2015 IEEE 15th International Conference on Data Mining Workshops

Crowdsourcing based API Search via Leveraging Twitter Lists Information

Tingting Liang[1], Liang Chen[1], Haochao Ying[1], Zibin Zheng[2], Jian Wu[1]
[1] College of Computer Science & Technology, Zhejiang University
Hangzhou, China
[2] School of Data and Computer Science, Sun Yat-sen University
Guangzhou, China
Email: {[1]liangtt, [1]cliang, [1]haochaoying, [1]wujian2000}@zju.edu.cn
[2]zbzheng@cse.cuhk.edu.hk

***Abstract*—With the rapid growth of open APIs on the Internet, searching appropriate APIs for a given query becomes a challenging problem. General API search systems, such as ProgrammableWeb, usually can not provide satisfactory results of API search due to the simple keywords matching between queries and API information offered by providers (e.g. name and description). In this paper, we propose a crowdsourcing based search approach named CrowdAPS to effectively find the appropriate APIs. Specifically, CrowdAPS leverages Twitter lists, which is a tool used by individual users to organize accounts that interest them on semantics. List meta-data, including list name and description, is generated from collective intelligence and can be used by Latent Semantic Indexing (LSI) model to acquire semantic similarity between APIs and queries. Furthermore, CrowdAPS exploits list number to infer the popularity of APIs. The final search result relies on the integration of semantic similarity and popularity. Comprehensive experiment based on real-world datasets crawled from ProgrammableWeb and Twitter demonstrates the effectiveness of CrowdAPS.**

I. Introduction

In the era of Web and mobile applications, a new kind of service called application programming interface (API) is springing up driven by SaaS (Software as a Service) and mobile applications. An API can be regarded as a protocol or application communication channel intended to be used as an interface by software components to communicate with each other. For example, consumers can access flight data from their mobile devices, use Google Maps from a hotel website and make payments online with smart phones, all of which are implemented by invoking open APIs. There are billions of API calls a day and the number is going to increase with the proliferation of smart devices as they start interacting over APIs. For instance, Twitter API calls reached 13 billion per day in May 2011, Netflix held 1.4 billion API calls per day in May 2012, and eBay API was invoked 1 billion per day in the first quarter of 2012 [1].

Compared to the traditional web services, APIs are in a more popular style. Anyone can create an API to share data or services and anyone can spot an opportunity for a new service and change it into a bigger framework with APIs. Thus APIs can make solution more accessible and more useful. For web services developers, APIs provide the advantage of external services and data to boost their offerings. These lead to an explosion in the number of APIs, and there exist many websites that are exclusively dedicated to manage APIs and mashups in current, such as ProgrammableWeb (PW)[1], Mashape[2], etc. According to the analysis made by PW, the number of APIs followed by it increased rapidly in recent years and has reached 10850 by October 2014. Due to the explosive increase of API number, how to discover the appropriate APIs becomes a hot issue. Currently, the search systems of most API management platforms (PW, Mashape, etc) could not find the best of breed APIs that consumers need as they work through simple keywords matching. Taking PW as an example, if a consumer wants to find an API that can offer him/her data or services about travel and takes "travel" as the query in PW search system, the top result is "Webcams.travel" which is a directory of touristic webcams and classified in *Video* category. Absolutely, "Webcams.travel" is not the objective API. The reason is that PW search system only seeks APIs whose names or descriptions contain the keyword "travel". That is to say, the general search systems for APIs consider the keywords match rather than the real semantic or topic information of APIs. Moreover, simple keywords matching neglects the popularity of APIs, which also limits the quality of API discovery.

Introducing crowdsourcing information to improve the quality of search methodology is a popular and novel strategy, and it has been proved effective in many other fields, such as information retrieval, social network, etc. Ghosh et al. [2] exploited crowdsourcing information to discover topic experts in Twitter social network. In [3], the author showed the necessity of applying social tags in music information retrieval. *Twitter Lists* was proposed in late 2009 to help users organize the Twitter accounts they follow [4]. It have been widely used to group sets of users into topical categories. Generally, a list is created with a name (required) and description (optional) and the creator can manage the list members. For a Twitter account, the names and descriptions of lists including it can be considered its semantic labels and the number of these lists reflects the popularity of the account. Through the information

[1]ProgrammableWeb: http://www.programmableweb.com
[2]Mashape: https://www.mashape.com

978-1-4673-8493-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDMW.2015.134

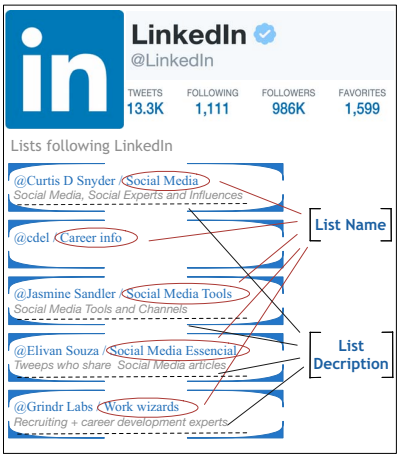
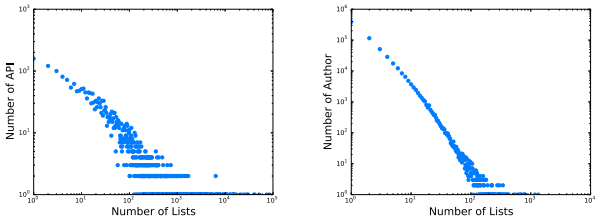


