Qian TY, Liu B, Li Q *et al.* Review authorship attribution in a similarity space. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(1): 200–213 Jan. 2015. DOI 10.1007/s11390-015-1513-6

Review Authorship Attribution in a Similarity Space

Tie-Yun Qian[1] (钱铁云), *Member, CCF, ACM*, Bing Liu[2] (刘 兵), *Fellow, IEEE*
Qing Li[3,*] (李 青), *Distinguished Member, CCF, Senior Member, IEEE*, and Jianfeng Si[4] (司建锋)

[1]*State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China*

[2]*Department of Computer Science, University of Illinois at Chicago, Chicago 60607, U.S.A.*

[3]*Multimedia Software Engineering Research Centre and Department of Computer Science, City University of Hong Kong Hong Kong, China*

[4]*Data Analytics Department, Institute for Infocomm Research, Singapore 138632, Singapore*

E-mail: qty@whu.edu.cn; liub@cs.uic.edu; itqli@cityu.edu.hk; thankjeff@gmail.com

Received February 19, 2014; revised November 14, 2014.

Abstract Authorship attribution, also known as authorship classification, is the problem of identifying the authors (reviewers) of a set of documents (reviews). The common approach is to build a classifier using supervised learning. This approach has several issues which hurts its applicability. First, supervised learning needs a large set of documents from each author to serve as the training data. This can be difficult in practice. For example, in the online review domain, most reviewers (authors) only write a few reviews, which are not enough to serve as the training data. Second, the learned classifier cannot be applied to authors whose documents have not been used in training. In this article, we propose a novel solution to deal with the two problems. The core idea is that instead of learning in the original document space, we transform it to a similarity space. In the similarity space, the learning is able to naturally tackle the issues. Our experiment results based on online reviews and reviewers show that the proposed method outperforms the state-of-the-art supervised and unsupervised baseline methods significantly.

Keywords authorship attribution, supervised learning, similarity space

1 Introduction

Authorship attribution (AA) has been studied by many researchers[1-12]. It was originally proposed to classify Bronte Sisters' novels, Shakespeare's plays, etc. Later on, it was applied to other literary work such as American and English literature and news articles. Recently, AA was applied to various types of online texts such as emails[13-14], blogs[13,15], forum posts[16] and reviews[17]. In this article, we consider AA for online reviews. One application is to find fake reviewers (also called opinion spammers) who register multiple user-IDs and write fake reviews (also called spam reviews)[18].

Most existing AA approaches are based on supervised learning. Let $A = \{a_1, \ldots, a_k\}$ be a set of k authors and $D = \{D_1, \ldots, D_k\}$ be k sets of documents with D_i being the document set of author $a_i \in A$. Each document is represented as a feature vector. Each feature represents a piece of information about the document itself, e.g., a word or a punctuation mark. The learning is done in the *document space* because each feature vector represents a document. A model or classifier is then built from the training data and applied to the test data to determine the author a of each test document d, where $a \in A$, meaning that the authors used in testing must also be from A. Applying this approach to the review domain has two difficulties.

1) Supervised learning needs a large set of documents from each author to serve as the training data. This is difficult for the review domain because most authors (reviewers) only write a few reviews. In [18], it

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61272275, 61232002, 61272110, 61202036, 61379004, 61472337, and 61028003, and the 111 Project of China under Grant No. B07037.

* Corresponding Author

©2015 Springer Science + Business Media, LLC & Science Press, China

is shown that on average each reviewer only has 2.72 reviews at amazon.com, and only 8% of the reviewers have more than four reviews. This number of reviews is too small for training an accurate classifier for supervised learning. As we will see in Subsection 6.2, with a small number of training reviews, the classification results are extremely poor using state-of-the-art supervised approaches[16].

2) The learned model cannot be applied to authors whose documents have not been used in training (we call them unseen authors, or test authors in an open set). Without seeing any documents of an author, there is no way for a model to recognize documents of the author. This, however, limits its application to the review domain because for any new author, a new model has to be built, which is inconvenient and impractical in the review domain since new reviewers (authors) join in constantly.

To solve the first problem of lack of training documents for an author, we have to somehow exploit documents of other authors in training in such a way that the learned classifier can also be applied to this author and help recognize his/her documents. If we can do that, then the second problem is automatically dealt with.

To achieve this, we propose a novel learning formulation in this article, which is based on the following key idea termed as LSS:

Learning in a Similarity Space (*LSS*). Learning/training is performed in a similarity space rather than the document space. That is, in this formulation, each training or testing example is a similarity vector (or s-vector) representing a set of similarities between a document d and a query q (which is also a document, see below). If d has the same author as q, we give its s-vector the class q-positive (denoted by 1). If d has an author different from q, we give its s-vector the class q-negative (denoted by -1). We use q-positive and q-negative here because we use q as the base and decide the class by comparing d's author with q's author.

With this transformation, we obtain a two-class classification problem. The two classes are q-positive and q-negative. Each training or testing case no longer belongs to any particular author as in the traditional formulation. And each author in the test set may or may not have appeared in training. In the following, we briefly look at the training and test data preparation.

Training Data Preparation. For each author $a_i \in AR$, we select a subset of his/her documents DR_i as the query documents QR_i. The rest of his/her documents $\{DR_i - QR_i\}$ are the q-positive documents as they have the same author as the queries. The documents from all other authors are q-negative documents, denoted by DR_{others}. A training instance is basically a similarity vector (or s-vector) produced by comparing a set of similarities of a query document from QR_i and a q-positive (or q-negative) document from $\{DR_i - QR_i\}$ (or from DR_{others}), and is attached with the class label 1 (or -1).

This is seemly a very expensive operation as every document can serve as a query or a non-query document. If the total number of training documents from all authors is w, the complexity is $O(w^2)$. However, in practice, we do not need all pairwise comparisons; instead, only a tiny subset is needed (see Section 6).

With the s-vectors for the two classes, we can use any learning algorithm to build a classifier. In this work, we use SVM.

Test Data Preparation. We have a set of test authors AT and a set of test documents DT. For each test author at_j, we must also have a set of known documents from at_j to serve as queries QT_j. Each test case is also an s-vector produced by comparing a set of similarities between a test document in DT and a query document in QT_j. If a test case is classified as q-positive (i.e., 1), then we know that its underlying document is written by the author of its corresponding query. We call this test author a seen author, or a test author in a closed set. If a test author did not appear in training, we call him/her an unseen author.

The similarity comparison above is done by a similarity function S, which consists of a set of similarity measures for two documents. Each measure produces one similarity feature (or s-feature). All the s-features together form an s-vector. In detail, each document d is first represented by a document space vector (or d-vector) based on d itself as in the traditional classification. Each feature in the d-vector is called a document-feature (or d-feature). A query document q is represented in the same way. We then produce a similarity vector (s-vector) $\boldsymbol{sv}$ for document d by comparing document d's d-vector and query q's d-vector based on the set of similarity measures in S. The d-vector in the document space is thus transformed to an s-vector $\boldsymbol{sv}$ in the similarity space.

Let us see an example, in which the d-vector of a query document q is as follows:

q: 1:1 2:1 6:2,

where $i{:}j$ is a d-feature representing that term i has a frequency j in q. We also have two non-query documents: one is d_1 which is also written by the author of query q and the other is d_2 which is not written by the query author q. Their d-vectors are given below:

d_1: 1:2 2:1 3:1,

d_2: 2:2 3:1 5:2.

