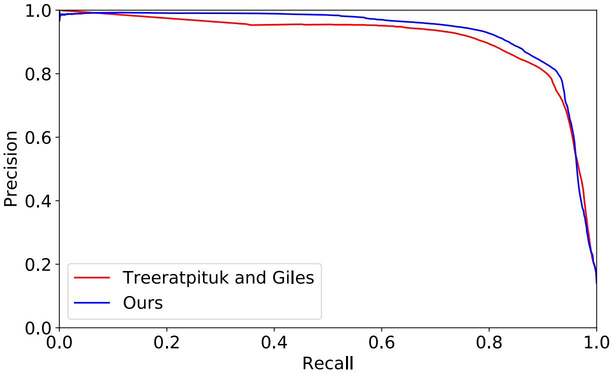


Fig. 3. Precision-Recall of Pairwise Classification for the Method of Treeratpituk and Giles [24]

We use a similar setting for each part in the pipeline as in Kim et al. [10]. The Blocking function combines *first initial and last name*. For the pairwise classification, we started with the state-of-art method suggested by Treeratpituk and Giles [24] and made a few changes to improve the result. We particularly choose this method because we proved that it gave the best results in several databases with a supervised setting (e.g. USPTO patent database [10], financial entity databases [9]), in which there are abundant labeled data to train a classifier. Note that one can also easily replace and use their own algorithm for the disambiguation, and use our query methods for the web service. First, we use a new feature set consisting of cosine distance of Bag-of-words (BoW) vectors weighted with term frequency-inverse document frequency (TF-IDF), and cosine distance of word embedding [15] trained on title and abstract of all PubMed publication, similar to the Müller [16]. Second, we use additional filtering as in Khabsa et al. [8], to filter out pairs that have incompatible first names and middle names. Finally, we use Gradient Boosted Trees [3] instead of Random Forest [2]; for the former, training and prediction time is faster and is known to be more robust to overfitting. Figure 3 shows the improvement in accuracy in pairwise classification result compared to that of Treeratpituk and Giles [24]. Evaluation is done with the same NIH dataset used in the evaluation of the web service. A detailed explanation of the dataset is provided in Chapter VI. For clustering, we use density-based clustering (DBSCAN) [4], which does not require prior guessing of the number of clusters.

### B. Search Engine

A search engine is used to index all attributes of the author records and store their publication lists, so that it handles the attribute-based queries. We use Elasticsearch[16] as our backend search engine. In section V-A, the selection of search engine and the handling of attribute-based queries is covered in detail.

### C. Record-to-Cluster Matching

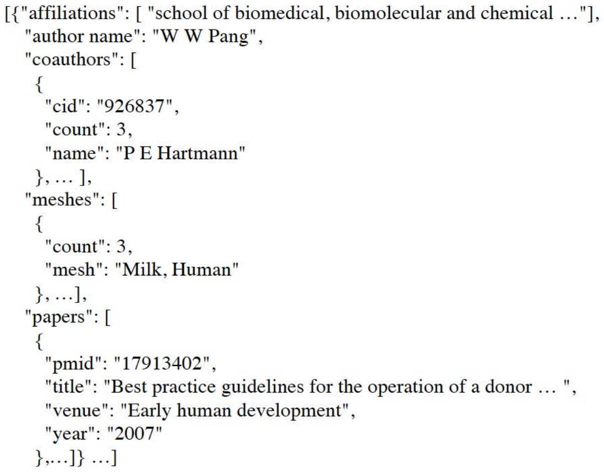
The record-to-cluster matching module is used to process the record-based queries. Given the author record uploaded by a user, it returns the disambiguated author cluster that is likely to be the author of the record. Our proposed record-to-cluster matching algorithm is used to retrieve the relevant disambiguated author. A detailed explanation of our proposed method is in Section V-B.

### D. Web Server with RESTful API

We run a web server using the micro web framework Python Flask[17] to handle all API requests from the users and to manage user resources. We choose Flask because it is light-weight, and does not require particular tools or libraries. However, it can be replaced easily with any web framework that can handle HTTP requests and that manages user file resources for record-based queries.

