iPLUG: Personalized List Recommendation in Twitter

Lijiang Chen[1], Yibing Zhao[2], Shimin Chen[3], Hui Fang[4], Chengkai Li[5], and Min Wang[6]

[1] HP Labs, Beijing, China
lijiang.chen@hp.com
[2] John Hopkins University, Baltimore, MD, USA
zyb009988@gmail.com
[3] Chinese Academy of Sciences, Beijing, China
chensm@ict.ac.cn
[4] University of Delaware, Newark, DE, USA
hfang@udel.edu
[5] University of Texas at Arlington, Arlington, TX, USA
cli@cse.uta.edu
[6] Google Research, Mountain View, USA
minwang@google.com

Abstract. A Twitter user can easily be overwhelmed by flooding tweets from her followees, making it challenging for the user to find interesting and useful information in tweets. The feature of *Twitter Lists* allows users to organize their followees into multiple subsets for selectively digesting tweets. However, this feature has not received wide reception because users are reluctant to invest initial efforts in manually creating lists. To address the challenge of bootstrapping Twitter Lists, we envision a novel tool that automatically creates personalized Twitter Lists and recommends them to users. Compared with lists created by real Twitter users, the lists generated by our algorithms achieve 73.6% similarity.

1 Introduction

Twitter is an instant content sharing service, through which users share their opinions and status by posting *tweet*s, i.e., short messages with less than 140 characters. As one of the most popular social networking services, Twitter boasts over 140 million active users and more than 340 million tweets per day[1]. With Twitter, a user (*follower*) can follow other users' (*followees*) tweets. A follow operation establishes a subscription-dissemination channel to send messages from a followee to a follower.

Twitter Lists. Users can easily be overwhelmed by flooding tweets from their followees since both the number of followees they subscribe to and the number of tweets their followees post on a daily basis could be very large. Thus, it is critical to help users organize their followees in order to more effectively digest and access the information posted by their followees.

Twitter Lists is one useful feature in Twitter for coping with this information overload problem. A user can organize her followees into multiple lists, each containing a subset

[1] http://blog.twitter.com/2012/03/twitter-turns-six.html

X. Lin et al. (Eds.): WISE 2013, Part II, LNCS 8181, pp. 88–103, 2013.

of followees. She can then opt to view the tweets from the followees in a particular list, thereby selectively digesting information from the list. A user can also subscribe to lists created by others without following the users in those lists.

Problem Definition. Albeit a potentially effective tool for social user organization and information selection in overloaded information space, Twitter Lists has not received wide reception from users [2]. One observation made from our empirical study is that over 80% of Twitter users have more than 100 followees, but only 35% of them have created at least one list and only 16% have more than 3 lists (cf. Observation 1). Moreover, among the users who have created lists, 49.6% of them included only less than 5% of their followees into their lists, and 76% of them included less then 10% of their followees in the lists (cf. Observation 2). This is not surprising, as other social media and social networks have encountered similar challenges in bootstrapping their services, including tagging, social bookmarking, and friend recommendation [5,1,4]. One particular reason for the low popularity of Twitter Lists is that users are reluctant to invest initial efforts in manually creating lists. Another reason is that users may have intrinsic difficulty in such a fuzzy grouping process.

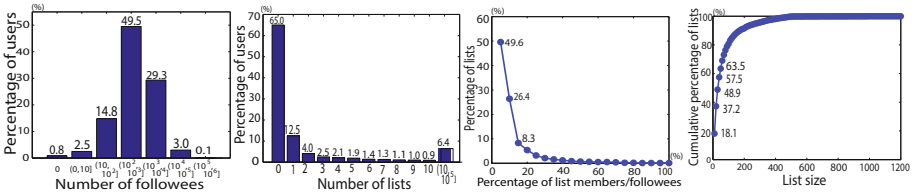
To tackle the low popularity of Twitter Lists and help achieve its full utility, we propose to automatically recommend personalized lists to Twitter users. This study can also shed light on solving similar problems in other leading social network services, which have features similar to Twitter Lists (e.g., *friend lists* in Facebook, *circles* in Google Plus).

Various definitions of the list recommendation problem may be worth studying. As an initial step towards effective list recommendation, this paper focuses on a particular problem definition— ***Given a Twitter user, recommend to the user new lists consisting of only the user's current followees.*** We consider the problem of recommending new followees a separate issue. We also do not simply recommend lists subscribed by many users, as such globally popular lists do not necessarily match a particular user's personal interests and social relationships.

Our Solution: iPLUG. We propose a solution that combines two approaches— a structure-based method and a content-based method, which recommend lists based on how users are co-listed in existing lists and how similar the contents of users' tweets are, respectively. After obtaining the initial list recommendations by the structure-based method and the content-based method, we improve the accuracy of the recommendations by performing both inter-list optimization and intra-list optimization. For inter-list optimization, we diversify top k recommended lists, to achieve a degree of overlap among these lists similar to the overlap exhibited by existing lists. For intra-list optimization, we study ways to prune unimportant members from the recommended lists. We call the resulting solution ***integrated PLUG (iPLUG)***.

The effectiveness of the proposed techniques is demonstrated by extensive experimental results. We evaluate iPLUG as well as various schemes that consist of only subsets of the proposed techniques. In a nutshell, the similarity between lists recommended by the proposed methods and those created by real users is as high as 73.6% for the largest lists and 72.2% for the largest five lists.