Fig. 1. Some examples of lists containing LinkedIn



(a) (b)

Fig. 2. Distribution of the number of APIs & Authors based on the number of relevant lists

of APIs provided in PW, it is noticed that most APIs have official Twitter accounts. Fig.1 shows several lists that contain LinkedIn, the twitter account of LinkedIn API linked in PW. Most of the list names are related to 'Social Media' and 'Career' along with the corresponding list descriptions, which distinctly reflect the topic of LinkedIn. We crawled 3877 APIs' official Twitter accounts and the lists involving them from Twitter. Fig.2 (a) shows the distribution of the number of APIs based on the number of lists involving them. One API could be involved in many lists and the most reaches 84511. 30.8% of total APIs are organized by more than 100 lists, and 70.1% are included by at least 10 lists. Thus the list information of most APIs is full of richness. Fig.2 (b) illustrates the distribution of the number of authors based on the number of lists created by them. The total amount of authors who created lists for the 3877 APIs is 658878, of which only 3.8% created more than 10 lists. It is obvious that most authors designed lists less than 10 and the sources of lists are in a wide range, which indicates the diversity of list semantics.

In this paper, we propose a crowdsourcing based search approach, named CrowdAPS, to accurately find the appropriate APIs. CrowdAPS leverages Twitter lists which are carefully managed by individual users to organize accounts (APIs' official Twitter accounts) that interest them and whose meta-data, including list names and descriptions, offers significant semantic cues to the topics of involved accounts. The critical idea of CrowdAPS is to infer an API's topics by analyzing the meta-data of the lists containing the API Twitter account. Latent Semantic Indexing (LSI) model is applied to map both the features extracted from lists and the given queries into a conceptual space to calculate the similarity between APIs and queries. Except for the semantic information provided by lists, CrowdAPS considers an API's popularity which can be measured by the number of lists so as to satisfy consumers' demand of popular APIs. For a given query, the final ranking scores of related APIs are decided by the combination of APIs' popularity and similarity between APIs and the query. The CrowdAPS based search engine is online available, users can use it to search APIs by visiting it[3].

In particular, the main contribution of our paper is summarized as follows:

1. We propose a novel approach named CrowdAPS, which utilizes lists in Twitter, a kind of collective intelligence of social media, to accurately search the appropriate APIs.
2. We integrate similarity and popularity obtained from lists to get the final ranking scores of the APIs related to the given query.
3. We utilize the information provided by Alexa to evaluate the performance of the proposed CrowdAPS with the datasets crawled from Twitter and ProgrammableWeb.

The remainder of paper is structured as follows: Section II reviews the related work. The architecture of CrowdAPS is introduced in Section III. Section IV describes the detailed procedure of the proposed API search approach. Section V shows a simple case study of CrowdAPS. The concrete evaluation process is illustrated in Section VI and Section VII makes a conclusion of this paper.

II. Related Work

Research in the fields of API and mashup have been widely published in recent years, specially including API and mashup recommendation, searching and ranking, etc. Zhong et al. [5] proposed a time-aware API recommendation approach for mashup creation with the combination of service evolution, collaborative filtering and content matching. A user-group item-based collaborative filtering method is proposed for personalized Open API recommendation in clouds by [6]. Chan et al. [7] modeled API invocations as an API graph to find an optimum connected subgraph and effectively improved API recommendation. In [8], Cao et al. presented an approach to recommend mashup services, which considers users' interests mined from their mashup usage history and the social network based on relationships of mashup services, Web APIs and tags. In [9], Dojchinovski et al. described a novel API selection method, which considers a user profile, user's preferences, temporal aspects and social links, to address the problem that the more recent, newly created APIs are impeded to adoption in some current selecting approaches. Different with

[3]http://zjumsi.com/projects/case/index.php

these service (API and mashup) recommendation approach that implemented in the same library, Zheng et al. [10] tried to recommend related APIs of different libraries through mining search results of Web search engines.

In addition to the research about API and mashup recommendation mentioned above, API search, which is the main work in this paper, is becoming another hot issue due to the explosive growth of APIs. Broadly speaking, API search is a kind of service discovery, which has been widely studied. Zhang et al. [11] presented WSExpress considering QoS characteristics of Web services for service discovery. Wu et al. [12][13] proposed a Web service clustering approach to improve service search. In [14], Gomadam et al. proposed a method for searching and ranking Web APIs that adopts document classification and faceted search, and the serviut score is used to rank APIs based on their utilization and popularity. Torres et al. [15] constructed collaboration network of APIs and proposed a API discovery approach based on social information. Lee et al. [16] showed how to syntactically define and semantically describe the characteristics of APIs and how to use these descriptions to easily search and composite APIs. An iterative approach to find APIs for building mashups was presented in [17], with the combination of semantic and social information obtained from the built collaborative social network of APIs. Bianchini et al. [18] provided a multi-perspective based API search framework for enterprise mashup design, in which a new perspective of the web designers' experience is used together with other Web API search techniques, relying on classification features, and technical features, such as the API protocols and data formats.