The web server is responsible for handling all APIs described in Table I. It handles the GET request by sending the queries to two corresponding modules introduced in Section V-A and V-B, and returns the query results in JSON format. Figure 4 shows an example of search results returned by our API. The server also manages user file resources created by POST and deleted by DELETE requests. The POST request accepts user resource data either in BibTeX[18] or PDF format. We use BibTeXParser[19] (for LaTeX input) and GROBID [14] (for PDF input) to extract metadata of the publication, and keep the metadata in JSON format for the file system. These resources are maintained with a unique ID generated with the Linux *mkstemp* command, to keep them secure and intractable. It sets each resource created by a user with time to live (TTL) to ensure that all resources are deleted after use, even without the user requesting the DELETE API explicitly. This allows a reasonable volume of storage to be maintained in the server.

The web server is also responsible for returning error codes of our web service, such as a bad request error (code 400) for an unknown API request, and a not found error (code 404) when the user requests GET API with an unknown resource ID.



```
[{"affiliations": [ "school of biomedical, biomolecular and chemical …"],
  "author name": "W W Pang",
  "coauthors": [
   {
    "cid": "926837",
    "count": 3,
    "name": "P E Hartmann"
   }, … ],
  "meshes": [
   {
    "count": 3,
    "mesh": "Milk, Human"
   }, …],
  "papers": [
   {
    "pmid": "17913402",
    "title": "Best practice guidelines for the operation of a donor … ",
    "venue": "Early human development",
    "year": "2007"
   },…]} …]
```

Fig. 4. Example of a Search Result in JSON Format

### E. Author Search Interface

As a demonstration application of our web service, we developed a web search interface named PubMedSeer , for PubMed disambiguated authors. The interface uses the proposed web APIs to handle all queries, supporting both attribute-based and record-based queries proposed in our web service.

## V. Query Processing

In this section, we describe how we process the two types of queries we proposed in the backend of our web service.

### A. Attribute-based Query

An attribute-based query uses one or multiple attributes in the author record to retrieve disambiguated authors. To handle this type of query, we index attributes of each disambiguated author with a search engine, including title, name, coauthors, affiliation, venue, and MeSH terms of each publications in offline mode. We reformulate the attribute-based query request to the appropriate query to the search engine, and return the result to users in JSON format.

Recently, Apache Solr[20] and Elasticsearch have been widely used for this purpose. Both use Apache Lucene Core[21] to index the data and provide similar functionalities. While Solr has good community support and is well documented, Elasticsearch is easy to use and light-weight. Elasticsearch allows us to make efficient filtering queries and aggregations while Solr is text search oriented. We particularly choose Elasticsearch because of its flexibility and scalability. Shards are used as index partitions for Lucene core. Elasticsearch has a cache per shard which is useful in case of rapidly changing data, while Solr uses a global cache. If data is changed in a single shard, only the corresponding cache is invalidated rather than the whole global cache, which makes the Elasticsearch flexible

[16] https://www.elastic.co/products/elasticsearch

[17] http://flask.pocoo.org/

[18] https://www.ctan.org/pkg/bibtex

[19] https://github.com/sciunto-org/python-bibtexparser

[20] http://lucene.apache.org/solr/

[21] https://lucene.apache.org/

TABLE IV
FEATURES USED FOR PAIRWISE CLASSIFICATION FOR RECORD-BASED QUERIES

| Attribute | Features |
|---|---|
| Title | Cosine (Bag-of-Words), Cosine (Word Embedding) |
| Abstract | Cosine (Bag-of-Words), Cosine (Word Embedding) |
| Afilliation | Cosine (Bag-of-Words) |
| Venue | Cosine (Bag-of-Words) |
| Coauthor | Cosine (Bag-of-Names (First Name Initial + Last Name)) |
| Year | Absolute Year Difference |

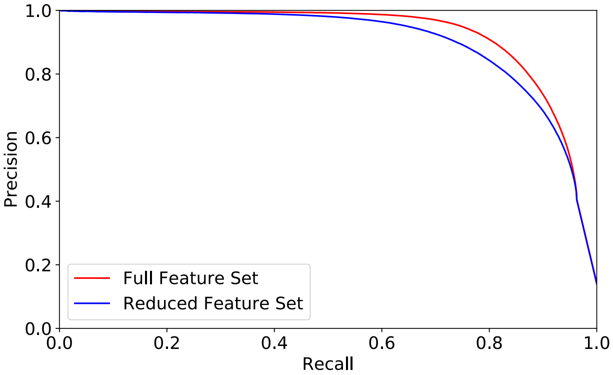


Fig. 5. Comparison of Pairwise Classification Results of Feature Sets for Disambiguation (Table III) and Record-based Queries (Table IV)

in dynamic environment. Scalability is important in our service because the scholarly databases grows exponentially, and also better flexibility let us to update the index easily if we need to update the schema.

