Identifying Multiple Userids of the Same Author

Tieyun Qian
State Key Laboratory of Software Eng.
Wuhan University
16 Luojiashan Road
Wuhan, Hubei 430072, China
qty@whu.edu.cn

Bing Liu
Department of Computer Science
University of Illinois at Chicago
851 South Morgan St., Chicago
IL, USA, 60607
liub@cs.uic.edu

Abstract

This paper studies the problem of identifying users who use multiple userids to post in social media. Since multiple userids may belong to the same author, it is hard to directly apply supervised learning to solve the problem. This paper proposes a new method, which still uses supervised learning but does not require training documents from the involved userids. Instead, it uses documents from other userids for classifier building. The classifier can be applied to documents of the involved userids. This is possible because we transform the document space to a similarity space and learning is performed in this new space. Our evaluation is done in the online review domain. The experimental results using a large number of userids and their reviews show that the proposed method is highly effective.

1 Introduction

It is common knowledge that some users in social media register multiple accounts/userids to post articles, blogs, reviews, etc. There are many reasons for doing this. For example, due to past postings, a user may become despised by others. He/she then registers another userid in order to regain his/her status. A user may also use multiple userids to instigate controversy or debates to popularize a topic to make it "hot" or even just to promote activities at a website. Yet, a user may also use multiple userids to post fake or deceptive opinions to promote or demote some products (Liu, 2012). It is thus important to develop technologies to identify such *multi-id users*. This paper deals with this problem based on writing style and other linguistic clues.

Problem definition: Given a set of userids $ID = \{id_1, \ldots, id_n\}$ and each id_i has a set of documents D_i, we want to identify userids that belong to the same physical author.

The main related works to ours are in the area of authorship attribution (AA), which aims to identify authors of documents. AA is often solved using supervised learning. Let $A = \{a_1, \ldots, a_k\}$ be a set of authors (or classes) and each author $a_i \in A$ has a set of training documents D_i. A classifier is then built to decide the author a of each test document d, where $a \in A$. We will discuss this and other related works in Section 2.

This supervised AA formulation, however, is not suitable for our task because we only have userids but not real authors. Since some of the userids may belong to the same author, we cannot treat each userid as a class because in that case, we will be classifying based on userids, which won't help us find authors with multiple userids (see Section 7 also).

This paper proposes a novel algorithm. To simplify the presentation, we assume that at most two userids can belong to a single author, but the algorithm can be extended to handle more than two userids from the same author. Using this assumption, the algorithm works in two steps:

1. *Candidate identification*: For each userid id_i, we first find the most likely userid id_j ($i \neq j$) that may have the same author as id_i. We call id_j the *candidate* of id_i. We also call this function *candid-iden*, i.e., id_j = *candid-iden*(id_i). For easy presentation, here we only use one argument for

* The work was mainly done when the first author was visiting the University of Illinois at Chicago.

candid-iden. In the computation, it needs more arguments (see Section 4).

2. *Candidate confirmation*: In the reverse order, we apply the function *candid-iden* on id_j, which produces id_k, i.e., $id_k = candid\text{-}iden(id_j)$.

   Decision making: If $k = i$, we conclude that id_i and id_j are from the same author. Otherwise, id_i and id_j are not from the same author.

The key of the algorithm is *candid-iden*. An obvious approach for *candid-iden* is to use an information retrieval method. We can first split the documents D_i of each id_i into two subsets, a *query set* Q_i and a *sample set* S_i. We then compare each query document in Q_i with each sample document in S_j from other userids id_j ($\in ID - \{id_i\}$). *Cosine* can be used here for similarity comparison. All the similarity scores are then aggregated and used to rank the userids in $ID - \{id_i\}$. The top ranked userid is the candidate for id_i. Note that partitioning the documents of a userid id_i into the query set Q_i and the sample set S_i is crucial here. We cannot use all documents in D_i to compare with all documents in D_j. If so and we get $candid\text{-}iden(id_i) = id_j$, we will definitely get $candid\text{-}iden(id_j) = id_i$ since the similarity function is symmetric.

This cosine similarity based method, however, does not work well (see Section 7). We propose a supervised learning method to compute the scores. For this, we need to reformulate the problem.

The idea of this reformulation is to learn in a similarity space rather than in the original document space as in traditional AA. In the new formulation, each document *d* is still represented as a feature vector, but the vector no longer represents the document *d* itself. Instead, it represents a set of similarities between the document *d* and a query *q*. We call this method *learning in the similarity space* (LSS).

Specifically, in LSS, each document *d* is first represented with a *document space vector* (called a *d-vector*) based on the document itself as in the traditional classification learning of AA. Each feature in the *d*-vector is called a *d-feature* (*document-feature*). A query document *q* is represented in the same way. We then produce a *similarity vector sv* (called *s-vector*) for *d*. *sv* consists of a set of similarity values between document *d* (in a *d*-vector) and query *q* (in a *d*-vector):

$$sv = Sim(d, q),$$

where *Sim* is a similarity function consists of a set of similarity measures. Thus, the *d*-vector for document *d* in the document space is transformed to an *s*-vector *sv* for *d* in the similarity space. Each feature in *sv* is called an *s-feature*. For example, we have the following *d*-vector for query *q*:

q: 1:1 2:1 6:2

where *x*:*z* represents a *d*-feature *x* (a word) and its frequency *z* in *q*. We also have two non-query documents, one is d_1 which is written by the author of query *q* and the other is d_2 which is not written by query author *q*. Their *d*-vectors are:

d_1: 1:2 2:1 3:1 $\quad\quad$ d_2: 2:2 3:1 5:2

If we use *cosine* as the first similarity measure in *Sim*, we can generate an *s*-feature 1:0.50 for d_1 ($cosine(q, d_1) = 0.50$) and an *s*-feature 1:0.27 for d_2 ($cosine(q, d_2) = 0.27$). If we have more similarity measures more *s*-features can be produced. The resulting two *s*-vectors for d_1 and d_2 with their class labels, 1 and -1, are as follows:

d_1: 1 1:0.50 … $\quad\quad$ d_2: -1 1:0.27 …

Class 1 means "written by author of query *q*", also called *q-positive*, and class -1 means "not written by author of query *q*", also called *q-negative*.

