Ranking-Based Name Matching for Author Disambiguation in Bibliographic Data

Jialu Liu[†] Kin Hou Lei[†*] Jeffery Yufei Liu[‡*] Chi Wang[†*] Jiawei Han[†]
[†]Department of Computer Science [‡]Department of Statistics
University of Illinois at Urbana-Champaign, Urbana, IL
{jliu64, klei2, liu105, chiwang1, hanj}@illinois.edu

ABSTRACT

Author name ambiguity is a frequently encountered problem in digital publication libraries such as Microsoft Academic Search. The cause of this problem mostly is that different authors may publish under the same name, while the same author could publish under various names due to abbreviations, nicknames, etc. Author disambiguation is exactly the goal of the Track II of KDD Cup Data Mining Contest 2013. In this paper we introduce our ranking-based name matching algorithm and system called RankMatch. One important feature of our solution is using heterogeneous meta-paths to evaluate the similarity between two potential duplicate authors whose names are compatible. We participated under team name "SmallData" and our final solution achieved a Mean F_1 score of 99.157%, ranking in the second place in the contest.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Application—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Name disambiguation, Name matching

1. INTRODUCTION

While you are searching for academic papers by an author name, have the returned papers ever surprised you? Sometimes you want to take a look at the publications of W. Wang, a theoretical physicist. Unfortunately, as you might have anticipated, this query leads to an intimidatingly long list of papers by all different authors publishing under "W. Wang", with topics ranging from neurology, chemistry to computer science. When you glance through the list and still couldn't find a paper you are particularly interested in, it suddenly occurs to you that the "missing" paper was in fact published under name "Wei Wang".

Author name ambiguity is indeed an often encountered problem in digital publication libraries such as Microsoft Academic Search. *Author disambiguation* [2, 10] is important for an apparent reason: it helps refine search results for end users and thus improves user experience and productivity. Another reason is that it empowers more accurate bibliometric analysis such as mining academic social networks. The Track II of KDD Cup Data Mining Contest 2013 hosted by Microsoft Academic Search and Kaggle precisely aims to solve this problem.

In the Microsoft Academic Search database each unique author is represented by one or multiple author IDs. The task of the Author Disambiguation Challenge is to enumerate all the duplicate cases for each author ID in the given index author ID set. We will briefly review the data sets provided, evaluation metrics as well as some challenges. Detailed task description can be found in [8] by the contest organizers.

The task is structured in a totally unsupervised manner, where no ground truth is provided to train the disambiguation model. Therefore it is up to the contestants to use their own judgment to determine whether two authors are duplicates. This is different from Treeratpituk and Giles [10] where they have training labels. Fortunately several additional data sets are provided to contestants to assist the decision making. The given data involves 2,293,837 unique author IDs and 2,258,482 papers which are published in 22,227 journals and 4545 conferences proceedings. By combining the given data we can derive a connected network or graph that links about 2.2 million papers with their authors, titles, keywords, publication years and venues. The derived network can also be used to generate deeper features of author's publishing patterns such as co-author and co-venue relationships, which are claimed to be useful in previous work [11].

To evaluate the disambiguation outcome the Mean F_1 score is adopted in this contest. In the literature of information retrieval F_1 score is a commonly used measure for quality of classification, search, etc. It is defined as the harmonic mean of precision and recall: $F_1 = 2pr/(p + r)$, where p is the precision and r is the recall. For instance, suppose for a given index author ID we retrieved 10 duplicate author IDs, out of which 8 are real duplicates. Then the precision

*Equal Contribution.

p is the proportion of correctly retrieved items with respect to all retrieved items $p = 8/10$. Also suppose the index author ID has a total 15 duplicates in the database. So recall r is the proportion of correctly retrieved duplicates among all duplicates, i.e., $r = 8/15$. The final Mean F_1 score is calculated as the average F_1 for the 247,203 indexing author IDs in the data set to be disambiguated. It's worth mentioning that the 247,203 indexing author IDs is a subset of all the 2,293,837 unique author IDs in the whole given data.

Several challenges underlying author disambiguation make it a difficult task. In addition to the obvious complication that different authors could share the same names, most of the challenges are due to the fact that many authors publish under inconsistent names due to abbreviations, nicknames, etc. For instance, publishing names *Michael Lewis, Mike Lewis, Michael James Lewis, Michael J. Lewis* and *M. J. Lewis* could all correspond to the same person. It's sensible to use similar/dissimilar publishing patterns such as co-authors, publication venues and years to identify/rule out duplicate authors, where an author's publishing patterns can be computed from the given data. This idea is actually adopted in our approach and how well it works to some extent depends on data quality. The data is overall very comprehensive and reliable, although we have seen some misspelling and extraction errors of author names, paper titles, etc. For some fields such as author affiliations there are frequent missing values. We also found some data inconsistency cases. For example, in the given author-paper data set, for some papers one can find two co-authors sharing identical names, while only one is the true author and the other is false. So a desired author disambiguation system should be robust to data inconsistency, exceptions and missing values.