If cosine is the first similarity measure in S, we can generate one s-feature 1:0.50 for d_1 and one s-feature 1:0.27 for d_2. If we have more similarity measures in S, we can generate other s-features. Finally, we have two s-vectors, $\boldsymbol{sv}_1$ and $\boldsymbol{sv}_2$, for d_1 and d_2 respectively with their class labels (1 and -1) attached:

$\boldsymbol{sv}_1$: 1 1:0.50 ...,

$\boldsymbol{sv}_2$: -1 1:0.27 ...,

where x:z is an s-feature representing the x-th similarity measure and its similarity value z between the query q and the document d_i.

Due to the use of query documents, the LSS formulation has some resemblance to information retrieval and document ranking①[19-21]. However, LSS is very different because it still performs supervised classification, rather than document ranking directly based on their similarities to queries as in information retrieval. The recent method for AA in [15] is based on the retrieval model. It does not learn a model/classifier. We will see in Section 6 that it performs poorly for our task. On the other hand, although a small set of query documents are still needed for each author in testing, we will see that a supervised baseline can hardly learn with such a small number of query documents as training examples (Section 6). Our experiments are conducted using a large number of reviews (documents) and their reviewers (authors) from Amazon.com. The proposed LSS formulation is tested based on the AA task and also an author retrieval task. The results show that it is highly effective, and it outperforms both the state-of-the-art supervised and unsupervised baselines significantly.

2 Related Work

2.1 Authorship Attribution

Authorship attribution (AA) has received a great deal of attention in recent years. Many approaches have been developed. Existing methods can be categorized into two main themes: finding good features to quantify author writing styles, and developing effective techniques to perform the AA task.

On finding good features (d-features in our case), function words were studied half a century ago[22]. Since then, various newer features have been proposed to model writing styles. The most promising features seem to be the function words[3,23] and rewrite rules[2]. Other features include length features[12,24], richness features[7], punctuation frequencies[12], character n-grams[1,4,25], word n-grams[9,26-27], POS (part-of-speech) n-grams[5,24,28], word endings[28], *k-ee* subtree patterns[17], topic models[13], and modality specific meta features (MSMF)[16]. Among them, MSMF reports the best results for AA with up to 100 authors. In this work, MSMF is used as one of our baselines.

Many other features like the length of URL and in-link (out-link) number are also introduced in machine-learned rankings in information retrieval②. However, since our task is to identify the authors of reviews (which have no links), we focus on writing style features of authors. When computing s-features for certain types of d-features, we use several similarity measures from information retrieval[29].

There are also a number of literatures that study the use of machine learning methods for attribution[30]. While an early work used Bayesian statistical analysis[22], recent research focused almost exclusively on classification, including discriminant analysis[11], PCA[31], Bayesian decision theory[25], multinomial logistic regression[28], neural networks[12,32], clustering[27], decision trees[33-34], SVM[4,7-8,15,35-36], and methods based on topic models[13]. Among them, supervised learning using SVM is regarded as one of the best approaches[9-10,17].

The open class authorship problem has been introduced at the PAN@CLEF conference series. However, the tasks there are quite different from ours, as they only return an answer of "other" or "unknown" for unseen authors. In contrast, we need to identify exactly which author is the candidate author. The most related work to ours is the work of Koppel *et al.*[15], which takes a retrieval approach, and does not use any training documents to build a model. Instead, it compares a query document with the test documents multiple times using cosine similarity. Each time, a subset of randomly sampled features are used. We will describe the algorithm in great detail in our experiment section. We will also see that this similarity-based approach without learning performs poorly for our problem.

① http://research.microsoft.com/en-us/um/beijing/projects/letor/, Nov. 2014.

② http://research.microsoft.com/en-us/projects/mslr/feature.aspx, Nov. 2014.

2.2 Similarity-Based Classification

The idea of learning in the similarity space is also related to the work done on the similarity-based classification[37-38], where data are represented as the similarity or dissimilarity relations between objects. Early work mainly used a nearest neighbor method. Later on, several other approaches such as density-based or SVM classifier were adopted for better classification accuracy[39-40]. Interestingness was also explored in finding good similarity or dissimilarity functions[41-43]. While results presented in the literature point to a data representation in terms of their (dis)similarity, the existing frameworks either use k-NN or traditional supervised classification. They estimate the class label for a test case based on its similarities to the training examples and its self-similarity, which means that the classes for test cases must be as same as or included in the training classes. In contrast, our proposed method can deal with the unseen classes in test which can be completely different from those in training.

3 Learning in Similarity Space

As discussed in the introduction, our proposed formulation is called learning in the similarity space (LSS). To facilitate understanding, we first describe the traditional formulation of the problem as supervised learning.

3.1 Traditional Formulation of AA

Training Data. We have a set of n training authors $AR = \{ar_1, \ldots, ar_n\}$ and a set of n training document sets $DR = \{DR_1, \ldots, DR_n\}$, where DR_i is the set of documents of author $ar_i \in AR$.

Test Data. We have a set of test documents DT with unknown authors. Their authors must be from the set AR, and none of the documents in DT has appeared in the training data.

Training Document Representation. Each training document $\boldsymbol{dr}$ is represented as a feature vector, which we call a document vector (or d-vector for short). We call each feature in the vector a d-feature (for document-feature), which represents a piece information about the document itself. Each training example is of the form $(\boldsymbol{dr}, y)$, where y is $\boldsymbol{dr}$'s class (or author).

Test Document Representation. Test documents are represented in the same way as the training documents except that their authors are unknown and need to be predicted.

Training. A model or classifier is learned using the training data.

Testing. The learned model/classifier is applied to the test data DT to assign an author ar ($\in AR$) to each document dt_j ($\in DT$).

3.2 Training in LSS Formulation

We now present the proposed LSS formulation of AA. The key differences from the traditional formulation are in 1) the test data and 2) about how each document is represented.

3.2.1 Training Data Representation

The training data is prepared as follows.

1) Each document is first represented by a d-vector in the traditional formulation.

2) From the d-vector representations of training documents, we produce s-training examples in a similarity space. We use the term s-training examples to distinguish our training examples from those in the traditional formulation. The algorithm is given in Fig.1 (where // indicates comment). The algorithm works as follows. For each author ar_i (line 1), it first selects a set of documents from the document set DR_i of the same author ar_i as the query documents Q_i (line 2). For each query $q_{ij} \in Q_i$ (line 3), it selects a set of documents DR_{ij} also from DR_i (excluding q_{ij}) of the same author (line 4) as the q-positive documents for q_{ij}. Then, for each document dr_{ijk} in DR_{ij}, a q-positive s-training example with the label 1 is generated for dr_{ijk} by computing the similarities of q_{ij} and dr_{ijk} using the similarity function S (line 6). It then selects a set of documents from other authors (line 7) as the q-negative documents for q_{ij}. For each document $dr_{ijk,\text{rest}}$ in $DR_{ij,\text{rest}}$ (line 8), a q-negative s-training example with the label -1 is generated for dr_{ijk} by computing the similarities of q_{ij} and $dr_{ijk,\text{rest}}$ using function S (line 9).

How to select Q_i, DR_{ij} and $DR_{ij,\text{rest}}$ (lines 2, 4 and 7) is left open intentionally to give flexibility in implementation.

Fig.2 shows what the s-training data looks like. For easy presentation, we assume that there are k queries in every Q_i, p documents in every DR_{ij}, and u documents in every $DR_{ij,\text{rest}}$. The number of authors is n. Each author ar_i generates $k \times (p + u)$ s-training examples.