LSS gives us a two-class classification problem. In this formulation, a test userid and his/her documents do not have to be seen in training as long as a set of known documents from this userid is available. Any supervised learning method can be used to build a classifier. We use SVM. The resulting classifier is employed to compute a score for each review to be used in the two-step algorithm above to find the candidate for each userid and then the userids with the same authors.

Due to the use of query documents, the LSS formulation has some resemblance to document ranking based on learning to rank (Li, 2011; Liu, 2011). However, LSS is very different because we turn the problem into a supervised classification problem. The key difference between learning to rank and classification is that ranking will always put some documents at the top even if the desired documents do not exist. However, classification will not return any document if the desired documents do not exist in the test data (unless there are classification errors). Our Type II experiments in Section 7 were specifically designed for testing such non-existence situations.

Using online review as the application domain, we conduct experiments on a large number of reviews and their author/reviewer userids from Amazon.com. The results show that the proposed algorithm is highly accurate and outperforms three strong baselines markedly.

2 Related Work

A similar problem was attempted in (Chen et al., 2004) in the context of open forums where users interact with each other in their discussions. Their method is based on post relationships and intervals between posts. It does not use any linguistic clues. It is thus not applicable to domains like online reviews. Reviews do not involve user interactions since each review is independent of other reviews. Novak et al. also solved the same problem under the name of "Anti-aliasing" (Novak et al., 2004). They used a clustering based method which assumed the number of actual authors is known. This is unrealistic in practice as there is no way to know which author has and does not have multiple ids.

Our work is also related to authorship attribution (AA). However, to our knowledge, our problem has not been attempted in AA. Existing works focused on two main themes: finding good writing style features, and developing effective classification methods. On finding good features (d-features in our case), it was found that the most promising features are function words (Mosteller, 1964; Argamon and Levitan, 2004; Argamon et al., 2007) and rewrite rules (Halteren et al., 1996). Length (Gamon 2004; Graham et al., 2005), richness (Halteren et al., 1996; Koppel and Schler, 2004), punctuations (Graham et al., 2005), character n-grams (Grieve, 2007; Hedegaard and Simonsen, 2011), word *n*-grams (Burrows, 1992; Sanderson and Guenter 2006), POS *n*-grams (Gamon, 2004; Hirst and Feiguina, 2007), syntactic category pairs (Narayanan et al., 2012) are also useful.

On classification, numerous methods have been tried, e.g., Bayesian analysis (Mosteller, 1964), discriminant analysis (Stamatatos et al., 2000), PCA (Hoover, 2001), neural networks (Graham et al., 2005; Zheng et al., 2006; Graham et al., 2005), clustering (Sanderson and Guenter, 2006), decision trees (Uzuner and Katz, 2005; Zhao and Zobel, 2005), regularized least squares classification (Narayanan et al., 2012), and SVM (Diederich et al., 2000; Gamon 2004; Koppel and Schler, 2004; Hedegaard and Simonsen, 2011). Among them, SVM was found to be most accurate (Li et al., 2006; Kim et al., 2011). Although we also use supervised learning, we do not learn in the original document space as these existing methods do. The transformation is important because it enables us to use documents from other authors in training. The traditional supervised learning (TSL) cannot do that. In our case, the only documents that TSL can use for training are the queries in the testing set. However, as we will see in our experiments, such a method performs poorly.

Since we use online reviews as our experiment domain, our work is related to fake review detection (Jindal and Liu, 2008) as imposters can use multiple userids to post fake reviews. Existing research has proposed many methods to detect fake reviewers (Lim et al., 2010; Wang et al., 2011; Mukherjee et al., 2012) and fake reviews (Jindal and Liu, 2008; Ott et al., 2011, 2012; Li et al., 2011; Feng et al., 2012). However, none of them identifies userids belonging to the same person.

3 Learning in the Similarity Space

We now formulate the proposed supervised learning in the similarity space (LSS), which will be used in the *candid-iden* function in our algorithm to be discussed in Section 4.

The key difference between LSS and the classic document space learning is in the document representation. Another difference is in the testing phase. We discuss testing first.

Test data: We are given:

- A query q from query author (userid) aq
- A set of test documents $DT = \{dt_1, \ldots, dt_m\}$.

Goal: classify the test documents into those authored by aq and those not authored by aq.

We note the following points:

i) This is like a retrieval scenario, but we use supervised learning to perform the task.

ii) Unlike traditional supervised classification, here the test query author aq does not have to be used in training. But we are given a query document q from aq. Clearly, in practice, we can have multiple query documents from aq, which we will discuss in Section 4.

Training document representation: As noted earlier, each document is represented with a similarity vector (*s-vector*) computed using a similarity

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},
($Sim(dr_{ijk}, q_{ij})$, 1)
// produce negative *s*-training examples
7. select a set of documents from the rest of authors
$DR_{ij,rest} \subseteq (DR_1 \cup \ldots \cup DR_n) - DR_i$
8. **For** each document $dr_{ijk,rest} \in DR_{ij,rest}$
9. produce an *s*-training example for $dr_{ijk,rest}$,
($Sim(dr_{ijk,rest}, q_{ij})$, -1)

Figure 1: Generating *s*-training examples

// Author ar_1 –
// positive (1) *s*-training examples
($Sim(dr_{111}, q_{11})$, 1), …, ($Sim(dr_{11p}, q_{11})$, 1)
…
($Sim(dr_{1k1}, q_{1k})$, 1), …, ($Sim(dr_{1kp}, q_{1k})$, 1)
// negative (-1) *s*-training examples
($Sim(dr_{111.rest}, q_{11})$, -1), …, ($Sim(dr_{11u.rest}, q_{11})$, -1)
…
($Sim(dr_{1k1.rest}, q_{1k})$, -1), …, ($Sim(dr_{1ku.rest}, q_{1k})$, -1)
…

Figure 2: *s*-training examples

function *Sim*. *Sim* takes a query document and a non-query document and produces a vector of similarity values or s-features to represent the non-query document. We present the detail below:

Let the set of training authors be $AR = \{ar_1, .., ar_n\}$. Each author ar_i has a set of documents DR_i. Each document in DR_i is first represented with a document vector (or *d*-vector). The algorithm for producing the training set, called *s-training set*, is given in Figure 1.