### B. *Record-based Query*

A record-based query retrieves the author who most likely wrote the publication queried by the user. To handle this type of query, we utilize the pairwise classification method used during the disambiguation process to find a match between the query record and disambiguated authors.

*1) Reduced Feature Set for Record-based Query:* Because users can provide any type of publication from a wide range of sources and scholarly databases, we train a new classifier using a set of features that exist in virtually all publication records. Table IV shows our reduced set of features which excludes PubMed-specific attributes from the original features used for PubMed disambiguation (Table III). Figure 5 shows the pairwise precision and recall of the new classifier compared to the original one used for the PubMed disambiguation, tested on NIH PI dataset (see Section VI for detail). The result shows some loss of accuracy in general, due to the limited information used for training the classifier. This loss can be thought as a trade-off to cover any type of publication record from various sources.

*2) Record-to-Cluster Pairwise Classification:* Khabsa et al. [8] proposed a method to find the most relevant author cluster for a publication using pairwise classification results. The method is originally used for online disambiguation, which dynamically disambiguates new author records from existing disambiguation results. Although our purpose is different, we can apply their method to handle record-based queries, and use their method as a baseline here.

They assign each new author record to an existing cluster according to the following process: 1) apply blocking on the author record, 2) find matches among all the records in the block by measuring record-wise similarity with the pairwise classifier (which we refer as *record-to-record pairwise classification* from now on), 3) assign the new record to the disambiguated author cluster with the highest vote. The time complexity of the algorithm is $O(mn)$, where $m$ is the number of clusters and $n$ is the largest number of author records among all the clusters in the block. Although this can solve our problem, the query time can be extremely slow if the block size is large, e.g. for some Asian names.

Our approach is to reduce the total number of required pairwise comparisons, by processing *record-to-record pairwise classification* for only those candidate clusters that have a high probability of being a match. To accelerate the process of finding those candidate clusters, we propose a *record-to-cluster pairwise classification* which estimates the similarity between the query record and each cluster (we refer this similarity as cluster-wise similarity) with a single classification operation. To enable this, we calculate the pairwise feature vector in Table IV between each cluster and the query record. The idea is to treat each cluster as one single record, where each attribute value is the union of all values of all records in the cluster. So for each feature in Table IV, Bag-of-words (BoW) vectors are calculated as the sum of the vectors within the cluster,

$$BoW_{attribute}(C) = \sum_{r \in C} BoW_{attribute}(r) \tag{1}$$

where r is an author record of a cluster C. Each vector is then used to generate features by calculating the cosine distance with the BoW vector of the query record. Word embedding features can also be calculated in the same manner. Then we predict the cluster-wise similarity using the pairwise classifier.

Our assumption underlying this method is that we can predict the cluster-wise similarity using the pairwise classifier trained for *record-to-record pairwise classification*. To verify this, we calculate the correlation between the actual match result and the cluster-wise similarity estimated with the proposed method. We get a strong positive correlation (0.87), so we can use the same pairwise classifier to estimate the cluster-wise similarity.

*3) Processing Record-based Query:* We process the query utilizing both the proposed *record-to-cluster pairwise classification* and previous *record-to-record pairwise classification*, as shown in Algorithm 1. First, we calculate the cluster-wise similarity using *record-to-cluster classification* for each cluster and filter out clusters beneath the threshold $t_{cluster}$. Then, to further increase accuracy, we process a re-ranking step. We process *record-to-record pairwise classification* for each

**Algorithm 1** Record-based Query

```
1:  function RECORDBASEDQUERY($q$)
2:      $B \leftarrow$ block with same blocking key as $q$
3:      $C \leftarrow$ all clusters in $B$
4:      $C' \leftarrow \{\}$
5:      for $c \in C$ do
6:          $cluster\_sim_c \leftarrow$ PAIRWISERECORDTOCLUSTER($q, c$)
7:          if $cluster\_sim_c > t_{cluster}$ then
8:              put $c$ in $C'$
9:          end if
10:     end for
11:     for $c \in C'$ do
12:         $record\_sim_c \leftarrow$ average of PAIRWISERECORDTORECORD($q, r$) of all $r \in c$
13:         $joint\_sim_c \leftarrow \alpha \times cluster\_sim_c + (1 - \alpha) \times record\_sim_c$
14:     end for
15:     return all $c \in C'$ in descending order
16: end function
```

author record in the remaining clusters to calculate record-wise similarities. The final joint similarity score is a linear combination of the cluster-wise similarity and average of record-wise similarity as in line 13. The parameters $\alpha, t_{cluster}$ is selected using grid search with the development set. From our experiment, we tested in the range $[0.0, 1.0]$ with an increment of 0.01, and we use $\alpha = 0.48, t_{cluster} = 0.54$.