Almost these API search approaches exploit semantic descriptions offered by API providers or social relationships between APIs which mainly measured by their utilization within the common mashups. The performance of API search will be limited by the lack of richness and diversity in these utilized information. Thus, in this paper, we propose a crowdsourcing based method for API search, in which collective intelligence of social media is utilized.

III. Framework of CrowdAPS

Fig.3 illustrates the framework of the proposed CrowdAPS. The list data for API search is crawled from Twitter after being matched with APIs crawled from ProgrammableWeb. Generally, the process of CrowdAPS could be divided into two parts: The first part is utilizing the name and description information of lists to compute semantic similarity between an API and a user query based on LSI model. The other part is about popularity calculation for each API on the basis of its list number, which reflects how popular an API is among Twitter users to a certain extent. Finally, the scores obtained by the two parts are integrated with a certain weight, then a list of ranked results is returned to the user.

IV. Crowdsourcing based API Search

In this section, we first introduce the content and form of Twitter *Lists*. Then the preprocessing of API search approach

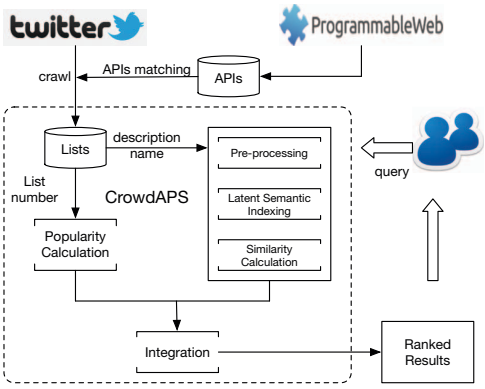


Fig. 3. Framework of crowdsourcing based API search

is described. Later we discuss a LSI based algorithm for text similarity computation. Finally, we present the strategy of integrating similarity with the popularity of lists to be the final rank score.

A. Leverage Twitter Lists

Lists is introduced for users to manage their followings and the Tweets they post. Users can group a set of accounts whom they think have the similar topics by creating a list. A list includes a list name (limited to 25 characters) and a list description that approximately shows the content of the list.

Table I shows illustrative examples of lists containing APIs, selected from our dataset (to be detailed in Section VI). As the examples present, Flickr API is included in lists with the name "Photography & design", "Photography", "Photography websites" and corresponding descriptions. LinkedIn and lastfm are in the similar situation. It can be observed that the list names and descriptions provide valuable semantic cues to the topics of the APIs in the lists. For instance, according the the list meta-data, we can associate Flickr API with photography or photo topic, LinkedIn API with social or profession topic, lastfm with music topic. In addition, the number of lists that including a certain API is a significant feature for measuring the popularity of this API. Thus, lists offer both semantic annotations and popularity to the APIs included by them. And these features are generated by arbitrary users, which means they reflect the collective knowledge of crowds.

The search method proposed in this paper mainly considers extracting the semantics and popularity contained in the crowdsourced lists to find appropriate APIs.

B. Data Pre-processing

Due to the arbitrariness of list meta-data (names and description), a pre-processing of lists is needed before applying LSI. The detailed strategy of extracting features from list meta-data consists of the following steps:

1. **Tokenization**: As list names are limited to 25 characters, users often combine words using *CamelCase* (e.g., PlayStation). It is essential to separate these into individual words by tokenization algorithm.

TABLE I
EXAMPLE OF APIS, THE NAME AND DESCRIPTION OF LISTS CONTAINING THEM

API Name	List Name	List Description
Flickr	Photography & design	Visual inspiration for my eyes to swallow
	Photography	The art of drawing with light
	Photography websites	Twitter accounts related to Photography you should be following
LinkedIn	Social Media	Bloggers and social media networks
	Professional-Resources	Writing and Design for Interactive Media
	Recruiters	Job Tweets
lastfm	Music	Flagging events or new music
	DJs, Producers, Labels	Music worth listening to
	Home Taping Is Killing Music	Bands and Music; go on, rock my world

2. **Stop Words Removal**: Words that are not meaningful for presenting APIs, such as *a*, *the*, *about*, etc, should be removed. In addition to the common stop words, a set of domain-specific stop words also should be filtered out (e.g., Twitter, Google, List).
3. **Nouns & Adjectives Identification**: It has been analyzed in previous work [19] that nouns and adjectives in list names and descriptions are particularly useful for semantics extraction. Thus we identify nouns and adjectives using a tagger tool.
4. **Stemming**: Extracting stem words is another important process. The stem words can be obtained by removing the commoner morphological and inflectional endings from words. For example, *travel*, *traveler*, *traveling* will be replaced with the same stem *travel*.
5. **Low Frequency Words Filtration**: As the definition of list names and descriptions is arbitrary, some words with low frequency are regarded as the impurity in the presentation of APIs and filtered out.