[2] `http://moreinmedia.com/2011/12/why-make-use-twitter-lists`



(a) Number of followees (b) Number of lists

Fig. 1. Statistics of Twitter lists

(a) List coverage (b) List size

Fig. 2. Live coverage and list size

2 Characteristics of Twitter Lists Created by Real Users

To understand the characteristics of real lists created by Twitter users and to gain insights towards our technical solution for personalized list recommendation, we collected a large Twitter data set and conducted a comprehensive study on its various statistics. Our data set contains $810,769$ Twitter users, $53,922,948$ lists, and 324 million tweets. The tweets were posted between December 27^{th}, 2007 and March 21^{st}, 2011. Details on the collection of the data set shall be described in the experiment section. In the rest of this section, we introduce our observations from the data set and provide analysis of the observations.

Observation 1. A Twitter user often follows a large number of other users, but rarely creates lists.

Figure 1(a) and Figure 1(b) show the distribution of the number of followees of a user and the distribution of the number of lists created by a user, respectively. From Figure 1(a), we observe that over 80% of Twitter users have 100 or more followees and almost half of all users have between 100 and 1000 followees. However, Figure 1(b) shows that 65% of users do not create any list. Only 16% of users have more than 3 lists, and nearly 93% have less than 10 lists. The sharp contrast between the large number of followees per user and the low popularity of Twitter Lists motivates our study of mechanisms for automatic list recommendation.

Observation 2. For users who have created lists, their lists often cover only a small fraction of their followees.

For those users that have lists, we measured the percentages of their followees in their lists. The results are shown in Figure 2(a). We observe that the lists created by a user often cover only a small fraction of the followees of the user. More specifically, 49.6% of the lists contain only less than 5% of the corresponding list owners' followees and 76% of the lists cover less than 10% of their followees. We also observe that lists are typically small in size. Figure 2(b) shows the cumulative distribution of list sizes (by increment of every 10 lists). The largest list has around 1200 members, while the most typical list sizes are [1, 10) and [10, 20), accounting for 18.1% and 19.1% of all the lists, respectively. 63.5% of the lists have less than 50 members. The observed small coverage and small size of existing lists further motivate the need for an automatic list recommendation approach.

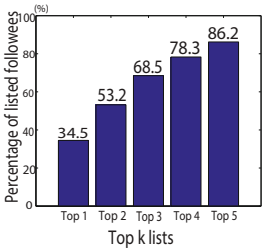


Fig. 3. The coverage of the largest 5 lists

User Set	Top 2	Top 3	Top 4	Top 5
Created $>$ 1 lists	13.2%			
Created $>$ 2 lists	13.7%	13.5%		
Created $>$ 3 lists	13.1%	12.6%	10.9%	
Created $>$ 4 lists	12.7%	11.8%	10.8%	10.0%

Fig. 4. Average redundancy in users' largest k lists

Observation 3. For users who have created multiple lists, a small number of their largest lists can cover most of their followees in their lists.

We further investigated the coverage of users' largest lists. We focused on the users who created at least 5 lists. For a user, we compute the coverage of her largest k lists by dividing the number of her followees in her largest k lists by all the followees in her lists. Figure 3 reports the average coverage across all these users while varying k from 1 to 5. As shown in Figure 3, the largest list of a user covers 34.5% of the followees in her lists on average, and the largest 5 lists cover an average of 86% of all followees in her lists. The statistics make it clear that a small number of lists cover most listed followees for a user. Assuming that creating longer lists takes more efforts, we speculate that users spend most efforts in creating largest 5 lists, which should be most important to them.

Observation 4. Although a user could put a followee into multiple lists, the number of overlapped followees among the lists is small.

We measured the redundancy in a user's largest k lists $l_1, \ldots, l_k$ as follows:

$$Redundancy = \frac{\sum_{j=1}^{k} |l_j| - |\bigcup_{j=1}^{k} l_j|}{|\bigcup_{j=1}^{k} l_j|} \quad (1)$$

where $|l_j|$ is the size of list l_j, i.e., the number of members in l_j. The $\bigcup$ denotes the set union operation.

Note that we only considered the largest k lists because many lists in the long tail only contain a few members and thus do not have much impact on the measured redundancy. We investigated the redundancy for different k values, ranging from 2 to 5. For each k value, we calculated the redundancy in the largest k lists, averaged across all users that have at least k lists. The result is shown in Table 4. We see that overall the average redundancy is small (10% to 13%) and slightly decreases while k increases.

Observation 3 and Observation 4 provide important characteristics of the lists created by real users. Given these characteristics, we aim to develop methods that generate lists exhibiting similar characteristics.

3 Personalized Twitter Lists Recommendation

Given a Twitter user u, let FE_u and FR_u denote the set of *followees* and the set of *followers* of u, respectively. Hence, $u_1 \in FE_{u_2} \Leftrightarrow u_2 \in FR_{u_1}$. A *list* l created by

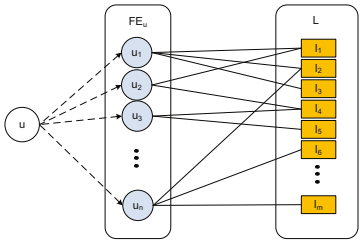


Fig. 5. The PLUG for a user u

user u is a subset of u's followees, i.e., $l \subseteq FE_u$. We use $L_u = \{l_1, l_2, \ldots, l_m\}$ to denote the set of lists created by u. Since a followee can be placed into multiple lists of u, it is possible that $l_i \cap l_j \neq \emptyset$. A followee of a user can also be followed by other users and included in their lists. The set of lists, to which a user u belongs, is denoted $L'_u = \{l \mid u \in l\}$.