The time complexity of the algorithm is $O(m)$ + $O(m_f n)$ where $m$ is the number of clusters and $n$ is the largest number of author records among all clusters in the block, and $m_f$ is the number of remaining clusters after filtering with cluster-to-record pairwise classification. It requires $m$ additional comparisons for cluster-wise similarity, and reduces $(m - m_f)n$ comparisons by filtering out those non-relevant clusters. Typically the latter is much larger than the former, because we try to filter non-relevant clusters as much as possible during the first step of our algorithm. Thus, our method is faster than the baseline method [8].

## VI. Experiments

We conducted experiments to measure the query time and accuracy of proposed attribute-based and record-based queries for our web service.

### A. Experiment Environment

For experiments, we use a single machine to run all components of the web service. It has an Intel Xeon CPU E5-2630 V3 @ 2.40 GHz, and runs concurrently up to 32 threads. We use no more than 64GB of memory, and the code is in python 2.7. We use Red Hat Enterprise Linux (RHEL) Server 7.4. We use PubMed raw xml files downloaded in late 2017.

### B. Experiments on Attribute-based Query

We measure the average query time of our attribute-based queries for various scenarios. We didn't measure accuracy of the attribute-based queries because the accuracy heavily relies on the attributes available on the data. Our data is indexed with Elasticsearch on a single machine and is divided into 5 shards. We generated three different types of queries with each type having 2000 queries. The first type performs searches only over the *name* attribute. The second query type uses both *name* and *MeSH* attributes as search criteria. The search key for the last type combines *name*, *MeSH* and *title* attributes. Average query time for single attribute, double attribute, and triple attribute queries are 2.02s, 3.04s and 5.35s respectively.

### C. Experiments on Record-based Query

We compare our method for the record-based query in Section V-B to the baseline method [8] which only uses *record-to-record pairwise classification* to retrieve the most relevant disambiguated author.

*1) Performance and Accuracy Evaluation with Labeled Dataset:* We use the NIH PI dataset to compare accuracy and performance. The dataset comprises (PI ID, PI name, list of publications in PubMed) tuples, and has 54,260 individuals and 1,178,459 records. We apply blocking to those records and then extract 100 popular blocks (which have the largest number of individuals), and then divide it into two sets for development and test sets. We use the largest blocks as an evaluation set to generate evaluation query samples as much as possible. The remaining blocks are used as a training set. The training set is used to train the pairwise classifiers for disambiguation (Section IV-A) and record-based queries (Section V-B). Approximately 10M positive and negative samples are generated from the training set. The development set is used to tune the parameters of the classifiers and the parameters in the record-based query (Algorithm 1) using a grid search.

The test set is used for the evaluation where we compare the baseline method [8] and our proposed method. We also compare our method without the re-ranking step, so that we only use cluster-wise similarity for ranking retrieved clusters. This is to see whether we lose accuracy by excessive filtering (first step of our algorithm), and also to see whether using both *record-to-cluster* and *record-to-record pairwise classification* improves the results (second step of our algorithm). We form author clusters with the test dataset, randomly sample 20% of all records and then remove them from author clusters. Removed records are used as test queries. To evaluate the accuracy, for each query we check whether a method retrieves the correct author cluster (to which the query record originally belonged). We use the metric *recall@k*, which checks whether the right author cluster appears in the top $k$ result of each query result. Also, we calculate the mean average precision (MAP), which is the average of inversed rank. If a query has the right author cluster in rank $k$, the precision is calculated as $1/k$. Since each query has only one true positive result in our experiment, the MAP result is identical to mean reciprocal rank.

