

# Summer Student Report: Author Clustering on Large Bibliographies

Christoph Sterz

CERN – European Council for Nuclear Research  
Meyrin, Switzerland

christoph.sterz@student.hpi.uni-potsdam.de

## ABSTRACT

We analyze and design an algorithm for clustering large sets of authors in Bibliographies. Not considering a distance function for a mutual comparison, but transforming the data into a multidimensional metric space, the algorithm described is similar to *locally sensitive hashing*. The task lies in the field of *Record-Linkage*.

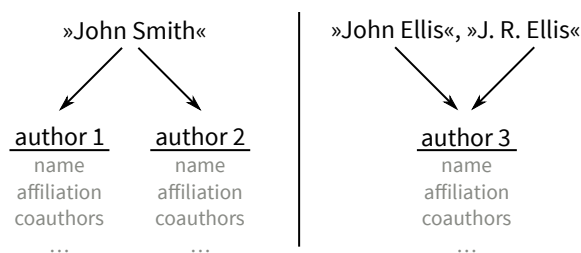
The algorithm was designed and performed based on the data of the *CERN Document Server*, consisting out of more than 1.7 million metadata entries and is part of the digital assets-managing-software *invenio*.

Meant as a prototype, the algorithm performs efficiently, clustering all authors on CDS in under 30 minutes. We will discuss extensions improving the recall rate, which still remains inferior to the currently used clustering-approach.

## INTRODUCTION

Nowadays, digital bibliographies of big organisations are managed with the help of relational database systems like MySQL. Originating in a library-dominated background though, data is sometimes still organized in an architecture resembling bibliographies and standards of physical libraries, for example, catalogues and indexing systematics like *MARC*[1].

Managing authorities in this document-centered databases is a challenging task, since for a given name there can be different Authorities—whereas, in contrast, different variations of an author name can belong to the same author.



**Figure 1: Illustrative example of the Task: The authorname »John Smith« is used for several authors, whereas »John Ellis« and »J. R. Ellis« both stand for the same author.**

We propose an algorithm considering multiple available fields as *clues*, and then successively transforming this data into a metric space with an optimized and weighted encoding; we compress the data applying a projection into a low-dimensional space of Eigenvectors determined by *Principal Components Analysis*; ultimately we use a *DBSCAN*[2]

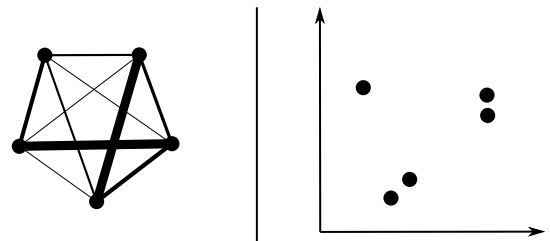
clustering, aided by a KD-tree datastructure to retrieve the similar authors from the Database.

## USING METRIC SPACE INSTEAD OF COMPARISONS

We use a metric space for representing our data.

Classical approaches use string distance metrics like Jaccard or Hemming distance. The Jaro-Winkler distance is specially engineered to cope with comparison of names[3].

Since the order this operation imposes on the data does not ensure transitivity, the use of specific string-based distances *forces* algorithms to compare every pair of datasets. This results in a complexity of  $O(n^2)$ .



**Figure 2: Visualisation of both approaches: On the left a pairwise similarities on the right neighbors forming clusters in a metric space.**

Working like this on datasets of over one million entries and thus performing  $1 \times 10^{12}$  comparisons (e.g. distance operations often being computationally intensive) is not practical.

To overcome this problem, we encode the data into a metric space. Doing this allows us to use efficient datastructures like KD- and ball-trees to perform a fast neighbor lookup, which is then needed by our clustering algorithm.

## ENCODING AND WEIGHTING THE DATA

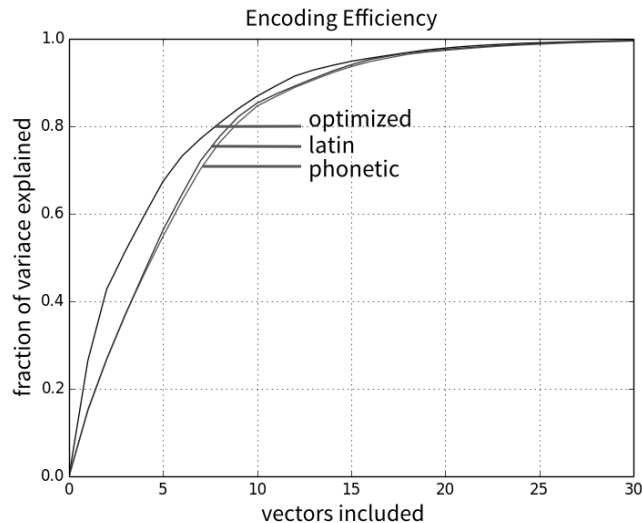
In Preparation of clustering, we have to encode the data to transform it into multidimensional vectors. We first retrieve the data and put it in a format just containing lowercase latin characters.