Problem Statement: Given a user u and her followees FE_u, the problem of *list recommendation* is to generate k new lists $l_1, \ldots, l_k$ for u, where $l_i \subseteq FE_u$ and $l_i \notin L_u$ for $1 \leq i \leq k$.

Note that the problem focuses on recommending top-k lists. It is not necessary that every followee of a user u is assigned to a new or existing list.

3.1 Solution Overview

We propose two methods to form and rank candidate lists. The structure-based method exploits the personalized list-user graph (PLUG) of a user and recommends new lists containing her followees that are often included together in existing lists. The content-based method recommends lists based on content similarity of the tweets posted by list members. The two methods are different on two aspects. First, the structure-based method can only cover *listed followees*— a user u's followees that are included in at least one existing list created by any user, including u. The content-based method can also cover *non-listed followees*. Second, the content-based method is explicitly geared towards recommending interest-oriented lists, in which the list members share similar interests on topics (e.g., music and sports) shown by their tweets. The structure-based method can also recommend relation-oriented lists, in which the list members have common real-world relations with their follower u (e.g., family, friends, colleagues).

To improve the accuracy of the initial recommendations from these two methods, an inter-list optimization is applied to diversify recommended top-k lists, and an intra-list optimization is applied to prune unimportant members from the recommended lists.

We also combine the results from both structure- and content-based methods. The resulting method is termed *integrated-PLUG* (*iPLUG* for short). Suppose the top-k ranked lists for user u by structure-based and content-based methods are Rec^s and Rec^c, respectively. For each list $l \in Rec^s \cup Rec^c$, its ranking scores in Rec^s and Rec^c are $Score^s_l$ and $Score^c_l$, respectively. (Note that $Score^s_l = 0$ if $l \notin Rec^s$ and $Score^c_l = 0$ if $l \notin Rec^c$.) The iPLUG method computes the new ranking score of l by the following equation. The top-k lists with highest new scores are recommended to u.

$$Score_l = \eta Score_l^s + (1-\eta) Score_l^c \quad (2)$$

In Equation (2), η is a regulatory factor (i.e. $0 \leq \eta \leq 1$) that controls the contributions of the two methods to the final result. Larger η values prefer more contributions from the structure-based method. In our implementation we tested an intuitive value for η, which is the percentage of u's listed followees, i.e., $\eta = |\{u_i | u_i \in FE_u \text{ and } L'_{u_i} \neq \emptyset\}| / |FE_u|$. The rationale is that, since the structure-based method can only cover listed followees, the percentage is used to determine the extent of the method's effect.

3.2 Structure-Based Method

Personalized List-User Graph (PLUG). The structure-based method takes advantage of crowd intelligence of Twitter users to recommend new lists. We call two users u_1 and u_2 *co-listed* in list l, if both users are members of l, i.e., $u_1 \in l$ and $u_2 \in l$. If an existing list l co-lists u_1 and u_2, l's creator indicates that u_1 and u_2 are related from her perspective. Therefore, if u_1 and u_2 are frequently co-listed by existing lists, then many existing users agree that u_1 and u_2 are related. The more frequent that two users are co-listed, the stronger that they are considered related among Twitter users. We exploit this intuition in the structure-based method.

Specifically, to recommend lists to a user u, we build a *personalized list-user graph* (PLUG) for u. As illustrated in Figure 5, it is an undirected bipartite graph between the followees of u, depicted as circles, and existing lists created by all users (including u), depicted as rectangles. An edge between a followee node u_i and a list node l_j captures the membership relation $u_i \in l_j$. Two followees are co-listed in a list if they are both connected to the list node. The concept of PLUG is defined as follows.

Definition 1 (Personalized List-User Graph)
The personalized list-user graph of a user u, $PLUG_u = (FE_u, L, E)$, is a bipartite graph between two node sets FE_u and L, where FE_u is the set of u's followees and L is the set of lists containing u's followees, i.e., $L = \bigcup_{u_i \in FE_u} L'_{u_i}$. In the edge set E, each edge captures the membership of a followee u_i in a list l_j, i.e., $E = \{(u_i, l_j) \mid u_i \in FE_u, l_j \in L, u_i \in l_j\}$.

The algorithm exploits an iterative random walk model [3] to propagate scores on the bipartite graph. It ranks the followees FE_u and the lists L based on their connections in $PLUG_u$ according to the following equations.

$$s_{u_i}^{t+1} = \sum_{l_j \in L'_{u_i}} \frac{Weight(u_i, l_j)}{\sum_{u_k \in FE_u} Weight(u_k, l_j)} Score_{l_j}^t \quad (3)$$

$$s_{l_j}^{t+1} = \sum_{u_i \in l_j \cap FE_u} \frac{Weight(u_i, l_j)}{\sum_{l_t \in L} Weight(u_i, l_t)} Score_{u_i}^t \quad (4)$$

$$Score_{u_i}^{t+1} = \frac{s_{u_i}^{t+1}}{\sum_{u_k \in FE_u} s_{u_k}^{t+1}} \quad (5)$$

$$Score_{l_j}^{t+1} = \frac{s_{l_j}^{t+1}}{\sum_{l_k \in L} s_{l_k}^{t+1}} \quad (6)$$

In Equations (3) and (4), the score of a followee u_i is collected from all lists containing u_i, where the score of such a list l_j is distributed into its members. The score of u_i collected from l_j is according to the transition probability, which captures the importance of l_j to u_i. Recursively, the score of a list is collected from all its members, where the score of a member is distributed into its containing lists. The score of l_j collected from u_i is according to the transition probability. Then the scores of followees and lists are both normalized, by Equations (5) and (6), respectively. The initial score of a followee is 1 normalized to the total number of followees. The initial score of a list is the list size after removing unrelated members, $|l_j \cap FE_u|$, normalized to the aggregate size of the lists. With the initial scores of followees and lists, our method iteratively updates their scores using the above equations. After the scores converge, the algorithm selects and returns the top k lists with the highest scores as the recommended lists.

