SocialSearch: Enhancing Entity Search with Social Network Matching*

Gae-won You, Seung-won Hwang
Pohang University of Science and Technology
Pohang, Republic of Korea
{gwyou,swhwang}@postech.edu

Zaiqing Nie, Ji-Rong Wen
Microsoft Research Asia
Beijing, P. R. China
{znie,jrwen}@microsoft.com

ABSTRACT

This paper introduces the problem of matching people names to their corresponding social network identities such as their Twitter accounts. Existing tools for this purpose build upon naive textual matching and inevitably suffer low precision, due to false positives (*e.g.*, fake impersonator accounts) and false negatives (*e.g.*, accounts using nicknames). To overcome these limitations, we leverage "relational" evidences extracted from the Web corpus. In particular, as such an example, we adopt Web document co-occurrences, which can be interpreted as an "implicit" counterpart of Twitter follower relationships. Using both textual and relational features, we learn a ranking function aggregating these features for the accurate ordering of candidate matches. Another key contribution of this paper is to formulate confidence scoring as a separate problem from relevance ranking. A baseline approach is to use the relevance of the top match itself as the confidence score. In contrast, we train a separate classifier, using not only the top relevance score but also various statistical features extracted from the relevance scores of all candidates, and empirically validate to outperform the baseline approach. We evaluate our proposed system using real-life internet-scale entity-relationship and social network graphs.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, experimentation, performance

Keywords

Entity search, graph matching, social network

1. INTRODUCTION

As the number of people with "Web presence" increases, more search engines provide object-level search results [11, 13], *e.g.*, by showing related news, images, products, and people frequently co-occurred in the news. To enable instant social interactions with such people entities, this paper develops SocialSearch, of mapping the given person name[1] with its corresponding entity in the commercial social networks, such as Twitter. For instance, such system will link more than 100 million people that can be crawled from the Web, to their corresponding Twitter accounts among 12 million users (number as of year 2009).

While there are existing services mapping Web entity with its social entity, such as a browser plug-in WebMynd [2], they build on an assumption that full names representing Web entities are used to describe the corresponding social entities and focus on textual matching.

However, we observe that such assumption holds only for a limited fraction of real-life matchings, as Figure 1 illustrates– For 627 randomly selected query names with notable Web presence (as explained in Section 4.1.2), full name matching is correct for only 16.5%, and fails for the remaining 83.5%, which falls into the following representative cases observed from this pie chart.

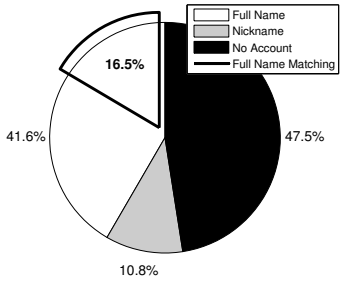

Figure 1: Failure of textual matching

Impersonator. As revealing full names attracts traffic through name search, many celebrities present their full names. However, for the same reason, many fake accounts impersonating celebrities also exist, which has resulted in many lawsuits for Twitter recently. To illustrate, in the pie chart, among 41.6% of accounts using full names, which naive textual matching identifies as correct matches, only 16.5% are authentic and the rest are fake or confused Twitter identities. To partially address this problem, Twitter manually distinguishes an authentic account and marked as "verified" by displaying a check mark next to the Twitter ID.

Nicknaming. For many people entities belonging to "long tail" [15], *i.e.*, not necessarily celebrities but with notable Web presence, attracting traffic is not as much of a concern. They thus often use nicknames, which cannot be reached by the naive name matching. Such account covers 10.8% of the real matches, shown as a gray pie in Figure 1.

*This research was supported by Microsoft Research and the MKE (The Ministry of Knowledge Economy), Korea, under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2010-C1810-1002-0003).

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

[1]This paper assumes that Web entities have no name ambiguity. The name disambiguation is out of our scope.

No account. While the above two cases cover the cases where there is a matching Twitter account but hard to find (confused by fake identities and the use of nicknaming), there is a significant fraction of cases with no matching Twitter account. As a black pie in Figure 1 shows, 47.5% of the random names do not have matching accounts in Twitter.

This paper aims at overcoming the limitations discussed above by exploiting "relational" evidences collected from the Web. More specifically, for given a person name as a query, we can extract related people to the query name using co-occurrences or link structure. We interpret such a relationship as an "implicit" counterpart to Twitter follower relationships and propose to leverage these relational features to complement textual features.

