Combination of Feature Engineering and Ranking Models for Paper-Author Identification in KDD Cup 2013

Chun-Liang Li, Yu-Chuan Su, Ting-Wei Lin, Cheng-Hao Tsai, Wei-Cheng Chang, Kuan-Hao Huang, Tzu-Ming Kuo, Shan-Wei Lin, Young-San Lin, Yu-Chen Lu, Chun-Pai Yang, Cheng-Xia Chang, Wei-Sheng Chin, Yu-Chin Juan, Hsiao-Yu Tung, Jui-Pin Wang, Cheng-Kuang Wei, Felix Wu, Tu-Chun Yin, Tong Yu, Yong Zhuang, Shou-de Lin, Hsuan-Tien Lin, Chih-Jen Lin.
National Taiwan University
{r01922001, r01922159, r01944011, r01922025, b99902019, b99902059, b99902073, b99902023, b97902055, b98902105, b99902109, r01944041, d01944006, r01922136, b98901044, r01922165, b98901037, b99902090, d00922023, r01922141, r01922139}@ntu.edu.tw, {sdlin, htlin, cjlin}@csie.ntu.edu.tw

ABSTRACT

The track 1 problem in KDD Cup 2013 is to discriminate between papers confirmed by the given authors from the other deleted papers. This paper describes the winning solution of team National Taiwan University for track 1 of KDD Cup 2013. First, we conduct the feature engineering to transform the various provided text information into 97 features. Second, we train classification and ranking models using these features. Last, we combine our individual models to boost the performance by using results on the internal validation set and the official Valid set. Some effective post-processing techniques have also been proposed. Our solution achieves 0.98259 MAP score and ranks the first place on the private leaderboard of Test set.

1. INTRODUCTION

Track 1 of KDD Cup 2013 requires distinguishing whether the assigned papers for a given author by Microsoft Academic Search are truly written by this author. The dataset [8], which is provided by Microsoft Academic Search, contains the information of confirmation and deletion of authors. The confirmation means the authors acknowledge they are the authors of the given paper; in contrast, the deletion means the authors claim that they are not the authors of the given paper. The confirmation and deletion information are split by organizers into three parts, including Train, Valid, and Test sets based on author IDs.

The Train set contains 3,739 authors. For each author, the AuthorId, ConfirmedPaperIds, and DeletedPaperIds are provided. The Valid set of 1,486 authors, each of which is with a sequence of assigned paper IDs without confirmation or deletion, is for public leaderboard evaluation. The answers (confirmation/deletion) in the Valid set were released two weeks before the end of the competition. Attenders were allowed to refine their algorithms based on the released answers of the Valid set, and were required to submit their models one week before the end of the competition. After the submission, the Test set of 2,245 authors was released for private leaderboard evaluation.

In addition to the Train set, other information is also provided. `Author.csv` contains author names and their affiliations. `Paper.csv` contains paper titles, years, conference IDs, journal IDs, and keywords. `PaperAuthor.csv` contains paper IDs, author IDs, author names, and affiliations. The `Journal.csv` and `Conference.csv` contain short names, full names, and home page information of journals and conferences, respectively. Unfortunately, the provided data are noisy and have missing values. For instance, `PaperAuthor.csv` contains the relations of authors and papers, but papers may be wrongly assigned to an author.

The goal of the competition is to predict which given papers are written by the given author. The evaluation criterion is mean average precision (MAP), which is commonly used for ranking problems. Before answers of the Valid set were released, each team was allowed to submit their results on the Valid set five times per day and MAP scores were shown on the public leaderboard. During the last week of the competition, each team was allowed to submit multiple results on the Test set, and select one result for the final standing.

The paper describes the approaches of team National Taiwan University. According to the announced result, our approach achieves the best result on the Test set with 0.98259 MAP score.

The paper is organized as follows. Section 2 outlines the framework of our approaches. Section 3 introduces the approaches to transform the given text information into meaningful features. Section 4 discusses the models we use. Section 5 describes how we combine different models and post-process the combined result to boost the performance. Finally, we conclude in Section 6. Our implementation is available at `https://github.com/kdd-cup-2013-ntu/track1`.

