IE304 Project: Learning to Estimate the Authors of Double-blind Submission in Scholarly Networks

Jie-Lin Qiu
Shanghai Jiao Tong University
{Qiu-Jielin@sjtu.edu.cn}

Abstract: **The problem is to estimate the authors of a double-blind submission paper with the help of scholar networks. First, we conduct the feature engineering to transform the various provided text information into 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 validation set. Some effective post-processing techniques have also been proposed. Our solution achieves 0.9113 MAP score of Test set.**

***Keywords*—Feature engineering, Paper-Author Identification, Scholar networks**

I. Introduction

The problem lies in how to estimate the author of papers which were submitted to double-blind process. The dataset, which is provided by Microsoft (Acemap[1] Group[2]), 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 paper has no information about its authors, institutions and any other information related to their personality. The confirmation and deletion information are split into three parts, including Train, Valid, an Test sets based on paper and author IDs.

The Train set contains 3739 authors. For each paper, the PaperId, ConfirmedAuthorIds, and DeletedAuthorIds are provided. The Valid set of 1486 authors, each of which is with a sequence of assigned author IDs without confirmation or deletion, is for public 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 (the real keywords and important words extracted from abstract). $PaperAuthor.csv$ contains paper IDs, author IDs, author names, and affiliations. The $Journal.csv$ and $Conference.csv$ contain short names and full names 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 project is to predict which given authors wrote the given papers. The evaluation criterion is mean average precision (MAP), which is commonly used for ranking problems.

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 postprocess the combined result to boost the performance. Finally, we conclude in Section 6.

II. 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 Valid set.

[1] http://acemap.sjtu.edu.cn/

[2] Thanks to excellent previous fundamental work achieved by Yuting Jia, Hao Wu, Weijie Tang, Wen Sui, Tao Ni, Hao Lu *et al.* of Acemap Group, especially the map group and the engineering group.

A. 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 csv 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, gcForest, and LambdaMART. For each individual model, to avoid overfitting, 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.

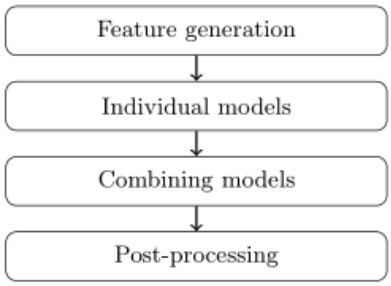

Fig. 1: The architecture of our apporach.

B. Validation Set

A validation set is useful to evaluate models and avoid over-fitting. 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 project, 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 Train set.

III. 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.

A. 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.

B. Features Using Author Information

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

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 project, 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.

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) 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; therfore, 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, keywords and important words extracted from abstract. Therefore, we compute the title and words similarity between the target paper and the potential authors other papers as features.

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.

C. 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.

D. Features Using Heterogeneous Bibliographic Networks

The work in [2] 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 project, finding whether a given paper is written by a author is the same as predicting a link between an author and a paper. Inspired by [2], we extract several useful features from the network to obtain the relation between nodes.

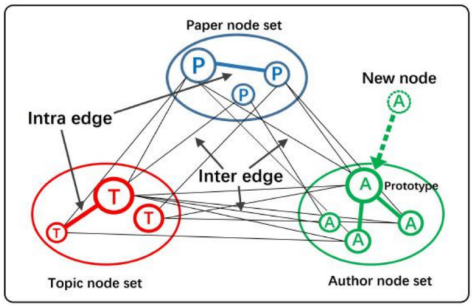

Fig. 2: Relationship among papers, authors and topics[1].

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 $V = P \cup A \cup C \cup J$ contains the set of papers P, the set of authors A, the set of conferences C and the set of journals 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

[1] Thanks to previous work achieved by Fengyu Deng, Lingkun Kong, Bo Wang, Jialu Wang *et al.*

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 $(p_i \leftrightarrow v_1 \leftrightarrow ... \leftrightarrow v_{k-1} \leftrightarrow a_j)$, where $v_1,...,v_{k1} \in V$ 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.

On the other hand, given an author pair (a_i, a_j), a length-k pseudo path is defined as $(a_i \sim a_1 \sim a_{k-1} \sim a_j)$ where $a_1, ..., a_{k1} \in A$. However, since there is no edge between two author nodes in our network, $\sim$ is the pseudoedge. 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 pseudoedge describes the possible co-authorship between two authors. By considering the pseudo paths, we can grasp different co-authorship information.

IV. Models

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

A. Random Forests

Random Forests is a tree based learning method introduced by Leo Breiman [3]. The algorithm constructs multiple decision trees using randomly subsampled 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 project.

We use the implementation in the scikit-learn package [4]. 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 [3].

In this project, 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. 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.

B. Gradient Boosting Decision Tree

Gradient Boosting Decision Tree (GBDT) [5] is also a tree-based learning algorithm. We use the same package scikit-learn [4]. The optimization goal of GBDT in [4] 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.