As such an example, we leverage an entity-relationship graph built upon co-occurrences in the Web corpus, obtained from object-level search engine Microsoft EntityCube [1, 13]. For instance, when a query is 'Bill Gates', it retrieves and visualizes the related people names such as 'Steve Jobs' frequently co-occurred with 'Bill Gates' in Web documents.

By leveraging this graph, our proposed problem of matching the person name to its Twitter account becomes to match a node from an entity-relationship graph to a Twitter graph. While textually comparing the person name and the Twitter name corresponds to using only node similarity, our proposed solution enables "holistic" matching of using both textual and relational evidences, which complements textual matching in the following two significant ways:

First, relational features, such as "A frequently co-occurs with B in Web documents" (extracted from an entity-relationship graph) and "A follows B' in Twitter" (extracted from a Twitter graph), work as an additional evidence to match B with its nickname B', even when there is not much textual similarity. Relational evidences can contribute to *not* matching two nodes that are textually identical to avoid matching impersonators. *Second,* unlike Twitter graph that is very manipulation-prone, as users can choose arbitrary user names and follow arbitrary accounts, the entity-relationship graph is more robust, representing the relationships extracted from the implicit feedbacks of Web document creators (or, leveraging the wisdom of crowds).

In particular, from these two graphs, we extract both textual and relational features and train two classifiers for (1) ranking candidates by relevance, for the given query keyword, and (2) classifying whether there exists a matching Twitter account among the candidates identified as follows:

- **Relevance Ranking:** We extract textual and relational features and use an SVM to learn a ranking function to accurately order the matching candidates.
- **Confidence Scoring:** We then quantify the confidence level on whether the candidates contain a match or not to help users to easily identify the case with no account.

To summarize, we believe this paper has the following key contributions:

- We formulate SocialSearch as a novel ranking problem and propose a learning model to rank the social entities. We leverage an entity-relationship graph extracted from Web document co-occurrences (1) to leverage relational features for matching, and (2) to enable more manipulation-resistant mapping. This unique formulation enables *holistic* and *robust* matching using both textual and relational features.
- We formulate confidence scoring as a separate problem from relevance ranking to help users to easily identify the case with no account. A baseline approach is to treat the relevance score of the top match itself as the confidence score. In contrast, we train a separate classifier using not only the top relevance score but also various statistical features extracted from the relevance scores of all candidates, and empirically validate to outperform the baseline approach.
- We validate the effectiveness of our approach using internet-scale real-life EntityCube and social network datasets.

2. RELATED WORK

To the best of our knowledge, no previous work has extracted implicit network structures from the Web corpus and matched with social networks. We survey related efforts on matching names with social network identities, categorized into those using (1) textual and (2) relational features.

Textual features. There have been several browser plug-ins such as WebMynd augmenting Web search results with social network search results. However, such approaches are solely based on textual similarity and thus identify only a limited fraction of matches as discussed in Section 1. Similarly, name or ID search typically provided from social network services, *e.g.*, Twitter name search, also builds on textual similarity and suffers from the same problems.

Relational features. Leveraging relations between nodes has been explored in a different problem context in regards to anonymity aspects. Several studies reported that relational features, such as friendship relationships in social network and citations in academic literature, reveal the identities (or real-life names), even when nodes are "anonymized" [6, 12]. While these works motivate the use of relational features, they cannot be adopted for our target problem because the precision is too low, 30.8% or 40-45%, respectively, by aiming at a different goal of validating de-anonymization (non-zero precision) with limited information. In contrast, our goal is to achieve the maximal precision leveraging all available information.

Alternatively, relational features have been adopted for disambiguating names and recommendation from (1) a Web graph [3], (2) bibliographic citations [14], and (3) a social network [4, 5]. First, link structure of the matching documents is used to select the right identity with high relational similarity. Second, coauthor information is used to identify and correct name variants. Third, the social relationships are exploited to recommend new friends (using friends-of-friends relationships), communities, or Web pages favored by social peers.

3. OUR FRAMEWORK

We design our solution as three main sequential procedures– (1) selecting candidate accounts, (2) ranking candidate accounts, and (3) computing confidence.

3.1 Candidate Selection

We select candidates relevant to the query name. A naive approach is to select all accounts that exactly match the query name. However, as we already discussed in Section 1, this approach can find only a limited fraction ($\sim$16.5%) of the real matches.

We thus relax candidate selection criteria to include partial matches. Our observation with real-life matches suggests that our proposed relaxation scheme covers 97.26% of real matches. More formally, given a query name $q = \{q^1, q^2, \cdots\}$, *i.e.*, an ordered set of words delimited by a space character, 97.26% of the names $c.n$ of the candidate account c fall into one of the following three types:

- **Exact containment (E-type):** The account name in this category is exactly matched with the query name, *i.e.*, $q = c.n$.