The remaining of the paper is organized as follows. In Section 2 we explain the intuition of our approach and lay out the system framework of RankMatch. In Section 3 we illustrate in details the major components in our system. We show empirical results in Section 4 and discussions in Section 5, followed by conclusion in Section 6.

2. SYSTEM OVERVIEW

Given the complex nature of this real world data challenge, it would be sensible to develop a set of methods that are carefully designed to tackle different aspects of the problem. In the following we briefly present our intuitions at designing our system and its major components. Figure 1 shows the diagram of our system framework.

Because the evaluation metric F_1 score is the harmonic mean of precision p and recall r, it is natural to attempt to maximize p and r simultaneously. However, directly doing so is difficult because maximizing one will usually compromise the other, so it is critical to find a balance point between p and r. We take a two-step strategy to first maximize r and then maximize p. Specifically, the two steps are as follows:

- r-step: enlarge candidate pool of duplicates. For each indexing author ID, we try to pull out all the author IDs whose author names are possible variations of the indexing author name. Ideally this pool covers all the duplicates and is as compact as possible. To come up with the pool we take into account a number of cases where names can mutate or be disturbed. For instance, *Mike Lewisg* can be safely assumed to be a noisy variant of *Mike Lewis* by an extra "g", while it is not the case for Chinese names *Wei Lin* and *Wei Ling*, as both "Lin" and "Ling" are valid last names. So we designed algorithms to specifically detect some names of different nationalities such as Chinese, Korean and Japanese names, and apply customized name similarity matching. The name matching and candidate pooling criteria will be covered in detail in the next section.

- p-step: trim the candidate pool based on authors' publication features (i.e., meta-paths) we have calculated. Examples of publication features include co-author network, publication venues, years, titles words and keywords. These features turn out to be discriminative for identifying real duplicates from the candidate pool.

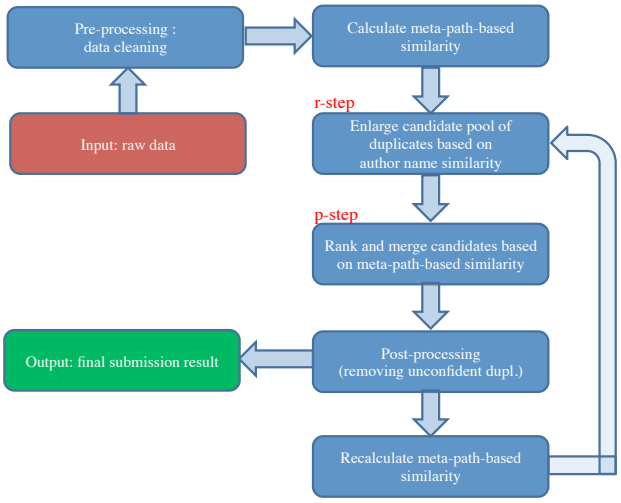

Figure 1: System diagram of RankMatch.

After we run the r- and p- steps, we feed the disambiguation results back to the authors' publication features (i.e., meta paths) calculation engine. This is a crucial step as it will help refine calculation of the features using more accurate data source. In addition to the major steps mentioned above, we also have a pre-processing procedure to handle noisy and missing values. At the end we have a post-processing step to remove unconfident duplicates based on certain measures. In the following section we will illustrate our algorithms in details.

3. ALGORITHM

In this section, we propose our ranking-based name matching algorithm following the intuition that it is more likely for string-based similar names to refer to the same author entity if they have higher similarity score inferred from the non-string-based information of the data set.

With this idea, as introduced in last section, we study two crucial aspects of the algorithm that are closely related to the performance measure: recall r and precision p. The r-

step aims to find similar name strings and corresponding author IDs, which is called string-based *name matching*. The p-step aims to accurately infer the true candidates with a *ranking-based* merging method based on meta-paths. These two steps are integrated to an iterative framework that can effectively identify the real duplicates of authors. In addition, the pre-processing and post-processing steps will be carefully explained to futher improve the results.

3.1 Pre-processing

Before going deeper to the p- and r-steps, we need to first clean the data to recover part of author names. We observe that many authors' names are noisy due to the imperfectness of the parser when obtaining the name strings from the raw bibliographic data. Fortunately, under the assumption that noises introduced by the parser can be diverse and the majority name strings from the parser are correct and consistent, we are able to recover the real names by learning such consistency from the dataset itself.