Tackling Sparsity of Co-listed Relation by List Clustering. For most users, we find that their PLUGs are sparse bipartite graphs, since many of the followees are placed in only a small number of existing lists because of the current low popularity of Twitter Lists. To tackle this sparsity problem of the co-listed relation, we improve the structured-based method by merging similar lists before the iterative computation of scores by Equation (3) and (4).

A natural similarity measure between two lists would have been based on the tweet contents of their members. However, such a similarity measure will only work for topic interest-oriented lists (e.g., food, music) rather than relation-oriented lists (e.g., colleagues in an organization), since the members of a relation-oriented list may not share the same interests and their tweets may not have similar contents. Such a similarity measure would increase the significance of interest-oriented lists, while relation-oriented lists will become relatively less significant. As a result, our algorithm would tend to recommend only interest-oriented lists, which is undesirable.

Instead, we merge lists according to the similarity of list names, thus making a balanced improvement for both kinds of lists. We pre-process list names by applying word stemming. Then we cluster the lists using the edit distance between stemmed list names as the distance function. Since we do not know the resulting number of clusters, we prefer hierarchical clustering to k-means. However, a regular hierarchical clustering algorithm is compute-intensive. Therefore, we instead use a greedy algorithm which aims to achieve the goal that the distance between any two final clusters is greater than or equal to a specified threshold T_{dist} ($T_{dist} = 0.4$ in our implementation). The algorithm randomly chooses a point (i.e., a list name) to form a single-point cluster, and then iteratively grows the cluster by adding other points whose average distances to the existing points in the cluster satisfy the threshold T_{dist}, until no more point can be included into the current cluster. The algorithm repeats this process to form multiple clusters. This greedy clustering algorithm computes the pair-wise distance between any two points at most once, and therefore has worst-case time complexity of $O(|L|^2)$.

Finally, we merge the multiple lists in each cluster into one list. The new PLUG of a user u is a bipartite graph between u's followees and the merged lists. Each merged list corresponds to a set of original lists. An edge exists between a followee u_i and a merged list l if u_i was in at least one of the constituent lists of l. In our data set, we start with $53,922,948$ original lists. The total number of resulting merged lists (clusters) for

all users is $240,572$, a nearly 99.5% reduction from the original lists. The new PLUG of a user becomes more compact and the extended co-listed relation helps improve our recommendation algorithm.

A new PLUG may not distinguish the importance of different followees in the merged lists. It is possible that a followee was included in multiple original lists within a merged list. For example, consider a musician u_1 that is in many music-related lists created by her fans. Such lists with similar names may be clustered into the same cluster and merged into a single list. In contrast, another followee u_2 may be in only one of the original lists before merging. It is clear that these two followees u_1 and u_2 should bear different importance in the resulting PLUG.

To distinguish the importance of different followees in the merged lists, we enhance PLUG to edge-weighted PLUG. A weight on an edge between a followee u_i and a merged list l_j represents the number of l_j's constituent lists that contain u_i:

$$Weight(u_i, l_j) = |L^0_{l_j} \cap L'_{u_i}| \tag{7}$$

where L^0_l is the set of constituent lists of l, i.e., the original lists that were merged into l. As discussed in the previous section, the edge weights are used to compute the transition probabilities for the iterative computation.

3.3 Content-Based Method

A limitation of the structure-based method is that it cannot cover *non-listed followees*—a user's followees that are not included into any existing list. This limitation is amplified by the low popularity of Twitter Lists. To tackle this problem, we also propose a content-based method that recommends lists based on the semantic similarity between the followees' tweets. Hence, this method can cover a non-listed followee as long as the followee has posted tweets.

To recommend k lists to a user, this method applies the k-means clustering algorithm to cluster a user u's followees into k clusters by the semantic similarity between their tweets. A virtual document is generated for each followee u_i, by concatenating the tweets posted by u_i. TF-IDF weighting is applied to represent each virtual document as a vector. The similarity between two followees is the cosine similarity between the corresponding two vectors, following the standard vector space model [12]. The clusters, i.e., new lists of followees, are ranked by the similarity between their centroids and the vector representing u's virtual document.

Since the content-based method is based on similarity of tweets, it is explicitly geared towards recommending interest-oriented lists. It tends to include two followees into the same list if their tweets exhibit similar topics. The structure-based method can implicitly recommend both relation-oriented and interest-oriented lists, since it captures the otherwise complex reasons for two followees to be included into the same list. A relation-oriented list represents real-world relations between a user and her followees (e.g., family, friends, colleagues), where the followees in the same list may not share common interests.