We randomly select a small set of queries Q_i from documents DR_i of each author ar_i (lines 1, and 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) to be the positive documents for q_{ij}, called *q-positive* and labeled 1. 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 *Sim* (lines 5, 6). In line 7, it selects a set of documents $DR_{ij,rest}$ from other authors to be the negative documents for q_{ij}, called *q*-negative and labeled -1. For each document $dr_{ijk,rest}$ in $DR_{ij,rest}$ (line 8), a *q*-negative *s*-training example with label -1 is generated for dr_{ijk} by computing the similarities of q_{ij} and $dr_{ijk,rest}$ using *Sim* (line 9). How to select Q_i, DR_{ij} and $DR_{ij,rest}$ (lines 2, 4 and 7) is left open intentionally to give flexibility in implementation.

1. **For** each document set D_i of $id_i \in ID$ **do**
2. partition D_i into two subsets:
(1) query set Q_i and (2) sample set S_i;
3. **For** each document set D_i of $id_i \in ID$ **do**
// step 1: *candidate identification*
4. id_j = candid-iden(id_i, *ID*), $i < j$;
// step 2: *candidate confirmation*
5. id_k = candid-iden(id_j, *ID*), $k \neq j$;
6. **If** $k = i$ **then** id_i and id_j are from the same author
8. **else** id_i and id_j are not from the same author

Figure 3: Identifying userids from the same authors

Function candidate-iden(id_i, *ID*)
1. **For** each sample document set S_j of $id_j \in ID$-{id_i} **do**
2. $pcount[id_j]$, $psum[id_j]$, $psqsum[id_j]$, $max[id_j] = 0$;
3. **For** each query $q_i \in Q_i$ **do**
4. **For** each sample $s_{jf} \in S_j$ **do**
5. $ss_{jf} = <(id_i, q_i), (Sim(s_{jf}, q_i), ?)>$;
6. Classify ss_{jf} using the classifier built earlier;
7. **If** ss_{jf} is classified positive, i.e., 1 **then**
8. $pcount[id_j] = pcount[id_j] + 1$;
9. $psum[id_j] = psum[id_j] + ss_{jf}.score$
10 $psqsum[id_j] = psqsum[id_j] + (ss_{jf}.score)^2$
11. **If** $ss_{if}.score > max[id_j]$ **then**
12. $max[id_j] = sr_{jf}.score$
// Four methods to decide which id_j is the candidate for id_i
13. **If** for all $id_j \in ID$-{id_i}, $pcount[id_i] = 0$ **then**
14. $cid = \arg\max_{id_j \in ID-\{id_i\}} (\max[id_j])$
15. **Else** $cid = \arg\max_{id_j \in ID-\{id_i\}} (\frac{pcount[id_j]}{|S_j|})$ // **1. Voting**
16. $cid = \arg\max_{id_j \in ID-\{id_i\}} (\frac{psum[id_j]}{|S_j|})$ // 2. **ScoreSum**
17. $cid = \arg\max_{id_j \in ID-\{id_i\}} (\frac{(psum[id_j])^2}{|S_j|})$ // 3. **ScoreSqSum**
18. $cid = \arg\max_{id_j \in ID-\{id_i\}} (\max[id_j])$ // **4. ScoreMax**
19. return *cid*;

Figure 4: Identifying the candidate

This formulation gives us a two-class classification problem. The classes are 1 (*q-positive* meaning "written by author of query q_{ij}") and -1 (*q-negative* meaning "not written by author of query q_{ij}." Figure 2 shows what the *s*-training data looks like. For easy presentation, we assume that there are *k* queries in every Q_i, and *p* documents in every DR_{ij} and *u* documents in every $DR_{ij,rest}$. The number of authors is *n*. Each author ar_i generates $k \times (p+u)$ *s*-training examples. As we will see in Section 7, *k* can be very small, even 1.

Complexity: In the worst case, every document

can serve as a query or a non-query document. Then we need to compute all pairwise document similarities. If the number of training documents is m, the complexity is $O(m^2)$, which is both space and computation expensive. However, in practice, we don't need all pairwise comparisons. Only a small subset is sufficient (see Section 7).

Test document representation: Like training documents, test documents are represented as *s*-vectors as well in the similarity space.

Given a query q from author aq and a set of test documents DT, each test document dt_i is converted to a *s*-vector $sv_i = Sim(dt_i, q)$. To reflect sv_i is computed based on query q from author aq, a *s-test* case is thus represented as $<(aq, q), (sv_i, ?)>$.

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

Testing: The classifier is applied to each *s*-test case $<(aq, q), (sv_i, ?)>$ (where $sv_i = S(dt_i, q)$) to give it a class *q-positive* or *q-negative*. Note that the classifier is only applied on sv_i.

In most cases, classification based on a single query is inaccurate. Using multiple queries of an author can classify much more accurately.

4 Identify Userids of the Same Author

We now expand the sketch of the two-step algorithm in Section 1 based on the problem statement in Section 1. The algorithm is given in Figure 3.

Lines 1-2 partitions the documents set D_i of each id_i in $ID = \{id_1, id_2, \ldots, id_n\}$, the set of userids that we are working on. How to do the partition is flexible (see Section 7). Line 4 is the step 1 of candidate identification, and line 5 is the step2 of candidate confirmation. Lines 6-8 is the decision making of step 2 (see Section 1). Line 6 produced a classification score using the classifier described in Section 3. The key function here is *candid-iden*. Its algorithm is in Figure 4.

The *candid-iden* function takes two arguments: the query userid id_i and the whole set of userids ID. It classifies each sample ss_{jf} in sample set S_j of $id_j \in ID\text{-}\{id_i\}$ to positive (q_i-positive) or negative (q_i-negative) (lines 4, 5, 6). We then aggregate the classification results to determine which userid is likely to have the same author as id_i.

One simple aggregation method is voting. We count the total number of positive classifications of the sample documents of each userid in $ID\text{-}\{id_i\}$. The userid id_j with the highest count is the candidate *cid* which may share the same author as query id_i. *cid* is returned as the candidate.

There are also other methods, which can depend on what output value the classifier produces. Here we propose four methods including the voting method above. The other three methods requires the classifier to produces a prediction score, which reflects the positive and negative certainty. Many classification algorithms produce such a score. Here we use SVM. For each classification, SVM outputs a positive or negative score indicating the certainty that the test case is positive or negative.

