Robust and Distributed Web-Scale Near-Dup Document Conflation in Microsoft Academic Service

Chieh-Han Wu
Microsoft Research, Redmond
One Microsoft Way
Redmond, WA, USA
chiewu@microsoft.com

Yang Song
Microsoft Research, Redmond
One Microsoft Way
Redmond, WA, USA
yangsong@microsoft.com

ABSTRACT

In modern web-scale applications that collect data from different sources, entity conflation is a challenging task due to various data quality issues. In this paper, we propose a robust and distributed framework to perform conflation on noisy data in the Microsoft Academic Service dataset. Our framework contains two major components. In the offline component, we train a GBDT model to determine whether two papers from different sources should be conflated to the same paper entity. In the online component, we propose a scalable shingling algorithm that can apply our offline model to over 100 million instances. The result shows that our algorithm can conflate noisy data robustly and efficiently.

KEYWORDS

Near-duplicate detection, shingling algorithm, n-gram, entity conflation

1. INTRODUCTION

Microsoft Academic Service (MAS) is a Microsoft's effort on the scientific article vertical search area which has been publicly available since 2008. The service was originally deployed as a research prototype[1], which was meant to study the relationship between different entity types such as authors, venues, affiliations, field-of-study, and so on. Recently, it has been integrated into Microsoft's Web search engine Bing as a vertical to showcase the power of Bing's dialog engine systems as well as the rich collection of scientific documents that can help academic researchers to facilitate their jobs on literature search and review.

MAS collects data primarily from three different sources. The large majority of the data comes from Web crawl of HTML meta tags. Most publishers (e.g., ACM, IEEE, ScienceDirect) follows Google Scholar's inclusion guidelines[2] to add specific meta tags on the paper's summary pages. E.g., using "citation_title" and "citation_author" to indicate the title and (one) author name of the current paper. Secondly, the MAS pipeline parses meta data from PDF files using a proprietary extractor to get the header information as well as the reference section inside PDF documents. Finally, MAS periodically gets bulk updates from certain publishers that often includes the meta data as well as the abstract and full text information of the recently published papers.

Table 1 lists the overall statistics of MAS as of July 2015. With over 100 million collections of papers collected from heterogeneous data sources, it is inevitable that many papers are duplications of each other. For example, we discovered that the famous topic-modeling paper "Latent Dirichlet Allocation" has 50 different versions in our data, while the ground-truth should only be 2 (one published in JMLR and the other in NIPS). Therefore, correctly conflating these different versions into the right clusters of papers becomes one of the most important data quality task.

Attribute Name	Count
Affiliation	3661025
Author	41863978
Conference	46757
FOS	52530
Journal	22794
Keywords	35938281
Title	118891522
Paper	126333772

Table 1: Statistics of different attributes in MAS.

Nevertheless, the variance of data quality from different publishers and domains are huge. We category the data quality issues into three aspects: missing data, ill-formatted data and bad data issues.

(**Missing data**): Table 2 lists the fields of meta data that may appear in a paper. We can observe that except title, authors and year[3] most other fields suffer from missing data. For example, DOI is the most important indicator of a unique paper, with only 56% percentage of DOI coverage, many of the papers cannot benefit from a simple DOI mapping-based conflation algorithm.

Attribute	Missing Rate
Affiliation	77.08%
Keyword	56.52%
Doi	53.56%
Venue	21.45%
Date	12.70%
Author	0.00%
Title	0.00%
Year	0.00%

Table 2: Statistics of missing attributes in MAS.

1. http://academic.research.microsoft.com/
2. https://scholar.google.com/intl/en/scholar/inclusion.html#indexing
3. These three fields are required by the Google Scholar standard for publisher's papers to be included.

(**Ill-formatted data**): data can become ill-formatted if the publishers do not follow the inclusion guidelines correctly or by using tools that format data in arbitrary ways. For example, many publishers choose to aggregate author names into one meta tags using arbitrary string concatenators (semi-colon, comma, "and" etc.) rather than separating them into individual ones as told. Other issues including the encoding of the text, the format of the Date field, the succinctness of author's affiliation and etc.