- **All containment (A-type):** The account name in this category contains all the words of the query name, *i.e.*, $\forall q^i \in q, q^i \subset c.n$[2]. In other words, this category permits the reordering of words or concatenation/augmentation. For example, given a query q = 'John Mccain', A-type matches include all E-type matches and also the matches like 'Mccain, John' (reordering words), 'JohnMccain' (concatenation), or 'JohnMccain2010' (concatenation and augmentation).
- **Some containment (S-type):** The account name in this category contains some words of the query name, *i.e.*, $\exists q^i \in q, q^i \subset c.n$. For example, for q = 'John Mccain', S-type matches include all E-type and A-type matches, and also the matches like 'john', 'jmccain', or 'mccain2010'.

Table 1 shows the coverage and the average number of candidates for the candidate selection methods exploiting three types. Specifically, for 329 random queries with matching Twitter accounts (as explained in Section 4.1.2), the matching account falls into E-, A-, and S-types with 79.33, 91.49, and 97.26%, respectively.

Table 1: Coverage of candidate selection

Type	E-type	A-type	S-type
Coverage	0.7933	0.9149	**0.9726**
Avg. # candidates	18.8	21.6	205.0

The increase in coverage, as E-, A-, and S-types are used for candidate selection, can be naturally explained by the containment in the definitions of three types– All E-type names naturally satisfy A-type and S-type containments, and all A-type names satisfy S-type, *i.e.*, $C_E \subset C_A \subset C_S$, where C_E, C_A, and C_S are the sets of candidates obtained from the three selection methods, respectively.

We adopt S-type for initial candidate selection to maximize the coverage, which we later narrow down using relevance ranking. However, the trade-off for maximizing coverage is a large candidate size, as Table 1 shows the average number of candidates which increases dramatically from 18.8 to 205.0, as the selection scheme relaxes from E-type to S-type. We partially address this problem with an additional filtering condition. To illustrate, for q='Bill Gates', S-type candidates include 'Bill Clinton', which is unlikely to be a candidate. To eliminate candidates that are highly likely to belong to another name, we drop the names that exactly match other names occurred in the Web page.

3.2 Ranking Candidates

Once the candidates are identified, we rank them with textual and relational features (Section 3.2.1) using the aggregated ranking function (Section 3.2.2).

3.2.1 Feature Extraction

This section reports the features that have a positive effect to the performance. Those features are computed for the candidates, with respect to the EntityCube entity matching a query keyword, designed to quantify (1) relevance and (2) popularity.

First, regarding *relevance*, for each Twitter account candidate, we compute the relevance in terms of (1) name revealed in the Twitter page (2) Twitter ID and (3) related nodes. Second, regarding *popularity*, we count the number of followers to measure how likely the corresponding Twitter account to match the Web identity with notable enough Web presence to qualify as an EntityCube entity.

Name relevance. The name type, among E-, A-, and S-types, suggests how likely the given Twitter account candidate matches the person name used in query, as Table 3 enumerates name type features as binary numbers.

ID relevance. Similarly, as many Twitter users use part of their names as Twitter ID, the ID type, classified into E-, A-, and S-types following the same convention of classifying names, also evidences the likelihood of the given candidate to be a match. However, unlike Twitter names, pre-filtered to belong to either E-, A-, or S-type, ID may not fall into any one of these categories. In such case, such ID is categorized into N-type. These four types are represented by four binary features shown in Table 3.

Relation relevance. We now discuss relational relevance. Given a query name 'Bill Gates', a candidate account 'BillGates' is more likely to be a match, if its related nodes (based on Twitter follower relationships) match the names of the adjacent nodes in the EntityCube entity-relationship graph (*i.e.*, people co-occurring frequently with 'Bill Gates' in Web documents). For this purpose, we first observe the discrepancy between co-occurrences and follower relationships– While the former is bidirectional, the latter is unidirectional, as Figure 2 illustrates. In this figure, given a query