To save space, all four alternative methods are given in Figure 4. Line 2 initializes some variables for recording the aggregated values for the final decision making. The four methods are as follows:

1). **Voting**: For each sample from userid id_j, if it is classified as positive, one vote/count is added to $pcount[id_j]$. The userid with the highest *pcount* is regarded as the candidate userid, *cid* (line 15). Note that the normalization is applied because the sizes of the sample sets S_j can be different for different userids. Lines 13 and 14 mean that if all documents of all userids are classified as negative ($pcount[id_j] = 0$, which also implies $psum[id_j] = psqsum[id_j] = 0$), we use method 4).

2). **ScoreSum**: This method works similarly to the voting method above except that instead of counting positive classifications, this method sums up all scores of positive classifications in $psum[id_j]$ for each userid (line 9). The decision is also made similarly (line 16).

3). **ScoreSqSum**: This method works similarly to ScoreSum above except that it sums up the squared scores of positive classifications in $psqsum[id_j]$ for each userid (line 10). The decision is also made similarly (line 17).

4). **ScoreMax**: This method works similarly to the voting method as well except that it finds the maximum classification score for the documents of each userid (lines 11 and 12). The decision is made in line 18.

5 D-features

We now compute *s*-features (similarity features)

for each non-query document based on a query document. Since *s*-features are calculated using *d*-features of a non-query document and a query document, we thus discuss *d*-features first, which are extracted from each document itself. We employ 26 *d*-features in four categories: length *d*-features, frequency based *d*-features, tf.idf based *d*-features, and richness *d*-features. Although many features below have been used in various tasks before, our key contribution is solving a new problem based on a new learning formulation (LSS).

Length d-feature: We derive three length *d*-features from each raw document: (1) *average sentence length* (in terms of word count); (2) *average word length* (in terms of character count in one word); (3) *average document length* (in terms of word count in one document).

Frequency based d-features: We extract lexical, syntactic, and stylistic tokens from the raw documents and the parsed syntactic trees to produce the following features:

- *Lexical tokens: word unigrams*
- *Syntactic tokens:* content-independent structures: POS n-grams ($1 \leq n \leq 3$) and rewrite rules (Halteren et al., 1996; Hirst and Feiguina, 2007). A rewrite rule is a combination of a node and its immediate constituents in a syntactic tree. For example, the rewrite rule for "the best book" is NP->DT+JJS+NN.
- *Common stylistic token: K*-length word ($1 \leq K \leq 15$), punctuations, and 157 function words (*www.flesl.net/Vocabulary/SinglewordLists/functionwordlist.php*).
- *Review specific stylistic tokens:* These tokens reflect styles of reviews: *all cap words, pairs of quotation marks, pairs of brackets, exclamatory marks, contractions, two or more consecutive non-alphanumeric characters, 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 these tokens as pseudo-words and count their frequency to form frequency *d*-features.

TF-IDF based d-feature: For the tokens listed in the frequency based features above, 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.

Richness d-features: This is a set of vocabulary richness functions used to quantify the diversity of vocabulary in text (Holmes and Forsyth, 1995). In this paper, we apply them to the counts of word unigrams, POS *n*-grams ($1 \leq n \leq 3$), 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, 1944; Burrows, 1992; Halteren et al., 1996) given in Table 1: Yule's characteristic (*K*), Hapax dislegomena (*S*), Simpson's index (*D*), Honorës measure (*R*), Brunet's measure (*W*), and Hapax legomena (*H*). They give us a set of richness *d*-features about word unigrams, POS n-grams, and rewrite rules.

Table 1. Richness metrics

$K = 10^4 * \frac{\sum_{m=1}^{v} (m^2 * V(m,T) - T)}{T^2}$	$S = \frac{V(2,T)}{V(T)}$
$D = \frac{\sum_{m=1}^{v} (m*(m-1)*V(m,T))}{T*(T-1)}$	$R = \frac{100*\log(T)}{1 - V(1,T)/V(T)}$
$H = V(1,T)$	$W = T^{V(T)^{-a}}, a = 0.17$

6 S-Features

The extracted *d*-features are transformed into *s*-features, which are a set of similarity functions on two documents. We adopt five types of *s*-features.

Sim4 Length s-features: This is a set of four similarity functions defined by us. They are used for *d*-feature vectors of length. The four formulae are given in Table 2, where l_{wq}. (l_{wd}), l_{sq}. (l_{sd}), and l_{rq}. (l_{rd}) denote the average word, sentence, and document length respectively, either in query *q* or non-query document *d*. They produce four *s*-features.

Table 2. Sim4 for computing length s-features

$1/(1+\log(1+\lvert l_{wq} - l_{wd} \rvert))$
$1/(1+\log(1+\lvert l_{sq} - l_{sd} \rvert))$
$1/(1+\log(1+\lvert l_{rq} - l_{rd} \rvert))$
$\sum_{m \in \{w,s,r\}} (l_{mq} * l_{md}) / \sqrt{\sum_{m \in \{w,s,r\}} (l_{mq})^2} * \sqrt{\sum_{m \in \{w,s,r\}} (l_{md})^2}$

Sim3 Sentence s-features: This is a set of three sentence similarity functions (Metzler et al., 2005). We apply them (called Sim3) to documents. Sim3 *s*-features are used for frequency based *d*-features. The three formulae are given in Table 3, where $f(t, s)$ is the frequency count of token t in a document s, and l_q and l_d are the average document length of the query and non-query document, respectively.

Table 3. Sim3 for computing sentence s-features

$\sum_{t \in q \cap d} f(t,d) / (\sum_{t \in q} f(t,q) + \sum_{t \in d} f(t,d) - \sum_{t \in q \cap d} f(t,d))$
$\log_{t \in q \cap d}(\frac{N}{f(t,d)}) * \frac{\sum_{t \in q \cap d} f(t,d)}{\sum_{t \in q} f(t,q) + \sum_{t \in d} f(t,d) - \sum_{t \in q \cap d} f(t,d)}$
$\frac{1}{1+\log(1+\lvert l_q - l_d \rvert)} * \sum_{t \in q \cap d} \frac{N * idf(t)}{1+\lvert f(t,q) - f(t,d) \rvert}$

Sim7 Retrieval s-features: This is a set of seven similarity functions (Table 4) applicable to all frequency based *d*-features. These functions were used in information retrieval (Cao et al., 2006).