C. Latent Semantic Indexing

Generally, information is retrieved by literally matching words in documents with those of the given query. However, lexical matching may be inaccurate since some words share the same concept (synonymy) and most words have multiple meanings (polysemy). Moreover, for list meta-data, the drawbacks of literal word matching become more obvious due to personal style and individual differences in word usage. Thus a better method should permit users to search information based on conceptual topics or semantics of a documents.

Latent Semantic Indexing (LSI) [20] is a model that mapping queries and documents into a space with latent semantic dimensions. In a latent semantic space, high cosine similarity of a query and a document of which words are conceptually similar can be obtained even if they do not share any words. LSI consists of the following four main steps, of which the first two steps are also applied in vector space models and the step of dimension reduction is the critical part.

1) Term-Document Matrix: A large collection of text is usually represented as a term-document ($t \times d$) matrix X, with each position x_{ij} corresponding to the frequency with which a term (a row i) occurs in a document (a column j). Note that the order of terms in the document is unconsidered in matrix X on the basis of "bag of words". Matrix X is typically sparse since most documents contain only a small percentage of total number of unique terms in the whole corpus.

2) Transformed Term-Document Matrix: Instead of dealing with raw term frequencies, the entries in the term-document matrix X are often transformed. An appropriate weighted technique is TF-IDF which is widely used as a weighting factor in text mining and other related fields. TF-IDF value of each word increases with its frequency in corresponding document, while offsets by its frequency in the whole collection.

3) Dimension Reduction (SVD): In traditional vector retrieval systems, documents and queries are denoted as vectors in n-dimensional space, where n is the number of indexed terms in the collection. And for applying LSI for document searching, Singular Value Decomposition (SVD) is used to decompose the term-document matrix $X_{t\times d}$ into the product of three matrices, $T_{t\times n}$, a term by dimension matrix, $S_{n\times n}$, a singular value matrix, $D_{d\times n}$, a document by dimension matrix:

$$X_{t\times d} = T_{t\times n} S_{n\times n} {D_{d\times n}}^T, \tag{1}$$

where t denotes the number of terms, d is the number of documents, $n = min(t, d)$, T and D have orthonormal columns:

$$TT^T = I, DD^T = I.$$

Suppose the rank of matrix X is r, the diagonal elements of S are ordered by magnitude,

$$S = diag(\sigma_1, \sigma_2, ..., \sigma_n),$$

where $\sigma_i > 0$ for $1 \leq i \leq r$, $\sigma_i = 0$ for $j \geq r + 1$.

LSI uses a truncated SVD, keeping the k largest singular values and their associated vectors, so

$$\hat{X}_{t\times d} = T_{t\times k} S_{k\times k} {D_{d\times k}}^T, \tag{2}$$

where k is the dimensionality of the reduced space and $k << n$. The resulting reduced-dimension SVD representation is the best k-dimensional approximation to the original matrix in the least-squares sense. The purpose of dimension reduction is to reduce the noise in the latent space, leading to a richer term relationship structure that reveals latent semantics. Through SVD, each term and document is represented as a k-dimensional vector in the latent semantic space. The rows in $T_{t\times k}$ are the term vectors and the rows in $D_{d\times k}$ are the document vectors in latent semantic space.

TABLE II
TOP 5 RESULTS BY PW SEARCH AND CROWDAPS FOR TWO QUERIES, ALONG WITH API DESCRIPTION, CATEGORY AND PROVIDER RANK

PW search				CrowdAPS			
API Name	**Extracts from Description**	**Category**	**Rank**	**API Name**	**Extracts from Description**	**Category**	**Rank**
Query: travel							
Webcams.travel	touristic webcams, Webcamse	Video	27869	TripAdvisor	travel site, hotels, flights, vacation, guides	Travel	217
Yahoo Travel	trip plans, retrieve	Travel	4	Expedia	travel, book hotels, flights, cars.	Travel	49537
mTrip	travel industry, agencies, tour operators	Travel	549004	British Airways	global airline, global route network	Travel	2400
Traxo	manage travel information, travel rewards	Travel	1141036	Travelocity	book travel, airline tickets, hotels, rental cars	Travel	2079
Extentia	custom software, education, travel, healthcare	Travel	329270	AirBNB	vocation rental, room rental, traveller	Travel	1146
Query: running							
Postcard on the Run	take photographs, print postcards	Photos	1462243	ESPN	sports news, statistics, team and player information	Sports	66
Cafe Press	make queries, run searches against Cafe Press' store	eCommerce	2438	Yahoo Fantasy Sports	fantasy sports provider, rich data on teams, player	Sports	4
CasperBox	run CasperJS scripts online	Developers	5113960	TrainingPeaks	training, nutrition software for athletes & coaches	Sports	28401
Mogotest	allows user run tests against their sites	Testing	1055323	Couchbase	NoSQL technology	NoSQL	52830
TripSay	run queries, share trips	Travel	8268718	Dailymile	social training log for runners	Sports	31615