(**Bad data**): this is by far the biggest issue MAS is facing when conflation papers. First of all, the assumption that the same papers should have exactly the same titles from different publishers is absolutely untrue. This mostly comes from typos or other bad separators in the paper titles. Given that, choosing a fuzzy string matching method and determining the correct cut-off threshold becomes quite important. Secondly, many of the crowd-sourced websites like researchgate and citeulike have no quality control of the user generated content. We discovered that on these websites, users can specify almost any meta tags at their own will: including specifying the year of the publication, the list and the order of authors in the papers, etc. These misleading information creates extra road-blocks for MAS to correctly identify same papers.

Given all those data quality issues and the importance of near-dup detection (or paper conflation) in MAS pipeline, we spent a lot of effort cleaning and organizing the data into correct format. In this paper, we propose a robust framework to perform near-dup detection given noisy input data. In general, our framework can be separated into two stages. During the offline model generation stage, we propose a machine-learning algorithm with manually crafted features to optimize the conflation performance. Our model leverages a set of hand-labeled training data that is sampled randomly from MAS data collection to make binary decisions of whether two papers should be conflated. Secondly, in order for our model to scale to over 100 million instances, we partition our data into many small chunks using a novel shingle-based partition mechanism. For each chunk that contains potential near dups, we then run our machine-learnt models on each pair and generate the final conflated papers. This step was carried out in a map-reduce like environment inside Microsoft.

2. RELATED WORK

The problem we are addressing in this paper is highly correlated to near-dup detection. In this section, we briefly review some of the important literature work.

Andrei Border [1] proposed a set-based shingling algorithm to detect near-duplicated documents on the Web. The algorithm converts 4-grams into unique ids and uses Jaccard similarity to measure the resemblance of two documents. Each document is represented using a 50-bytes vector.

Moses Charikar [2] characterizes Border's work as min-wise independent permutations and shows that it is equivalent to locality sensitive hashing (lsh) scheme. In his work, Charikar proposed to construct lsh-based document signatures that contains only 0's and 1's. This method allows the documents to be summarized into a much succinct fashion: resulting roughly only 64 bits for each documents. The document signatures are then compared using hamming distances and treated as near-dup if the distances is less than k bits.

Manku et al. [3] leveraged Charikar's simhash algorithm to detect near-duplicates in Google's Web index of multi-billion documents. They first show that using 64-bit simhash fingerprints, setting k-3 can effectively detect near-duplicates. The authors then proposed an algorithm to scale the algorithm which can efficiently compute the hamming distances among billions of documents. Their algorithm performs random permutation of the fingerprint blocks by splitting it into different sized blocks (10 bits, 16 bits…) and duplicating the process to create redundant tables for each setting. It was shown that this process can reduce the scanning time dramatically and thus makes the algorithm practically useful.

Our approach is similar to the previous work in the sense that we also leverage n-grams to construct document shingles. However, the biggest difference comes from the data processing architecture. Due to the nature of the near-duplicate detection problem, most of the previous algorithms work on a non-distributed fashion, which means that none of them can be efficiently deployed to a map-reduce system which can process massive amount of the data in parallel. Our method, on the other hand, address the limitation and can be easily deployed to any map-reduce systems like Hadoop or Spark and deal with billions of documents efficiently.

Furthermore, previous algorithms work well when the full-text of the documents is accessible. When it comes to only short titles, the Jaccard similarity can be very low even if two titles differ in a single character, which may fail to detect the near duplicate. In comparison, our approach is more robust and capable of addressing short titles effectively even if full text is unavailable.

3. BASELINE APPROACH

A simple conflation approach is to leverage paper's title and year for matching. However, due to different data quality issue, this will cause both over-conflation and under-conflation.

Table 3 lists some example papers for better understanding this issue. The "Latent Dirichlet Allocation" paper was published in both NIPS (in 2002) and JMLR (in 2003) with the same title. However, due to the Year error in the meta tag, paper ID 2 and 4 will be treated as the same paper despite of different venue, which causes over-conflation. On the other hand, paper 1 and 2 will be treated as different papers due to different years, so does paper 3 and 4.