2. FRAMEWORK

This section first provides the architecture of our system. Then it discusses the self-split internal validation set from the Train set, which is a crucial component in the architecture. The internal validation is not only useful for offline validating the model performance and combining different models, but also important for avoiding over-fitting the

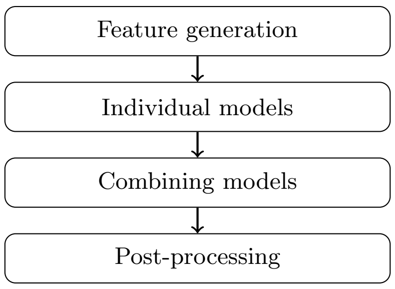

Figure 1: The architecture of our approach

Valid set.

2.1 System Overview

Our system can be divided into four stages: generating features, training individual models, combining different models, and post-processing as shown in Figure 1. In the first stage, since the given data are all text files, we explore different approaches to generate various features, which capture different aspects of the given information. In the second stage, we mainly employ three models, including Random Forests, Gradient Boosting Decision Tree and LambdaMART. For each individual model, to avoid over-fitting, we carefully conduct the parameter selection by using the internal validation set. In the third stage, we combine the three different models by using results on the internal validation set and the official Valid set. In the last stage, we post-process the combined result to further improve the performance by exploiting the characteristic of one feature which is not fully utilized by the models.

2.2 Validation Set

A validation set is useful to evaluate models and avoid over-fitting. Given that the answer of the official Valid set is not available in the early stage of the competition, we construct an internal validation set for verifying our models. It is also useful to avoid over-fitting leaderboard results on the Valid set. In this competition, official Train, Valid and Test sets are generated by randomly shuffling authors with ratio 5:2:3. Therefore, we randomly split the Train set to have 2,670 authors as the internal training set and 1,069 authors as the internal validation set. In our experiments, the MAP score on the internal validation set is usually consistent with the one computed by five-folds cross validation on the official Train set.

3. FEATURE ENGINEERING

To determine the confirmation or deletion of each author-paper pair, we transform `Train.csv` into a binary classification training set. Each confirmation of an author-paper pair is a training instance with label 1; each deletion of of an author-paper pair is a training instance with label -1. We then generate 97 features for each instance and apply the learning algorithms described in Section 4. Subsequently, in describing the feature generation for each author-paper paper, we refer to the author and the paper as the target author and target paper, respectively. In this section, we describe our approaches of transforming the given information into features. For the full feature list, please refer to the Appendix.

3.1 Preprocessing

Since many features are mainly based on string matching, we conduct simple preprocessing to clean the data. We first remove or replace non-ascii characters. Then, we remove stop words in affiliations, titles and keywords, where the stop-word list is obtained from the NLTK package [1]. Finally, we convert all characters into lowercase before comparison.

3.2 Features Using Author Information

This type of features stems from user profile, such as user names or affiliations. Based on the information we try to capture, these features can be classified into the following three groups.

3.2.1 Confirmation of Author Profiles

An intuitive method to confirm that a paper is written by a given author is to check whether the name appears in the author section of the paper. However, a more careful setting is to check also the consistency of other information such as affiliations. In the competition, author affiliations are provided in `Author.csv` and `PaperAuthor.csv`. One basic assumption about `Author.csv` and `PaperAuthor.csv` is that `Author.csv` contains the author profiles maintained by Microsoft Academic Search, while the author information in `PaperAuthor.csv` is extracted from the paper without confirmation. The assumption is based on our observation on the given files as well as the online system. When there exists a conflict between `Author.csv` and `PaperAuthor.csv`, the author information in the online system is usually the same as that in `Author.csv`. Therefore, the features of author profile confirmation comprise of comparisons of author name and affiliation between `Author.csv` and `PaperAuthor.csv`. The comparisons are done by string matching, and various string distances are used as features, including Jaro distance, Levenshtein distance, Jaccard distance (of words) and character match ratio. These features are simple but useful; for example, by using only the affiliation Levenshtein distance as a feature, we can achieve 0.94 MAP score on the Valid set.