Complexity. As discussed in Section 1, in the worst case, every document can serve as a query or a non-query document. Then we need to compute all pairwise similarities. If the total number of training documents is w, the complexity is $O(w^2)$, which is expensive. In practice, however, we do not need all pairwise comparisons. Instead, only a tiny subset is needed (see Section 6).

```
1. For each author $ar_i \in AR$
2.   Select a set of query documents $Q_i \subseteq DR_i$
3.   For each query $q_{ij} \in Q_i$
             // Produce positive s-training examples
4.     Select a set of documents from author $ar_i$
         $DR_{ij} \subseteq DR_i - \{q_{ij}\}$
5.     For each document $dr_{ijk} \in DR_{ij}$
6.         Produce an s-training example for $dr_{ijk}$,
             $(S(dr_{ijk}, q_{ij}), 1)$
             // Produce negative s-training examples
7.     Select a set of documents from the rest of the authors
         $DR_{ij,\text{rest}} \subseteq (DR_1 \cup \ldots \cup DR_n) - DR_i$
8.     For each document $dr_{ijk,\text{rest}} \in DR_{ij,\text{rest}}$
9.         Produce an s-training example for $dr_{ijk,\text{rest}}$,
             $(S(dr_{ijk,\text{rest}}, q_{ij}), -1)$
```

Fig.1. Generating s-training examples.

```
// Author $ar_1$
//   Positive (1) s-training examples
       $(S(dr_{111}, q_{11}), 1), \ldots, (S(dr_{11p}, q_{11}), 1)$
       ...
       $(S(dr_{1k1}, q_{1k}), 1), \ldots, (S(dr_{1kp}, q_{1k}), 1)$
//   Negative (−1) s-training examples
       $(S(dr_{111,\text{rest}}, q_{11}), -1), \ldots, (S(dr_{11u,\text{rest}}, q_{11}), -1)$
       ...
       $(S(dr_{1k1,\text{rest}}, q_{1k}), -1), \ldots, (S(dr_{1ku,\text{rest}}, q_{1k}), -1)$
...
// Author $ar_n$
//   Positive (1) s-training examples
       $(S(dr_{111}, q_{n1}), 1), \ldots, (S(dr_{11p}, q_{n1}), 1)$
       ...
       $(S(dr_{1k1}, q_{nk}), 1), \ldots, (S(dr_{1kp}, q_{nk}), 1)$
//   Negative (−1) s-training examples
       $(S(dr_{111,\text{rest}}, q_{n1}), -1), \ldots, (S(d_{11u,\text{rest}}, q_{n1}), -1)$
       ...
       $(S(dr_{1k1,\text{rest}}, q_{nk}), -1), \ldots, (S(dr_{1ku,\text{rest}}, q_{nk}), -1)$
```

Fig.2. S-training examples.

3.2.2 *Training in LSS*

A binary model/classifier is learned using the s-training data. Each s-training example is represented by $(\boldsymbol{sv}, y)$, where $\boldsymbol{sv}$ is an s-vector and y $(\in \{1, -1\})$ is its class. Any supervised learning algorithms, e.g., SVM, can be applied.

3.3 Authorship Attribution in LSS

After a classifier is learned, we can then test it by performing the authorship attribution task. We are given the following data:

1) A set of m test authors $AT = \{at_1, \ldots, at_m\}$. Note that in traditional supervised learning, AT must be from AR. In contrast, in our new LSS framework, AT can be totally different from AR. We differentiate these two types of test authors as unseen authors, i.e., $AT \cap AR = \emptyset$, and seen authors, $AT \subseteq AR$.

2) A set of m given query document sets $Q = \{Q_1, \ldots, Q_m\}$ of the m authors, where Q_j is the set of query documents of author $at_j \in AT$.

3) A set of r test documents $DT = \{dt_1, \ldots, dt_r\}$ with unknown authors. Their authors are from the set AT. None of the documents in DT has appeared in training.

Objective. Using the learned classifier to determine the author of each $dt_k \in DT$.

Note that if at_j $(\in AT)$ has appeared in training, its query set can be his/her documents used in training.

3.3.1 *Test Data Representation*

The data is prepared as follows.

1) Each test document in DT and each query document in Q_i are first represented by their document feature vectors (or d-vectors) as in the traditional formulation.

2) From the d-vector representation of each test document, we produce test cases in the similarity space. Each test case in the space is called an s-test case. A test document will generate multiple s-test cases. The algorithm is given in Fig.3, which works as follows. For each test author at_i in AT (line 1), each query document q_{ij} in Q_i (line 2) and each test document dt_f in DT (line 3), it produces an s-test case by computing the similarity of q_{ij} and dt_f using function S (line 4). Each s-test case is $\langle(at_i, q_{ij}), (S(dt_f, q_{ij}), ?)\rangle$ as we need to remember the author and the query for later author assignment. $?(\in \{1, -1\})$ denotes the unknown class to be predicted.

```
1. For each test author $at_i \in AT$
2.   For each query $q_{ij} \in Q_i$
3.     For each test document $dt_f \in DT$
4.       Produce an s-test case $\langle(at_i, q_{ij}), (S(dt_f, q_{ij}), ?)\rangle$
```

Fig.3. Generating s-test cases.

Fig.4 shows the set of s-test cases denoted by ST. For easy presentation, we assume that there are k queries in every Q_i. We have $r(= |DT|)$ test documents, and $m(= |AT|)$ test authors. Note that the classifier is only applied to the s-vector produced by $S(dr_j, q_{ij})$. Each query q_{ij} generates $r \times m$ s-test cases. The total number of s-test cases for each author is $r \times m \times k$.

```
// Test data $DT = \{dt_1, \ldots, dt_r\}$
    // Author $at_1$
$\langle(at_1, q_{11}), (S(dt_1, q_{11}), ?)\rangle, \ldots, \langle(at_1, q_{11}), (S(dt_r, q_{11}), ?)\rangle$
...
$\langle(at_1, q_{1k}), (S(dt_1, q_{1k}), ?)\rangle, \ldots, \langle(at_1, q_{1k}), (S(dt_r, q_{1k}), ?)\rangle$
    ...
    // Author $at_m$
$\langle(at_m, q_{m1}), (S(dt_1, q_{m1}), ?)\rangle, \ldots, \langle(at_m q_{m1}), (S(dt_r, q_{m1}), ?)\rangle$
...
$\langle(at_m, q_{mk}), (S(dt_1, q_{mk}), ?)\rangle, \ldots, \langle(at_m, q_{mk}), (S(dt_r, q_{mk}), ?)\rangle$
```

Fig.4. S-test cases.

As we can see, the process of producing s-test cases is similar to that for s-training examples. However, there are some differences.

1) For testing, the query sets and the test set (lines 2 and 3 in Fig.3) are pre-determined, unlike in training where every document can be a query or a non-query document.

2) In line 4 of Fig.3, the resulting s-vector of $S(dt_f, q_{ij})$ is associated with its query q_{ij} and author at_i, and the latter two are not used in training. After the s-training data in the similarity space is generated, training simply builds a model for the q-positive and the q-negative classes, and does not concern either individual authors or their documents.