### Choosing the Encoding

As we found out, the representation of a letter to a Number is crucial for a clustering task. While the vowels "o" and "i" are different and thus should have a big distance in encoding, "f" and "v" are quite similar, originating from transliterations of names out of the slawic language families for example. Hence, choosing the right alphabet can provide an efficient clustering-metric, this way.

Because we are no experts in the field of linguistics and phonetics, we purely relied on the data. We considered our sample of 1.7 million names being *representative* to contain most of the similarities. Also, characters that are not used frequently can be used to spread the distance between the more frequent, non-similar characters.

We used the *explained variance* of the next step of the algorithm (*PCA*) to measure the performance of the current encoding. Making use of *simulated annealing* parameter optimisation, we retrieved a better alphabet, explaining more variance in datasets even on a few vectors in the PCA:



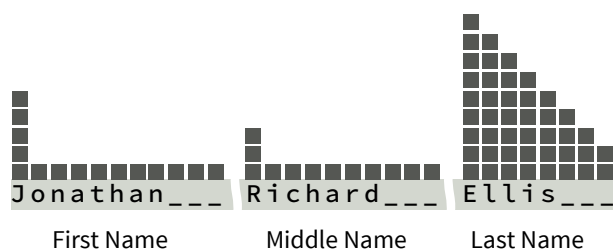
**Figure 3: Difference in explained fraction of data between phonetic(green), latin(red) and our optimized(blue) alphabet.**

We also considered n-Grams for encoding, which showed promising results, but remains future work to look at.

### Weighing Data

We use a weighting-mechanism to put emphasis on important parts of the names and other information retrieved from the database. For this, we simply scale the according elements on the vectors.

Figure 4 shows an exemplary weighting on a name feature, that is actually used.



**Figure 4: A possible distribution of weights. Note how the first letters are ephasized, since often Initials are given. The last name, which is most prominent in the database, is weighted decreasingly.**

Other fields being retrieved can be weighted similarly after they have been transformed into numerical data.

We normalize the Vectors, such that the vectors independent on their "content" have the same impact on the data compression in the next step.

### Encoding Missing Data

One of the drawbacks a metric featurespace introduces is the encoding of missing Data. It is obvious, that in a single dimension encoding the distance of missing data "\_" to all the other characters can never be fair, as presented in the following figure:

```
_ a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m _ n o p q r s t u v w x y z
_a _b _c _d _e _f _g _h _i _j _k _l _m _
```

**Figure 5: No matter where the missing data is kept, the distance to characters is always unfair: in the first row, missing data is defined as zero which can make comparisons irrelevant in geometric distances. The second row tries to have a fair distribution of distances. On the third row missing data is encoded with a random odd string, while actual existing data is encoded with even numbers, trying to achieve statistical fairness.**

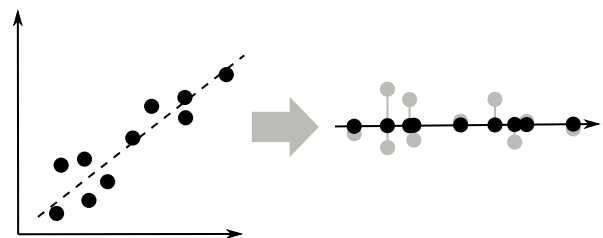
We decided to use the unfair, but faster encoding with missing data in the center of the encoding alphabet since we could not use a geometric distance computation in specialized datastructures providing a logarithmic neighbor search behavior. Also we decided against the statistically fair last approach.

### EIGENVECTOR BASED COMPRESSION

Text-centered data is highly correlated. In fact, in our case, correlation is bloating the recognition and clustering of authors. In our case, we decided to keep vectors of length of the original data, consisting out if the first, last and two middle names, as well as—if available—the affiliation, e-mail, and IDs provided in other Library systems. This 100-dimensional vectors are then reduced to 5 to 10 dimensions. We thus avoid the so-called *curse of dimensionality* in clustering and fast neighbor search algorithms.

### Principal Components Analysis

We use a *Principal Components Analysis*, which compresses the data down to the space of its first and most expressive Eigenvectors, being retrieved with the help of Singular Value Decomposition.