Here we list two main types of noises we successfully detected and recovered.

Noisy First or Last Names: Some examples of noisy names belonging to this type are *Eytan H. Modiano* and *Eytan Modianoy*, *Nosrat O. Mahmoodo* and *Nosrat O. Mahmoodiand*, *UniversityofBonn Andreu Mas Colell* and *Andreu Mas Colell*. To recover the correct first and last names from the noisy observations, we first build statistics of single name units. Later for rare last and first names, we compare them with the possible sub-strings by discarding the characters at the beginning or end of the name unit. If theses sub-strings appear frequent in the data set, we believe these name strings should get recovered. For instance, if name unit "Modianoy" only appears one time but "Modiano" appears many times, it is likely that "Modianoy" should be replaced by "Modiano".

Mistakenly Separated or Merged Name Units: Name units are defined as elements split by whitespace or other punctuation marks like dash as delimiters. Sometimes, different name units are mistakenly separated or merged in the data set. Examples include *Sazaly Abu Bakar* and *Sazaly AbuBakar*, *Vahid Tabataba Vakili* and *Vahid Tabatabavakili*. To handle this kind of noise, similar to the previous type, we still build statistics of name units. But this time co-occurrences of name units appearing within the same author name are recorded. By referring to the times of appearance of concatenated name units, we are able to separate or merge name units if they are originally mistakenly merged or separated.

3.2 The r-Step: Improving the Recall

Improving the recall of the algorithm means that given an author ID, one should find as many potential duplicates as possible. In our model, we propose to enhance the recall via two main categories of considerations. The implementation details for efficiency consideration are also provided.

3.2.1 String-based Consideration:

The first category is purely based on the name string similarity. We consider two names to be potential duplicates once they have close distance defined based on the strings.

Levenshtein Edit Distance: Levenshtein edit distance [7] between two strings is the minimum number of single character edits required to change one string into the other. Thus, we can see that two strings of name are similar if they have small edit distances. For example, misspelled names will have a small edit distance and thus they are similar.

Soundex Distance: Soundex algorithm [4] is a phonetic algorithm that indexes words by their pronunciation in English. For example, names like "Michael", "Mickel" and "Michal" share the same Soundex distance because their pronunciation are very close to each other. Besides the Soundex algorithm, there are other phonetic algorithms available like Metaphone Coding Method [6] and Guth Name-Matching [3].

Overlapping Name Units: Two strings are viewed as similar if one name string shares most of its name units with the other. For example, name strings like: *Wing Hong Onyx Wai* and *Onyx Wai Wing Hong* are viewed as similar because each unit in one string is identical to another, the only difference is in the order of the name units.

3.2.2 Name-Specific Consideration

The second category relies on human knowledge about the rules of names and is difficult to be inferred from the above string-based similarity.

Name Suffixes and Prefixes: Name prefixes like "Mr" and "Miss", generational titles like "Jr", "I", "II" are ignored during name comparison. For example, *Frederick P. Brooks Jr* and *Frederick P. Brooks* are considered identical because the only difference between them is the generational title.

Nicknames: We search for common nicknames from the world wide web to form our nickname knowledge base because some researchers prefer to use nicknames and original names for different papers. It is worth noting that the mapping between original names and nicknames is not transitive. For example, "Chris" could be a nickname of "Christian" or "Christopher" but "Christian" will not be compatible with "Christopher".

Name Initials: Due to the diverse format of research papers, it is not surprising to see name initials used in title, in the content, and/or the citation of research papers. For example, *Kevin Chen-Chuan Chang* could be written as name initials *K. C.-C. Chang*, *Kevin C. Chang*, and others. One thing that one needs to be careful with is that some initials refer to the same name unit even if they are not the same. One example is that "B." and "W." is compatible sometimes because "Bill" is the nickname of "William".

Asian Names and Western Names: Asians have different name structure from westerns. One significant feature is that Asian names usually lack the middle names and their first and last names could contain more than one name unit where for Western names each unit represents one meaningful part of the whole name. To see the differences, let us take two names as our example: *Andrew Chi-Chih Yao* and

Michael I. Jordan. For *Andrew Chi-Chih Yao*, the first name is "Andrew Chi-chih" or "Chi-Chih" and last name is "Yao". But for *Michael I. Jordan*, the first name is "Michael", the middle name is "I." and the last name is "Jordan". Thus, we cannot use the same approach for names from different regions. In order to differentiate Asian names and Western names, we utilize a list of common Asian names as a knowledge base to detect them first. Then for Asian names, we define some rules to extract first/last names, and for Western names, we define another set of rules to extract first/middle/last names. Additionally, Asian and western names usually have totally different settings for the above mentioned ideas. For instance, the thresholds for two name strings to be viewed as similar in terms of edit distance are different. *Mike Leqwis* can be safely assumed to be a noisy variant of *Mike Lewis* by an extra "q", while this is not the case for Chinese names *Wei Wan* and *Wei Wang*, as both "Wan" and "Wang" are valid last names. Besides this, Soundex distance is designed for English only and cannot even be applied to Asian names.