3.3.2 Authorship Attribution

The learned classifier is applied to the s-test set ST. Each s-test case in ST is represented by $\langle(at, q), (\boldsymbol{sv}, ?)\rangle$, where $\boldsymbol{sv}$ is computed using the similarity function S based on the query document q of author at. The classifier is applied to $\boldsymbol{sv}$. However, this classification alone is unable to make an author assignment to each test document $dt_f \in DT$. The reason is that for each dt_f, multiple s-test cases are generated due to multiple test authors and multiple query documents of each author (see Fig.4). The labels produced by the classifier may not be consistent. For example, dt_1 can be classified as 1 and -1 for q_{11} and q_{1k}, respectively. Let the set of s-test cases for a test document dt_f be ST_f. Some s-test cases in ST_f may be classified as q-positive for some queries of an author but q-negative for other queries of the same author. Furthermore, there are also multiple authors. Hence, additional algorithms are needed to decide the final author attribution for each test document dt_f.

One simple general method is voting. Depending on what output value the classifier produces, there are also other methods. Here we propose three methods including the voting method. The other two methods require the classifier to produce predicted score, which can reflect the positive and negative certainty. Many classification algorithms produce such a score, e.g., SVM, logistic regression, and naïve Bayesian. Here we use SVM as an example. For each test case, SVM outputs a positive or negative score which can be interpreted as the certainty that a test case is positive or negative. The three methods are all given in Fig.5 to save space.

```
1.  Classify all s-test cases in $ST$
2.  **For** each test document $dt_f \in DT$
3.      **For** each test author $at_i \in AT$
4.          $pcount[at_i, dt_f]$, $psum[at_i, dt_f]$, $max[at_i, dt_f] = 0$
5.          **For** each query $q_{ij} \in Q_i$
6.              **If** $st = \langle(at_i, q_{ij}), (S(dt_f, q_{ij}), ?)\rangle$ $(\in ST)$
                    is classified positive (i.e., $? = 1$) **then**
7.                  $pcount[at_i, dt_f] = pcount[at_i, dt_f] + 1$
8.                  $psum[at_i, dt_f] = psum[at_i, dt_f] + st$.score
9.              **If** $st$.score $> max[at_i, dt_f]$ **then**
10.                 $max[at_i, dt_f] = st$.score
// Three alternative methods to determine the author of $dt_f$
11. **For** each test document $dt_f \in DT$
12.     **If** for all $pcount[at_i, dt_f] = 0$ **then**
13.         $dt_f$.author $= \arg\max_{at_i \in AT}(max[at_i, dt_f])$
14.     **Else** $dt_f$.author $= \arg\max_{at_i \in AT}(\frac{pcount[at_i, dt_f]}{|Q_i|})$ // Voting
15.         $dt_f$.author $= \arg\max_{at_i \in AT}(\frac{psum[at_i, dt_f]}{|Q_i|})$ // ScoreSum
16.         $dt_f$.author $= \arg\max_{at_i \in AT}(max[at_i, dt_f])$ // ScoreMax
```

Fig.5. Model testing: authorship attribution.

1) *Voting*. The learned model/classifier is first applied to classify all s-test cases in ST (line 1). Then for each test document $dt_f \in DT$ (line 2), the author assignment works as follows. For each test author $at_i \in AT$ (line 3), it counts the number of queries of the author for which the test document dt_f is classified as positive (line 7). In other words, it counts the number of s-test cases that are classified as positive. The count value is stored in $pcount[at_i, dt_f]$. It then assigns dt_f to the author with the highest count (line 14). Lines 12 and 13 mean that if dt_f's s-test cases generated from all queries of all authors are classified as negative, we use method 3 ScoreMax below. The normalization is used because each test author may not have the equal number of queries.

2) *ScoreSum*. The second method works similarly to the voting method above except that instead of counting positive classifications, it sums up all scores of positive classifications (line 8). The sum is stored in $psum[at_i, dt_f]$. The decision is also made similarly (line 15). Again, normalization is applied. Note that if $pcount[at_i, dt_f] = 0$ (line 12), it implies that $psum[at_i, dt_f] = 0$.

3) *ScoreMax.* This method also works similarly except that it finds the maximum classification score (lines 9 and 10). The decision is also made similarly in line 16.

4 D-Features

We now describe how to compute s-features (similarity features) for each non-query document based on a query document. Since s-features are calculated using d-features (document features) of a non-query document and a query document, we first discuss d-features, which are extracted from each document itself. We employ 26 d-features and classify them into four categories: length d-features, frequency-based d-features, tf.idf-based d-features, and richness d-features. Note that although many features listed below have been used in various tasks before, our main contribution here is to look at the AA problem from a new angle and formulate it differently, which helps deal with the issues of existing approaches as discussed in the introduction.

4.1 Length D-Feature

We derive three length d-features from each raw document, namely:

average sentence length: in terms of word count in one sentence;

average word length: in terms of character count in one word;

average review length: in terms of word count in one document.

4.2 Frequency-Based D-Features

We first extract lexical, syntactic, and stylistic tokens from the raw documents and the parsed syntactic trees.

• *Lexical tokens:* word unigrams.

• *Syntactic tokens* (content-independent structures): POS n-grams ($1 \leqslant n \leqslant 3$) and rewrite rules[2,5]. In linguistics, a rewrite rule is the combination of a node and its immediate constituents in a syntactic tree[2]. For example, the rewrite rule for the noun phrase "the best book" is NP->DT+JJS+NN.

• *Common stylistic token:* K-length word ($1 \leqslant K \leqslant 15$), punctuations, and 157 function words such as "we" and "after"③.

• *Review specific stylistic tokens:* reflect styles specific to reviews: all cap words, pairs of quotation marks, pairs of brackets, exclamatory marks, two or more consecutive non-alphanumeric characters, contractions, model auxiliaries (e.g., should, must), word "recommend" or "recommended", sentences with the first letter capitalized, sentences starting with "This is (this is)" or "This was (this was)".

We then treat all these tokens as pseudo-words and count their frequency to form the frequency-based d-features.

4.3 TF-IDF Based D-Feature

For the tokens listed in Subsection 4.2, we also compute their tf.idf values. We list these two kinds of d-features separately because they will be used for different s-features later.

4.4 Richness D-Feature

This set of d-features is a set of vocabulary richness functions, which were used to quantify the diversity of the vocabulary in a text[2]. In this work, we apply the richness metrics to the counts of word unigrams, POS n-grams ($1 \leqslant n \leqslant 3$), and rewrite rules. Here POS n-grams and rewrite rules are treated as pseudo-words. Let T be the total number of tokens (words or pseudo-words), and $V(T)$ be the number of different tokens in a document, v be the highest frequency of occurrence of a token, and $V(m,T)$ be the number of tokens which occur m times in the document. We use the following six richness measures: Yule's characteristic (K)[44], Hapax dislegomena (S), Simpson's index (D), Honorës measure (R), Brunet's measure (W), and Hapax legomena (H)[24]. The formulae for these richness metrics are listed as below:

$$\begin{aligned}
K &= 10^4 \times \frac{\sum\limits_{m=1}^{v}(m^2 \times V(m,T) - T)}{T^2},\\
S &= \frac{V(2,T)}{V(T)},\\
D &= \frac{\sum\limits_{m=1}^{v}(m \times (m-1) \times V(m,T))}{T \times (T-1)},\\
R &= \frac{100 \times \log(T)}{1 - V(1,T)/V(T)},\\
H &= V(1,T),\\
W &= T^{V(T)^{-a}}, a = 0.17.
\end{aligned}$$

These give us a set of richness d-features about word unigrams, POS n-grams, and rewrite rules.

③ https://www.flesl.net/Vocabulary/SinglewordLists/functionwordlist.php, Nov. 2014.