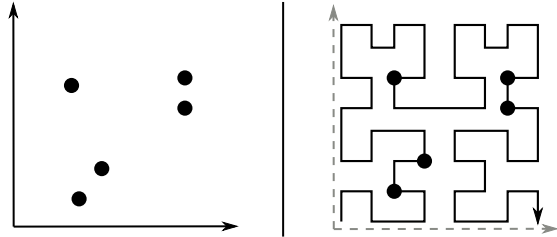


**Figure 6: Illustrative Example of decorrelation and compression based on a *projection* to the most prevalent eigenvectors. For explanatory reasons, here two highly correlated dimensions are compressed.**

Note, this compression also weights data based on the availability of variance in the vector. This makes sure the less available datafields in the vector are considered less which reduces the missing-data-encoding error explained in the previous section.

### Space Filling Curves

Traditionally, spacial hashes are implemented or improved by space-filling curves, which preserve the spacial distance up to a certain error, while reducing dimensionality. A Hilbert Curve as depicted in the following figure would be one example:



**Figure 7: A Hilbert Curve reducing the dimensionality of a spacial distribution, while trying to preserve locality of the given points.**

One advantage of this concept is the good fitting to our vectorized, but discrete data. Accessing one field of data instead of two for a fast neighbor search also reduces cache-pollution in the execution of the program.

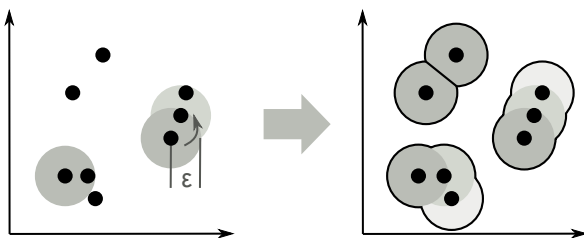
Still, we decided against using this concept since it is only practical for dimensionalities of two and three. In higher dimensions the overhead of index computation becomes too expensive and the minimum-locality-error in high dimensionalities becomes too large.

### DBSCAN CLUSTERING AND SIMILARITY SEARCH

We use the DBSCAN algorithm for clustering.

#### DBSCAN

The algorithm considers dense regions within the data as clusters, that it then further extends based on a minimum cluster-distance  $\epsilon$ . It might occasionally visit datapoints multiple times in the course of the analysis. Still the overall complexity is dominated by the range-query of neighbors[2].



**Figure 8: The core step in DBSCAN: Points retrieved by a region-query of a radius  $\epsilon$  are added to a cluster recursively including their regions. The result is presented on the right.**

### Using Spatial Indexing Structures

As discussed before, the region-query is the dominating factor in the complexity of the algorithm. The Scikit implementation used, offers several options: We found the use of ball-trees as the option of the spacial index is optimal for our needs. The DBSCAN paper states, this reduces the complexity to  $O(n \log n)$ .

Although we rely on the aforementioned ball-trees, the runtime behavior of our algorithm appears quadratic in the measurements. We did not investigate this further, because the runtime behavior already fitted our needs.

### Similarity Search

As a sideproduct we also implemented a similarity search based on a simple *K-Nearest Neighbor* search, again supported by the use of a spacial index structure. This can now be used for interactive suggestions in searches in the style of big search-engines, because it is able to deliver results in the sub-millisecond timeframe.

### IMPLEMENTATION REMARKS

The code resembles the described steps in this paper.

We implemented the algorithm fully in python using the sklearn package provided under the Scikit framework[4]. Except for feature extraction we rely on the packages' implementations provided for PCA, Fast Nearest Neighbor Search, and DBSCAN. Scikit is based on, and uses numpy[5].

The normalisation of the preprocessing-step is taken from the already existing work done in the bibauthorid package of the software *invenio*.

### RESULTS

#### Data Availability and Retrieval

Firstly, we investigated the availability of data suitable to be used as "clues" within the *Cern Document Server*:

	authors [100]	coauthors [700]
instances (names)	368.988	551.643
with available		
Affiliation	23.274	39.525
INSPIRE-ID	1.393	11.632
CCID	1.578	3.621
CCID/ALICE-ID	513	4.809

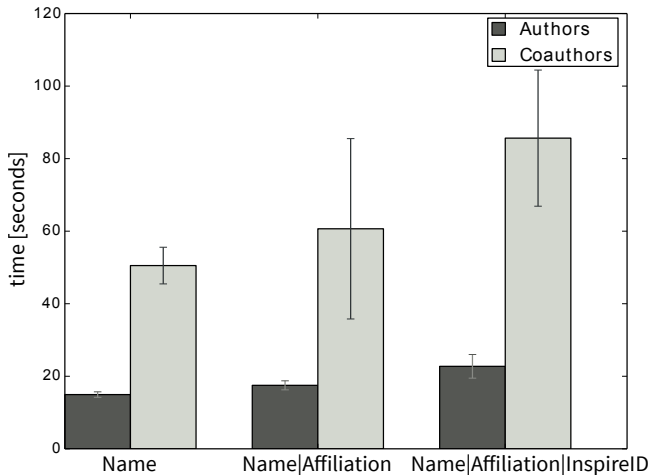
**Table1: Availability of metadata about authors in the CDS Database, that can be used immediately for disambiguation.**