An issue in author-name matching is to handle abbreviated names, which are very common in `PaperAuthor.csv`. In contrast, author names in `Author.csv` are usually in complete format. The string distance between an abbreviated name and a full name may be large even if the two names are the same. Two different approaches are used to overcome the problem. The first one is to convert all names into an abbreviated format before the comparison; in our approach, the conversion is done by retaining only the last name and first character of first and middle names. The second approach is to split the author name into first, last and middle names, and compare each of them separately. The two approaches are applied independently to obtain different features.

Another challenge of name matching comes from the inconsistency of the name order. There are two main name orders in the provided data, the Western order and the Eastern order. The Western-order means that given names precede surnames; in contrast, the Eastern-order means that surnames precede given names. While most of the names are in the Western order, names in the Eastern order also frequently appear to cause failed comparisons. Although it is possible to check the name order and transform the Eastern-order names to Western-order ones before comparisons, such checking might be difficult and is prone to error. Instead, two different features are generated for the same distance measure. One assumes that names from `Author.csv` and `PaperAuthor.csv` are in the same name order. The other assumes that names are in opposite orders, so the name order in `Author.csv` is changed before string comparisons. Specifically, the order change is done by exchanging the first word and last word in the name. However, this setting may wrongly consider two different author names as the same; for example, **Xue Yan** (PID:1224852) and **Yan Xue** (PID:482431) are considered as the same person given the second feature. Fortunately, because the number of Eastern-order name is relatively small in the data set, our approach still improves the overall performance.

3.2.2 Coauthor Name Matching

Features matching coauthor names are inspired by an observation of the dataset: in many deleted papers, there exist coauthors with names similar to the target author. For example, two authors (174432 and 1363357) of the deleted paper 5633 are the same as the target author **Li Zhang**. Therefore, having such coauthors is an important trait of deleted papers. To capture the information, we take the minimum string distance of names between the target author and his/her coauthors as a feature. Similar to the feature generation in Section 3.2.1, we also need to address the issue of abbreviated names and name orders.

Another problem for matching coauthor names is to decide names for comparison. For a given author identifier, corresponding names may appear in both `Author.csv` and `PaperAuthor.csv`. In fact, multiple names under this identifier may appear in `PaperAuthor.csv`. These names may be different because of abbreviations, typos or even parsing errors of the Microsoft system. For example, author 1149778 is **Dariusz Adam Ceglarek** in `Author.csv`, while it corresponds to **Dariusz Ceglarek** and **D. Ceglarek** under paper 770630 in `PaperAuthor.csv`. Besides, some authors in `PaperAuthor.csv` do not appear in `Author.csv`. To handle the problem, multiple features are generated, where each feature is computed by using different combinations of name sources. For instance, the target author name could be from `Author.csv` and `PaperAuthor.csv`, and coauthor names could be from `PaperAuthor.csv`. Then the distances of all possible combinations of the author and each coauthor names from different sources are computed. We select the minimum distance among all possible combinations to represent the name distance between the author and his/her coauthors.

3.2.3 Author Consistency

Understandably, information in the dataset should be consistent across papers and authors. Author-consistency features try to measure such information in author profiles. In particular, we measure the coauthor-affiliation consistency and research-topic consistency as features. Affiliation consistency is based on the assumption that authors with the same affiliation are more likely to co-work on a paper; therefore, we compute the affiliation string distance as well as the number of coauthors with the same affiliation as the target author. Similar to coauthor name matching, the affiliation may come from different sources, so we compute multiple features.

Research-topic consistency assumes that the author should work on similar topics across different papers. Although the research-topic or field information is not given in the dataset, we infer it from the paper titles and keywords. Therefore, we compute the title and keyword similarity between the target paper and the target author's other papers as features.