5 S-Features

The extracted d-features are transformed into s-features in a similarity space for classification. S-features are a set of similarity functions on two documents. In this work, we adopt five types of s-features.

5.1 Sim4 Length S-Features

Defined by us, this is a set of four similarity functions used for d-feature vectors of length

$$1/(1+\log(1+|l_{wq}-l_{wd}|)),$$
$$1/(1+\log(1+|l_{sq}-l_{sd}|)),$$
$$1/(1+\log(1+|l_{rq}-l_{rd}|)),$$
$$\frac{\sum\limits_{m\in\{w,s,r\}}(l_{mq}\times l_{md})}{\sqrt{\sum\limits_{m\in\{w,s,r\}}(l_{mq})^2}\times\sqrt{\sum\limits_{m\in\{w,s,r\}}(l_{md})^2}},$$

where $l_{wq}.(l_{wd})$, $l_{sq}.(l_{sd})$, and $l_{rq}.(l_{rd})$ denote the average word, sentence, and review length respectively, either in query q or non-query document d. The four formulae produce four s-features.

5.2 Sim3 Sentence S-Features

This is a set of three similarity functions designed for sentence similarity[45]. Since the length of online review can be short, we apply this set (called Sim3) to the query and the non-query reviews. Sim3 s-features are used for frequency-based d-features. The three formulae are given as below:

$$\sum_{t\in q\cap d} f(t,d)/\left(\sum_{t\in q} f(t,q)+\sum_{t\in d} f(t,d)-\sum_{t\in q\cap d} f(t,d)\right),$$

$$\log_{t\in q\cap d}\left(\frac{N}{f(t,d)}\right)\times\frac{\sum\limits_{t\in q\cap d} f(t,d)}{\sum\limits_{t\in q} f(t,q)+\sum\limits_{t\in d} f(t,d)-\sum\limits_{t\in q\cap d} f(t,d)},$$

$$\frac{1}{1+\log(1+|l_q-l_d|)}\times\sum_{t\in q\cap d}\frac{N\times idf(t)}{1+|f(t,q)-f(t,d)|},$$

where $f(t,s)$ denotes the frequency count of token t in a document s, and l_q and l_d denote the average review length of the query and the non-query document, respectively.

5.3 Sim7 Retrieval S-Features

This is a set of seven similarity functions. These functions were originally used for text information retrieval[42], and in this work, they are applied to all frequency-based d-features:

$$\sum_{t\in q\cap d}\log(f(t,d)+1),$$
$$\sum_{t\in q\cap d}\log\left(\frac{|D|}{f(t,d)}+1\right),$$
$$\sum_{t\in q\cap d}\log(idf(t)),$$
$$\sum_{t\in q\cap d}\log\left(\frac{f(t,d)}{|d|}+1\right),$$
$$\sum_{t\in q\cap d}\log\left(\frac{f(t,d)}{|d|}\times idf(t)+1\right),$$
$$\log(BM25score),$$
$$\sum_{t\in q\cap d}\log\left(\frac{f(t,d)}{|d|}\times\frac{|D|}{f(t,d)}+1\right),$$

where $f(t,d)$ denotes the frequency count of token t in a non-query document d, q denotes the query document, D is the entire collection, $|.|$ is the size of a set, and *idf* is the inverse document frequency. These seven formulae can produce seven s-features.

5.4 SimC tf-idf S-Feature

This is the cosine similarity measure used for the d-vectors represented by the tf.idf-based d-features. SimC tf-idf produces one s-feature.

5.5 SimC Richness S-Feature

This is also the cosine similarity measure, but it is applied to the richness d-feature vectors. SimC richness produces one s-feature.

6 Experimental Evaluation

To evaluate the proposed approach, we first introduce the experiment setup, and then present the experimental results under various settings and compare them with two state-of-the-art baselines. All our experiments use the SVM$^{\text{perf}}$ classifier[46] with default parameter settings.

6.1 Experiment Setup

6.1.1 Dataset

We use a set of documents (reviews) and their authors (reviewers) collected from Amazon.com as our experiment data. We select the authors who have posted more than 20 reviews in the book category. All duplicate and near duplicate reviews are removed by comparing their text similarity using cosine. After preprocessing, we have 1 903 authors in our data. We randomly select 1 803 authors for training and 100 authors as unseen authors for testing. For seen author testing, we use reviews from some of the 1 803 authors that are not used in training. The numbers of reviews in the training set and in the unseen test author set are 99 055 and 5 380, respectively. The average review length is 223 words. We use the Stanford PCFG parser[47] to generate the grammar structure of sentences in each review for extracting syntactic d-features.

Training Data. We randomly choose a small number of reviews, i.e., one review or two reviews for each author as the query and 70% of his/her other reviews as q-positive reviews (the remaining 30% are used for seen author testing). The q-negative reviews consist of reviews randomly selected from the other 1 802 authors, two reviews per author. We also try choosing two and more queries from each author, but they make little difference in the final results.

Test data. We use two types of test data. One is for unseen authors, and the other is for seen authors. For both the seen and the unseen author data, we prepare the test set for each author as follows. We randomly choose 9 reviews from the author as the query set and other randomly selected 10 reviews as his/her q-positive test set. The q-negative set is from the other authors. The query and the test sets are disjoint. We do not use more queries or test reviews from each author since in the review domain, most authors only have a few reviews.

We vary the number of test authors, the number of queries, and the number of q-positive and q-negative reviews in testing. We use the following format to describe our test data:

$$[U|S]<n>\_Q<n>D<n>,$$

where U denotes unseen author, S seen author, Q query, D test document, and n a number. For example, $U50\_Q9D10$ stands for the test data with 50 unseen authors, 9 queries and 10 test documents from each author. * or # means a wildcard.

6.1.2 Baseline Methods

We now give two state-of-the-art baselines.

$MSMF + FLF$[16]. This latest supervised method for authorship attribution (AA) has reported the best results so far for AA with up to 100 authors. Besides the first level features (FLF), i.e., stylistic, lexical, perplexity, and syntactic features, it also generates a set of modality-based meta-features (MSMF). SVM is used as the learning algorithm.

SBM[15]. This method uses the cosine similarity comparison for AA with a single query. Each document is represented as a vector of frequencies of character 4-grams. SBM works in a retrieval-based method as follows. It randomly chooses 40% of character 4-grams to compute cosine similarity between the query document and each test document. This procedure repeats for 100 times. If a test document has the highest similarities for at least $\sigma \times 100$ times, the document is labeled as positive. Otherwise, it is labeled as negative.

6.2 Authorship Attribution

This is the traditional AA task. Traditional AA can only work with seen authors. However, our approach can perform the task for both seen and unseen authors. Below, we report the experimental results.

1) *Effects of Positive/Total Ratio in Training Set.* Here we evaluate the results of different proportions of positive s-training examples. We have experimented with different ratios of 0.3, 0.4, 0.5, 0.6, and 0.7. U^*_Q9D10 is used as the test data. The number of test authors (*) varies from 3, 5, 10, 30, 50, to 100. We use the ScoreSum method as it is the best (see below). Note that unseen authors are employed for testing to highlight the major advantage of our approach, but we will see shortly that for seen authors, our system achieves similar results.

We observe a stair-like best results in Table 1. For the datasets containing fewer authors, a higher positive(p)/total(t) ratio is more helpful. However, for data with more authors, a lower ratio is slightly better. This is mainly because a relatively low positive proportion enables the learner to observe more negative samples from other authors. Thus the generated model has the ability to differentiate reviews from more authors. For balanced results, all our experiments below use the model trained with the 0.5 ratio.