One may argue that adding venue as an additional feature may solve the problem, which is unfortunately not the case for two major reasons:

1. Many of the papers do not possess venue information as we shown in Table 3, which makes it difficult to be used as a grouping criteria.
2. Besides year, other fields such as paper titles may also have issues, e.g., PDF parsing errors. In Table 3, paper 5 and 6 have the same year and venue information but the titles are slightly different due to

the use of different parsers, which will cause them to be under-conflated.

ID	Title	Year	Venue
1	Latent Dirichlet Allocation	2003	JMLR
2	Latent Dirichlet Allocation	*2001*	JMLR
3	Latent Dirichlet Allocation	2002	NIPS
4	Latent Dirichlet Allocation	*2001*	NIPS
5	gapped blast and psi blast a new generation of protein database search programs	1997	NAR
6	gapped blast and psi blast a new generation of protein database searchprograms	1997	NAR

Table 3: Some example papers with data quality issues. Underlined words and years are noisy data.

4. PROPOSED FRAMEWORK

In this section, we discuss how we train a model to determine whether two papers should be conflated together, and then introduce how we apply the model to a large collection of papers.

4.1 A Machine Learning Framework

We formulate the near-dup detection problem as a binary-classification problem: given two papers, determine whether they are the same paper from different sources or not. To train a machine learning model, we first randomly sampled a set of paper pairs from the entire MAS data preserve the distribution of the original data set. We then manually labels each pair to be positive or negative. Overall, we generated 1370 positive pairs and 8641 negative pairs for training.

We extract the following four kinds of feature from a pair of papers. Each feature value is between 0 and 1.

(**AuthorSimilarity S_A**): Name disambiguation has been a challenging question. We leverage a modified version of string matching algorithm proposed by Chin et al. [4], which considers Chinese and non-Chinese names separately because of their different naming conventions, to determine whether two authors share the same name. We then compute the Jaccard similarity between two author sets.

(**TitleSimilarity S_T**): this calculates the normalized Levenshtein distance by dividing the distance by the maximum length of two titles.

(**VenueSimilarity S_V**): this compares whether two papers share the same venue: 1 if venues are matched, 0 if not matched. Since many venues are missing in the raw data, we impute this feature to be 0.5 if either one of the venues is missing.

(**YearSimilarity S_Y**): this computes the distance between the publication years of two papers. Since years are messy in our dataset, instead of expecting years to be exactly matched, we assign S_Y to be 1 if the year difference is within 3 years, otherwise 0.

We use the gradient boosted decision tree (GBDT) [5] as our classifier. GBDT has shown to be a powerful tool for classification and ranking in many tasks. Besides, the GBDT model is quite interpretable since each sub-model is a classification tree that can be visualized and understood easily.

For the model training, we first split the data randomly into 4-fold for cross validation. We then performed a grid search over the parameters of the model, including: the number of leaves N, the number of trees/iterations M, minimum number of documents in a node D and the learning rate L. The best performance was achieved with the parameter setting of N = 8, M = 50, D = 10, and L = 0.1.

For efficiency concern during online computation, we also computed the performance of a single-tree model. The results shown in Table 4 have indicated a slight worse accuracy but not significant. Therefore, we have decided to use the single-tree model for our map-reduce framework. The final model can be summarized as Figure 1. Note that S_Y doesn't appear in our final model, which truly represents the noise of year extraction from our data.