The evaluation of the baseline method and our proposed method tested with 3,538 queries is shown in Table V. Even without the re-ranking step our method tends to find more

TABLE V
ACCURACY AND AVERAGE QUERY TIME OF RECORD-BASED QUERIES ON THE NIH PI DATASET. R@K IS RECALL@K, MAP IS MEAN AVERAGE PRECISION, AND TIME IS AVERAGE QUERY TIME.

| Method | r@1 | r@5 | r@10 | MAP | Time |
|---|---|---|---|---|---|
| Baseline [8] | 0.9562 | 0.9898 | 0.9901 | 0.9726 | 27.37ms |
| Ours w/o re-ranking | 0.9401 | 0.9958 | 0.9966 | 0.9673 | 8.39ms |
| Ours w/ re-ranking | 0.9661 | 0.9960 | 0.9966 | 0.9817 | 8.72ms |

TABLE VI
AVERAGE QUERY TIME FOR 300 RECORD-BASED QUERIES ON PUBMED

| Method | Time (w/o caching) | Time (w/ caching) |
|---|---|---|
| Baseline [8] | 247.559s | 23.329s |
| Ours | 195.017s | 5.815s |

relevant clusters than the baseline, although it looses some accuracy when finding them in the first rank. This shows that our method successfully filters out only those clusters that are less likely to be a match during the *record-to-cluster pairwise classification*. The result with the re-ranking shows that the second step of our algorithm further improves the accuracy using both cluster-wise and record-wise similarity, and when compared to the baseline, our final method has better accuracy for all ranges of *recall@k*. Also, we can see that our final method has better MAP compared to the baseline and method without re-ranking. This shows that the relevant cluster can be found in a higher rank in our method, and also less frequently entirely missed from the query result. Note that the overhead of re-ranking is tolerable, and compared to the baseline, our method is 3.14 times faster.

*2) Actual Performance Evaluation:* Since the NIH PI dataset consists of a small portion of all the disambiguated author data, the query time above does not reflect the actual query time that users can expect from our service. To compare and measure the actual performance, we tested on the entire disambiguated PubMed data. We tested with 300 queries with each query associated with a different block. We carefully selected blocks to have varying sizes in order to accurately measure processing times. Table VI shows the comparison of the query time between the baseline method and ours. As we can see from the query time without caching, the speed improvement from our method is much lower. The lower gain is due to excessive database queries, so we store and cache the input vectors for pairwise classification to improve the query time. We can see that our method is 4.01 times faster than the baseline method with caching, which shows a similar result to the experiment we conduct on the labeled dataset. One can use the threshold $t_{cluster}$ to adjust the balance between the accuracy and query speed. Higher threshold improves the speed by filtering clusters more aggressively, while lower threshold improves the accuracy by considering more clusters.

## VII. CONCLUSION

We propose a web service with a RESTful API for searching disambiguated authors in scholarly databases. Two types of queries are supported in our service. An attribute-based query searches appropriate disambiguated authors using attributes of the author record. A record-based query retrieves the disambiguated author most likely to be the author of the publication record provided as a query. The two queries need to be processed differently since the former uses internal resources, and the latter uses external resources. Resource management is provided with RESTful APIs. We studied two different search engines to handle attribute-based queries and proposed a novel *record-to-cluster pairwise classification* and algorithm to accelerate record-based queries. Our results show that the record-based query is four times faster than the baseline method.

Future work could explore record linkage between disambiguated authors in other databases and construct a web service containing unified profiles of multiple scholarly databases. As an example for our web service with PubMed, we could match each cluster to Google Scholar and ORCID profiles.

## ACKNOWLEDGMENT

Giles, Kim and Weinberg gratefully acknowledge support from NIA, OBSSR, and NSF SciSIP through P01 AG039347. Weinberg gratefully acknowledges support from EHR DGE 1348691, 1535399, 1760544; and the Ewing Marion Kauffman and Alfred P. Sloan Foundations. Weinberg was supported on P01 AG039347 by the NBER directly and on a subaward from NBER to Ohio State.

## REFERENCES

[1] M. Ahmadpoor and B. F. Jones, “The dual frontier: Patented inventions and prior scientific advance,” *Science*, vol. 357, no. 6351, pp. 583–587, 2017.

[2] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[3] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD’96)*, vol. 96, no. 34, 1996, pp. 226–231.

[5] A. A. Ferreira, M. A. Gonçalves, and A. H. Laender, “A brief survey of automatic methods for author name disambiguation,” *Acm Sigmod Record*, vol. 41, no. 2, pp. 15–26, 2012.

[6] B. F. Jones and B. A. Weinberg, “Age dynamics in scientific creativity,” *Proceedings of the National Academy of Sciences*, vol. 108, no. 47, pp. 18 910–18 914, 2011.