3.4 Inter-list Optimization: Reducing List Redundancy through Diversified Ranking

Given the ranked lists by both structure- and content-based methods, we discovered that many top-ranked lists share a large fraction of common members. One reason is that popular users are likely to be placed into multiple popular lists which makes both popular users and lists ranked high by our methods. However, we have observed that real lists created by users do not overlap much (Observation 4). To reduce the overlap among recommended lists and improve their diversity, we employ a Maximum Marginal Relevance algorithm [2]. It re-ranks lists by trading off between their original scores and diversity.

$$Score_l^d = Arg\max_{l \in S}[\lambda Score_l - (1-\lambda)\max_{l_i \in L} Sim(l, l_i)] \quad (8)$$

The new score of a list, $Score_l^d$, is calculated by Equation (8). Suppose L is the set of selected lists so far. We are to select the next list among a set of candidate lists S. $Score_l$ is the score of a list l, by Equation (2). $Sim(l, l_i)$ is the similarity between two lists. (Specifically, we use Jaccard similarity, i.e., $Sim(l, l_i) = \frac{|l \cap l_i|}{|l \cup l_i|}$.) λ is the parameter to balance the original list score and list diversity. A proper λ value is selected through experiments. According to Observation 4, the redundancy of the real lists created by users is about 10–13%. In choosing a proper value of λ, we set the list redundancy round 10%, because our goal is to make recommended lists exhibit similar statistics to that of the lists created by real users. Our diversification method chooses the list with the highest original score as the first recommended list, then iteratively applies Equation (8) to recommend the next list, until k lists are selected.

3.5 Intra-list Optimization: Pruning Unrelated Members from Merged Lists

Both structure-based and content-based methods may produce long lists, especially due to clustering and merging of original lists. The resulting merged lists may contain many members that have little relevance to the lists. They might be included just because they were co-listed with other members that are more relevant. For example, a followee u_f may be put into a list by another user u with whom she happened to play tennis once. The followee u_f in fact may not like tennis that much. However, since the list also contains many other followees that are true tennis fans, u_f may be included in a merged list corresponding to the tennis interest of u. In another example, user u_{1f} may be listed by a user u_1 in a list named "colleague", while user u_{2f} may be listed by a user u_2 in a list that happens to have the same name "colleague". Our list clustering algorithm will merge the two lists and u_{1f} and u_{2f} will be co-listed in the resulting merged list although they are not quite related.

Given a result list produced by the structure-based or content-based method, we compute a ranking score for each individual followee u_i in the list. With regard to the content-based method, the ranking score captures the relevance of an individual followee to the list. We compute the score as the cosine similarity between the vectors corresponding to the centroid of l and u_i. Recall that the vectors model the tweets posted by the users.

$$Score^c(u_i, l) = Cosine(Vector_{u_i}, Vector_l) \quad (9)$$

With regard to the structure-based method, the ranking score is the sum of co-occurrence counts for u_i and all other members in l, given by the following equation:

$$Score^s(u_i, l) = \sum_{u_j \in l \wedge u_i \neq u_j} |L'_{u_i} \cap L'_{u_j}| \tag{10}$$

We sort the followees in a list l by their ranking scores according to the above definitions, and then find a threshold t to divide the sorted followees into two classes, members ($C_m = \{u_1, \ldots, u_t\}$) and non-members ($C_n = \{u_{t+1}, \ldots, u_{|l|}\}$). We then prune all non-members from the list. Instead of tuning a pruning threshold by experiments and using the same threshold invariably for all lists, we apply the Otsu Thresholding Method [10] to automatically select an optimum threshold. This method is widely used for threshold selection from gray-level histograms in computer vision and image processing. The idea is to exhaustively search for the threshold t that minimizes the weighted intra-class variance for the two classes, defined as follows:

$$\sigma_w^2(t) = \omega_{C_m}(t)\sigma_{C_m}^2(t) + \omega_{C_n}(t)\sigma_{C_n}^2(t) \tag{11}$$

where weights $\omega_{C_m}(t)$ and $\omega_{C_n}(t)$ are the probabilities of the two classes separated by the threshold t, which are given by $\omega_{C_m}(t) = Pr(C_m) = \sum_{i=1}^{t} p_i$ and $\omega_{C_n}(t) = Pr(C_n) = \sum_{i=t+1}^{|l|} p_i$, where p_i is a normalized probability based on the followee's score $Score(u_i, l)$. $\sigma_{C_m}^2(t)$ and $\sigma_{C_n}^2(t)$ are the variances of scores in the member class and the non-member class, respectively. More details of this method can be found in [10].

4 Evaluation

4.1 Experimental Design

Twitter Data Set. We collected a Twitter data set through the Twitter API. Our implementation is based on an open source Java library– twitter4j[3].

We started with around 10 thousand randomly selected seed Twitter users. We call them level-1 users. We retrieved all the followees of the level-1 users. We call these followees level-2 users. We also retrieved all the followees of the level-2 users, which we call level-3 users. The union set of all level-1, level-2, and level-3 users is our Twitter user set. For each user in the set, we crawled the user's Twitter Lists, her followees, her followers, and her latest $3,200$ tweets[4]. The collected data set contains $810,769$ Twitter users, $53,922,948$ Twitter lists, and 324 million tweets. The earliest tweet was created on December 27, 2007 and the latest one was dated March 21, 2011.

We studied the characteristics of Twitter lists on the full data set. To quantitatively evaluate the effectiveness of the proposed methods, we construct an evaluation data set based on the collected data. The evaluation set contains a focus group with $8,614$ users and together they have $550,793$ followees. Our task is to recommend personalized lists for each user in the focus group based on his or her followees. The real lists created by these users are used as ground truth for our evaluation. The number of lists created by these users is $20,016$. On average around 25% followees of a user in the focus group have been included in the lists.