Table 1. Accuracy for Different p/t Ratios in Training

p/t	Number of Test Authors					
	3	5	10	30	50	100
0.3	80.00	78.00	65.00	56.00	49.60	**42.10**
0.4	80.00	78.00	66.00	56.67	**52.00**	**42.10**
0.5	83.33	82.00	**71.00**	**57.67**	50.80	41.50
0.6	**90.00**	**86.00**	70.00	56.33	48.60	39.60
0.7	83.33	84.00	67.00	54.00	40.20	37.20

2) *Effects of Different Numbers of Queries per Training Author.* Fig.6 shows the results with 1 query ($Q1$) and 2 queries ($Q2$) randomly chosen from the training data under 0.4($R0.4$) and 0.5($R0.5$) positive/total ratio settings. *AuthorNum* means the number of test authors. Other results are similar and omitted for clarity. We can see that $Q1$ and $Q2$ give very similar results. This indicates that our model is quite stable and not sensitive to the selection of query documents. Hence the time overhead for training can be very small. We use a model trained from 1 query in all other experiments.

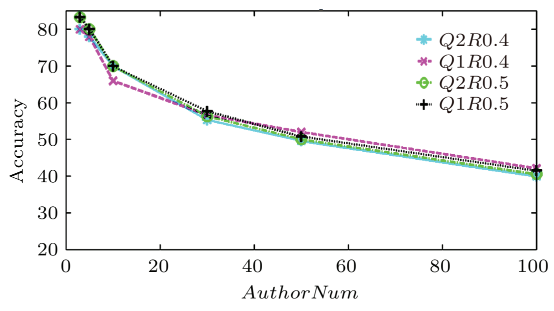

Fig.6. AA by different numbers of queries per training author.

3) *Effects of Different Decision Methods.* We now show the effects of the three proposed decision methods: Voting, ScoreSum and ScoreMax using our basic data of U^*_Q9D10 with varied number of test authors. Fig.7 shows that ScoreSum is the best. We also observe that Voting performs better than ScoreMax in all cases. This indicates that the decision made from a number of scores, either Voting or ScoreSum, is much more reliable than that made from only one score. ScoreSum is our default method.

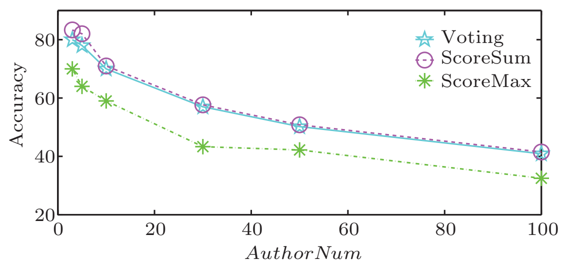

Fig.7. AA using different decision methods.

4) *Results of Seen and Unseen Authors.* Since our method does not differentiate reviews from seen or unseen authors, we study how it performs on seen and unseen authors.

Fig.8 shows the comparison results using the U^*_Q9D10 (unseen) and S^*_Q9D10 (seen) datasets. Fig.8 shows that our method performs similarly for seen and unseen authors, suggesting that the influence of training reviews of seen authors is limited. We can say that LSS generalizes well for both seen and unseen authors. Note that the two results are not strictly comparable as they use different author sets (seen and unseen), but we can see the same general trend.

5) *Effects of Different Number of Queries per Author in Testing.* Fig.9 shows that the accuracies, on all datasets, consistently increase when the number of queries (*QueryNum*) increases from 1 to 9. This indicates the positive effects of the number of queries, which is easy to understand.

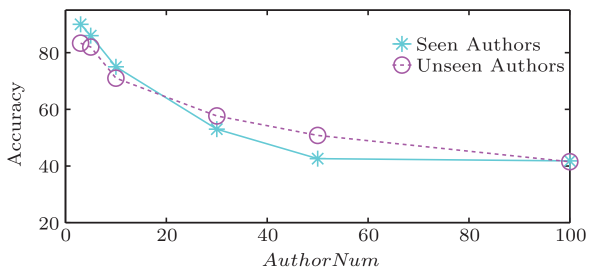

Fig.8. AA with seen and unseen authors.

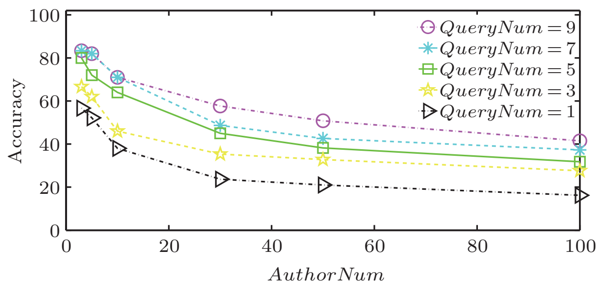

Fig.9. AA by different numbers of queries in testing.

6) *Effects of Number of Test Documents (Reviews) per Author.* We have experimented with 2, 4, 6, 8, and 10 test reviews per author.

Fig.10 shows the results on $U\#\_Q9D^*$ ($\# = 10$, 50, 100, and $^* = 2$, 4, 6, 8, 10), where *DocNum* denotes the number of test documents per author. From Fig.10, the other numbers of authors follow the same trend. From Fig.10, we find that a larger number of test documents/reviews have a slightly negative effect on classification. This is understandable. Since with more test documents, the task becomes harder, which brings down the performance slightly.

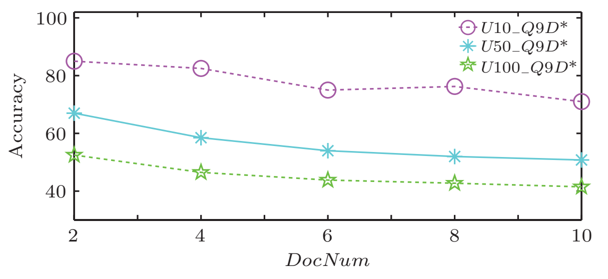

Fig.10. AA by changing the number of test reviews per author.

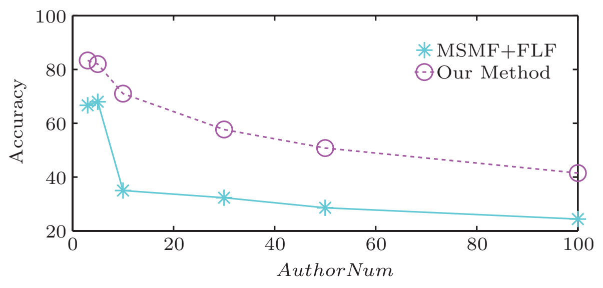

Fig.11. Comparison with MSMF+FLF on AA.

7) *Impact of Individual S-Feature Sets.* To show the effectiveness of individual s-feature sets, we use U^*_Q9D10 as an example. From Table 2, we see that SimCTfidf s-features are extremely important. Removing SimCTfidf drops the accuracy from 70% to 0%. Sim7Retrieval s-features are also useful. The impacts of other s-features are small. For 10 authors, without Sim4Len, the accuracy actually improves, but for 50 and 100 authors, Sim4Len is useful to some extent. Hence we use all features in all other experiments.