[7] M. Khabsa, P. Treeratpituk, and C. L. Giles, “Large scale author name disambiguation in digital libraries,” in *IEEE International Conference on Big Data*, 2014, pp. 41–42.

[8] ——, “Online person name disambiguation with constraints,” in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries(JCDL’15)*, 2015, pp. 37–46.

[9] K. Kim and C. L. Giles, “Financial entity record linkage with random forests,” in *Proceedings of the Second International Workshop on Data Science for Macro-Modeling*. ACM, 2016, p. 13.

[10] K. Kim, M. Khabsa, and C. L. Giles, “Random forest dbscan clustering for uspto inventor name disambiguation and conflation,” in *IJCAI-16 Workshop on Scholarly Big Data: AI Perspectives, Challenges, and Ideas*, 2016.

[11] M. Levin, S. Krawczyk, S. Bethard, and D. Jurafsky, “Citation-based bootstrapping for large-scale author disambiguation,” *Journal of the American Society for Information Science and Technology*, vol. 63, no. 5, pp. 1030–1047, 2012.

[12] S. G. Levin and P. E. Stephan, “Research productivity over the life cycle: Evidence for academic scientists,” *The American Economic Review*, pp. 114–132, 1991.

[13] W. Liu, R. Islamaj Doğan, S. Kim, D. C. Comeau, W. Kim, L. Yeganova, Z. Lu, and W. J. Wilbur, “Author name disambiguation for pubmed,” *Journal of the Association for Information Science and Technology*, vol. 65, no. 4, pp. 765–781, 2014.

[14] P. Lopez, “Grobid: Combining automatic bibliographic data recognition and term extraction for scholarship publications,” in *International Conference on Theory and Practice of Digital Libraries*. Springer, 2009, pp. 473–474.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[16] M.-C. Müller, “Semantic author name disambiguation with word embeddings,” in *International Conference on Theory and Practice of Digital Libraries*. Springer, 2017, pp. 300–311.

[17] M. Packalen and J. Bhattacharya, “Age and the trying out of new ideas,” National Bureau of Economic Research, Tech. Rep., 2015.

[18] Y. Petinot, C. L. Giles, V. Bhatnagar, P. B. Teregowda, H. Han, and I. Councill, “A service-oriented architecture for digital libraries,” in *Proceedings of the 2nd international conference on Service oriented computing*. ACM, 2004, pp. 263–268.

[19] Y. Qian, Q. Zheng, T. Sakai, J. Ye, and J. Liu, “Dynamic author name disambiguation for growing digital libraries,” *Information Retrieval Journal*, vol. 18, no. 5, pp. 379–412, 2015.

[20] Q. Shen, T. Wu, H. Yang, Y. Wu, H. Qu, and W. Cui, “Nameclarifier: A visual analytics system for author name disambiguation,” *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 141–150, 2017.

[21] D. K. Simonton, “Creative productivity: A predictive and explanatory model of career trajectories and landmarks.” *Psychological Review*, vol. 104, no. 1, p. 66, 1997.

[22] R. Sinatra, D. Wang, P. Deville, C. Song, and A.-L. Barabási, “Quantifying the evolution of individual scientific impact,” *Science*, vol. 354, no. 6312, p. aaf5239, 2016.

[23] V. I. Torvik and N. R. Smalheiser, “Author name disambiguation in medline,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 3, p. 11, 2009.

[24] P. Treeratpituk and C. L. Giles, “Disambiguating authors in academic publications using random forests,” in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries(JCDL’09)*, 2009, pp. 39–48.

[25] K. Williams, L. Li, M. Khabsa, J. Wu, P. C. Shih, and C. L. Giles, “A web service for scholarly big data information extraction,” in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 105–112.

[26] H. Zhao and P. Doshi, “Towards automated restful web service composition,” in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 189–196.

[27] N. Zolas, N. Goldschlag, R. Jarmin, P. Stephan, J. Owen-Smith, R. F. Rosen, B. M. Allen, B. A. Weinberg, and J. I. Lane, “Wrapping it up in a person: Examining employment and earnings outcomes for ph. d. recipients,” *Science*, vol. 350, no. 6266, pp. 1367–1371, 2015.

[28] H. Zuckerman and R. K. Merton, “Age, aging, and age structure in science,” *Higher Education*, vol. 4, no. 2, pp. 1–4, 1972.