[3] `http://twitter4j.org/en/index.jsp`

[4] Due to Twitter API constraint, we can only retrieve at most 3200 tweets of a given user.

Table 1. Overview of list recommendation methods to evaluate

Algorithm	Basic Model		List Merging	Diversification	Member Pruning (Sec 3.5)		Integration
	Structure (Sec 3.2)	Content (Sec 3.3)	(Sec 3.2)	(Sec 3.4)	User Co-occurrence	Tweet Similarity	Model (Sec 3.1)
Content		√					
Content-Div		√		√			
Content-LMP		√		√		√	
PLUG-Basic	√						
PLUG-Merge	√		√				
PLUG-Div	√		√	√			
PLUG-LMP	√		√	√	√		
iPLUG	√	√	√	√	√	√	√

Methods To Be Evaluated. Several list recommendation methods are formed by combining the proposed techniques. Table 1 summarizes the techniques included in these methods which are compared in the evaluation.

Evaluation Methodology. When evaluating the above algorithms, we hide all the original lists created by users from the focus group and generate recommended lists for each of them. We then compare the recommended lists with the original ones. The reported performance is computed by taking the average of the performance for all the users in the focus group. Since we observed that the largest 5 lists of a user often cover 86% of the followees, we focus on comparing the top 5 recommended lists with the largest 5 lists created by users. In order to better understand the performance of our algorithms, we evaluate the results under two scenarios: (1) *Member Set (M-Set):* We recommend lists based on the followees who were list members in the original lists. (2) *Full Set (F-Set):* We recommend lists based on all the followees of a user.

We report the following measures in our evaluation:

• **Redundancy:** As defined in the previous section, it is a measure to quantify the overlaps among a set of lists. The statistics in Observation 4 show that the redundancy from the largest 2 to the largest 5 original lists are stably around 10%. Therefore, in our experiments, we choose a proper λ setting to keep the redundancy around 10%.

• **List Similarity:** We measure the similarity between two lists by Jaccard coefficient, i.e., $Sim(l_i, l_j) = \frac{|l_i \cap l_j|}{|l_i \cup l_j|}$. With this definition, the larger the similarity value is, the more similar the two lists are. To calculate the similarity between two sets of lists, we compute the average pairwise-similarity of all list pairs from the two sets.

4.2 Experiment Results

Top-1 List Recommendation Results. As shown in Observation 3, the largest list (top-1) covers more than $\frac{1}{3}$ followees of a user on average. Thus, it is the most representative list of a user. We compare the members of this list with the members of the top-1 recommended list, in terms of precision, recall, F1-measure and list similarity. The results on *M-Set* are shown in Table 2. We omit all results of Content-Div and PLUG-Div because in our experiments the diversification algorithm always selects the top-1 list as the first candidate to recommend. We also omit the results on *F-Set* because they exhibit similar trend.

Table 2. Experimental results of the top-1 lists

Algorithm	Precision	Recall	F1	Similarity
Content	0.321	0.276	0.297	0.295
Content-LMP	0.533	0.552	0.544	0.532
PLUG-Basic	0.051	0.065	0.095	0.048
PLUG-Merge	0.456	0.393	0.357	0.359
PLUG-LMP	0.694	**0.757**	0.724	0.727
iPLUG	**0.718**	0.742	**0.731**	**0.736**

Table 3. List similarity results on Member Set

Algorithm	top-2	top-3	top-4	top-5
Content	0.354	0.372	0.319	0.273
Content-Div	0.457	0.482	0.441	0.424
Content-LMP	0.556	0.581	0.612	0.574
PLUG-Basic	0.043	0.034	0.036	0.026
PLUG-Merge	0.456	0.393	0.357	0.359
PLUG-Div	0.563	0.592	0.601	0.552
PLUG-LMP	0.731	0.729	0.705	0.690
iPLUG	**0.735**	**0.746**	**0.733**	**0.722**

Table 4. List similarity results on Full Set

Algorithm	top-2	top-3	top-4	top-5
Content	0.143	0.153	0.139	0.086
Content-Div	0.238	0.253	0.261	0.251
Content-LMP	0.321	0.357	0.365	0.343
PLUG-Basic	0.012	0.009	0.009	0.008
PLUG-Merge	0.216	0.203	0.197	0.199
PLUG-Div	0.275	0.291	0.286	0.285
PLUG-LMP	0.436	0.462	0.476	0.426
iPLUG	**0.452**	**0.508**	**0.542**	**0.524**

Table 5. Redundancy of recommended lists

Algorithm	top-2	top-3	top-4	top-5
Content	73.5%	71.4%	69.2%	67.4%
Content-Div	18.6%	17.5%	16.6%	15.9%
Content-LMP	14.5%	13.1%	12.4%	11.3%
PLUG-Merge	72.4%	70.6%	68.2%	67.3%
PLUG-Div	16.2%	15.8%	14.6%	13.3%
PLUG-LMP	13.1%	12.4%	11.6%	9.9%
iPLUG	11.2%	10.3%	9.6%	9.3%

Table 2 shows that the iPLUG algorithm outperforms other algorithms in terms of precision, F1-measure, and list similarity. It achieves slightly worse recall compared to PLUG-LMP, but is dramatically better than other algorithms. Overall, iPLUG achieves 73.6% in list similarity and 73.1% in F1-measure. We also see that the performance of the PLUG-Basic model is poor due to the sparsity of the co-listed relation. List merging can effectively improve the performance of PLUG-Basic by about 5 to 10 times. Then list member pruning yields as much as 200% improvement in performance, indicating that irrelevant members often exist in the initial recommended lists.