This results show a poor availability of direct metadata for author instances. In future work we thus have to rely on more indirect information such as similarity of coauthors, which itself is a catch-22 situation: *In order to cluster based on coauthors the identity of those coauthors must be known.*

One of the better sources for information is the metadata of the authors' document itself. Especially year, title, and category look promising and are widely available.

We implemented custom-tailored SQL statements to retrieve the data efficiently from the database. Here are the results

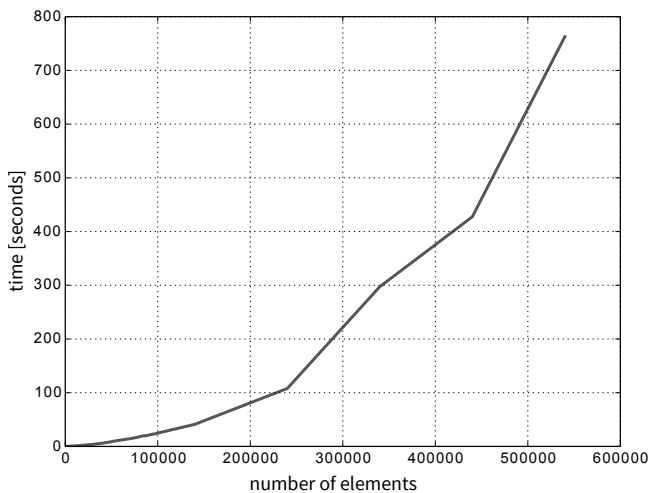
for a query of up to 3 metadata entries, totalling 18 million queried record-fields of the Database:



**Figure 9: Data aquisition from the DB: additional fields do not add a lot of runtime, since the comparison-operation is shadowed by the I/O-intense SELECT\* statement.**

The preprocessing of data (for example the transliteration of digrams) then takes about one minute. We do not provide detailed benchmarks on this step, since this code is taken from the already existing bibauthorid package.

We present an exemplary clustering-benchmark on increasing datasets. Note, that the number of datapoints being clustered are already substantially lower than the 1.7 million for author and coauthor entries. The reason for this is the exact similarity of entries, we can collate without changing the result of the algorithm.



**Figure 10: DBSCAN clustering performance on Single Core (single measurement)**

The consumption of memory at all times was lower than 200MB which can be explained by the size of the search index consuming about 90 % of this size.

## Quality

The clustering quality of the algorithm is hard to measure. Firstly, because we do not have sufficiently large sets of labeled, "ground-truth" data. Secondly, we cannot compare the existing algorithm directly with our results, since the data availability in the *INSPIRE-HEP* database of the existing algorithm is different.

Also optimal parameters e.g. for clustering distance still have to be determined. Once this is done, a more serious and precise quality measurement can be performed.

## CONCLUSION AND FUTURE WORK

We presented a prototype for fast clustering on large bibliographies on the example of the *CERN Document Server*. By discussing all the important design decisions and explaining the chosen concepts we provide both a theoretical documentation as well as justification of the algorithms in each step.

Considering resource needs, the algorithm outperforms the existing one in time and memory usage up to a factor of 5.000. Because the quality of the results stays inferior to the existing algorithm, some work still needs to be done in tuning the parameters as well as in including more relevant information for the clustering step.

We recommend to investigate the approach further, especially concerning interactive searches and fast identity checks in the database.

Aside from this two main steps, the future work can be in the implementation of a multi-threaded version, which for known subclusters, like historical and recent article promises an even better runtime behavior.

## ACKNOWLEDGEMENTS

I would like to thank my Supervisor Nikolaos Kasioumis as well as Samuele Carli, Ludmilla Marian, Toni Mattis, and Gjergji Kasneci for their helpful advice. Special Thanks to Avraam Tsantekidis for his cheerful words during my Summer Student time.

## REFERENCES

- [1] Library of Congress, Network Development and MARC Standards Office, <http://www.loc.gov/marc/>
- [2] Ester, M., 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 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996
- [3] Cohen, William W., Ravikumar P., Fienberg, Stephen E., "A comparison of string distance metrics for name-matching tasks", 2003
- [4] Scikit-learn package webpage, <http://scikit-learn.org>
- [5] Numpy package webpage, <http://www.numpy.org>