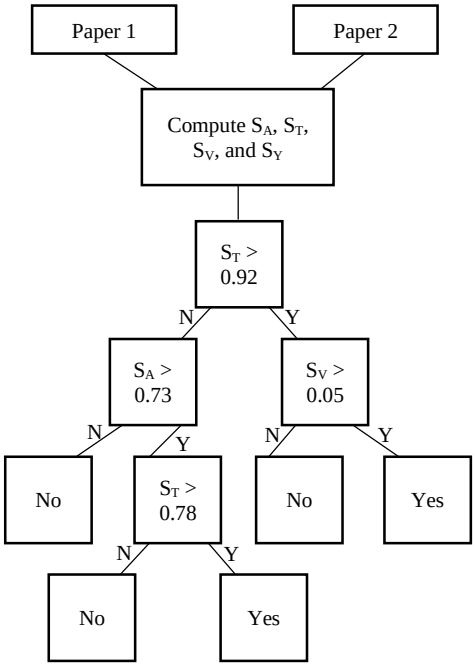

Figure 1: The single-tree model used in our map-reduce framework

M	Accuracy	Positive Precision	Positive Recall	Negative Precision	Negative Recall
50	0.9805	0.9683	0.9821	0.9895	0.9794
1	0.977	1	0.9387	0.9654	1

Table 4: Comparison between the best GBDT model and the single-tree model

4.2 Online (map-reduce) shingle algorithm

One of the biggest challenges in our pipeline is how to scale up our model to a large collection of papers. Instead of using brute-force approach to make pair-wise comparisons between all papers which is certain to cause performance issues, we propose a scalable shingling algorithm to partition the data into small near-dup chunks before applying the model.

Given a positive integer k and a sequence of words in a paper title, the k-shingles is the set of all consecutive sequences of k words in the title. The main assumption in our

shingling algorithm is that a noisy title and a correct title would have at least one common k-shingle. We describe this algorithm in detail using k = 5 and the following two titles as examples: a correct title **T_1** "gapped blast and psi blast a new generation of protein database search programs" and a noisy title **T_1'** "gapped blast and psi blast a new generation of protein database searchprograms".

Step 1: Add dummy words in titles

We first preprocess all paper titles by adding dummy words to titles that have less than 2 * k words. Consider the worst case that the noisy word occurs in the middle of the title, for example, the third word of a 5-word title such as "indexing by latent semantic analysis". If we select k = 5, we should add 3 dummy words at the beginning and the end of the title so that the noisy title and the correct one still have common shingles, for example, "# # # indexing by" and "semantic analysis # # #". Table 5 shows the correspondence between the number of dummy words and the length of title. In our examples, since both **T_1** and **T_1'** have more than 9 words, we don't need to add dummy words in the titles.

Title length	Number of dummy word
2	4
3	4
4	3
5	3
6	2
7	2
8	1
9	1
>9	0

Table 5: Correspondence between title length and number of dummy words.

Step 2: Generate k-shingles for each paper

For each paper, we generate a set of k-shingles. For example, the 5-shingles of the correct title **T_1** and the noisy title **T_1'** are shown in table 5. We can observe that even **T_1'** has a noisy pattern "searchprogram", **T_1** and **T_1'** still have common 5-shingles such as "gapped blast and psi blast", "blast and psi blast a", etc.

Paper	Shingle	Common Shingle
T_1	gapped blast and psi blast	gapped blast and psi blast blast and psi blast a … new generation of protein database
	blast and psi blast a	
	…	
	new generation of protein database	
	generation of protein database search	
	of protein database search programs	
T_1'	gapped blast and psi blast	
	blast and psi blast a	
	…	
	new generation of protein database	
	generation of protein database searchprograms	

Table 6: 5-shingles of the correct title T_1 and the noisy title T_1'

Step 3: Aggregate papers for each k-shingle

For each k-shingle, we then group all papers that possess such shingle into the same cluster. For example, if **T_1**, **T_1'**, **T_2**, etc. have a common shingle "gapped blast and psi blast", they should be grouped into the same cluster {**T_1**, **T_1'**, **T_2**, ...}. Papers within the same cluster are considered candidates to be conflated. Table 7 shows the paper cluster and the cluster size of each shingle.