3.2.4 Missing Value Handling

The missing value problem is an important issue of string matching. A common situation in comparing author affiliations or author names is that both strings are empty. The resulting zero string distance wrongly indicates an identical match. Then papers with missing values tend to be ranked higher in prediction. To conquer this problem, we consider values other than zero in calculating the distance. If both strings for comparison are empty, we define their Jaro distance as 0.5, Jaccard distance as 0.5 and Levenshtein distance as the average length of the field. Besides, we use some indicators as features; examples include the number of coauthors without affiliation information.

3.3 Features Using Publication Time

Publication-time features are related to the publication year provided in `Paper.csv`. The intuition of these features is that an author can be active in a specific period, and papers written outside this period are likely authored by others. We include several features to capture the publication-time information, such as the exact publication year, publication-time span and publication year differences with other papers of the target author.

To determine whether the provided year is valid is an issue to resolve before we can generate year features. In the dataset, some papers' publication years are obviously invalid, such as 0, -1 and 800190. Besides, experiments on the internal validation set show that excluding publication years earlier than 1800 A.D. improves the overall performance. Therefore, we set the valid interval to be between 1800 A.D. and 2013 A.D. and ignore publication years outside the interval.

Removing invalid publication years incurs the missing value problem. To fill the missing year values, we utilize the publication-year information of coauthors. The basic concept is to replace a missing value with the average of the mean publication years of all coauthors of the paper. This average, however, is not computable because coauthors may also have missing information on publication years. An iterative process is used to solve the problem as follows. First, papers with invalid years are ignored and mean of available publication years is calculated for each author. The mean value is then used to fill the missing value of the author. These new values can be incorporated to calculate the new mean value of the publication years. Therefore, the mean publication years and missing values are computed alternatively until convergence.

3.4 Features Using Heterogeneous Bibliographic Networks

The work in [9] introduces the concept of Heterogeneous Bibliographic Network which captures the different relations between authors and papers, and demonstrates the effectiveness of link prediction. In this competition, finding whether a paper is written by a given author is the same as predicting a link between an author and a paper. Inspired by [9], we extract several useful features from the network to obtain the relation between nodes.

Heterogeneous Bibliographic Network is a graph $G = (V, E)$, where V is the vertex set and E is the edge set. According to the given data, the vertex set $V = \mathcal{P} \cup \mathcal{A} \cup \mathcal{C} \cup \mathcal{J}$ contains the set of papers $\mathcal{P}$, the set of authors $\mathcal{A}$, the set of conferences $\mathcal{C}$ and the set of journals $\mathcal{J}$. The set E consists of two kinds of edges. Based on `PaperAuthor.csv`, if author a_i writes paper p_j, then we create the edge e_{ij}; based on `Papers.csv`, if paper p_m belongs to conference c_n or journal j_n, then we create the edge e_{mn}. Note that, because information in `PaperAuthor.csv` may be incorrect, some links are wrongly generated in the network.

After generating the network, we could extract basic features, such as the number of publications of an author, and the number of total coauthors of an author.

To utilize the heterogeneous bibliographic network, we further define the "path" to describe node relationship. Given the paper-author pair (p_i, a_j), a length k *meta path* is defined as $(p_i \leftrightarrow v_1 \leftrightarrow \cdots \leftrightarrow v_{k-1} \leftrightarrow a_j)$, where $v_1, \cdots, v_{k-1} \in V$ and $\leftrightarrow$ means two nodes are connected by an edge. For example, given (p_i, a_j), $S_{ij} = \{(p_i \leftrightarrow j \leftrightarrow \bar{p} \leftrightarrow a_j)\}$ is a set of length-3 meta paths which captures all papers of author a_j published in the same journal j as p_i. In Appendix A, we list all kinds of meta paths we used for feature engineering.