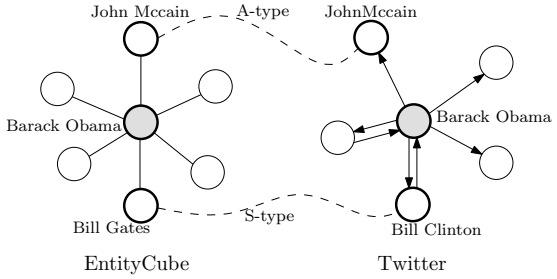

Figure 2: Features on relation

'Barack Obama', 'Bill Clinton' with a bidirectional follower relationship may have different strength, compared to 'John Mccain' with a unidirectional follower relationship, which we thus separate into two different features. For each type, we count the number of common neighbors as features between neighbors in EntityCube graph and those in Twitter graph, as Table 3 enumerates. For matching and counting common neighbors, we use A-type and S-type textual matches respectively, which explains the A-type and S-type features in Table 3 for each type. In this neighbor matching process, if one-to-many matches exist, we select one (1) satisfying stricter matching criteria (E-type matching for counting A-type common neighbors, and A-type for counting S-type matches) and (2) if no such match exists, we arbitrarily select one. Lastly, we take the logarithm of the count, as counts tend to follow power law, *i.e.*, $\log(|R|+1)$, where $|R|$ is the number of relations.

Table 2: Average number of common neighbors for unidirectional (U) and bidirectional (B) edges

	A-type (U)	S-type (U)	A-type (B)	S-type (B)
Match	**2.0217**	**9.3746**	**0.4386**	**1.7554**
Non-match	0.4897	3.4200	0.0672	0.3236

To motivate the use of common neighbors as features to distinguish matches from non-matches, Table 2 illustrates the average number of common neighbors for the matches and non-matches, counted using E- and A-type match (respectively) for uni- and bidirectional neighbors (respectively). We can observe that the common neighbors of the matches significantly outnumber those of non-matches.

Popularity. In addition to the above mentioned three features related to relevance, we also consider the popularity of the corresponding Twitter account, quantified by the number of follower inlinks. As EntityCube entities correspond to people with notable Web presence, this metric evidences how likely the corresponding

[2]We use the notation $\subset$ to represent a substring matching. For example, given two strings s_1 and s_2, $s_1 \subset s_2$ means that s_1 is a substring of s_2.

Twitter account to have notable Web presence. We similarly take the logarithm of the inlink count, *i.e.*, $\log(indegree(c)+1)$, as the indegree also follows power law [7].

3.2.2 Learning to Rank: Combining Features

This section suggests how to combine the multiple features observed in the previous section and learn a ranking function $f_{\overrightarrow{w}}$ to rank candidates $\mathcal{C}_i$. In particular, we adopt Ranking SVM algorithm [9, 10]. More formally, let a training set $S = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n\}$ with respect to a set of queries $Q = \{q_1, q_2, \cdots, q_n\}$, where $\mathcal{C}_i = \{c_{i,1}, c_{i,2}, \cdots c_{i,|\mathcal{C}_i|}\}$ is a set of candidates for each query q_i. Without loss of generality, we consider $c_{i,1}$ as the correct answer. This is an optimization problem as follows:

$$minimize: \qquad V(\overrightarrow{w}, \overrightarrow{\xi}) = \frac{1}{2}\overrightarrow{w} \cdot \overrightarrow{w} + C\sum \xi_{i,j} \qquad (1)$$

$$subject\ to:$$

$$\forall i, \forall c_{i,j} \in \mathcal{C}_i - \{c_{i,1}\}:$$

$$\overrightarrow{w}\Phi(q_i, c_{i,1}) > \overrightarrow{w}\Phi(q_i, c_{i,j}) + 1 - \xi_{i,j} \qquad (2)$$

$$\forall i \forall j: \xi_{i,j} \geq 0 \qquad (3)$$

where $\overrightarrow{w}$ is a weight vector, $\Phi(q_i, c_{i,j})$ is a mapping onto features between a query q_i and a candidate account $c_{i,j}$ explained in the previous section, and $\sum \xi_{i,j}$ means total training error. We can thus control a permitted limit of training error by adjusting the parameter C. Note that Eq. (2) means the pairwise comparison between the correct answer $c_{i,1}$ and other candidates $c_{i,j}$ such that $j \neq 1$.

One challenge in training this classifier is a large bias in terms of candidate size. We observe that the number of candidates for queries, which widely varies by orders of magnitude, *e.g.*, from 6×10^3 to 1. Due to this bias, using all candidates for training can lead to overfitting for queries with a large set of candidates. To address this problem, we use only top-20 candidates with the maximum in-links for training.