4) Retrieval in Latent Semantic Space: When LSI is used for retrieval, a query must be represented as a vector in k-dimensional latent semantic space that the document collection is represented in. A query is seen as a set of words like a document, and it can be represented by multiplying the transpose of the query's term vector with $T_{t\times k}$ and ${S^{-1}}_{k\times k}$:

$$\hat{q} = q^T T_{t\times k} {S^{-1}}_{k\times k}, \tag{3}$$

where the term $q^T T_{t\times k}$ reflects the sum of these k-dimensional term vectors, and the right multiplication with $S^{-1}_{k\times k}$ weights the separate dimensions distinguishingly. By representing query in this way, the distance between documents and given query can be measured using cosine similarity. The returned relevant documents are ranked according to cosine similarity. Thus the semantics based rank score of list meta-data of an API returned for a query could be defined as:

$$\begin{aligned} score_s(q, API_i) &= sim(q, d_i) \\ &= \frac{\sum_j w_{q,j} w_{i,j}}{\sqrt{\sum_j {w^2}_{q,j}} \sqrt{\sum_j {w^2}_{i,j}}}, \end{aligned} \tag{4}$$

where **w** denotes the term vector of a document or query in latent semantic space.

D. Integration of Similarity and Popularity

Returning the semantics matched APIs for given queries is essential, while popular APIs are also demanded by users. Except for semantic information, Twitter lists of APIs imply popularity information. The intuition behind it is that the more lists including an API, the more popular this API is. In other words, the popularity of an API is decided by the number of lists that contain the API. Considering the large gap between the numbers of different APIs' lists, we normalize the number of lists into $[0, 1]$ with its logarithmic form. Therefore, the popularity based rank score of an API is defined as follows:

$$score_p(API_i) = \frac{\log_{10} N - \log_{10} min}{\log_{10} max - \log_{10} min} \tag{5}$$

where N represents the number of lists including the ith API, min and max respectively denote the minimal and maximal numbers of APIs' lists.

The final rank score of an API is decided by the integration of semantic similarity between API and a query and API's popularity, since the two measures synthetically satisfy the demands of users. So combining (4) and (5), we define the final rank score as:

$$\begin{aligned} score(q, API_i) = \lambda score_s(q, API_i) \\ + (1-\lambda) score_p(API_i), \lambda \in [0, 1] \end{aligned} \tag{6}$$

where λ is a weight to balance the importance of semantic similarity and popularity.

V. CASE STUDY

This section illustrates retrieval cases for two queries: *travel* and *running*, in order to better understand the distinction of ProgrammableWeb's own search system (PW search) and the proposed CrowdAPS that leverages crowd knowledge. For the two queries, Table II reveals the comparison between the top 5 results from PW search and the top 5 results generated by CrowdAPS, along with the description, category and provider rank. Provider rank represents the ranking of the site of API's provider, which is provided by Alexa[4]. The intuition is that an API generated by a provider whose site ranked higher would be regarded as the relatively popular one.

As mentioned in Introduction, the first result of query *travel* returned by PW search is about touristic webcams and classified in *Video*, which brought by simple lexical matching. Note that while the top results of PW search contain APIs that offer general services about management, retrieval and software, etc, the top CrowdAPS results are much more specific in the requirements of travel. And for query *running*, the top results are generated by PW search mainly due to the occurrence of keyword "run" in API's name or description. Most of the top

[4]Alexa: http://www.alexa.com/

results obtained from CrowdAPS are connected with *sport*, the most reasonable topic related to *running*. Furthermore, the provider rank in CrowdAPS is overall higher than that in PW search. The popularity of two cases are respectively improved 37 times 140 times with the average provider ranks, 11076 and 22584, which indicates that lists information assuredly improves the popularity of returned APIs.

VI. Experiment & Discussion

A. Dataset and Setup

The datasets used in this paper are respectively crawled from Twitter and ProgrammableWeb, and named as D_p and D_t. First we crawl 10,850 APIs from PW, then 3,877 API official Twitter accounts are crawled from Twitter matching with APIs in PW. 414 out of total 3,877 APIs with no list are removed since the kernel of CrowdAPS is leveraging list meta-data. As the key attribute for evaluation, category can not be empty and 11 APIs are filtered out for this reason. The amount of APIs is 3,453, and D_p contains API name, description, category. D_t includes API name, list name, list description, and the number of lists in D_t reaches 1,712,777, 36.0% of which have descriptions. Besides, we crawl an dataset about API provider rank mentioned in section V from Alexa as an evaluation criteria for the ranking of APIs.

All the experiments in this paper are implemented with Python 2.7, conducted on a MacBook Pro with an *2.2 GHz Intel Core i7 CPU* and *16 GB 1600 MHz DDR3 RAM*, running *OS X Yosemite*.