Paper	Shingle	Cluster	Size	Common Shingle
T_1	gapped blast and psi blast	{ **T_1**, **T_1'**, …}	68	gapped blast and psi blast blast and psi blast a … new generation of protein database
	blast and psi blast a	{ **T_1**, **T_1'**, …}	68	
	…			
	new generation of protein database	{ **T_1**, **T_1'**, …}	71	
	generation of protein database search	{ **T_1**, …}	70	
	of protein database search programs	{ **T_1**, …}	69	
T_1'	gapped blast and psi blast	{ **T_1**, **T_1'**, …}	68	
	blast and psi blast a	{ **T_1**, **T_1'**, …}	68	
	…			
	new generation of protein database	{ **T_1**, **T_1'**, …}	71	
	generation of protein database searchprograms	{ **T_1'**}	1	

Table 7: Paper cluster for each shingle of the correct title T_1 and the noisy title T_1'

Step 4: Select the best cluster for each paper

The next challenge is to select the best cluster for each paper. There are two kinds of paper cluster that are considered as bad candidates:

(**General shingle cluster**): a general shingle shared by a lot of titles, such as "to the study of the", has a large paper cluster. If we select a cluster that is too large, it may results in comparing a lot of irrelevant papers and therefore decrease efficiency, which is an important factor especially in a large-scale application like MAS.

(**Noisy shingle cluster**): a shingle that contains noisy patterns, such as "generation of protein database searchprograms", which usually only have a few papers in the cluster. If we select a cluster that is too small, the algorithm may fail to conflate noisy papers with the correct ones since the small cluster may only contains papers with noisy titles.

To prevent selecting these two kinds of paper clusters, we leverage DBSCAN algorithm proposed by Ester et al. [6] to filter out bad candidate. For example, we apply DBSCAN with the following parameter setting to **T_1** and **T_1'**:

1. Density distance eps = 20
2. Minimum number of points to form a cluster = 2

All paper clusters of **T_1** is in the same density region since they are all density-connected. Similarly, all paper clusters of **T_1'** are in the same density region except the paper cluster formed by the noisy shingle "generation of protein database searchprograms", which only has one paper in the cluster and therefore is not density-connected with other paper clusters of

T_1'. As a result, the bad cluster associated with the noisy shingle can be filtered out by leveraging DBSCAN.

Since the clusters will serve as the keys for the reducers performing conflation, two papers in different clusters will never be compared with each other at the conflation stage. At this step, our goal is to assign all near-dup entities to the same cluster. To this end, we select the largest paper cluster (i.e. the cluster associated with "a new generation of protein" in our example) within the density-connected region since it is more likely to be shared by all near-dup entities. After selecting the best cluster for each paper, we can apply our GBDT model in a map-reduce framework using paper clusters as the keys for reducers.

Paper	Shingle	Cluster	Size	Common Shingle
T_1	a new generation of protein	{ T_1, T_1', …}	82	a new generation of protein new generation of protein database gapped blast and psi blast …
	new generation of protein database	{ T_1, T_1', …}	71	
	generation of protein database search	{ T_1, …}	70	
	…			
	blast a new generation of	{ T_1, T_1', …}	69	
T_1'	a new generation of protein	{ T_1, T_1', …}	82	
	new generation of protein database	{ T_1, T_1', …}	71	
	gapped blast and psi blast	{ T_1, T_1', …}	68	
	…			
	generation of protein database searchprograms	{ T_1'}	1	

Table 8: Sorted paper cluster of the correct title T_1 and the noisy title T_1'

5. RESULTS AND DISCUSSION

In our experiment, the running time of the baseline algorithm and the shingling algorithm is roughly 73 minutes and 169 minutes, respectively. The time gap is caused by the fact that the shingling algorithm first partitions all papers to different paper clusters and then uses paper clusters as the keys for reducers to perform conflation, while the baseline algorithm uses title and year directly as the keys for reducers.

With our shingling algorithm, we successfully reduce the under-conflation rate from 12% to 0.7%. We mainly address the under-conflation issue caused by the following two reasons:

(**Noisy titles**): Our algorithm can conflate two papers even if their titles are noisy due to typos or other bad separators. Table 9 shows some examples of under-conflated papers in the baseline approach can be correctly conflated together by our framework.