Table 3: Feature table for ranking candidates (R) and confidence classification (C)

No.	Features		Type	R	C
1	Name type	E-type	Binary	✓	
2		A-type	Binary	✓	✓
3		S-type	Binary	✓	✓
4	ID type	E-type	Binary	✓	
5		A-type	Binary	✓	
6		S-type	Binary	✓	
7		N-type	Binary	✓	
8	Relation (unidirectional)	A-type	Real	✓	
9		S-type	Real	✓	
10	Relation (bidirectional)	A-type	Real	✓	
11		S-type	Real	✓	
12	Popularity		Real	✓	
13	Top-1 ranking value		Real		✓
14	Average ranking value		Real		✓
15	Standard deviation		Real		✓
16	Difference		Real		✓
17	Eccentricity		Real		✓

3.3 Confidence Scoring

This section discusses how to quantify the confidence of the top candidate found, to assist user- or application-level decision on whether to return the top candidate or to return 'no account' as result.

3.3.1 Baseline: Using Relevance as Confidence

A straightforward approach is to simply use the relevance score of the top match itself as a confidence score. However, we observe that, to make a more robust decision, we need to refer the relevance of, not just the top match, but its relative standing in the overall relevance score distribution, as we will empirically validate in Section 4. We thus propose to separate confidence scoring, to consider the overall relevance score distribution as features.

3.3.2 Proposed Approach: Confidence Scoring

This section proposes to train a separate classifier using not just (1) the relevance score of top-1 candidate but also (2) distributional features for relevance scores of other candidates as follows:

Name type. The name type of the top-1 candidate, using the same categorization into E-, A-, and S-types used for relevance scoring.

Top-1 ranking value. The relevance score of the top-1 candidate, *i.e.*, $\max_{c_{i,j} \in C_i} f_{\overrightarrow{w}}(c_{i,j})$.

Average ranking value. The average relevance score of all candidates.

Standard deviation σ. The standard deviation of the relevance scores of all candidates.

Difference. The difference between the relevance scores of the top-1 and the top-2 candidates, *i.e.*, $\max_1 - \max_2$, where $\max_1$ and $\max_2$ are the relevance scores of the top-1 and the top-2 candidate, respectively.

Eccentricity. The eccentricity of the top-1 candidate. This feature measures how outstanding the top-1 score is, compared to other scores, which we normalize as $(\max_1 - \max_2)/\sigma$.

Table 3 summarizes the details of the features explained above.

We train an SVM classifier [8] using the above six features. Such classifier outputs negative values for empty accounts and positive values for non-empty account using the maximum margin hyperplane as the boundary value 0. This classifier can thus be used both for (a) classification and (b) confidence scoring– Using the sign of the output value, we can distinguish non-empty and empty accounts, and when positive, the magnitude can be used as confidence scoring.

4. EXPERIMENTS

This section reports our experimental results to evaluate our proposed approach. First, Section 4.1 reports our experimental setting. Second, Section 4.2 validates the effectiveness of our approach over a real-life dataset. Our experiments were carried out on a machine with Intel Xeon 2.33 GHz CPU and 4GB RAM running Windows. All implementations were written in C#.

4.1 Experimental Settings

We first describe how we collect Twitter and EntityCube datasets in Sections 4.1.1. We then report how to build test datasets for evaluating our framework in Section 4.1.2.

4.1.1 Twitter and EntityCube

We collected 10 million Twitter accounts from 20th May to 15th September in 2009, from which, we crawled ID, name, following list, and verified account tag from each front page of the Twitter accounts. Among these accounts, 692 accounts were tagged by Twitter as "verified", which suggests that twitter.com manually verified the accounts match the person name used. Based on the information collected, we build a "directed unweighted" graph $G_t = (V_t, E_t)$, where each directed edge represents that a user explicitly refers to another user.

Meanwhile, we collected a web co-occurrence graph (an undirected graph) from EntityCube with about 100 million people with notable Web presence. More formally, based on this information, we build an "undirected weighted" graph $G_e = (V_e, E_e)$. The weight of each edge represents the degree of co-occurrence between two entities (how frequently two entity names occur in the

same web pages).

4.1.2 Building Test Data

We now need "ground-truth" matches between Entity and Twitter nodes to test our proposed approaches. We build one test set $\mathcal{V}$ using verified pairs provided from twitter.com. However, this set, by design, cannot cover the case where the EntityCube identity does not have a matching Twitter account ('empty account'), which naturally motivates to build another test set $\mathcal{E}$ including the empty account cases based on $\mathcal{V}$. More specifically,

Set $\mathcal{V}$: We crawled 692 verified accounts from Twitter and manually found the matching real people names in EntityCube. We denote this set of pairs (v_e, v_t) as $\mathcal{V}$, where $v_e \in V_e$ is a node in EntityCube G_e, and $v_t \in V_t$ is a node in Twitter G_t.