B. Evaluation Methodology

For evaluation, a method of binary judgment should be applied to identify whether the returned API is relevant to the given query. The queries used for evaluations could be chosen from a given set of 52 sample queries that are extracted from topics of the 12 categories shown in Table III. The idea of binary judgment in the evaluation is: a returned result for a given query can be judged as relevant once its category is identical with the category of the query. Except for binary judgment, which is utilized for evaluating the impact of semantics provided by lists, we adopt provider rank mentioned in section V to estimate the influence of popularity information mined from lists. We use the following three metrics to evaluation CrowdAPS.

1) Precision@k: Precision measures the proportion of returned documents that are relevant. Precision at cutoff k (P@k) is calculated as:

$$\text{P@}k(r) = \frac{1}{k}\sum_{i=1}^{k} r_i, \tag{7}$$

where r_i indicates the relevance of the ith ranked result scored as either 0 (not relevant) or 1 (relevant).

2) nDCG@k: Normalized discounted cumulative gain (nDCG) is an explicitly rank-weighted metric. For a ranked list retrieved for the user, the further down the list a result occurs, the more its value to the user is discounted. If binary relevance is determined, with R relevant results for a query, the formula of nDCG@k is:

$$\text{nDCG@}k(r) = \frac{\sum_{i=1}^{k} r_i \cdot w_{DCG}(i)}{\sum_{i=1}^{\min\{k,R\}} w_{DCG}(i)}, \tag{8}$$

where r_i indicates the relevance of the ith ranked result, and

$$w_{DCG}(i) = \begin{cases} 1/\log_2(i) & \text{if } i > 2; \\ 1 & \text{otherwise.} \end{cases}$$

3) Average Rank@k: In our evaluation, Average Rank at cutoff k (AR@k) is used for measuring popularity of search results. AR@k is calculated as:

$$\text{AR@}k(r) = \frac{1}{k}\sum_{i=1}^{k} rank_i, \tag{9}$$

where $rank_i$ denotes the ranking score of the ith ranked result. A lower AR@k indicates a higher rank.

In our experiment, the cutoff k is set as 10, and the value of $rank_i$ is obtained from the provider rank of each API as mentioned in section V.

TABLE III
The 52 Sample Queries for Evaluation

Category	Sample queries
Music	audio, band, lyrics, singer, sound, music
Travel	hotel, tourism, flight, holiday, voyage, travel
Financial	stock, investment, price, finance, economics
eCommerce	shopping, eCommerce
Mapping	GIS, geography, GPS, mapping, traffic
Sports	NFL, fitness, sport, running, athlete
Photos	photo, image, picture, camera
Government	policy, congress, department, government, law
Game	player, game
Education	education, training, student, school
Enterprise	sales, hr, leader, enterprise
Social	social network, media, social

C. Performance of CrowdAPS

In this part, we use dataset D_p and D_t that consist of description information and list meta-data of APIs to verify effective performance of CrowdAPS based on three metrics.

1) Comparison of PW search & CrowdAPS: In this evaluation, we compare the performance of CrowdAPS and ProgrammableWeb search system (PW search). Here we set $\lambda = 0.6$. As the API collection of PW for searching is larger and includes the dataset of CrowdAPS, we filter out the APIs which have no official Twitter accounts from the returned results by PW for fair. Table IV partially reports the performance of PW search and CrowdAPS in terms of three metrics for 15 queries. It is evident that for most queries, CrowdAPS outperforms PW search. On average, the values of P@10 and nDCG@10 of CrowdAPS are higher than those of PW search, which caused by the significant semantics mined from lists. And for AR@10, the performance of CrowdAPS is better as an API developed by the provider with a higher website rank is probably more popular. Note that provider rank has been normalized and its value is quite small due to the wide range between the maximum and minimum rank.

TABLE IV
COMPARISON OF PW SEARCH AND CROWDAPS FOR 15 QUERIES

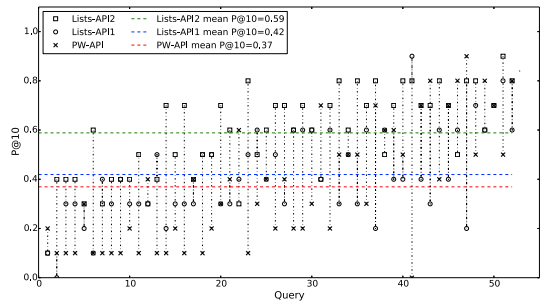
Query	P@10		nDCG@10		AR@10(10^{-2})	
	PW search	Crowd-APS	PW search	Crowd-APS	PW search	Crowd-APS
sales	0.40	**0.70**	0.35	**0.75**	3.27	**0.73**
media	0.10	**0.80**	0.07	**0.75**	7.24	**0.01**
social	**1.00**	0.80	**1.00**	0.75	23.06	**0.01**
audio	0.60	0.60	0.57	**0.72**	9.49	**1.39**
music	0.60	**0.70**	0.59	**0.79**	4.63	**1.98**
hotel	**1.00**	0.90	**1.00**	0.94	3.29	**0.05**
tourism	0.60	**0.80**	0.72	**0.88**	1.43	**0.07**
travel	**0.90**	0.80	0.81	**0.88**	1.63	**0.07**
fitness	0.60	0.60	0.51	**0.69**	7.15	**0.84**
running	0.20	**0.50**	0.13	**0.66**	6.96	**0.14**
image	0.60	**0.70**	0.54	**0.79**	2.32	**0.06**
picture	0.40	**0.60**	0.47	**0.71**	2.94	**0.26**
government	0.40	**0.70**	0.25	**0.63**	16.62	**1.02**
education	**0.70**	0.60	**0.75**	0.74	11.24	**0.74**
school	0.60	**0.70**	0.59	**0.79**	6.85	**0.76**
...	...	...	...	...	...	...
Average	0.55	**0.59**	0.56	**0.61**	5.82	**1.66**