Table 2. Results on U^*_Q9D10 by Using Different S-Features

	Accuracy (10 Authors)	Accuracy (50 Authors)	Accuracy (100 Authors)
All features	**71.00**	**50.80**	**41.50**
No Sim4Len	74.00	50.60	40.20
No SimCRich	70.00	50.20	41.60
No SimCTfidf	0.00	0.20	0.10
No Sim7Retr	66.00	43.80	34.90
No Sim3Sent	68.00	49.20	38.40

8) *Comparison with Baselines.* We first compare LSS with MSMF+FLF[16] which cannot work with unseen authors. However, since in our LSS formulation, we need query documents in testing, we can use these query documents to train a classifier. This comparison is fair in the sense that our methods see only the same queries. It is also "unfair" because our model is trained from reviews of other authors, but the supervised learning method in MSMF+FLF is unable to exploit training documents from other authors. This shows the major advantage of our approach. For this set of experiments, we again use the data U^*_Q9D10, all features and the ScoreSum method. We vary neither the number of training reviews (the queries), nor the number of test documents from each author since for MSMF+FLF, these maximum numbers of queries and test documents already do very poorly. The comparison results are given in Fig.11.

From the results, it is clear that our LSS method dramatically outperforms MSMF+FLF on all data. The average accuracy increases 36.12% on all datasets. This can be explained since the supervised classification approaches like MSMF can hardly learn with such a small number of training documents (queries in our test data). In contrast, our LSS formulation can exploit other authors in training and thus greatly improve the accuracy.

We have also used seen authors in testing, but they make no difference.

Next we compare our method with SBM[15]. SBM can deal with unseen authors as it is a retrieval method based on cosine similarity. The original SBM does not perform the authorship assignment task for multiple authors. It only uses a single query to find another (one) document authored by the same author from a large set of documents. For multiple authors, we adopt the method. We first save the outputs of SBM and compute the average or total values of cosine similarities. Then we use this as the input and apply the same three decision methods to obtain the final author assignment for each test document for SBM. The details are as follows.

In the original SBM, each query has only one test document to identify, but we need to identify n ($n = 10$ in our case) test documents for each query. We have to extend SBM. Instead of identifying one, we make it to identify n (the number of test documents of each author) documents. Clearly, this is unrealistic because n is not known in practice. Our method does not use n. For the adapted SBM, as long as a document appears at least $\sigma \times 100$ times in any of the top n rank positions, we label it positive (written by the query author), and otherwise negative. σ is set to 0.9 in [15], but 0.9 performs very poorly with our data. We thus try $\sigma = 0.9$, 0.8, 0.7, 0.6, 0.5, 0.4, 0.2, 0.1. We found that a value higher than 0.5 will result in about 0% accuracy. We believe the reason is that in their test[15], the query and the positive test document are actually from the same document: the query is the first half and the test is the second half of the document. Due to the content similarity of the same document, their classification is much easier. In contrast, our queries and test documents are all individual and different reviews.

We have tried both the average and total value of cosine similarities for the three decision methods and conducted a series of experiments. Average-based SBM is very poor. Neither Voting nor ScoreMax is good. Here we only present the experimental results for SBM using the total value with the ScoreSum method.

In Fig.12, we compare our method with SBM based on the σ values which give the best results. This is unrealistic, but even in this unrealistic situation (in favor of SBM), our method performs dramatically better than SBM. For example, our method has a 50% accuracy increase over SBM on data $U10\_Q9D10$.

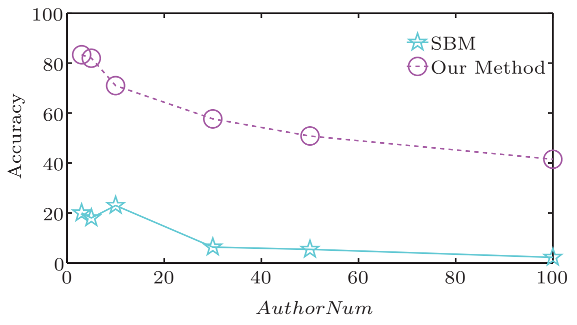

Fig.12. Comparison with SBM on AA.

In summary, we can conclude that our LSS method dramatically outperforms the two state-of-the-art supervised and unsupervised baselines. The reason, as we believe, is that LSS can exploit the documents of a large number of other authors to gain knowledge in the similarity space, while the two existing methods are unable to do that. It is also very important to note that for all our experiments (seen or unseen), our method uses only one classifier.

7 Conclusions

Although authorship attribution (AA) has been studied by many researchers, the problem is still an open one. In the domain of user-generated contents such as reviews or posts in the forums, there are some special characteristics, e.g., lack of training documents and unable to deal with authors not used in training, which make it difficult for current classification approaches to work effectively. In this paper, we proposed a novel formulation LSS of the AA problem to deal with the two issues by transforming learning from the original document space to a similarity space. In this new space, we can use documents from other authors to learn a model to help classify the documents of current test authors, regardless of whether the test authors are seen or unseen in training, as long as we have some query documents from the test authors. Our experimental results based on a large number of authors and their reviews show that the proposed LSS approach is highly effective, and it outperforms two state-of-the-art existing approaches by large margins.

References

[1] Grieve J. Quantitative authorship attribution: An evaluation of techniques. *Literary and Linguistic Computing*, 2007, 22(3): 251-270.

[2] Baayen H, van Halteren H, Tweedie F. Outside the cave of shadows: Using syntactic annotation to enhance authorship attribution. *Literary and Linguistic Computing*, 1996, 11(3): 121-132.

[3] Argamon S, Whitelaw C, Chase P, Hota S R, Garg N, Levitan S. Stylistic text classification using functional lexical features: Research articles. *Journal of the Association for Information Science and Technology*, 2007, 58(6): 802-822.

[4] Hedegaard S, Simonsen J G. Lost in translation: Authorship attribution using frame semantics. In *Proc. the 49th ACL*, June 2011, pp. 65-70.

[5] Hirst G, Feiguina O. Bigrams of syntactic labels for authorship discrimination of short texts. *Literary and Linguistic Computing*, 2007, 22(4): 405-417.

[6] Holmes D I, Forsyth R S. The federalist revisited: New directions in authorship attribution. *Literary and Linguistic Computing*, 1995, 10(2): 111-127.

[7] Koppel M, Schler J. Authorship verification as a one-class classification problem. In *Proc. the 21st ICML*, July 2004.

[8] Diederich J, Kindermann J, Leopold E, Paass G. Authorship attribution with support vector machines. *Applied Intelligence*, 2000, 19(1/2): 109-123.

[9] Escalante H J, Solorio T, Montes-y-Gómez M. Local histograms of character n-grams for authorship attribution. In *Proc. the 49th ACL*, June 2011, pp. 288-298.

[10] Li J, Zheng R, Chen H. From fingerprint to writeprint. *Communications of the ACM*, 2006, 49(4): 76-82.

[11] Stamatatos E, Fakotakis N, Kokkinakis G. Automatic text categorization in terms of genre and author. *Computational Linguistics*, 2000, 26(3): 471-495.

[12] Graham N, Hirst G, Marthi B. Segmenting documents by stylistic character. *Natural Language Engineering*, 2005, 11(4): 397-415.

[13] Seroussi Y, Bohnert F, Zukerman I. Authorship attribution with author-aware topic models. In *Proc. the 50th ACL*, July 2012, pp. 264-269.

[14] de Vel O, Anderson A, Corney M, Mohay G. Mining e-mail content for author identification forensics. *ACM SIGMOD Record*, 2001, 30(4): 55-64.

[15] Koppel M, Schler J, Argamon S. Authorship attribution in the wild. *Language Resources and Evaluation*, 2011, 45(1): 83-94.

