Harnessing Twitter to Support Serendipitous Learning of Developers

Abhishek Sharma[1], Yuan Tian[1], Agus Sulistya[1], David Lo[1], and Aiko Fallas Yamashita[2]
[1]School of Information Systems, Singapore Management University
[2]Oslo and Akershus University College of Applied Sciences
Email: {abhisheksh.2014,yuan.tian.2012,aguss.2014,davidlo}@smu.edu.sg, Aiko.Yamashita@hioa.no

***Abstract*—Developers often rely on various online resources, such as blogs, to keep themselves up-to-date with the fast pace at which software technologies are evolving. Singer et al. found that developers tend to use channels such as Twitter to keep themselves updated and support learning, often in an undirected or serendipitous way, coming across things that they may not apply presently, but which should be helpful in supporting their developer activities in future. However, identifying relevant and useful articles among the millions of pieces of information shared on Twitter is a non-trivial task. In this work to support serendipitous discovery of relevant and informative resources to support developer learning, we propose an unsupervised and a supervised approach to find and rank URLs (which point to web resources) harvested from Twitter based on their informativeness and relevance to a domain of interest. We propose 14 features to characterize each URL by considering contents of webpage pointed by it, contents and popularity of tweets mentioning it, and the popularity of users who shared the URL on Twitter. The results of our experiments on tweets generated by a set of 85,171 users over a one-month period highlight that our proposed unsupervised and supervised approaches can achieve a reasonably high Normalized Discounted Cumulative Gain (NDCG) score of 0.719 and 0.832 respectively.**

***Index Terms*—Online Resources; Recommendation System; Social Media for Software Engineering**

I. Introduction

Software development is a field which evolves rapidly, so software developers always need to keep themselves up to date with new knowledge and methodologies. Learning continuously and serendipitously may help them to solve new, unseen and/or complex challenges that they may encounter during their software development tasks. Storey et al. found that *keeping up with new technologies* is a major challenge faced by software developers today [19]. They also found that developers use media such as Twitter to keep them up to date with the latest trends and to extend their software knowledge [18].

In this work, we present an approach to support the serendipitous learning of developers by harnessing Twitter as a knowledge repository. Past research has shown that Twitter is used by software developers to share important information with other fellow developers [3], [18], [20]. Sharing links in the form of URLs (Uniform Resource Locators) of various software related articles and multimedia is a popular activity in software engineering Twitter space [17]. Twitter has been found to be better at serendipitously exposing developers to latest updates and developments in technology when compared to search engines [18]. Also, consideration of the URLs on Twitter allows us to reduce the search space for finding popular and relevant URLs, and also to infer the social approval of links shared. Unfortunately, even on Twitter, finding URLs to relevant and useful articles for a particular domain of interest (e.g., Java) is not an easy task. Developers need to identify many relevant Twitter users to follow, and sieve through a large amount of tweets that they may generate, which often result in information overload. These challenges have been validated by Singer et al. in their survey with developers [18].

To address the above mentioned challenges, we propose an unsupervised and a supervised approach to harvest and rank URLs linked to contents that are popular and relevant to a particular domain of interest from Twitter. Both output a sorted list of URLs sorted based on their likelihood to be popular and relevant to the domain of interest, where domain is characterized by a set of keywords (e.g., {"Java"}). The supervised approach also requires as an input a small training set, which contains URLs that are manually assigned with relevance ratings ranging from 0 (highly irrelevant) to 3 (highly relevant). Both of the two approaches characterize a URL in terms of 14 features that are grouped into three families: content features, popularity features, and network features. Our unsupervised approach makes use of Borda count [1], a popular data fusion technique, to rank URLs based on their features. Our supervised approach makes use of *Learning to Rank* [8], a popular information retrieval technique, to build a ranking model from the labeled URLs, which can then be applied to rank a set of URLs based on their likelihood to be informative.

In this preliminary study, we evaluate the two proposed approaches on a dataset of 577 unique URLs found among 2,104 tweets posted by people potentially interested in software development. These 2,104 tweets were filtered from about 3,980,397 tweets posted in Novemeber 2015 based on the condition that they contain the keyword "Java". We measure the effectiveness of our approaches in ranking these URLs in terms of Normalized Discounted Cumulative Gain (NDCG) [8]. NDCG scores are computed based on the relevance ranks of the URLs which were manually labeled by two study participants. The participants label the data independently and then resolve their differences in order to create the final ground truth.