Top-2 to top-5 Recommended Lists. The list similarity for top-2 to top-5 recommended lists for *M-Set* and *F-Set* are shown in Table 3 and Table 4. The two tables exhibit result trends similar to those in Table 2. Overall, the iPLUG algorithm achieves the best performance among all the methods. PLUG-LMP is quite close to iPLUG. There is only a 4% performance difference between iPLUG and PLUG-LMP on *M-Set*. This means that PLUG-LMP is good enough for most users in *M-Set*.

Comparing Table 3 with Table 4, we see that the performance of all algorithms deteriorates on *F-Set*. iPLUG is significantly better than all other methods, including the second best PLUG-LMP. The improvement of iPLUG compared with PLUG-LMP on *F-Set* is as much as 20%. It indicates that the content-based method has significant effect on iPLUG's performance when non-listed followees exist.

Diversification. To show the effectiveness of the technique of diversified list ranking, we compare the redundancy of the recommended lists of our major algorithms and show the results of top-$2, 3, 4, 5$ lists in Table 5. We find that the redundancy of PLUG-Div and Content-Div decreases by more than 70%, compared with PLUG-Merge and

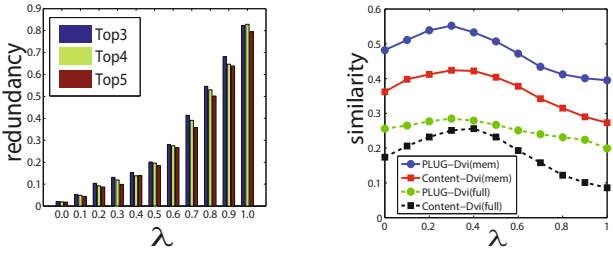
Content, respectively. This demonstrates the effectiveness of our diversified list ranking technique.

From results shown in Table 3 and Table 4, we see that the list similarity of both PLUG-Div and Content-Div improves significantly compared to PLUG-Merge and Content. The performance gap increases as k increases. The reason is that the top-k recommended lists are highly redundant before diversification. For example, the redundancy is 67–73% in the lists recommended by PLUG-Merge and Content. With diversified ranking, we are able to keep the redundancy to a relatively low level (around 10%), which is near the level of the lists created by real users.

4.3 Parameter Sensitivity Analysis

Parameter Settings of λ. In our diversified list ranking algorithm, the parameter λ plays the role of balancing list ranking from basic models (structure-based and content-based) and list diversification. As discussed earlier, the redundancy of lists captures list diversification. Two lists with less redundant members are considered more diverse. Therefore, we vary λ from 0 to 1, and report the redundancy of the top-k ($k = 3, 4, 5$) recommended lists, as shown in Figure 6(a). We see that λ has significant impact on the redundancy of the results. According to the statistics discussed in Observation 4, the redundancy of original lists in the real data is around 10%. From the results of Figure 6(a), it is clear that when λ is around 0.3, the redundancy of our recommended lists is close to 10%.

To understand the impact of the settings of λ on list similarity, we compute list similarity between the largest 5 original lists and the top-5 recommended lists while varying λ values for both *M-Set* and *F-Set*. The results of both structure-based and content-based algorithms are shown in Figure 6(b). The results indicate that the PLUG-Div and Content-Div algorithms achieve the best performance on *M-Set* when $\lambda = 0.3$. On *F-Set* Content-Div achieves the best performance when $\lambda = 0.4$. That is to say, setting λ to 0.3 (or 0.4) results in recommending lists most similar to the ones created by real users. Therefore, we set $\lambda = 0.3$ in our experiments.



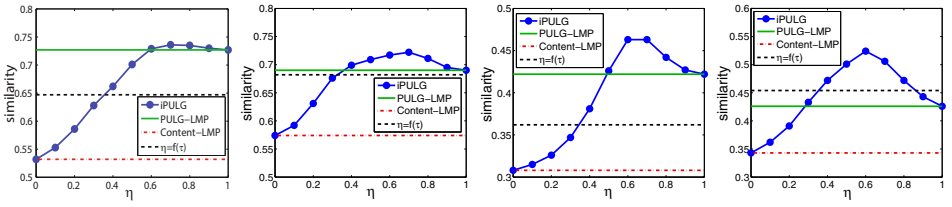
(a) List redundancy (b) Similarity of top-5 lists

Fig. 6. Impact of λ values on list recommendation

Parameter Settings of η. We study the impact of different settings of η on the performance of the integration algorithm (iPLUG). η is the factor that controls the contributions of the structure-based model and the content-based model to iPLUG. A larger η indicates that iPLUG sees higher contribution from the structure-based model, and lower contribution from the content-based model. We vary η from 0 to 1 and report the corresponding list similarity of iPLUG. Figure 7 shows the list similarity results of top-1 and top-5 lists. When $\eta = 0$, iPLUG becomes the content-based model. On the other hand, when $\eta = 1$, iPLUG uses only the structure-based model.