On the other hand, given an author pair (a_i, a_j), a length-k *pseudo path* is defined as $(a_i \sim a_1 \sim \cdots \sim a_{k-1} \sim a_j)$, where $a_1, \cdots, a_{k-1} \in \mathcal{A}$. However, since there is no edge between two author nodes in our network, $\sim$ is the pseudo-edge. If author node a_j is reachable from a_i on the network by traversing non-author nodes, then we consider there is a pseudo-edge between a_i and a_j. In other words, the pseudo-edge describes the possible co-authorship between two authors. By considering the pseudo paths, we can grasp different co-authorship information.

4. MODELS

After generating features, we apply classification methods to train the data set. To enhance the diversity, we explore three state-of-the-art algorithms as described in this section.

4.1 Random Forests

Random Forests is a tree based learning method introduced by Leo Breiman [2]. The algorithm constructs multiple decision trees using randomly sub-sampled features and outputs the result by averaging the prediction of individual trees. The use of multiple trees reduces the variance of prediction, so Random Forests are robust and useful in this competition.

We use the implementation in the scikit-learn package [7]. The package provides a parallel module to significantly speed up the tree building process. Note that the scikit-learn implementation combines classifiers by averaging probabilistic prediction instead of a voting mechanism in [2].

In this competition, the variance may influence the standing on the leaderboard significantly. For example, with different random seeds and fewer trees, the performance of Random Forests can vibrate from 0.981 to 0.985 on the Valid set. On the public leaderboard, the scores of top 20 places are from 0.98554 to 0.98130. Our experiments show that using more trees leads to better validation scores due to lower variance. After some trials, we use 12,000 trees and a fix random seed 1 in our Random Forests model, which could achieve 0.983340 MAP score on the Valid set.

4.2 Gradient Boosting Decision Tree

Gradient Boosting Decision Tree (GBDT) [5] is also a tree-based learning algorithm. We use the same package scikit-learn [7]. The optimization goal of GBDT in [7] is to optimize "deviance" which is same as logistic regression. Unlike Random Forests, GBDT combines different tree estimators in a boosting way. A GBDT model is built sequentially by using weak decision tree learners on reweighted data. Then it combines built trees to generate a powerful learner. The main disadvantage of GBDT is that it cannot be trained in parallel, so we only use 300 trees to build the final ensemble model of GBDT. This is much smaller than 12,000 for Random Forests. With the above parameters, the GBDT model could achieve 0.983046 MAP score on the Valid set.

4.3 LambdaMart

We choose LambdaMART [3] because of its recent success on Yahoo! Learning to Rank Challenge [4]. LambdaMART is the combination of GBDT and LambdaRank. The main advantage is that LambdaMART use LambdaRank gradients to consider highly non-smooth ranking metrics. We use the implementation in the JForests [6], which optimizes the NDCG metric. To avoid over-fitting, we train 10 LambdaMART models with random seeds from 0 to 9, and average the output confidence scores. The number of leaves is set to 32, the feature sample rate is 0.3, the minimum instance percentage per leaf is 0.01, and the learning rate is 0.1. With the above parameters, the LambdaMART model could also achieve 0.983047 MAP score on the Valid set.

5. ENSEMBLE AND POST-PROCESSING

To further boost our performance, we ensemble results of different models and conduct a post-processing procedure.

5.1 Ensemble

In our system, we calculate the simple weighted average after scaling the decision values to be between 0 and 1. Because only three models described in Section 4 were built, we search a grid of weights to find the best setting.

To see the performance under a setting of weights, we check the results on the internal validation set and the official Valid set. We train three models on the internal training set, and predict on the internal validation set. Then we combine the results according to the weights to check the improvement. Similarly, we train three models on the Train set (internal train set + internal valid set) and predict on the Valid set. Then we check whether results are further improved. The final weights are 1 for both Gradient Boosting Decision Tree and LambdaMART, and 5 for Random Forests.