Set $\mathcal{E}$: As $\mathcal{V}$ cannot include empty accounts by design, it cannot be used to verify the accuracy of confidence scoring. We thus build another test set $\mathcal{E}$ including empty account cases. More specifically, from the verified pairs in $\mathcal{V}$, we manually selected 20 seed accounts representing a wide range of occupation domains. From each of this "seed pair" (v_e, v_t), we randomly select the neighboring nodes v'_e of v_e, specifically among those one or two hops away from v_e. From 20 seed pairs, we collect the total 627 neighboring nodes. For these 627 EntityCube nodes identified, we asked human assessors to find the matching Twitter accounts using search engines and to mark the cases where no Twitter account can be found. As a result, 52.5% of EntityCube entities were matched to the corresponding Twitter accounts (which we denote as $\mathcal{E}_n$ for non-empty matches) while the remaining 47.5% of entities correspond to empty accounts (which we denote as $\mathcal{E}_e$).

4.2 Experimental Results

This section reports our experimental results using the evaluation datasets explained in previous section. We first validate our ranking method only with non-empty accounts in Section 4.2.1. We then validate the accuracy of confidence scoring with both non-empty and empty accounts in Section 4.2.2.

4.2.1 Relevance Ranking

For evaluating the accuracy of ranking, we use the non-empty accounts $\mathcal{V}$ and $\mathcal{E}_n$.

As the evaluation metrics, we use the following three widely adopted metrics for ranking effectiveness– (1) P@1, ratio of the query results such that top-1 candidate is the correct answer (precision), (2) R@5, ratio of query results such that the correct answer is contained within top-5 candidates (recall), and (3) MRR [16], average of the reciprocal ranks of the query results as this formula:

$$\text{MRR} = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{rank_q} \quad (4)$$

where Q is a set of queries, and $rank_q$ is the rank of the correct answer for $q \in Q$.

We then compare our proposed method with the following three natural baselines:

- E+POP: For the given name, we use E-type textual match to identify candidates, from which we rank Twitter candidates by popularity, *i.e.*, the number of follower in-links.
- A+POP: We use A-type match for candidate selection then rank by popularity.
- S+POP: We use S-type match for candidate selection then rank by popularity

In clear contrast, our proposed method uses S-type match for candidate selection, then ranks the candidates by the SVM model leveraging both textual and relational features, which we thus denote as as S+SVM.

Table 4 shows the coverage of top-k candidates, *i.e.*, R@k, over $\mathcal{E}_n$ for the three baseline approaches.

Table 4: R@k of top-k candidates

k	1	5	10	20	All
E+POP	0.7447	0.7872	0.7933	0.7933	0.7933
A+POP	0.8602	0.9058	0.9149	0.9149	0.9149
S+POP	0.3343	0.6596	0.7477	0.8815	**0.9726**

Recall from our discussion in Section 3.1 that 97.26% of all matching name/Twitter pairs satisfy S-type textual match. More formally, this can be interpreted that R@All, *i.e.*, the ratio of query results such that the correct answer is contained within all candidates, for selecting candidates using S-type is 97.26%. In other words, 0.9726 is the theoretical optimum value R@k that can be achieved by S+POP, as we marked in bold in Table 4. Meanwhile, R@k results of S+POP is very low, until k reaches 20, which explains the drawback of having high recall for S+POP. That is, as the coverage of candidates increases, the chance of including false positives also increases. The recall solely depends on the ranking accuracy, while the baseline ranking is not very accurate.

As a result, A+POP, achieving a moderate balance of recall and accuracy, is the winner for the baseline ranking, which we compare with our proposed approach S+SVM in Table 5 using P@1, R@5, and MRR metrics. Specifically, we use $\mathcal{V}$ and $\mathcal{E}_n$ as the training and test data, interchangeably, for S+SVM.