Table 4. Sim7 for computing retrieval *s*-features

$\sum_{t \in q \cap d} \log(f(t,d)+1)$	$\sum_{t \in q \cap d} \log(\frac{\lvert D \rvert}{f(t,d)}+1)$
$\sum_{t \in q \cap d} \log(idf(t))$	$\sum_{t \in q \cap d} \log(\frac{f(t,d)}{\lvert d \rvert}+1)$
$\sum_{t \in q \cap d} \log(\frac{f(t,d)}{\lvert d \rvert} * idf(t)+1)$	$\log(BM25score)$
$\sum_{t \in q \cap d} \log(\frac{f(t,d)}{\lvert d \rvert} * \frac{\lvert D \rvert}{f(t,d)}+1)$	

In Table 4, $f(t, d)$ denotes the frequency count of token t in a non-query document d, q denotes the query, D is the entire collection, $|.|$ is the size of a set, and *idf* is the inverse document frequency. These 7 formulae can produce 7 *s*-features.

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

SimC Richness s-feature: This is also cosine similarity. However, it is applied to the richness *d*-feature vectors, and produces one *s*-feature.

7 Experimental Evaluation

We now evaluate the proposed approach and compare it with baselines. All our experiments use the SVM^{perf} classifier (Joachims, 2006).

7.1 Experiment Setup

Experiment Data: We use a set of reviews and their authors/reviewers from Amazon.com as our experiment data. We select the authors who have posted more than 30 reviews in the book category. After cleaning, we have 831 authors, 731 authors for training and 100 authors for testing. The numbers of reviews in the training and test author set are 59256 and 14308, respectively. We use the Stanford parser (Klein and Manning, 2003) to generate the grammar structure of review sentences for extracting syntactic *d*-features. Note that the authors here are in fact userids. However, since they are randomly selected from a large number of userids, the probability that two sampled userids belong to the same person is very small. Thus, it should be safe to assume that each userid here represents a unique author.

Training data: We randomly choose 1 (one) review for each author as the query and all of his/her other reviews as *q*-positive reviews. The *q*-negative reviews consist of reviews randomly selected from the other 730 authors, two reviews per author. We also tried to use more queries from each author, but they make little difference.

Test data: The test authors are all unseen, i.e., their reviews have not been used in training. We prepare the test case for each author as follows.

We first divide the reviews of each author into two equal subsets. The purpose is to simulate the situation where there are two userids id_{ia} and id_{ib} from the same author a_i. Our objective is that given one userid id_{ia} and its query set, we want to find the other userid id_{ib} from the same author.

For the review subset of id_{ia} (or id_{ib}), we randomly select 9 reviews as the *query set* and another 10 reviews as the *sample set* for the userid. The two sets are disjoint. We don't use more queries or sample reviews from each author since in the review domain most authors do not have many reviews (Jindal and Liu, 2008). In the experiments, we will vary the number of test userids, the number of queries, and the number of samples. We use the following format to describe each test data: T<n>_Q<n>S<n>, where T denotes the total number of test userids, Q the query set and S the sample set, and <n> a number. For example, T50_Q9S10 stands for a test data with 50 userids, and for each userid, 9 reviews are selected as queries and 10 reviews are selected as samples. * rep-

resents a wildcard whose value we can vary.

Note that we use this "artificial" data rather than manually labeled data for our experiments because it is very hard to reliably label any gold-standard data manually in this case. The problem is similar to labeling fake reviews. In the fake review detection research, researchers have manually label fake reviews and reviewers (Yoo and Gretzel 2009; Lim et al., 2010; Li et al., 2011; Wang et al., 2011). However, based on the actual fake reviews written using Amazon Mechanical Turk, Ott et al. (2011) have showed that the accuracy of human labeling of fake reviews is very poor. We also believe that our test data is realistic for evaluation as we can image that the two sets of reviews are from two accounts (userids) of the same author (reviewer).

Two types of experiments: For each author with two userids, we conduct two types of tests.

- **Type I**: Identify two userids belong to the same author. The experiment runs iteratively to test every userid. In each iteration, we plant one userid of an author in the test set and use the other userid of the same author as the query userid. That is, in the i^{th} run, the test data consist of the following two components:
  1. Query userid id_{ia} and its query set Q_{ia}
  2. Test userids $\{id_{1a}, \ldots, id_{(i-1)a}, id_{ib}, \ldots, id_{ma}\}$ and their corresponding sample review sets $\{S_{1a}, \ldots, S_{(i-1)a}, S_{ib}, \ldots, S_{ma}\}$.

  Note that the query userid id_{ia} and the test userid id_{ib} are from the same author. Our objective is to use Q_{ia} to find id_{ib} through S_{ib}.

  Evaluation measure: We use precision, recall, and F_1 score to evaluate Type I experiments as we want to identify all matching pairs. The errors are "no pair" and "wrong pair" found.
- **Type II**: Type II experiments test the cases when no pair exists. That is, we do not plant any matching userid for the query userid. Then, the algorithm should not find anything. For the i^{th} run, the test data has these components:
  1. Query userid id_{ia} and its query set Q_{ia}
  2. Test userids $\{id_{1a}, \ldots, id_{(i-1)a}, id_{(i+1)a}, \ldots, id_{ma}\}$ and their sample review sets $\{S_{1a}, \ldots, S_{(i-1)a}, S_{(i+1)a}, \ldots, S_{ma}\}$. id_{ib} is not planted.

  Evaluation measure: Here we cannot use precision and recall because we are not trying to find any pairs. We thus use accuracy as our measure. For each id_i, if no pair is found, it is correct. If a pair is found, it is wrong.