3.2.3 Practical Considerations in RankMatch

The ideal way to make use of the ideas discussed above is to process any pairs of author IDs in the dataset which is of time complexity $O(n^2)$. One can use map-reduce to handle this data scale, but due to the limited resources, we resort to explore efficient implementation on a single machine even though we must sacrifice some performance.

One obvious operation is to compare among author names instead of author IDs by building the one-to-many mapping between author names and IDs. Later, our idea for decreasing time complexity is to reduce the search space of potential duplicates for each name string. Generally, we borrow the idea from SimHash[1] that potential similar strings can be mapped to the same hash bin with a large probability. However, we do not use SimHash directly because it is originally designed for long texts and our name strings are too short to be hashed in order to achieve good performance. Therefore we design our own hashing method by projecting each author name in two kinds of bins: name initials and individual name unit. We assume that potential duplicates could share the same initials or at least one name unit in most cases.

Of course this way of mapping from names into initials and name units for finding potential duplicates is "lossy" and we may miss some real duplicates. Thus we incorporate the transitive rule, i.e., if name string a is similar to b and b is similar to c, then the name pair a and c needs to be checked to see whether they are similar or not.

3.3 The p-Step: Improving the Precision

Once we have found potential duplicates for each candidate author name, the next critical task is to infer the real author entity shared by one or more author IDs. Consider the following toy example as in Table 1: Simply based on the string-based name matching, it is clear that authors 2 and 3 cannot be the same person. However, it is impossible to infer the accurate matching between them and their shared author IDs, i.e., 1, 4 and 5. As we can see authors 2 and 3 co-occur in the matched author lists of authors 1, 4 and 5, we call this phenomenon **conflict** because they themselves are clearly not the same author.

Table 1: Lists of matched author IDs.

Name	Author ID	Matched Author IDs
Michael Lewis	1	2, 3, 4, 5
Michael J. Lewis	2	1, 4, 5
Michael P. Lewis	3	1, 4, 5
Michael Lewis	4	1, 2, 3, 5
M. Lewis	5	1, 2, 3, 4

Additionally, it is questionable for considering two author IDs to be the duplicates even though they share the same or similar names. It is quite possible that authors 1 and 4 in the above case point to different researchers, reflecting the **uncertainty** lying in the data set.

Now we propose a ranking-based method in cascaded stages to solve these two problems simultaneously.

3.3.1 Meta-Path-based Similarity

In the network schema extracted from the data set depicted in Figure 2, two authors can be connected via different paths. For example, two authors can be connected via "author-paper-author" path, "author-paper-venue-paper-author" path, and so on. Intuitively, the semantics underneath different paths imply different similarities. Formally, these paths are called meta-paths[9], defined as follows.

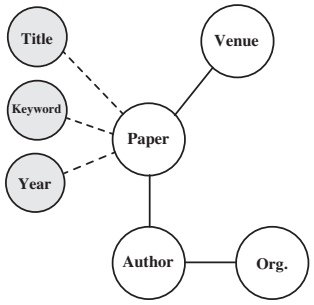

Figure 2: Schema of the Microsoft Academic Search data set. Circles in grey and dashed lines represent attributes belonging to the object.

Definition 1. **Meta-path**. A meta-path $\mathcal{P}$ is a path defined on the graph of a network schema $(\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_{1,2}} A_2 \xrightarrow{R_{2,3}} \cdots \xrightarrow{R_{l-1,l}} A_{l+1}$, which defines a composite relation $\mathcal{R} = R_{1,2} \circ R_{2,3} \circ \cdots \circ R_{l-1,l}$ between types A_1 and A_{l+1}, where $\circ$ denotes the composition operator on relations.

For simplicity, we also use type names denoting the meta-path if there exist no multiple relations between the same pair of types: $\mathcal{P} = (A_1 A_2 ... A_{l+1})$. For example, in this competition data set, the co-author relation can be described using the length-2 meta-path $Author \xrightarrow{writing} Paper \xrightarrow{written-by} Author$, or short as APA if there is no ambiguity. Other types in this data set include K for keyword, Y for year, T for titles, V for venues (conferences/journals) and O for organizations/affilications.