5.2 Post-Processing

5.2.1 Using Strong Features

In Section 3.4, we describe the concept of Heterogeneous Bibliographic Network. Even if there is an edge between the author node a and the paper node p, a may not be the author of p because of the incorrect information in `PaperAuthor.csv`. To get confidence on each link, we observe from PaperAuthor.csv that there are some duplicated paper-author pairs. For example, lines 147,035 and 147,036 record the same author-paper pair. We observe that duplicates highly correlate with the confirmation. Therefore, we let the number of duplicates be the weight of the edge between a paper and an author. We use weighted edges in two ways. First, we add a feature to illustrate the number of duplicates before the training procedure to obtain models described in Section 4. Second, according to the number of duplicates, we divide the given papers of each author into two groups; those having more than one duplicate and those having only one. Then in our prediction, we rank the first group before the second. For each group, we rank its members according to their decision values.

5.2.2 Duplicated Paper ID

In the Test set, the assigned papers of an author may contain duplicates. For example, author 100 has five papers 1, 2, 2, 3 and 4 to be ranked, and confirmed papers are 1, 2, 2 and 4. According to the algorithm provided by Kaggle for calculating MAP, only one of these duplicated paper IDs will be calculated in MAP. Therefore, the list 1, 2, 4, 3, 2 has a higher MAP than the list 1, 2, 2, 4, 3 because the second paper with ID 2 is treated as a deleted paper in the evaluation algorithm of Kaggle. To handle this situation, we put all duplicated paper IDs to the end of the ranked list as deleted papers.

6. CONCLUSION

In this paper, we introduce the approaches of team National Taiwan University for track 1 of KDD Cup 2013. We successfully transform the given text information into several useful features and propose techniques to address the issue of noisy texts for making features robust. We then apply several state-of-the-art algorithms on the generated features. To further improve the performance, we conduct a simple weighted average ensemble and a post-processing procedure by utilizing some strong features. During each stage, we cautiously use the internal validation or the official Valid set to potentially avoid the over-fitting issue. This step is crucial for us to get the best performance on the private leaderboard for predicting data in the Test set.

7. ACKNOWLEDGMENT

We thank the organizers for holding this interesting competition. We also thank the College of Electrical Engineering and Computer Science as well as the Department of Computer Science and Information Engineering at National Taiwan University for their supports and for providing a stimulating research environment. The work was also supported by National Taiwan University under Grants NTU 102R7827, 102R7828, 102R7829, and by National Science Council under Grants NSC 101-2221-E002-199-MY3, 101-2628-E002-028-MY2, 101-2628-E002-029-MY2.

8. REFERENCES

[1] S. Bird, E. Klein, and E. Loper. Natural language processing with Python, 2009.

[2] L. Breiman. Random forests. *Machine Learning*, 2001.

[3] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. 2010.

[4] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research - Proceedings Track*, 2011.

[5] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 2002.

[6] Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, 2011.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[8] S. B. Roy, M. D. 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 2013 workshop*, 2013.

[9] Y. Sun, R. Barber, M. Gupta, C. C. Aggarwal, and J. Han. Co-author relationship prediction in heterogeneous bibliographic networks. In *Advances in Social Networks Analysis and Mining*, 2011.

APPENDIX

A. FEATURE LIST

Since our team members are divided into several subgroups internally, some features are repeatedly generated. For these features, we denote the n times repeats by (*n) at the end of the description.

A.1 Features Using Author Information

Confirmation of Author Profile

1. The Levenshtein distance between the names of the target author in Author.csv and PaperAuthor.csv.
2. The Levenshtein distance between the affiliations of the target author in Author.csv and PaperAuthor.csv (*2).
3. The ratio of matched substring between the names of the target author in Author.csv and PaperAuthor.csv.
4. The ratio of matched substring between the affiliations of the target author in Author.csv and PaperAuthor.csv.
5. The ratio of matched substring between the abbreviated names of the target author in Author.csv and PaperAuthor.csv.

Coauthor Name Matching

1. The maximum Jaro distances between the target author's name and each coauthor's name. The names are from PaperAuthor.csv under the target paper.
2. The maximum Jaro distances between the last names of the target author and each coauthor. The names are from PaperAuthor.csv under the target paper.
3. The maximum Jaro distances between the target author's name and each coauthor's name. The names are from PaperAuthor.csv under the target paper. Coauthors having the same affiliation with the target author are ignored during the comparison.
4. The minimum Levenshtein distances between the target author's name and each coauthor's name. The names are from PaperAuthor.csv under the target paper. Coauthors that are in the same affiliation of the target author are ignored during comparison.