Table 5: Effectiveness of ranking methods (Training set $\rightarrow$ Test set)

Measure	P@1	R@5	MRR
A+POP	0.8602	0.9058	0.8783
S+SVM	**0.8906**	**0.9574**	**0.9169**

(a) $\mathcal{V} \rightarrow \mathcal{E}_n$

Measure	P@1	R@5	MRR
A+POP	0.9152	0.9390	0.9253
S+SVM	**0.9226**	**0.9628**	**0.9400**

(b) $\mathcal{E}_n \rightarrow \mathcal{V}$

Table 5(a) first reports the results when $\mathcal{V}$ is used for training and $\mathcal{E}_n$ for testing (denoted as $\mathcal{V} \rightarrow \mathcal{E}_n$). Observe that, in all settings, S+SVM outperforms A+POP. We also stress that the accuracy of S+SVM is close to the theoretical optimum– The R@5 result of S+SVM is 95.74%, close to the optimum value, *i.e.*, 97.26%, while that of baseline is 90.58%. Table 5(b) then reports the results when $\mathcal{E}_n$ is used for training and $\mathcal{V}$ for testing (denoted as $\mathcal{E}_n \rightarrow \mathcal{V}$). In this setting, of using Twitter verified accounts $\mathcal{V}$ for testing, which are heavily biased to celebrities, the baseline using popularity for ranking is relatively more effective compared to Table 5(a), when using less-biased $\mathcal{E}_n$ for testing. However, even in this unfavorable setting, our proposed method S+SVM consistently outperforms baseline A+POP. Summing up, from these evaluations, we observe that, even though there is no wide margin to improve to the optimum, our proposed method manages to outperform baselines and performs closely to the theoretical optimum.

4.2.2 Confidence Scoring

This section discusses how we evaluate the accuracy of confidence scoring, by comparing baseline REL (using the relevance score) and our proposed scheme CNF (using separate the confidence score) respectively, using test set $\mathcal{E}$, including both non-empty and empty accounts, *i.e.*, $\mathcal{E} = \mathcal{E}_n \cup \mathcal{E}_e$.

As the ground-truth is given as the binary class of non-empty and

empty results, to evaluate the accuracy of confidence scoring using this test set, we set the confidence score threshold θ for both REL and CNF to make the binary decision. We validate the accuracy of this classification using 5-fold cross validation on $\mathcal{E}$. More specifically, $\mathcal{E}$ is divided into five equal-size disjoint subsets $\mathcal{E}^1, \cdots, \mathcal{E}^5$ such that $\mathcal{E}^i \subset \mathcal{E}$. We then use $\mathcal{E} - \mathcal{E}^i$ as a training set for learning the SVM classifier, and then test the result on $\mathcal{E}^i$.

We now discuss how we set θ for REL and CNF. For baseline approach REL, such threshold θ, for each fold $\mathcal{E}^i$, can be estimated as the value that maximizes F1-measure for the training set $\mathcal{E} - \mathcal{E}^i$. Our proposed approach CNF, as this scheme builds on an SVM classifier, naturally suggests threshold θ, as the boundary value 0 (*i.e.*, conceptually the maximum margin hyperplane). We thus do not need a tuning procedure for CNF.

In contrast, for REL, we observe a challenge in tuning θ. Figure 3 plots the threshold value θ found for REL from each fold $\mathcal{E}^i$. We observe that, the optimal threshold values of five folds are scattered in a wide range, *i.e.*, {2.05, 2.06, 2.18, 2.8}, which suggests that the confidence scoring of the baseline is not very robust, being heavily dependent on testing set.

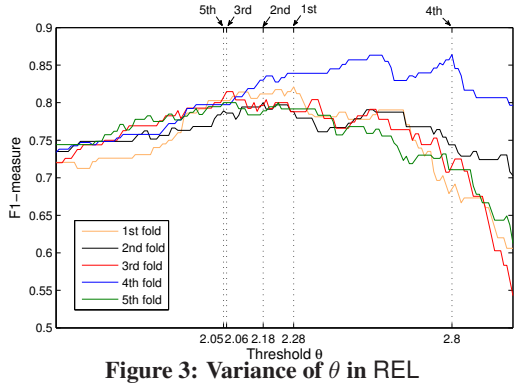

Figure 3: Variance of θ in REL

We measure the classification effectiveness using the following widely-adopted metrics– (1) Precision and Recall on the non-empty accounts $\mathcal{E}_n$, (2) F1-measure, harmonic mean of precision and recall on $\mathcal{E}_n$, and (3) Accuracy on $\mathcal{E}$, ratio of the query results that return the correct non-empty accounts in $\mathcal{E}_n$ or correctly classify the empty accounts in $\mathcal{E}_e$. Note that we report the average scores of 5 folds for the above four measures.

We implement the following four different algorithms. For selection and ranking, all these algorithms adopt the winner approach S+SVM, discussed in the previous section. Their names vary, depending on the confidence scoring algorithm, which we denote using the naming convention of [selection]+[ranking]+[confidence scoring] scheme used:

- S+SVM+NUL using no confidence scoring,
- S+SVM+REL using REL for confidence scoring,
- S+SVM+CNF using CNF for confidence scoring,
- S+SVM+CNF+F1 using CNF optimized for F1 measure.