Once the topologies defined by meta-paths are determined, the next stage is to propose measures on these meta-paths to compute similarities for paired authors. The reason for computing these meta-path-based similarities is to obtain knowl-

edge beyond string-based name matching and guide us later to distinguish name entities among the candidate pool. In terms of similarity measures, we use the normalized count of the path instances due to its efficiency. Other possible measures between any two objects given the meta-path include random walk-based measures, SimRank[5], PathSim[9] and so on.

For a single relation in $R \in \mathcal{R}$, the measure matrix is an adjacency matrix M of the sub-network extracted by R. Given a composite relation in a meta-path, the measure matrix can be calculated by the matrix multiplication combined with normalization of the partial relations.

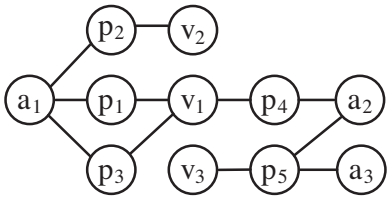

Figure 3: Illustration for computing meta-path-based similarity.

Assume we are given a tiny sub-network as shown in Figure 3 composing three authors, five papers and three venues. It is easy to check the adjacency matrix $M_{A,P}$ and $M_{P,V}$ depicting the relations for Author-Paper and Paper-Venue are:

$$M_{A,P} = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad M_{P,V} = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Then, the measure matrix between Author and Venue is

$$\overline{M_{A,V}} = Normalize(M_{A,P} \times M_{P,V})$$

Here $normalize(\cdot)$ is applied to make the input matrix ℓ_2-normalized such that the self-maximum property[1] can be achieved.

Now the measure matrix between authors based on meta-path $APVPA$ is

$$\overline{M_{A,A}} = \overline{M_{A,V}} \times \overline{M_{A,V}}^T = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\ \begin{bmatrix} 1.0000 & 0.6325 & 0 \\ 0.6325 & 1.0000 & 0.7071 \\ 0 & 0.7071 & 1.0000 \end{bmatrix} \end{array} \tag{1}$$

From now on, the similarity scores for any pairs of authors can be simply referred to by looking up the corresponding matrix values.

To support multiple meta-paths, we adopt the linear combination strategy:

$$Sim(a_i, a_j) = \sum W_{path} Sim_{path}(a_i, a_j)$$

where W_{path} is a positive weight for the specific *path*. The larger the weight is, the more dominant the path will be.

[1]Self-maximum: $Sim(a_i, a_j) \in [0, 1]$, and $Sim(a_i, a_i) = 1$.

Until now, we obtain the similarity score for all pairs of authors in the data set independent of the string-based similarity and we are ready to combine it with the previous results generated by the r-step.

3.3.2 Ranking-based Merging

To start with, recall that for each author ID we have generated a list of matched author IDs in the r-step. Examples can be seen in Table 1. We now reorganize them as a set of author ID pairs. For simplicity we only consider IDs 1, 2 and 3. Thus, the set generated from Table 1 should be $((1,2),(2,3),(1,3))$. By looking up the similarity score of all the ID pairs from Eq. 1 in the set, sorting is applied to them as demonstrated in Table 2.

Table 2: Rank of author ID pairs.

Author ID Pair	Similarity	Rank
$(1,2)$	0.6325	2
$(1,3)$	0	3
$(2,3)$	0.7071	1

Then we do a scan from the top ranked ID pair to the lower ranked ones to help infer the author entity. Let us show this for the example we have. The highest similarity is from pair $(2,3)$. Because the names of these two IDs are in conflict, we skip this candidate. Now the next pair is $(1,2)$. As this is the first pair we encounter which not only has high similarity but also passes the name matching comparison, we believe these two IDs having high probability to be the real duplicate. Here comes the interesting part for the last pair $(1,3)$. Although its similarity is 0 which makes it no sense for us to consider ID 1 and ID 3 to be the same person, they are still viewed as possible duplicate and let us see how our algorithm responds: When pair $(1,3)$ comes, our algorithm will check the existing duplicates from those higher ranked pairs, i.e., $(1,2)$ from the last operation. By testing whether there is conflict within the merged ID group $(1,2,3)$ of $(1,2)$ and $(1,3)$, the algorithm is able to reject $(1,3)$ because ID 2 and ID 3 are in conflict. As a result, sets $(1,2)$ and (3) are returned as two duplicate author ID groups.