5. The number of authors having the same name as the target author in the entire dataset.
6. The maximum Jaro distances between the abbreviated names of the target author and each coauthor. The names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
7. The minimum among Levenshtein distances between the abbreviated names of the target author and each coauthor. The names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
8. The minimum substring matched ratios between the target author's last name and each coauthor's last name. The author's name is form `Author.csv`, and coauthors' names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
9. The minimum substring matched ratios between the target author's first name and each coauthor's first name. The author's name is form `Author.csv`, and coauthors' names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
10. The minimum substring matched ratios between the target author's reversed name and each coauthor's name. Middle name is ignored, and the target author's first name and last name are exchanged before comparison. The author's name is form `Author.csv`, and coauthors' names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
11. The minimum substring matched ratios between the target author's middle name and each coauthor's middle name. The author's name is form `Author.csv`, and coauthors' names are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
12. The maximum Jaro distances between the target author's last name and each coauthor's last name. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
13. The maximum Jaro distances between the target author's first name and each coauthor's first name. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
14. The maximum Jaro distances between the target author's name and each coauthor's name. Middle name is ignored, and the target author's first name and last name are exchanged before comparison. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
15. The maximum Jaro distances between the abbreviated names of the target author and each coauthor. Middle name is ignored, and the target author's first name and last name are exchanged before abbreviation. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
16. The maximum Jaro distances between the abbreviated names of the target author and each coauthor. Middle name is ignored, and the coauthor's first name and last name are exchanged before abbreviation. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
17. The minimum Levenshtein distances between the target author's last name and each coauthor's last name. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
18. The minimum Levenshtein distances between the target author's first name and each coauthor's first name. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
19. The minimum Levenshtein distances between the target author's name and each coauthor's name. Middle name is ignored, and the target author's first name and last name are exchanged before comparison. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
20. The minimum Levenshtein distances between the abbreviated names of the target author and each coauthor. Middle name is ignored, and the target author's first name and last name are exchanged before abbreviation. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
21. The minimum Levenshtein distances between the abbreviated names of the target author and each coauthor. Middle name is ignored, and the coauthor's first name and last name are exchanged before abbreviation. The names are from `PaperAuthor.csv` under the target paper. Coauthors in the same affiliation as the target author are ignored during comparison.
22. The maximum of affiliation Jaro distances times name Levenshtein distances betweein target author and coauthors. Both author name and affiliation are from `PaperAuthor.csv`.
23. The maximum Jaro distances between the target author's name and each coauthor's name. The name of target author is from `Author.csv`, and that of coauthors are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.
24. The minimum Levenshtein distances between the target author's name and each coauthor's name. The name of target author is from `Author.csv`, and that of coauthors are from `PaperAuthor.csv` under the target paper. Coauthors that are in the same affiliation of target author are ignored during comparison.

Author Consistency

1. The maximum Jaro distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `Author.csv`.
2. The maximum Levenshtein distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `Author.csv`.
3. The maximum Jaro distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `PaperAuthor.csv` under the target paper.
4. The minimum Levenshtein distance between the affiliation of the target author and affiliations of coauthors in

the paper. The affiliations are from `PaperAuthor.csv` under the target paper.

5. The maximum Jaro distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `PaperAuthor.csv` under all papers published by a given author.
6. The maximum Levenshtein distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `PaperAuthor.csv` under all papers published by a given author.
7. The maximum Jaccard distance between the affiliation of the target author and affiliations of coauthors in the paper. The affiliations are from `PaperAuthor.csv` under all papers published by a given author.
8. The number of coauthors in the same affiliation as the target author. The affiliations are from `PaperAuthor.csv` under the target paper.
9. The number of authors with no affiliation information in `PaperAuthor.csv` under the target paper.
10. The percentage of authors with no affiliation information in `PaperAuthor.csv` under the target paper.
11. Maximum paper title Jaro distance of the target paper and papers written by the author.
12. Minimum paper title Levenshtein distance of the target paper and papers written by the author.
13. Maximum keywords Jaccard distance of the target paper and papers written by the author.