(**Different years**): Due to noisy year extracted from different sources, it is unrealistic to expect that years are the same for the same paper. With the GBDT model that considers all attributes such as author and venue, our algorithm leverage the heterogeneous characteristics of MAS to conflate entity robustly. Table 10 lists the case, which we fail to conflate using the baseline approach due to different years, can be identified as two different entities by our framework (i.e., one entity published in JMLR, the other published in NIPS).

The main advantage of shingling algorithm is that we no longer rely on the correctness of data sources to perform entity conflation, which is a more robust and flexible framework in web-scale applications, such as MAS, that aggregate information from different sources.

6. CONCLUSION

In this paper, we presented an entity conflation framework that consists of two stages: a GBDT model that considers different attributes in the MAS dataset to perform conflation robustly, and a shingling algorithm to make the offline model scalable to millions of entities. Experiments showed that our proposed framework can address various data quality issues, such as noisy titles and incorrect years, which caused serious under-conflation in the baseline approach.

REFERENCE

[1] Andrei Z. Broder, Identifying and Filtering Near-Duplicate Documents, Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, p.1-10, June 21-23, 2000.

[2] Moses S. Charikar, Similarity estimation techniques from rounding algorithms, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, May 19-21, 2002, Montreal, Quebec, Canada.

[3] Gurmeet Singh Manku , Arvind Jain , Anish Das Sarma, Detecting near-duplicates for web crawling, Proceedings of the 16th international conference on World Wide Web, May 08-12, 2007, Banff, Alberta, Canada.

[4] W.-S. Chin, Y.-C. Juan, Y. Zhuang, F. Wu, H.-Y. Tung, T. Yu, J.-P. Wang, C.-X. Chang, C.-P. Yang, W.-C. Chang, K.-H. Huang, T.-M. Kuo, S.-W. Lin, Y.-S. Lin, Y.-C. Lu, Y.-C. Su, C.-K. Wei, T.-C. Yin, C.-L. Li, T.-W. Lin, C.-H. Tsai, S.-D. Lin, H.-T. Lin, C.-J. Lin. Effective String Processing and Matching for Author Disambiguation, Proceedings of KDD Cup 2013 Workshop at KDD2013.

[5] J. H. Friedman. Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5):1189-1232, 2001.

[6] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial database with noise. In Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-96), Portland, Oregon, August 1996.

Baseline Approach			Shingling Algorithm		
Title	**Year**	**Venue**	**Title**	**Year**	**Venue**
gapped blast and psi blast a new generation of protein database search programs	1997	NAR	gapped blast and psi blast a new generation of protein database search programs	1997	NAR
gapped blast and psi blast a new generation of protein database searchprograms	1997	NAR			
gapped blast and psiblast a new generation of protein database search programs	1997	NAR			
clustal w improving the sensitivity of progressive multiple sequence alignment through sequence weighting position specific gap penalties and weight matrix choice	1994	NAR	clustal w improving the sensitivity of progressive multiple sequence alignment through sequence weighting position specific gap penalties and weight matrix choice	1994	NAR
clustal w improving the sensitivity of progressive multiple sequence alignment through sequence weighting positions specific gap penalties and weight matrix choice	1994	NAR			
clustal w improving the sensitivity of progressive multiple sequence alignment through sequence weighting position specific gap penalties and weigh t matric choice	1994	NAR			

Table 9: Comparison of baseline approach and shingling algorithm on noisy titles. Underlined words are noisy data.

Baseline Approach			Shingling Algorithm		
Title	**Year**	**Venue**	**Title**	**Year**	**Venue**
Latent Dirichlet Allocation	2003	JMLR	Latent Dirichlet Allocation	2003	JMLR
Latent Dirichlet Allocation	*2001*	JMLR			
Latent Dirichlet Allocation	2002	NIPS	Latent Dirichlet Allocation	2002	NIPS
Latent Dirichlet Allocation	*2001*	NIPS			

Table 10: Comparison of baseline approach and shingling algorithm on different years. Underlined years are noisy data.