Conflict and uncertainty are two main problems for the author disambiguation task as introduced at the beginning of this subsection. However, the above algorithm is still not able to solve them perfectly especially for uncertainty. Specifically, it is not able to differentiate compatible names when they have low similarity. One simple patch is to define a threshold such that once both of the two IDs have multiple publications and low meta-path-based similarity score, the algorithm can reject their merging request.

Another important strategy is to expand the author names corresponding to the IDs once we are confident about two IDs to be the duplicate. For example, as authors 1 and 2 are highly possible to be the same person and the name of author 2 has better quality than that of author 1, we can replace the name of author 1 to be *Michael J. Lewis*. This idea is useful because it can help avoid the mistakenly detected conflicts. Suppose the full name of author 2 to be *Michael James Lewis* and we have a new author with name *James Lewis*. If we do not adopt this name expanding mechanism, obviously author 1 and this new author are in conflict and

Table 3: Test cases for name matching.

Author Name A	Author Name B	Expected Result
Jiawei Han	*Jia Han*	In Conflict
Xiang Li	*Xiang Lin*	In Conflict
Gordon D. Moskowitz	*Gordon Blaine Moskowitz*	In Conflict
H. Murray-Rust	*D. M. Murray-Rustt*	In Conflict
Deliang L. Wang	*Liang Wang*	In Conflict
Takeshi Mori	*Taketoshi Mori*	In Conflict
Tadashi Suzuki	*Takashi Suzuki*	In Conflict
Hong-Hu Zhu	*H. H. Zhu*	Compatible
Ralph Mac Nally	*RalphMac Nally*	Compatible
V. Scott Gordont	*V. Scott Gordon*	Compatible
Jeff W. Hughes	*Jeffrey W. Hughes*	Compatible
William Hughes	*Bill Hughes*	Compatible
William Hughes	*B. Hughes*	Compatible
Valli Kumari Vatsavayi	*V. Valli Kumari*	Compatible
Mercedes Femandez-Redondo	*Mercedes Fernandez Redondo*	Compatible
Aliaa Abdel-Haleim Abdel-Razik Youssif	*Aliaa A. A. Youssif*	Compatible

the recall will be affected[2].

3.3.3 Practical Considerations in RankMatch

As there is no validation set provided by the host, we selected meta-paths and corresponding weights simply based on our prior knowledge. The selected meta-paths are APA, AOA, $APAPA$, $APVPA$, $APKPA$, $APTPA$ and $APYPA$. The weights for them are decreasing progressively.

As measure matrices such as $\overline{M_{A,A}}$ are decided by the data set itself, they are computed offline in consideration of efficiency. For titles of papers as well as organizations, we view them as bags of words and compute the similarity based on the number of overlapped single words between two authors. The TF-IDF idea is applied to organizations and titles to remove the top frequent words appearing in most of the phrases.

3.4 Post-processing

Unconfident duplicate author IDs should be removed even though their names are compatible and their meta-path-based similarity scores are acceptable. This step is crucial in that the later iterative framework requires highly confident output to gradually refine the results. It is also tricky since through empirical study we realize it is difficult to define "unconfident".

As a result, we only introduce our heuristics and try to leave the flexibility of the strategy to be tunable. In this way, one can be as aggressive as possible if highly confident output is needed in the case of iterative refinement. Meanwhile, one can also be conservative to remove the most unconfident duplicates to achieve the best performance in the last iteration[3]. To begin with, we only focus on small duplicate groups. The reason underneath is that within the large group, the information usually is rich and most unconfident duplicates have already been removed in the p-step. Next we define "unconfident" to have two factors: the difference between name strings in terms of unmatched name units to be large and the meta-path-based similarity score to be small. This is relying on both measures: string-based and meta-path-based, different from the previous r-step and p-step which treat the problem in individual pipelines.

3.5 Iterative Framework

Assume we are able to find highly confident duplicate author IDs using the aggressive strategy introduced in the last subsection, what can we do with this to further improve the performance? The answer is clear: An iterative framework which takes the detected duplicates of the last iteration as part of the input. There are two reasons to do this:

First, we are able to generate much better meta-path-based similarity scores by adding up the elements of the duplicate author IDs in the adjacency matrices describing different relations. This definitely can help the p-step to increase the precision.

Second, the other useful aspect led by this framework is related to the r-step. Recall the name expansion module introduced at the end of the p-step. One shortcoming of that module is the irreversibility of the conflict detection. That is, once we detect conflicts within a group of author IDs, the merge must be rolled back even though later the conflicted name strings are expanded to be compatible. Suppose *Michael Lewis* and *James Lewis* are the names of the top-ranked IDs and get skipped, even though later both of them are expanded to *Michel James Lewis*, they have no chance to get merged again. However, if we adopt this iterative framework and expand author names at the very beginning, it is impossible to miss this performance gain.