Baseline methods: As mentioned eariler, there are only two works that tried to identify multi-id users. The first is that in (Chen et al., 2004). However, as we discussed in related work, their approach is not applicable to reviews. The other is that in (Novak et al., 2004), which used clustering but assumed that the number of actual authors (or clusters) is known. This is unrealistic in practice. Thus we designed three new baselines:

TSL: This baseline is based on the traditional supervised learning (TSL). We use it to evaluate how the traditional approach performs in the original feature space. In this case, each document in TSL has to be represented as a vector of d-features or traditional n-gram features. For each test userid *id*, we build a SVM classifier based on the *one vs. all* strategy. That is, for training we use *id*'s queries in T*_Q*S10 as the positive documents, and all queries of the other test userids (e.g., 99 userids if the test data has 100 userids) as the negative documents. Note that TSL cannot use the 731 userids for training as in LSS because they do not appear in the test data. In testing, userid *id*'s sample (non-query) documents in T*_Q*S10 are used as positive documents, and the sample documents of all other test userids are used as negative documents.

SimUG: It uses the word unigrams to compare the cosine similarity of queries and samples. Cosine similarity with unigrams is the most widely used document similarity measure.

SimAD: It uses all *d*-features to compare the cosine similarity of queries and samples.

For both *SimUG* and *SimAD*, their cosine similarity values are used in place of SVM scores of LSS or TSL. We then apply the same 4 strategies to decide the final author attribution except voting as cosine similarity cannot classify.

7.2 Results and analysis

1) Effects of positive/total ratio in training set: Since our data is highly skewed and too many negative cases may not be good for classification, we thus performed this experiment to find a good ratio. Table 5 shows the results for Type I experiments. From Table 5, we can see that the results are highly accurate. Even for 100 userids, our method can correctly identify 85% cases. Here we use the data sets T*_Q9S10 and the decision method is ScoreSqSum, which produces the best result. The

results for Type II experiments (Table 6) are also accurate. In most cases, the values of accuracy are higher than 90%. For all our experiments below, we use the model/classifier trained with 0.4 ratio.

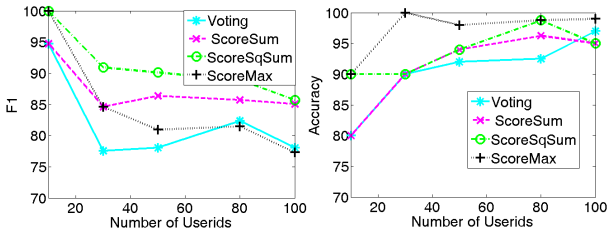
Table 5. Positive(p)/total(t) ratio in training (Type I)

	p/t	10	30	50	80	100
F1	0.3	**100.00**	84.62	86.36	**88.89**	83.72
	0.4	**100.00**	**91.91**	90.11	**88.89**	85.71
	0.5	**100.00**	90.91	**91.30**	**88.89**	87.01
	0.6	94.74	82.35	87.64	85.71	86.36
	0.7	94.74	84.62	86.36	86.53	**87.64**

Table 6. Positive(p)/total(t) ratio in training (Type II)

	p/t	10	30	50	80	100
Accuracy	0.3	90.00	90.00	92.00	97.50	94.00
	0.4	**90.00**	**90.00**	**94.00**	**98.75**	**95.00**
	0.5	80.00	86.67	94.00	97.75	95.00
	0.6	80.00	86.67	90.00	93.75	92.00
	0.7	80.00	86.67	90.00	95.00	92.00

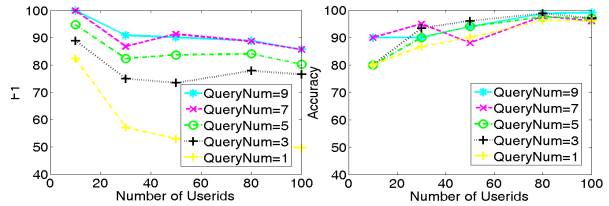
(2) Effects of different decision methods: We show the results of the four proposed decision methods: Voting, ScoreSum, ScoreSqSum, and ScoreMax, using our basic data of T*_Q9S10 with varied number of test userids. Figure 5(a) shows that ScoreSqSum is the best for Type I experiments. Figure 5(b) shows ScoreMax is the best for Type II, but ScoreSqSum also does very well. Below, ScoreSqSum is used as our default method because Type I is more important than Type II.



(a) Type I (b) Type II

Figure 5: Effect of different decision methods

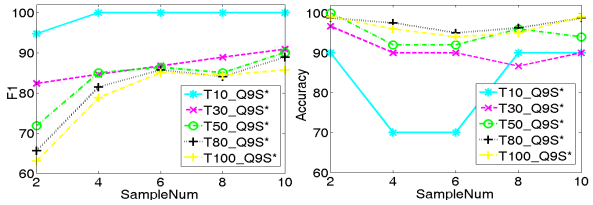
(3) Effects of number of queries per userid: Figure 6 shows the results of different numbers of queries. We see that more queries give better results, which is easy to understand because more queries give more information. We use 9 queries per userid in all other experiments.



(a) Type I (b) Type II

Figure 6: Effect of different numbers of queries

(4) Effects of number of samples per userid: We tried 2, 4, 6, 8, 10 samples per userid. Although there are some fluctuations for Type II (Fig.7(b)), we can see an upward trend for Type I in Fig. 7(a). This indicates that more sample documents give better results in general. The main reason again is that more samples from a userid give more identifying information about the userid. We use 10 test documents (samples) per userid in all experiments.



(a) Type I (b) Type II

Figure 7: Effect of different number of samples

(5) Impact of individual s-feature sets: Here we show the effectiveness of individual s-feature sets. From Table 7, we see that Sim7Retrieval s-features are extremely important for Type I test. Removing Sim7Retrieval causes about 10% to 20% F_1 score drop on different datasets. SimCTfidf s-features are also useful. The impacts of other s-features are small. The same applies to Type II test (Table 8). On average, using all features is the best. Hence we use all features in all other experiments above.

Table 7. Using different s-features (Type I)

T*_Q9S10	F1 10	F1 30	F1 50	F1 80	F1 100
All features	**100.00**	**90.91**	**90.11**	**88.89**	**85.71**
No Sim4Len	100.00	88.89	86.36	87.32	85.06
No SimCRichness	100.00	88.89	91.30	88.89	85.71
No SimCTfidf	100.00	80.00	86.36	86.53	83.72
No Sim7Retrieval	82.35	72.34	75.80	78.79	77.30
No Sim3Sent	94.74	84.62	86.36	88.11	87.64

Table 8. Using different s-features (Type II)

T*_Q9S10	Acc. 10	Acc 30	Acc. 50	Acc 80	Acc. 100
All features	**90.00**	**90.00**	**94.00**	**98.75**	**99.00**
No Sim4Len	90.00	93.33	96.00	96.25	96.00
No SimCRichness	90.00	90.00	94.00	96.25	96.00
No SimCTfidf	90.00	86.67	94.00	93.75	97.00
No Sim7Retrieval	80.00	90.00	94.00	94.00	96.00
No Sim3Sent	90.00	93.33	92.00	98.75	93.00

(6) Comparing with the three baselines: Similar to our method, the training data for TSL is highly skewed as it uses a *one-vs.-all* strategy. Hence we also investigate the effect of p/t ratio in training for TSL. Results show that 0.4 ratio is the best setting.