The contributions of this paper are as follows:

1) We propose an unsupervised and supervised approach to support developer serendipitious learning using Twitter by ranking URLs to online resources. To the best of our knowledge, no prior study has helped developers in this task. Our approaches sieve through a large number of tweets to automatically extract and rank URLs relevant to a particular domain of interest. Our preliminary evaluation shows that they can achieve reasonably high Normalized Discounted Cumulative Gain (NDCG) scores on a dataset of 577 URLs related to the keyword 'Java'.
2) We propose 14 features from three categories, i.e., content features, popularity features, and network features, to comprehensively characterize a URL given a set of keywords describing a domain of interest.

The structure of the remainder of this paper is as follows. In Section II, we describe our proposed approach that extracts and ranks informative URLs from Twitter. In Section III, we present our experiment settings and results. Related work is presented in Section IV. We finally conclude and mention future work in Section V.

II. Approach

Our approach has four steps, i.e., Data Acquisition, Feature Extraction, Unsupervised Recommendation and Supervised Recommendation, as shown in Figure 1.

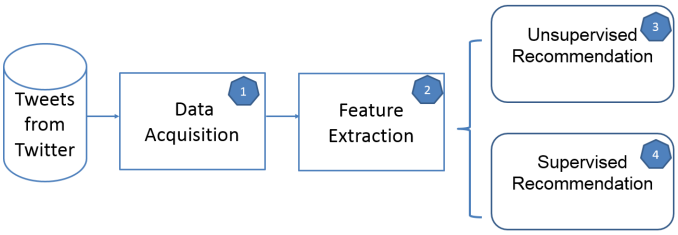

Fig. 1: Approach Overview

Data Acquisition. We first identify some users on Twitter who are potentially interested in software development. We start with a set of well known software developers who are also present on Twitter. We then process the profile of these seed users to find all the other users who follow or are followed by at least n of these seed users. The approach has been used in several previous works [16], [17], [21]. We then download and process the tweets of these identified Twitter users on a period of time, filtering tweets using keywords that characterize a domain of interest. Next, we extract URLs shared in these tweets. These URLs are typically shortened by Twitter itself or by users using a URL shortening service, e.g., https://goo.gl/. If a URL has been shortened by Twitter, it maintains a reference of the expanded URL in the tweet's meta data. In case the Twitter user had used an external service to shorten the URL, we use a browser to expand the short URLs to their expanded forms. Then we remove the duplicates among expanded URLs. We also remove URLs which correspond to broken links and error pages. In the end, we have a set of valid URLs along with the other associated information such as tweet content and user data.

Feature Extraction. We extract features which help us to find useful URLs w.r.t. a particular domain of interest represented by a set of keywords. We have categorized the features into three broad categories: *Content Features*, *Popularity Features*, and *Network Features*. These are explained below.

Content Features. These features are based on the similarity between the input keywords, which characterize the domain of interest, and various textual contents that are linked to a URL.

- CosSimT: This feature corresponds to the cosine similarity between the keywords related to our domain of interest and the combined text of all the tweets which mention a particular URL.
- CosSimW: Through this feature, we measure the cosine similarity score between the keywords and the text contents on the webpage which a URL link resolves to.
- CosSimP: This feature measures the cosine similarity between the input keywords and the combined text from all the profile data of users who posted a particular URL.

Popularity Features. These features measure the popularity of a URL Link.

- NumOfT: This feature counts the number of tweets or retweets generated by a community of software enthusiasts on Twitter (i.e., users tracked in the data acquisition step) which contain a particular URL.
- NumOfU: This feature counts the number of unique users in a community of software enthusiasts who have shared a particular URL in their tweets. This feature differs from *NumOfT* feature as a user may post the same URL link in multiple tweets. For calculation of *NumOfU* we only consider a user once.
- NumOfRT: This feature counts the sum of the retweet counts of all the original tweets that contain the URL link.
- NumOfF: This feature counts the sum of the favourite counts of all the tweets and retweets that contain the URL link.

Network Features. We take the network of all Twitter users in our dataset who have posted at least a tweet containing the domain related URL and then infer the network importance of each user present, considering each user as a network node. To measure the importance of user, we use popular centrality metrics proposed in web and social network mining communities [4], [5], [9], [24]. We compute the features by using Jung (http://jung.sourceforge.net/). We provide a brief description below. (For a complete description please refer [23]).