C. gcForest

The gcForest is a model proposed by Zhou *et al.* 2017 [6], which which generates deep forest holding layer-by-layer processing, in-model feature transformation and sufficient model complexity. This is a decision tree ensemble approach, with much less hyper-parameters than deep neural networks, and its model complexity can be automatically determined in a data-dependent way. Figure 3 summarizes the overall procedure of gcForest. Suppose that the original input is of 400 raw features, and three window sizes are used for multi-grained scanning. For m training examples, a window with size of 100 features will generate a data set of 301 $\times m$ 100-dimensional training examples. These data will be used to train a completely-random tree forest and a random forest, each containing 500 trees. The transformed training set will then be used

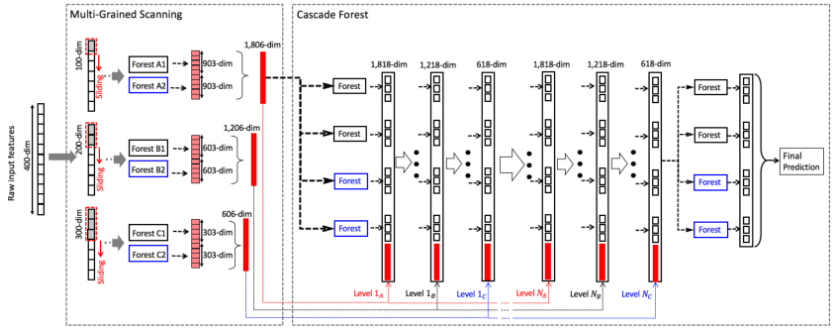

Fig. 3: The model of gcForest.

to train the 1st-grade of cascade forest. Similarly, sliding windows with sizes of 200 and 300 features will generate 1,206-dimensional and 606-dimensional feature vector, respectively, for each original training example. The transformed feature vectors, augmented with the class vector generated by the previous grade, will then be used to train the 2nd-grade and 3rd-grade of cascade forests, respectively. This procedure will be repeated till convergence of validation performance. In other words, the final model is actually a cascade of cascade forests, where each level in the cascade consists of multiple grades (of cascade forests), each corresponding to a grain of scanning. Note that for difficult tasks, users can try more grains if computational resource allows. Given a test instance, it will go through the multi-grained scanning procedure to get its corresponding transformed feature representation, and then go through the cascade till the last level. The final prediction will be obtained by aggregating the four 3-dimensional class vectors at the last level, and taking the class with the maximum aggregated value.

In experiments, gcForest is using the same cascade structure: each level consists of 4 completely-random tree forests and 4 random forests, each containing 500 trees. Three-fold cross validation is used for class vector generation. The number of cascade levels is automatically determined. In detail, we split the training set into two parts, i.e., growing set and estimating set 2; then we use the growing set to grow the cascade, and the estimating set to estimate the performance. If growing a new level does not improve the performance, the growth of the cascade terminates and the estimated number of levels is obtained. Then, the cascade is retrained based on merging the growing and estimating sets. For all experiments we take 80% of the training data for growing set and 20% for estimating set.

D. *LambdaMart*

We choose LambdaMART [7] because of its recent success on [8]. 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 [9], which optimizes the NDCG metric. To avoid over-fitting, we train 10 Lamb-daMART 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.

V. Ensemble and Post-Processing

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

A. *Ensemble*

In our system, we calculate the simple weighted average after scaling the decision values to be between 0 and 1. Because only four 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 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, 4 for fcForest, and 5 for Random Forests.

B. Post-Processing

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.

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 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. To handle this situation, we put all duplicated paper IDs to the end of the ranked list as deleted papers.

VI. Results and Conclusion

The comparison results of different approaches are shown in Fig. 4, where RF is short for random forest, GBDT is short for Gradient Boosting Decision Tree, LbM is short for LambdaMART, gcF is short for gcForest, and EM is short for ensemble method. As it is shown, gcForest achieved better results than the other three and ensemble method took advantage of complementarity of different approaches and achieved better results than single model.

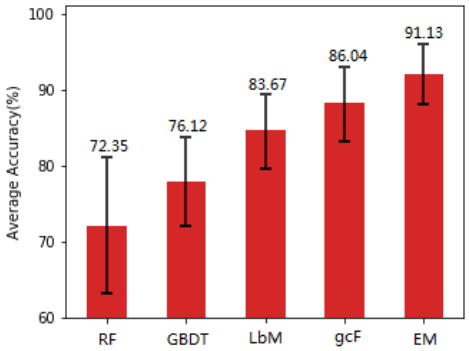

Fig. 4: Comparison results of different approaches.

In this paper, we address the problem of estimating the authors of double-blind submission in scholarly networks. We successfully transform the given paper information into several useful features and propose techniques to address the issue of noisy features for making features robust. We then apply several useful 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, which achieved better results.

References

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

[2] Y. Sun *et al.*, "Co-author relationship prediction in heterogeneous bibliographic networks," *2011 International Conference on Advances in Social Networks Analysis and Mining*, pp. 121–128, 2011.

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[4] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] J. H. Friedman, "Stochastic gradient boosting," 1999.

[6] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," in *IJCAI*, 2017.

[7] C. J. C. Burges, "From ranknet to lambdarank to lambdamart: An overview," 2010.

[8] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *Yahoo! Learning to Rank Challenge*, 2011.

[9] Y. Ganjisaffar, R. Caruana, and C. V. Lopes, "Bagging gradient-boosted trees for high precision, low variance ranking models," in *SIGIR*, 2011.