Thus this setting is adopted for TSL in the following experiments. Note that we cannot conduct p/t ratio experiments for SimAD and SimUG as they are unsupervised methods. We use ScoreMax for TSL, ScoreSqSum for SimUG and SimAD, respectively, since they perform the best for their corresponding approaches. Tables 9 and 10 show the results of our LSS method and the baseline methods for Type I and II tests respectively. For TSL, we use all *d*-features. Unigram features gave TSL much worse results and are thus not included here.

Table 9: Comparison with baselines (Type I)

		10	30	50	80	100
LSS	**Pre**	100.00	100.00	100.00	100.00	98.68
	Rec	100.00	83.33	82.00	80.00	75.76
	F1	**100.00**	**90.91**	**90.11**	**88.89**	**85.71**
TSL	**Pre**	50.00	50.00	33.33	0.00	0.00
	Rec	11.11	3.45	2.08	0.00	0.00
	F1	18.18	6.45	3.92	0.00	0.00
SimUG	**Pre**	100.00	100.00	100.00	100.00	100.00
	Rec	70.00	46.67	48.00	48.75	43.00
	F1	82.35	63.64	64.86	65.55	60.14
SimAD	**Pre**	100.00	75.00	100.00	33.33	0.00
	Rec	20.00	10.35	2.00	1.28	0.00
	F1	33.33	18.18	3.92	2.47	0.00

Table 10: Comparison with baselines (Type II)

Accuracy	**10**	**30**	**50**	**80**	**100**
LSS	90.00	90.00	94.00	98.75	95.00
TSL	90.00	96.67	98.00	98.75	99.00
SimUG	96.00	93.33	96.00	96.25	97.00
SimAD	90.00	96.67	98.00	98.75	99.00

From Tables 9 and 10, we can make the following observations.

- For Type I, F_1 scores of LSS are markedly better than those of the three baselines. The results of SimUG also drop more quickly than LSS with the increased number of userids. SimAD's results are extremely poor. These show that LSS is much more superior to the unsupervised methods. TSL performed the worst, indicating that traditional supervised learning is inappropriate for this task. There are two main reasons: First, for *one vs. all* learning, the negative training data actually contain positive documents which are written by the same author using another userid as the positive data, which confuses the classifier. Second, TSL is unable to build an accurate classifier using the small number of queries (which are training data). In contrast, our LSS method can exploit a large number of other authors who do not have to appear in testing and thus achieves the huge improvements.
- For Type II, LSS also performs very well. The baselines perform well too and even better, which is not surprising because they have difficulty in finding matching pairs for Type I. Since Type II datasets have no author with multiple userids, naturally the baselines will do well for Type II. But that is useless because when there are authors with multiple usersids (Type I), they are unable to find them well.

In summary, we can conclude that for Type I tests (there are authors with multiple userids), LSS is dramatically better than all baseline methods. For Type II tests (there is no author with multiple userids), it also performs very well.

8 Conclusion

This paper proposed a novel method to identify userids that may be from the same author. The core of the method is a supervised learning method which learns in a similarity space rather than the document space. This learning method is able to better determine whether a document may be written by a known author, although no document from the author has been used in training (as long as we have some documents from the author to serve as queries). To the best of our knowledge, there is no existing method based on linguistic analysis for solving the problem. Our experimental results based on a large number of reviewers and their reviews show that the proposed algorithm is highly accurate. It outperforms three baselines markedly.

Acknowledgements

We are grateful to the anonymous reviewers for their thoughtful comments. Tieyun Qian was supported in part by the NSFC Projects (61272275, 61272110, 61202036), and the 111 Project (B07037). Bing Liu was supported in part by a grant from National Science Foundation (NSF) under no. IIS-1111092.

References

Shlomo Argamon and Shlomo Levitan. 2004. Measuring the usefulness of function words for authorship attribution. *Literary and Linguistic Computing* 1-3.

Shlomo Argamon, Casey Whitelaw, Paul Chase, Sobhan Raj Hota, Navendu Garg, and Shlomo Levitan. 2007. Stylistic text classification using functional lexical features: Research articles. *J. Am. Soc. Inf. Sci. Technol*. 58:802-822.

John F. Burrows. 1992. Not unless you ask nicely: The interpretative nexus between analysis and information. *Literary and Linguistic Computing* 7:91-109.

Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking svm to document retrieval. *Proc. of SIGIR*, Pages 186-193.

Hung-Ching Chen, Mark K. Goldberg, Malik Magdon-Ismail. 2004. Identifying multi-ID users in open forums. *Intelligence and Security Informatics*, Pages 176-186.

Joachim Diederich, Jörg Kindermann, Edda Leopold, and Gerhard Paass, 2000. Authorship attribution with support vector machines. *Applied Intelligence* 19:109-123.

Hugo Jair Escalante, Thamar Solorio, and Manuel Montes-y-Gómez. 2011. Local histograms of character n-grams for authorship attribution. *Proc. of ACL-HLT*, Volume I: 288-298.

Song Feng, Longfei Xing, Anupam Gogar, and Yejin Choi. 2012. Distributional Footprints of Deceptive Product Reviews. *Proc. of ICWSM*.

Michael Gamon. 2004. Linguistic correlates of style: authorship classification with deep linguistic analysis features. *Proc. of Coling.*

Neil Graham, Graeme Hirst, and Bhaskara Marthi. 2005. Segmenting documents by stylistic character. *Natural Language Engineering*, 11:397-415.

Jack Grieve. 2007. Quantitative authorship attribution: An evaluation of techniques. *Literary and Linguistic Computing* 22:251-270.