- Barycenter Centrality: This feature is computed by taking the reciprocal of the sum of shortest distance of a node to each other node in a network.
- Betweenness Centrality: This feature counts the number of shortest paths from all nodes to all others that pass through a node.
- Closeness Centrality: This feature is computed by taking the reciprocal of the average shortest distance of a node to all the other nodes in a network.

- Eigenvector Centrality: This feature measures the importance of a node based on the importance of its neighboring nodes. The values of eigenvector centrality for nodes in the network is computed as follows:

$$\alpha(I - \beta R)^{-1} R1$$

In the above equation, α is a scaling vector for normalizing the score, I is the identity matrix, R is the adjacency matrix representing the network, β is the weighting factor for the adjacency matrix, and $R1$ is a matrix where the contents of all its cells are ones. Since the value of this metric is often very small, in this work we compute the reciprocal of this metric. We use the default values of α and β in Jung.
- Hubs and Authorities: Hubs and Authorities are two scores to measure node importance in network. They are computed based on the Hyperlink-Induced Topic Search (HITS) algorithm proposed by Kleinberg [7]. These two scores of a Twitter user u are computed as follows:

$$Hub(u) = \sum_{i=1}^{n} Auth(u),$$

$$Auth(u) = \sum_{i=1}^{n} Hub(u)$$

In the equation, n is the total number of users in a network, $Hub(u)$ computes the hub score for node u, and $Auth(u)$ computes the authority score for node u.
- PageRank: PageRank (PR) is a node importance measurement proposed by Brin and Page [9]. The PR algorithm computes a probability to represent the likelihood of a particular node being visited while randomly traversing edges.

Unsupervised Recommendation. Based on the 14 feature scores, we use Borda Count [1] to arrive at a combined score for a URL and then rank the URLs based on this combined score.

Borda Count works by first assigning a *rank* for each feature score to a URL. For each feature score, we create a list of all URLs that we have harvested in the data acquisition step, and sort them in descending order of their feature scores. The *rank* of a URL for a feature is then defined as the position of the URL in the sorted list. Next, for each feature score, after we have the rank of a URL, we can compute its *ranking point*. It is calculated by subtracting the *rank* of the URL from the total number of URLs. After we have the *ranks* and *ranking points* for all URLs and features, we can compute the URL's combined score. Let u_i denotes the i^{th} URL and $rp_j(u_i)$ denotes the ranking point assigned to u_i for the j^{th} feature. Also, let N_f denotes the number of feature scores per URL and N_u denotes the total number of URLs in our data set. The combined score of a URL u_i can be calculated as follows:

$$BordaScore(u_i) = \frac{\sum_{j=1}^{N_u}(rp_j(u_i))}{N_f \times N_u}$$

In the above equation, the combined score is the summation of all the *ranking points* divided by the product of N_f and N_u. After obtaining the combined score, we rank the URLs in the descending order of their combined scores. The URL having the highest combined feature score is considered the most relevant, and the URL having the lowest score is considered as the most irrelevant.

Supervised Recommendation. We use *Learning to Rank* [8] approach to train a supervised model which is then used to assign ranks to URLs. In the learning phase of our supervised approach, we consider a set of URLs as training data and based on the feature scores of these URLs and their corresponding manually assigned labels, we learn a ranking function $f(u)$. This function $f(u)$ can be considered as the weighted sum of all the features of a URL u, and during the learning phase it tries to learn these weights or parameters of the features through optimization. This ranking function when applied to unseen test URLs (also represented as their corresponding feature vectors) assigns scores to the URLs. Based on the scores provided by $f(u)$, all the test URLs can be ranked in the descending order. This sorted list is considered as the recommended result. In this work, we make use of a popular off-the-shelf implementation of a learning to rank algorithm, SVM^{rank}, which is made available from https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html.

III. Preliminary Experiments and Results

In this section, we first present the process of creating ground truth set. Next, we describe our experiment setting and evaluation metric. Finally, we present our research questions and the results of our experiments which answer the questions.

A. Dataset

In the data acquisition step, as the seed set of Twitter users, we use a list of top 100 popular developers on Twitter given in: http://noop.nl/2009/02/twitter-top-100-for-software-developers.html. We set n as 5 (i.e., we find all other users who follow or are followed by at least 5 of these seed users). Moreover, we collect tweets made on November 2015, and filter tweets using keyword "*Java*". We are able to extract 2,104 of such tweets and 577 unique and valid URLs along with their associated information. More URLs could be gathered if we expand our Twitter user base and the period of time the tweets were made. We leave the gathering of an extended dataset for a more comprehensive evaluation as future work.