4. EXPERIMENTS

We implemented our algorithm in Python and the code repository of RankMatch is on Github[4]. We have tested it on a desktop with Intel Core I7 2600 and 16GB memory. It takes about one hour for that machine to finish one iter-

[2]Though this strategy is closely related to the recall, we consider it to be appropriate to appear in this subsection due to the context.

[3]The details of the iterative framework will be discussed in Section 3.5.

[4]https://github.com/remenberl/KDDCup2013

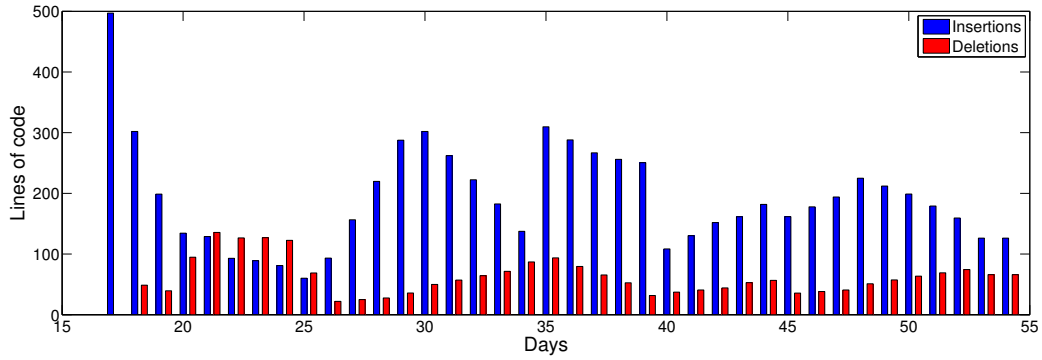

Figure 4: The smoothed everyday code frequency within the competition period.

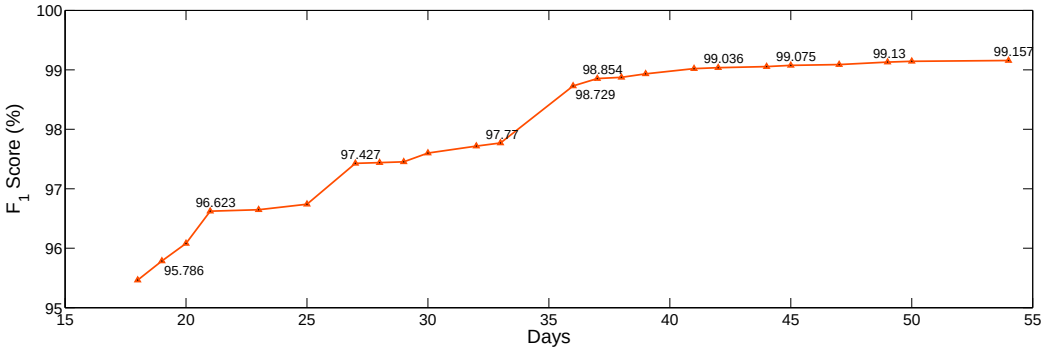

Figure 5: The performance of the submission files within the competition period.

ation. For detailed experimental settings, interested readers can refer to the code.

As no additional testing sets can be directly obtained, we treat this competition as unsupervised. Before every submission we simply conduct manual checking of the results. In addition, we have a name conflict test to ensure our name matching system works reasonably well. Here we list a subset of test cases in Table 3 to show how well our model can perform.

It is also worth mentioning that the data set provided by Microsoft Academic Search contains additional training and validation set for the Track I competition. But unfortunately we have not found an effective way to utilize them.

We also draw the histogram of code frequency extracted from the git-commit history as in Figure 4. The corresponding line chart of the performance was plotted according to the submission log in Figure 5. Table 4 gives a brief introduction of some important commits and one can relate it to Figures 4 and 5 to have an impression about their workloads and performance gains. Note that some important modules such as Ranking-based Merging and Levenshtein Edit Distance are not incorporated in the table due to the fact that we implement and refine these modules within a relative longer time period for several times. This makes it difficult to show statistics for the mentioned modules as we are short of the ground truth labels. For the same reason, we cannot ensure the statistics of the current list of modules in Table 4 to be very accurate.

5. DISCUSSION

Through thorough analysis and experiments of the data set, we realize that the competition is really challenging not only due to the absence of training data to help fit our model but also because the "ground truth" based upon which contestants' submissions are evaluated is not always correct.

Moreover, we feel there are several promising directions that can further refine the solutions and we regret not to try and implement them. Here we hope to share the ideas with you.