2) Effectiveness of Twitter lists: In order to evaluate the effectiveness of Twitter lists leveraged by CrowdAPS, we apply the algorithm used in CrowdAPS to three scenarios which have different datasets:

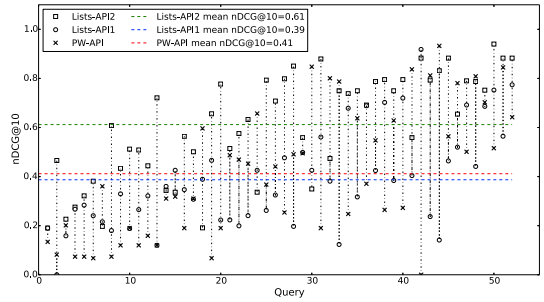
- **PW-API.** In this scenarios, API names and descriptions crawled from PW are utilized to search API based on semantics.
- **Lists-API[1].** We make use of the semantic information provided by lists (name and description) to find API in this scenario. That is the proposed CrowdAPS with $\lambda = 1$ in equation (6).
- **Lists-API[2].** Both semantic and popularity information provided by lists (name, description and number of lists of each API) are adopted for API search in this scenario. Here we set λ as 0.6.

Fig.4 shows the comparison of three scenarios in terms of P@10, nDCG@10 and AR@10. Queries in Fig.4 (a) and (b) have been ordered by the mean of the per-query metric score for the three scenarios. It is observed that both P@10 and nDCG@10 of Lists-API[2] are obviously higher than the other two for most queries, which benefits from the semantic and popularity information offered by lists. Lists-API[1] works better than PW-API in terms of P@10, while mean of nDCG@10 value of PW-API is a litter higher. It indicates that Lists-API[1] obtains a list of result containing more relevant APIs though it performs a little weaker when considering the rank of results. Therefore, the advantage of lists is that (i) the semantics of them facilitates a high proportion of relevant APIs in the former part of the returned results, and (ii) popularity information helps the relevant APIs included in more lists become the top of the ranked results.

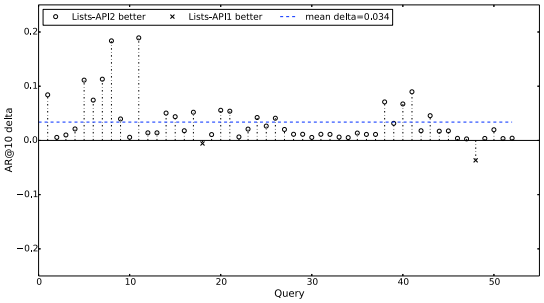
Additionally, we use AR@10 to compare Lists-API[2] with Lists-API[1] based on provider rank so as to assess the performance of popularity provided by lists. In Fig.4 (c), AR@10 delta represents the difference between AR@10 of Lists-API[2] and Lists-API[1]. A positive delta indicates Lists-API[2]



(a)



(b)



(c)

Fig. 4. Performance comparison in terms of (a) P@10, (b) nDCG@10 and (c) AR@10

outperforms Lists-API[1], a negative delta the reverse. It reveals that 50 out of 52 queries have positive delta values and the mean delta is 0.034. Thus, utilizing list meta-data could better meets users' requirements of appropriate and popular APIs.

3) Impact of λ: This part we analyze the impact of parameter λ, the weight to balance the importance of semantic similarity and popularity in CrowdAPS. We compute the average values of three metrics for 52 queries with the change of λ. Fig.5 demonstrates the influence of λ in terms of P@10, nDCG@10 and AR@10. As λ increases, both P@10 and nDCG@10 first go up then go down, and AR@10 first decreases then increases. It means the performance of CrowdAS gets better with the increase of λ, and decreases when λ reaches a certain value. It can be observed that the

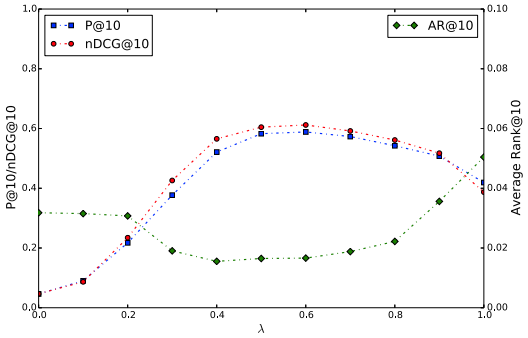


Fig. 5. Impact of λ

best performance is achieved when the value of λ is around the range between 0.5 and 0.6. Furthermore, Fig.5 shows AR@10 achieves the worst performance when only the semantic similarity is considered($\lambda = 1.0$), which demonstrates the importance of popularity information mined from lists.