Next, we manually assign relevance score labels on a scale of 0 to 3 for each of the selected 577 unique URLs we extracted. The data is labeled by 2 persons, both having more than 4 years of professional programming experience in Java. The labellers are provided with the 577 URLs and asked to browse the websites pointed to by the URLs and then have to assign a score to the URL, with a score of 3 being assigned if the content linked with the URL is highly relevant and shareable, 2 being assigned if the content is relevant but not worth sharing, 1 being assigned if URL content was marginally

relevant and not shareable, and 0 being assigned if the content is highly irrelevant. For the URLs where the two labellers have a disagreement, they have to sit down together to discuss and agree to a final label. Table I shows the distribution of the labels for the 577 URLs.

TABLE I: Distribution of Scores for the 577 URLs

Label Assigned	0	1	2	3	Total
#URLs	115	77	184	201	577

B. Experiment Setting and Evaluation Metrics

By default, we perform 10-fold cross validation to investigate the effectiveness of our approach. As an evaluation metric, we make use of Normalized Discounted Cumulative Gain (NDCG). NDCG measures the performance of a recommendation system by evaluating its capability to recommend more relevant URLs as the top results and less relevant ones as the bottom results. NDCG gives a score between 0 and 1 to the recommender system it evaluates. The closer the NDCG score of a system is to 1, the more effective it is at recommending informative URLs. We use the following formula to calculate NDCG:

$$NDCG = \frac{DCG - WDCG}{IDCG - WDCG}$$

In the above formula, DCG is a Discounted Cumulative Gain score [8] of the URL relevance, IDCG is the ideal DCG score (i.e., informative URLs are listed before less informative ones), and WDCG is the worst DCG score (i.e., all less informative URLs are listed before more informative ones). The following equation is used to compute DCG, where rel_i is the rating assessment for the URL at position i in the ranking:

$$DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log(i)}$$

The main concept of DCG is that relevant documents (in our case, relevant URLs) appearing lower in a search result list corresponds to a poorer result.

C. Research Questions

RQ1: How effective are our unsupervised and supervised approaches in recommending informative URLs?

In this research question, we investigate the effectiveness of our two approaches based on the NDCG metric. Table II shows the NDCG scores of our approaches. The NDCG score for the unsupervised approach is 0.719 while that for the supervised approach is 0.832. The supervised approach can outperform the unsupervised one by 15.71%. Table III shows some examples of URLs that are recommended by our approach.

TABLE II: NDCG Scores of Our Proposed Approaches

Approach	**NDCG Score**
Unsupervised	0.719
Supervised	0.832

TABLE III: Some Examples of Recommended URLs

URL
www.infoq.com/articles/Java-The-Missing-Features
http://github.com/zeroturnaround/java-fundamentals
www.adam-bien.com/roller/abien/entry/java_8_infinite_stream_of

RQ2: How sensitive is our supervised approach on the amount of training data?

In this research question, we investigate the impact of reducing the amount of training data on the effectiveness of our supervised approach by performing k-fold cross validation and varying the value of k from 2 to 10. From Table IV, we can see that the performance of our supervised approach remains stable across various values of k. and is not overly sensitive.

TABLE IV: NDCG Scores for Different k

k	**NDCG**	**k**	**NDCG**	**k**	**NDCG**
10	0.832	7	0.845	4	0.837
9	0.825	6	0.834	3	0.847
8	0.833	5	0.842	2	0.843

D. Threats to Validity

Threats to *internal validity* refer to experimenter biases. We have tried to mitigate this threat by asking two persons to independently rate the relevance of the webpages pointed to by the URLs, and later meet to resolve their disagreements. Threats to *external validity* refer to the generalizability of our findings. For this preliminary work, we have considered one domain of interest, namely Java programming language, using the keyword "Java" to characterize this domain. In the future, we plan to reduce this threat further by considering other domains and/or keywords in addition to Java domain. Threats to *construct validity* correspond to the suitability of our evaluation metric. In this work, we make use of Normalized Discounted Cumulative Gain (NDCG) which is a standard information retrieval metric and has also been used in many past software engineering studies, e.g., [6], [15]. Therefore, we believe that threat to construct validity is minimal.

IV. Related Work

In this section, we present studies that analyze Twitter data from a software engineering perspective and studies on recommendation systems for software engineering (RSSE). Due to page limitation, the survey here is by no means complete.