In addition to CNF optimized for accuracy, being designed to minimize the misclassification error according to the definition of the SVM classifier [8], we also implement CNF+F1 optimized for F1-measure, by readjusting θ obtained from the SVM classifier (maximizing accuracy) into θ' maximizing F1-measure.

Table 6 shows the results of all four approaches categorized above on the four measures. Bold numbers indicate the best performance for each metric.

Observe that our proposed schemes using the confidence scoring (S+SVM+CNF and S+SVM+CNF+F1) significantly outperform S+SVM+NUL without confidence scoring. While S+SVM+NUL achieves high recall, it suffers from low precision, *i.e.*, less than 50%, by failing to identify "empty account" cases. In clear contrast, in all settings, S+SVM+CNF and S+SVM+CNF+F1 outperform S+SVM+REL in all metrics, with S+SVM+CNF+F1 being the winner in F1 metric, and S+SVM+CNF in precision.

Table 6: Effectiveness of the proposed methods on F1-measure, Precision, Recall, and Accuracy

Method	F1	Precision	Recall	Accuracy
S+SVM+NUL	0.6336	0.4839	**0.9241**	0.4839
S+SVM+REL	0.8062	0.7762	0.8403	0.8183
S+SVM+CNF	0.8148	0.**7917**	0.8415	0.8282
S+SVM+CNF+F1	**0.8178**	0.7856	0.8543	**0.8299**

5. CONCLUSION AND FUTURE WORK

This paper studied how to enhance entity search by matching the corresponding social network entities. This problem is challenging, as existing tools, using textual matching, suffer from low precision. In contrast, we holistically leveraged both textual and relational features and proposed a learning model to rank the matching accounts by the aggregated relevance. Meanwhile, as nearly half of Web entities do not have matching social entities, we observed that computing confidence for deciding whether to return the top match or return 'no matching account' is critical. For this purpose, we proposed to separately compute a confidence score of the match found, which was empirically more robust than using the relevance itself as a confidence measure. Our evaluation results empirically validated the accuracy of our algorithm over the real-life datasets.

Meanwhile, this paper does not address name disambiguation problem, which we leave as future work to investigate how relational features can be used for the disambiguation purpose.

6. REFERENCES

[1] EntityCube. http://www.entitycube.com.

[2] WebMynd. http://www.webmynd.com.

[3] R. Bekkerman and A. McCallum. Disambiguating web appearances of people in a social network. In *proc. WWW*, pages 463–470. ACM, 2005.

[4] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy. Make new friends, but keep the old: recommending people on social networking sites. In *proc. CHI*, pages 201–210. ACM, 2009.

[5] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *RecSys*, pages 53–60. ACM, 2009.

[6] S. Hill and F. Provost. The Myth of the Double-blind Review?: Author Identification Using Only Citations. *SIGKDD Explorations Newsletter*, 5(2):179–184, 2003.

[7] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: understanding microblogging usage and communities. In *proc. WebKDD/SNA-KDD*, pages 56–65. ACM, 2007.

[8] T. Joachims. Making large-scale support vector machine learning practical. pages 169–184, 1999.

[9] T. Joachims. Optimizing search engines using clickthrough data. In *proc. SIGKDD*, pages 133–142. ACM, 2002.

[10] T. Joachims. Training linear svms in linear time. In *proc. SIGKDD*, pages 217–226. ACM, 2006.

[11] J. Lee, S. won Hwang, Z. Nie, and J.-R. Wen. Query result clustering for object-level search. In *proc. SIGKDD*, pages 1205–1214. ACM, 2009.

[12] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *proc. S&P*, pages 173–187. IEEE Computer Society, 2009.

[13] Z. Nie, J.-R. Wen, and W.-Y. Ma. Object-level vertical search. In *proc. CIDR*, pages 235–246, 2007.

[14] B.-W. On, D. Lee, J. Kang, and P. Mitra. Comparative Study of Name Disambiguation Problem using a Scalable Blocking-based Framework. In *Proc. JCDL*, pages 344–353. ACM, 2005.

[15] B. Taneva, M. Kacimi, and G. Weikum. Gathering and ranking photos of named entities with high precision, high recall, and diversity. In *proc. WSDM*. ACM, 2010.

[16] E. M. Voorhees. The trec question answering track. *Natural Language Engineering*, 7(4):361–378, 2001.