D. Discussion

CrowdAPS, the proposed API search approach, leverages list meta-data and applies LSI model to calculate the similarity of APIs and the given query. To further improve the performance of API search, other effective topic models (PLSA, LDA, etc.) could be tried to calculate the texts similarity based on the information extracted from lists. The social information in Twitter, such as the relationship of users who interest in APIs, could be used to boost the quality of API search. As lists of APIs contain much information about semantics and popularity, which has been proved in the evaluation of CrowdAPS, it also could be applied to the fields of APIs clustering, API recommendation, etc.

VII. Conclusion

With the rapid growth of APIs, searching appropriate APIs has been an imperative problem. In this paper, we propose a novel approach for API search named CrowdAPS by exploiting Twitter lists, which are carefully managed by individual users to include APIs' official Twitter accounts that interest them. CrowdAPS applies LSI model to mine semantic information of an API from its list name and description, and utilizes the number of its lists to calculate API's popularity. The integration of semantic similarity and popularity leads to the final ranked search result for a given query. To evaluate the performance of CrowdAPS, two real datasets crawled from PW and Twitter are employed, respectively. We compare the overall searching performance of CrowdAPS and PW search system, and evaluate the impact of Twitter *Lists*. The experimental results demonstrate the effectiveness of the proposed CrowdAPS approach.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China under grant of 61173176, Science and Technology Program of Zhejiang Province under grant of 2013C01073, National High-Tech Research and Development Plan of China under Grant No. 2013AA01A604.

References

[1] J. Musser, "Open apis, whats hot, whats not," in *Keynote at Glue Conference*, 2012, pp. 23–24.

[2] S. Ghosh, N. Sharma, F. Benevenuto, N. Ganguly, and K. Gummadi, "Cognos: crowdsourcing search for topic experts in microblogs," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2012, pp. 575–590.

[3] P. Lamere, "Social tagging and music information retrieval," *Journal of New Music Research*, vol. 37, no. 2, pp. 101–114, 2008.

[4] N. Kallen, "Twitter blog: Soon to launch: Lists," 2009.

[5] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation in an evolving service ecosystem," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 25–32.

[6] H. Sun, Z. Zheng, J. Chen, W. Pan, C. Liu, and W. Ma, "Personalized open api recommendation in clouds via item-based collaborative filtering," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE, 2011, pp. 237–244.

[7] W.-K. Chan, H. Cheng, and D. Lo, "Searching connected api subgraph via text phrases," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 10.

[8] B. Cao, J. Liu, M. Tang, Z. Zheng, and G. Wang, "Mashup service recommendation based on user interest and social network," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 99–106.

[9] M. Dojchinovski, J. Kuchar, T. Vitvar, and M. Zaremba, "Personalised graph-based selection of web apis," in *The Semantic Web–ISWC 2012*. Springer, 2012, pp. 34–48.

[10] W. Zheng, Q. Zhang, and M. Lyu, "Cross-library api recommendation using web search engines," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 480–483.

[11] Y. Zhang, Z. Zheng, and M. R. Lyu, "Wsexpress: A qos-aware search engine for web services," in *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 91–98.

[12] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering web services to facilitate service discovery," *Knowledge and information systems*, vol. 38, no. 1, pp. 207–229, 2014.

[13] L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng, "Wt-cluster: Utilizing tags for web services clustering," in *Service-Oriented Computing*. Springer, 2011, pp. 204–218.

[14] K. Gomadam, A. Ranabahu, M. Nagarajan, A. P. Sheth, and K. Verma, "A faceted classification based approach to search and rank web apis," in *Web Services, 2008. ICWS'08. IEEE International Conference on*. IEEE, 2008, pp. 177–184.

[15] R. Torres, B. Tapia, and H. Astudillo, "Improving web api discovery by leveraging social information," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 744–745.

[16] Y.-J. Lee and J.-S. Kim, "Automatic web api composition for semantic data mashups," in *Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on*. IEEE, 2012, pp. 953–957.

[17] B. Tapia, R. Torres, and H. Astudillo, "Simplifying mashup component selection with a combined similarity-and social-based technique," in *Proceedings of the 5th International Workshop on Web APIs and Service Mashups*. ACM, 2011, p. 8.

[18] D. Bianchini, V. De Antonellis, and M. Melchiori, "A multi-perspective framework for web api search in enterprise mashup design," in *Advanced Information Systems Engineering*. Springer, 2013, pp. 353–368.

[19] R. Pochampally and V. Varma, "User context as a source of topic retrieval in twitter," in *Workshop on Enriching Information Retrieval (with ACM SIGIR)*, 2011, pp. 1–3.

[20] B. Rosario, "Latent semantic indexing: An overview," *Techn. rep. INFOSYS*, vol. 240, 2000.