[16] Solorio T, Pillay S, Raghavan S, y Gómez M M. Modality specific meta features for authorship attribution in Web forum posts. In *Proc. the 5th IJCNLP*, Nov. 2011, pp. 156-164.

[17] Kim S, Kim H, Weninger T, Han J, Kim H D. Authorship classification: A discriminative syntactic tree mining approach. In *Proc. the 34th SIGIR*, July 2011, pp. 455-464.

[18] Jindal N, Liu B. Opinion spam and analysis. In *Proc. WSDM*, Feb. 2008, pp. 219-230.

[19] Rudin C. The p-norm push: A simple convex ranking algorithm that concentrates at the top of the list. *The Journal of Machine Learning Research*, 2009, 10: 2233-2271.

[20] Yih W, Meek C. Improving similarity measures for short segments of text. In *Proc. AAAI*, Nov. 2007, pp. 1489-1494.

[21] Agichtein E, Brill E, Dumais S T, Ragno R. Learning user interaction models for predicting web search result preferences. In *Proc. the 29th SIGIR*, Aug. 2006, pp. 3-10.

[22] Mosteller F, Wallace D L. Inference and Disputed Authorship: The Federalist. Addison-Wesley, 1964.

[23] Argamon S, Levitan S. Measuring the usefulness of function words for authorship attribution. In *Proc. the 2005 ACH/ALLC Conference*, June 2005.

[24] Gamon M. Linguistic correlates of style: Authorship classification with deep linguistic analysis features. In *Proc. the 20th COLING*, Aug. 2004, Article No. 611.

[25] Peng F, Schuurmans D, Wang S, Keselj V. Language independent authorship attribution using character level language models. In *Proc. EACL*, April 2003, pp. 267-274.

[26] Burrows J F. Not unless you ask nicely: The interpretative nexus between analysis and information. *Literary and Linguistic Computing*, 1992, 7(2): 91-109.

[27] Sanderson C, Guenter S. Short text authorship attribution via sequence kernels, Markov chains and author unmasking: An investigation. In *Proc. EMNLP*, July 2006, pp. 482-491.

[28] Madigan D, Genkin A, Lewis D, Argamon S, Fradkin D, Ye L. Author identification on the large scale. In *Proc. CSNA*, June 2005.

[29] Cao Y, Xu J, Liu T, Li H, Huang Y, Hon H. Adapting ranking SVM to document retrieval. In *Proc. the 29th SIGIR*, Oct. 2006, pp. 186-193.

[30] Stamatatos E. A survey of modern authorship attribution methods. *Journal of the Association for Information Science and Technology*, Aug. 2009, 60(3): 538-556.

[31] Hoover D L. Statistical stylistics and authorship attribution: An empirical investigation. *Literary and Linguistic Computing*, 2001, 16(4): 421-444.

[32] Zheng R, Li J, Chen H, Huang Z. A framework for authorship identification of online messages: Writing style features and classification techniques. *Journal of the Association for Information Science and Technology*, 2006, 57(3): 378-393.

[33] Uzuner Ö, Katz B. A comparative study of language models for book and author recognition. In *Proc. the 2nd IJCNLP*, Oct. 2005, pp. 969-980.

[34] Zhao Y, Zobel J. Effective and scalable authorship attribution using function words. In *Proc. the 2nd Asia Information Retrieval Symposium*, Oct. 2005, pp. 174-189.

[35] Luyckx K, Daelemans W. Authorship attribution and verification with many authors and limited data. In *Proc. the 22nd COLING*, Aug. 2008, pp. 513-520.

[36] Vapnik V N. Statistical Learning Theory. Wiley-Interscience, 1998.

[37] Graepely T, Herbrichz R, Bollmann-Sdorraz P, Obermayery K. Classification on pairwise proximity data. In *Proc. NIPS*, Jan. 1999, pp. 438-444.

[38] Chen Y, Garcia E K, Gupta M R, Rahimi A, Cazzanti L. Similarity-based classification: Concepts and algorithms. *The Journal of Machine Learning Research*, 2009, 10: 747-776.

[39] Pezkalska E, Duin R P W. Dissimilarity representations allow for building good classifiers. *Pattern Recognition Letters*, 2002, 23(8): 943-956.

[40] Liao L, Noble W S. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proc. the 6th RECOMB*, April 2002, pp. 225-232.

[41] Wang L, Yang C, Feng J. On learning with dissimilarity functions. In *Proc. the 24th ICML*, June 2007, pp. 991-998.

[42] Balcan M F, Blum A, Srebro N. A theory of learning with similarity functions. *Machine Learning*, 2008, 72(1/2): 89-112.

[43] Kar P, Jain P. Similarity-based learning via data driven embeddings. In *Proc. the 25th NIPS*, Dec. 2011.

[44] Yule G U. The Statistical Study of Literary Vocabulary. Cambridge University Press, 1944.

[45] Metzler D, Bernstein Y, Croft W B, Moffat A, Zobel J. Similarity measures for tracking information flow. In *Proc. the 14th CIKM*, Oct. 2005, pp. 517-524.

[46] Joachims T. Training linear SVMs in linear time. In *Proc. the 12th KDD*, Aug. 2006, pp. 217-226.

[47] Klein D, Manning C D. Accurate unlexicalized parsing. In *Proc. the 41st ACL*, July 2003, pp. 423-430.

Tie-Yun Qian is an associate professor at the State Key Laboratory of Software Engineering at Wuhan University. She received her B.S. degree in computer science from Wuhan University of Technology in 1991, and her Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2006. Her current research interests include text mining, web mining, and natural language processing. She has published over 20 papers in top conferences including ACL, EMNLP, SIGIR, etc. She is a member of CCF and ACM. She has served as program committee member of many leading conferences: WWW, COLING, DASFAA, WAIM, and APWeb.

Bing Liu is a professor of computer science at the University of Illinois at Chicago (UIC). He received his Ph.D. degree in artificial intelligence from the University of Edinburgh. Before joining UIC, he was a faculty member at the National University of Singapore. His current research interests include sentiment analysis and opinion mining, data mining, machine learning, and natural language processing (NLP). He has published extensively in top conferences and journals. He is also the author of two books: "Sentiment Analysis and Opinion Mining" (Morgan and Claypool) and "Web Data Mining: Exploring Hyperlinks, Contents and Usage Data" (Springer). In addition to research impacts, his work has also made important social impacts. Some of his work has been widely reported in the press, including a front-page article in The New York Times. On professional services, Liu has served as program chair of many leading data mining related conferences of ACM, IEEE, and SIAM: KDD, ICDM, CIKM, WSDM, SDM, and PAKDD, as associate editor of several leading data mining journals, e.g., TKDE, TWEB, DMKD, and as area/track chair or senior technical committee member of numerous NLP, data mining, and Web technology conferences. He currently also serves as the chair of ACM SIGKDD, and is an IEEE fellow.

Qing Li is a professor at the City University of Hong Kong, where he also directs the Multimedia Software Engineering Research Centre. His research interests include multimedia databases, social networks and recommender systems. Prof. Li has authored/co-authored over 90 journal papers, and 240 conference publications in these and related areas. He is a fellow of IET, a distinguished member of CCF, and a senior member of IEEE. He is a steering committee member of ACM RecSys, DASFAA, ER, ICWL, and WISE Society.

Jianfeng Si received his Ph.D. degree in computer science from the City University of Hong Kong, China, in 2013. He is currently a research scientist in the Data Analytics Department of Institute for Infocomm Research, Singapore. His research interests include text mining, natural language processing, and social media analysis.