To start with, we could make use of machine learning techniques to train a classifier for detecting duplicate author ID pairs. The features for this classifier can be extracted from existing modules in our algorithm like edit distance and meta-path-based similarity scores. The name expansion and iterative ideas can also be adapted to improve this classifier. However, this classifier may need a large number of labels and is impracticable for us to implement within the time limit of this competition.

Another direction could be to support multiple name strings for an author ID. Again due to the time limit of this competition, our implementation only allows one author ID to be mapped to one name string. This is because we expect to expand every author ID's corresponding name string to have high quality and to build a powerful name matching function by using any kinds of ideas introduced in previous sections. Now it seems to us that if time allows, a one-to-many mapping between author ID and name strings could achieve better recall and is easier for development.

Table 4: Module descriptions and corresponding performance gains.

Performance	Gain	New Module(s)	Days
95.376	-	Same Author Name Benchmark	-
95.786	0.410	+ Meta-path: Coauthor	19
96.623	0.837	+ Name Initials, Omitted Middle Name	21
97.427	0.804	+ Meta-path: Covenue	27
97.770	0.343	+ Nicknames + Asian names handling	33
98.729	0.959	+ Accepting name-compatible author pair even with zero meta-path-based similarity	36
99.020	0.291	+ Name reordering + noisy last/first name pre-processing	37
99.036	0.016	+ Rough post-processing	42
99.075	0.039	+ SoundEx distance	45
99.130	0.055	+ Name units breaking/merging pre-process, + name expansion	49
99.157	0.027	+ Iterative framework + more aggressive post-processing	54

The last direction is to stick with improving the current modules. For example, the SoundEx algorithm only supports English names and we should find algorithms with similar functions to support all kinds of world languages. Secondly, more language specific settings could be designed. Currently we only support Mandarin, Taiwanese, Cantonese and Korean as discussed in Section 3.2. The similar ideas can be extended to Indian, Japanese, Arabic and some western languages like French, German, Russian and so on. Thirdly, within the iterative framework, we can model on all the authors appeared in the data set instead of a subset which is requested in the submission file for evaluation. Finally, better models of affiliation and keywords could be designed to replace the current bag-of-word model by TF-IDF.

6. CONCLUSION

In this paper, we have proposed our ranking-based name matching algorithm and system RankMatch to author disambuigation task on the Microsoft Academic Search data set. We have developed an iterative framework mainly composing two steps. The first step consists of a set of string-based name matching mechanisms for increasing recall and the second step constains a ranking-based merging technique based on meta-paths for the sake of precision. Experimental results on the data sets demonstrated the effectiveness of the various techniques proposed by this paper.

Acknowledgement

We would like to acknowledge SIGKDD for organizing this exciting and inspiring competition. We also thank Microsoft Academic Search and Kaggle for their generous supports with data, platform and all kinds of resources. We benefited a lot from the discussions with the data mining group at UIUC and Jing Gao from University at Buffalo.

This work was supported in part by U.S. National Science Foundation grants CNS-0931975 and IIS-1017362, the U.S. Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA) and W911NF-11-2-0086, DTRA, MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC, and a Microsoft Graduate Fellowship. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

7. REFERENCES

[1] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[2] A. Culotta, P. Kanani, R. Hall, M. Wick, and A. McCallum. Author disambiguation using error-driven machine learning with a ranking loss function. In *Sixth International Workshop on Information Integration on the Web (IIWeb-07), Vancouver, Canada*, 2007.

[3] G. J. A. Guth. Surname spellings and computerised record linkage. *Historical Methods. Newsletter*, 10(1):10–19, 1976.

[4] D. O. Holmes and M. C. Mccabe. Improving precision and recall for soundex retrieval, insertions and reversals. In *Proceedings of International Symposium on Information Technology*, pages 22–26, 1995.

[5] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings of eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543, 2002.

[6] P. Lawrence. The double metaphone search algorithm. *C/C++ Users Journal*, 18(6):38–43, 2000.

[7] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

[8] S. Roy, M. De Cock, V. Mandava, S. Savanna, B. Dalessandro, C. Perlich, W. Cukierski, and B. Hamner. The microsoft academic search dataset and kdd cup 2013. In *KDD-Cup Workshop*, 2013.

[9] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

[10] P. Treeratpituk and C. L. Giles. Disambiguating authors in academic publications using random forests. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 39–48, 2009.

[11] X. Yin, J. Han, and P. Yu. Object distinction: Distinguishing objects with identical names. In *Proceedings of 2007 IEEE International Conference on Data Engineering*, pages 1242–1246, 2007.