Twitter and Software Engineering. Singer et al. did a survey about 271 developers from GitHub and found that Twitter is used by developers to keep them updated with the latest developments in software engineering field [18]. Other studies have also reported that Twitter is utilized by software developers for coordination of efforts, sharing of knowledge, etc. [3], [21], [22]. Tian et al. have done a manual categorization of 300 tweets into 10 groups, which are commercial, news, tools and code, question & answer, events, personal, opinion, tips, job, and miscellaneous [20]. Several techniques to filter software

relevant tweets have also been proposed [12], [16]. Different from the above mentioned work, we propose an approach that support developer serendipitious learning using Twitter by ranking URLs to online resources.

Recommender Systems for Software Engineering. Bacchelli et al. and Ponzanelli et al. have developed Eclipse plugins to automatically recommend Stack Overflow posts based on the context of the code being edited in the IDE [2], [10], [11]. Rahman et al. have proposed context based recommendation and search engines for searching programming errors and exceptions [13], [14] For a detailed description of various types of recommender systems for software engineering please refer to the RSSE book [15]. Our work is different from these and many other RSSE works as we build a recommender system that returns a list of URLs based on a set of keywords characterizing a domain of interest by leveraging crowd knowledge shared via Twitter.

V. Conclusion and Future Work

Software developers using channels such as Twitter serendipitously learn about new methodologies and keep their skills and knowledge up to date. Unfortunately, given the huge number of choices developers have at their disposal, identifying which resources and channels to follow and what to ignore is a major challenge for them [19].

We propose two approaches, one unsupervised and one supervised, to search and rank URLs harvested from Twitter which can support developers in their serendipitous learning tasks. These approaches are based on 14 features which characterize a URL's relevance and informativeness from three dimensions: 1) *content* features which capture similarity of the input domain specific keyword with the textual contents of tweets, webpages pointed to by the URLs, and user profiles, 2) *popularity* features which characterize the popularity of the tweets containing the URL on Twitter, 3) *network* features which characterize the importance of the user posting the URL on Twitter. In our preliminary experiments, we evaluate the two approaches on a set of 577 URLs. The experiments show that our unsupervised and supervised approaches can achieve a reasonably high Normalized Discounted Cumulative Gain (NDCG) score of 0.719 and 0.832 respectively.

As a future work, we plan to improve the effectiveness of our approach further by the incorporation of additional features and the design of more sophisticated algorithms. We would also like to enlarge the scale of our experiments to consider more tweets collected over a longer period of time and also to add more channels to mine URLs. Moreover, we plan to build a site that shares URLs of informative resources that are harvested by our proposed approach and gets continuously updated in real time.

Acknowledgement. This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

References

[1] J. A. Aslam and M. Montague, "Models for metasearch," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2001, pp. 276–284.

[2] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, 2012, pp. 26–30.

[3] G. Bougie, J. Starke, M.-A. Storey, and D. M. German, "Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions," in *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, 2011.

[4] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008.

[5] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1979.

[6] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, 2011, pp. 323–332.

[7] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[8] T. Liu, *Learning to Rank for Information Retrieval*. Springer, 2011.

[9] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." *Technical Report, Stanford InfoLab*, 1998.

[10] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1295–1298.

[11] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 102–111.

[12] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E.-P. Lim, "Automatic classification of software related microblogs," in *ICSM*, 2012.

[13] M. M. Rahman, S. Yeasmin, and C. K. Roy, "Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 2014, pp. 194–203.

[14] M. M. Rahman and C. K. Roy, "Surfclipse: Context-aware meta-search in the ide," in *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 617–620.

[15] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds., *Recommendation Systems in Software Engineering*. Springer, 2014.

[16] A. Sharma, Y. Tian, and D. Lo, "Nirmal: Automatic identification of software relevant tweets leveraging language model," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 449–458.

[17] ——, "What's hot in software engineering twitter space?" in *Software Maintenance and Evolution (ICSME),*. IEEE, 2015, pp. 541–545.

[18] L. Singer, F. M. F. Filho, and M. D. Storey, "Software engineering at the speed of light: how developers stay current using twitter," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, 2014, pp. 211–221.

[19] M.-A. Storey, A. Zagalsky, L. Singer, D. German *et al.*, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Transactions on Software Engineering*, 2016.

[20] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim, "What does software engineering community microblog about?" in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 247–250.

[21] Y. Tian and D. Lo, "An exploratory study on software microblogger behaviors," in *MUD*, 2014.

[22] X. Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald, "Microblogging in open source software development: The case of drupal and twitter," *Software, IEEE*, 2013.

[23] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press.

[24] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003.