As we see in Figure 7, the performance monotonically increases from $\eta = 0$ to the peak ($\eta = 0.7$ on *M-Set* and $\eta = 0.6$ on *F-Set*). Afterwards, it decreases and finally is equal to the performance of PLUG-LMP algorithm ($\eta = 1$). That is, η has significant impact on the performance of iPLUG. It also shows the trend that generally taking in more contribution of the structure-based model results in better performance for iPLUG. The best setting of η for both top-1 and top-5 cases on *M-Set* are 0.7 while on *F-Set* are 0.6, we therefore set $\eta = 0.7$ and 0.6 respectively in our experiments.

As mentioned before, an intuitive value for η is the proportional of the followees who are listed, i.e., $\eta = \frac{\tau}{|FE_u|}$, where τ is the number of the followees that appear in at least one list. This method prefers more contribution from the structure-based model if a user has a larger fraction of listed followees. We set η separately for each individual user using this method. The results are shown as the black dash line in Figure 7. We see that in all four figures, the performance of this setting is between the performance of the settings of $\eta = 0.3$ and $\eta = 0.4$. The reason is that in our data set only 24.52% of followees of the focus group users have been listed. That is on average the iPLUG model takes only 24.52% of contribution from the PLUG model, and the fine-grained η settings for each user improves the performance more than a fixed setting of $\eta = 0.3$. However, we observe that it is far from the best performance result in our data set. Hence, $\eta = \frac{\tau}{|FE_u|}$ is not an optimal setting for η.



(a) Top-1, M-Set (b) Top-5, M-Set (c) Top-1, F-Set (d) Top-5, F-Set

Fig. 7. Impact of η values on lists similarity

5 Related Work

Our goal is to help Twitter users automatically group their followees and put them into different lists. The problem is similar to a recent study on classifying users accordingly to their interests [11]. This study is limited to bifacial interests detection such as detecting whether a user is democrat or republican or whether a user likes Starbucks or

not in business affinity detection. Twitter List can be considered as a tag for each followee. Tag and Tag-based entity (e.g. photos, music, videos, web pages) classification or clustering [9,6,15] have been used to improve the performance of web page indexing, music categorization, news filtering and content recommendation. Compared with these studies, the difference of our method is to take advantage of both structural and content information for list recommendation.

Social recommendation is becoming prevalent in online services [1,14,7]. Social tags help users better understand, interact with and propagate variety of content. However, user-generated tags with uncontrolled vocabulary can sometimes be ambiguous, obscure, inadequate and redundant. To tackle this problem, [13] proposed a personalization algorithm based on content-dependent variant of hierarchical tag clustering. We observed the same drawback of list names as [13] discovered with user-generated tags. Also, the topic selection algorithm in [8] helps to eliminate the redundancy between topics and focus on the important ones. In our algorithm, instead of using the semantic information from list name, we take advantage of the information of users and lists.

6 Conclusions and Future Work

In this paper, we propose the iPLUG algorithm for recommending lists for Twitter users, which leverages both the structure of list-user graphs and the tweets of users. Experiments over a Twitter dataset show that iPLUG is capable of recommending lists similar to the ones created by real users. There are many interesting directions for future work. First, it is interesting to study how to help users automatically discover and add new followees to existing lists. Second, it would be interesting to study how list recommendation changes user behavior on Twitter.

Acknowledgments. The work of Li is partially supported by NSF grants 1018865, 1117369, and 2011, 2012 HP Labs Innovation Research Award.

References

1. Belém, F., Martins, E., Pontes, T., Almeida, J., Gonçalves, M.: Associative tag recommendation exploiting multiple textual features. In: SIGIR (2011)
2. Carbonell, J., Goldstein, J.: The use of mmr, diversity-based reranking for reordering documents and producing summaries. In: SIGIR (1998)
3. Deng, H., Lyu, M.R., King, I.: A generalized co-hits algorithm and its application to bipartite graphs. In: SIGKDD (2009)
4. Guan, Z., Wang, C., Bu, J., Chen, C., Yang, K., Cai, D., He, X.: Document recommendation in social tagging services. In: WWW (2010)
5. Guy, I., Zwerdling, N., Ronen, I., Carmel, D., Uziel, E.: Social media recommendation based on people and tags. In: SIGIR (2010)
6. Khudyak, A., Kurland, O.: Cluster-based fusion of retrieved lists. In: SIGIR (2011)
7. Lu, C., Hu, X., Chen, X., Park, J.-R., He, T., Li, Z.: The topic-perspective model for social tagging systems. In: SIGKDD (2010)

8. Rowe, M., Wagner, C., Strohmaier, M., Alani, H.: Measuring the topical specificity of online communities. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 472–486. Springer, Heidelberg (2013)
9. Meeder, B., Karrer, B., Sayedi, A., Ravi, R., Borgs, C., Chayes, J.: We know who you followed last summer: inferring social link creation times in twitter. In: WWW (2011)
10. Otsu, N.: A threshold selection method from gray-level histograms. In: IEEE TSMC (1979)
11. Pennacchiotti, M., Popescu, A.-M.: Democrats, republicans and starbucks afficionados: user classification in twitter. In: SIGKDD (2011)
12. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM (1975)
13. Shepitsen, A., Gemmell, J., Mobasher, B., Burke, R.: Personalized recommendation in social tagging systems using hierarchical clustering. In: RecSys (2008)
14. Venetis, P., Koutrika, G., Garcia-Molina, H.: On the selection of tags for tag clouds. In: WSDM (2011)
15. Yin, Z., Li, R., Mei, Q., Han, J.: Exploring social tagging graph for web object classification. In: SIGKDD (2009)