A.2 Features Using Publication Time

1. Earliest publication year of the author (*2).
2. Latest publication year of the author (*3).
3. Publication year of the paper, and the invalid year is replaced by 0 (*3).
4. Indicator to see if the publication year of the paper is missing.
5. Publication year after filling missing value.
6. Mean publication year of all papers of the author.
7. Standard deviation of publication year of all papers of the author.
8. Mean publication year of the authors' papers in the same conference as the target paper.
9. Standard deviation of the publication year of the authors' papers in the same conference as the target paper.
10. Mean publication year of the authors' papers in the same journal as the target paper.
11. Standard deviation of the publication year of the authors' papers in the same journal as the target paper.
12. Mean publication year of all papers in the same conference as the target paper.
13. Standard deviation of the publication year of all papers in the same conference as the target paper.
14. Mean publication year of all papers in the same journal as the target paper.
15. Standard deviation of the publication year of all papers in the same journal as the target paper.
16. The difference between target author's the latest publication year and the earliest publication year.
17. The difference between the target paper's publication year and the median of the publication year of all the papers of the target author.
18. The maximum publication-year difference between the target paper and papers of the target author.

A.3 Features Using Heterogeneous Bibliographic Network

1. Total number of papers published by the target author (*3).
2. Total number of coauthors of the target author (*4).
3. Number of authors of the target paper (*3).
4. Number of occurrences of the (PID,AID) pairs in `PaperAuthor.csv` (*2, and used for post processing).
5. Number of papers the author published in the conference of the target paper (*3).
6. Number of papers the author published in the journal of the target paper (*3).
7. Number of conference papers of the author (*2).
8. Percentage of conference papers of the author.
9. Number of conferences the author has papers in.
10. Number of journal papers of the author (*2).
11. Percentage of journal papers of the author.
12. Number of journals the author has papers in.
13. Average paper number of the author in conferences he/she has published in.
14. Average paper number of the author in journals he/she has published in.
15. Total number of papers written by coauthors of the target author.
16. Average paper number of coauthors of the target author.
17. The variance of paper number of coauthors of the target author.
18. Total number of papers written by coauthors in the target paper.
19. Average paper number of coauthors in the target paper.
20. The variance of paper number of coauthors in the target paper.
21. Indicator of journal papers.
22. Indicator of conference papers.
23. The difference between the number of conference papers and journal papers written by the target author.
24. The number of coauthors in the paper that have coauthored other papers with the target author.
25. The percentage of papers that are coauthored with at least one of the coauthors of the target paper.
26. Maximum number of coauthored papers with coauthors of the target paper.
27. Maximum percentage of coauthored papers (with respect to total number of papers written by the target author) with coauthors of the target paper.
28. Number of coauthors that appear more than once under the target paper in `PaperAuthor.csv`.
29. Indicator of whether the paper has only one author.
30. Number of papers published by the author which has duplicated (PID, AID) in `PaperAuthor.csv`.
31. Number of coauthored papers of the target author with all the coauthors of the target paper.
32. Number of coauthored papers of the target author with all the coauthors of the target paper, divided by the total number of coauthored papers of the target author with each coauthor of the target paper.
33. Number of coauthored papers of the target author with all the coauthors of the target paper (excluding the target paper).
34. Number of coauthored papers of the target author with all the coauthors of the target paper, divided by total number of coauthored papers of the target author with all coauthors of the target paper (excluding the target paper).
35. Total number of coauthored papers of the target author with all possible coauthors (*2).
36. Average number of coauthored papers of the target author with each coauthor of the target paper (*2).
37. Number of coauthored papers of the target author with all the coauthors of the target paper.