Hans van Halteren, Fiona Tweedie, and Harald Baayen. 1996. Outside the cave of shadows: using syntactic annotation to enhance authorship attribution. *Literary and Linguistic Computing* 11:121-132.

Steffen Hedegaard and Jakob Grue Simonsen. 2011. Lost in translation: authorship attribution using frame semantics. *Proc. of ACL-HLT, short papers* - Volume 2, 65-70.

Graeme Hirst and Ol'ga Feiguina. 2007. Bigrams of syntactic labels for authorship discrimination of short texts. *Literary and Linguistic Computing* 22:405-417.

David I. Holmes and R. S. Forsyth. 1995. The Federalist Revisited: New Directions in Authorship Attribution, *Literary and Linguistic Computing*, 10(2): 111-127.

David L. Hoover. 2001. Statistical stylistics and authorship attribution: an empirical investigation. *Literary and Linguistic Computing* 16:421-424.

Nitin Jindal and Bing Liu. 2008. Opinion Spam and Analysis. *Proc. of WSDM*, California, USA.

Thorsten Joachims. 2006. Training linear svms in linear time. *Proc. of KDD*.

Sangkyum Kim, Hyungsul Kim, Tim Weninger, Jiawei Han, and Hyun Duk Kim. 2011. Authorship classification: a discriminative syntactic tree mining approach. *Proc. of SIGIR*, Pages 455-464.

Dan Klein, and Christopher D. Manning. 2003. Accurate unlexicalized parsing. *Proc. of ACL*, 423-430.

Moshe Koppel and Jonathan Schler. 2004. Authorship verification as a one-class classification problem. *Proc. of ICML*.

Moshe Koppel, Jonathan Schler, Shlomo Argamon. 2011. Authorship attribution in the wild. *Lang Resources & Evaluation*, 45:83-94

Fangtao Li, Minlie Huang, Yi Yang and Xiaoyan Zhu. 2011. Learning to identify review Spam. *Proc. of IJCAI*.

Hang Li. 2011. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool publishers.

Jiexun Li, Rong Zheng, and Hsinchun Chen. 2006. From fingerprint to writeprint. *Communications of the ACM*, 49:76-82.

Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, Hady W. Lauw. 2010. Detecting product review spammers using rating behaviors. *Proc. of CIKM*, 2010.

Bing Liu. 2012. *Sentiment Analysis and Opinion Mining*, Morgan & Claypool publishers.

Tieyan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer.

Kim Luyckx, Walter Daelemans. 2008. Authorship Attribution and Verification with Many Authors and Limited Data. *Proc. of Coling*, pages 513-520.

David Madigan, Alexander Genkin, David D. Lewis, Shlomo Argamon, Dmitriy Fradkin, and Li Ye. 2005. Author Identification on the Large Scale. *Proc. of CSNA.*

Donald Metzler, Yaniv Bernstein, W. Bruce Croft, Alistair Moffat, and Justin Zobel. 2005. Similarity measures for tracking information flow. *Proc. of CIKM*. Pages 517-524.

Frederick Mosteller, David Lee Wallace. 1964. *Inference and disputed authorship: The Federalist*. Addison-Wesley.

Arjun Mukherjee, Bing Liu, and Natalie Glance. 2012. Spotting Fake Reviewer Groups in Consumer Reviews. *Proc. of WWW*, Pages 191-200.

Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, et al. 2012. On the feasibility of internet-scale author identification. *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. Pages 300-314

Jasmine Novak, Prabhakar Raghavan, Andrew Tomkins. 2004. Anti-aliasing on the web. *Proc. of WWW*, Pages 30-39

Myle Ott, Yejin Choi, Claire Cardie, Jeffrey T. Hancock. 2011. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. *Proc. of ACL*.

Myle Ott, Claire Cardie, Jeffrey T. Hancock. 2012. Estimating the prevalence of deception in online review communities. *Proc. of WWW.*

Fuchun Peng, Dale Schuurmans, Shaojun Wang, and Vlado Keselj. 2003. Language independent authorship attribution using character level language models. *Proc. of EACL,* Pages 267-274.

Conrad Sanderson and Simon Guenter. 2006. Short text authorship attribution via sequence kernels, markov chains and author unmasking: an investigation. *Proc. of EMNLP*, Pages 482-491.

Yanir Seroussi, Fabian Bohnert, Ingrid Zukerman. 2012. Authorship Attribution with Author-aware Topic Models. *Proc. of ACL*, 2:264-269.

Thamar Solorio, Sangita Pillay, Sindhu Raghavan, Manuel Montes y G´omez. 2011. Modality Specific Meta Features for Authorship Attribution in Web Forum Posts. *Proc. of IJCNLP*, Pages 156-164.

Efstathios Stamatatos. 2009. A Survey of Modern Authorship Attribution Methods. *Journal of the American Society for Information Science and Technology*, 60(3):538-556, Wiley.

Efstathios Stamatatos, George Kokkinakis, and Nikos Fakotakis. 2000. Automatic text categorization in terms of genre and author. *Comput. Linguist.* 26:471-495.

Özlem Uzuner and Boris Katz. 2005. A comparative study of language models for book and author recognition. *Proc. of IJCNLP*, Pages 969-980.

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. Wiley-Interscience, NY.

O. de Vel, A. Anderson, M. Corney and G. Mohay. 2001. Mining Email Content for Author Identification Forensics. *Sigmod Record*, 30:55-64.

Kyung-Hyan Yoo and Ulrike Gretzel. 2009. Comparison of Deceptive and Truthful Travel Reviews. *Information and Communication Technologies in Tourism*, Pages 37-47.

Georgy Udnv Yule. 1944. *The statistical study of literary vocabulary*. Cambridge University Press.

Guan Wang, Sihong Xie, Bing Liu, Philip S. Yu. 2011. Review Graph based Online Store Review Spammer Detection. *Proc. of ICDM*.

Ying Zhao and Justin Zobel. 2005. Effective and scalable authorship attribution using function words. *Proceeding of Information Retrival Technology*, Pages 174-189.

Rong Zheng, Jiexun Li, Hsinchun Chen, and Zan Huang. 2006. A framework for authorship identification of online messages: Writing style features and classification techniques. *Journal of the American Society of Information Science and Technology* 57:378-393.