PocketView: A Concise and Informative Data Summarizer

Yihai Xi[1,2], Ning Wang[1,2*], Shuang Hao[1,2], Wenyang Yang[1,2], Li Li[1,2]
[1]*School of Computer and Information Technology, Beijing Jiaotong University, China*
[2]*Beijing Key Laboratory of Traffic Data Analysis and Mining, China*
{xiyihai, nwang, haoshuang, yangwy, bjtulili}@bjtu.edu.cn

***Abstract*—A data summarization for the large table can be of great help, which provides a concise and informative overview and assists the user to quickly figure out the subject of the data. However, a high quality summarization needs to have two desirable properties: presenting notable entities and achieving broad domain coverage. In this demonstration, we propose a summarizer system called `PocketView` that is able to create a data summarization through a pocket view of the table. The attendees will experience the following features of our system: (1) time-sensitive notability evaluation - `PocketView` can automatically identify notable entities according to their significance and popularity in user-defined time period; (2) broad-coverage pocket view - Our system will provide a pocket view for the table without losing any domain, which is much simpler and clearer for attendees to figure out the subject compared with the original table.**

***Index Terms*—data summarization, pocket view, notability, domain coverage**

I. Introduction

There are lots of semantic recovery or tagging tasks in crowdsourcing platforms (e.g., annotating column labels). Due to the limitation of crowdsourcing tasks, requesters are recommended to reduce the table size. The most common way is random selection, as used in [1]. It mainly suffers from two limitations. On the one hand, random selection neglects the notability of different entities. Given a large table, it would be hard for workers to undertake a task in face of the unfamiliar data. On the other hand, it does not consider the coverage of domain information. The random selection may lose lots of domains covered by the original table and cause misunderstanding. In contrast, a high-quality data summarization would pick out notable entities and be more representative for covering most of domains, which will contribute to the semantic understanding of the table for crowd workers.

We capture the quality of a data summarization through the notions of notability and coverage. Notability refers to the property of an entity in the data summarization to be of a high degree of popularity and significance. Wikipedia uses the importance scale that contains four levels 'Top', 'High', 'Mid', 'Low' to describe the importance of an entity, which is manually evaluated by project watchers. [2] proposed a notability determination system by training a Boolean classifier, which is a coarse-grained evaluation where each entity finally get a yes/no category label. However, we need a fine-grained notability ranking algorithm in order to create a high quality summarization. Coverage measures the amount of different domains covered in the data summarization. A dataset usually has a topic but cover many domains. For example, a *Company* table may contain *Internet*, *Media*, *Retail* domains, etc. And an entity may cover more than one domain such as *Samsung* is a magnate that covers *Electronics*, *RealEstate*, *Chemical*, etc. The domain of each entity can be obtained by existing knowledge bases (KBs) or question answering system. Given a table T where each tuple refers to some facts of an entity and a size limitation n, we try to create a concise and informative summarization T^s consisting of n entities in T, which covers all domains and achieves maximal notability.

Challenges. Although important, generating a high quality summarization is *not* an easy task. Firstly, it is challenging to evaluate the notability of each entity in a fine-grained manner, which is sensitive to time. Secondly, taking both notability and coverage into consideration is hard since they are independent targets *i.e.,* notable entities in the table may focus on a small number of domains while broad coverage entities may not be notable.

Our Methodology. Inspired by Google PageRank [3], we observe that notable entities get more links from others in Wikipedia pages. Consequently, we first utilize the Name Entity Linking technology [4] to map the entities in T to the articles in Wikipedia, and then calculate the notability of each entity iteratively. In addition, the entity notability is also related to its page views in a certain period of time. We combine the factor to propose a time-sensitive notability evaluation algorithm. With the notability of each entity, we model our problem into a weighted bipartite graph. Each vertex in the left part represents an entity in T and the vertex weight is its notability. Each vertex in the right part represents a domain under the topic of T. If an entity covers a domain, we add an edge between them. Then we simplify the data summarization problem to the maximum weight set N-covering problem (MWSNCP), which is to find a set of entities with size not larger than n that cover all domains and have maximum total vertex weight. Since it can be proved to be NP-hard by a variant of the set covering problem, we propose an efficient heuristic algorithm to find the vertex set.

Related Work. Previous work mostly focuses on schema summarization [5], which produces an overview of a schema with representative elements. In contrast, our goal is to create

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00159

IEEE computer society

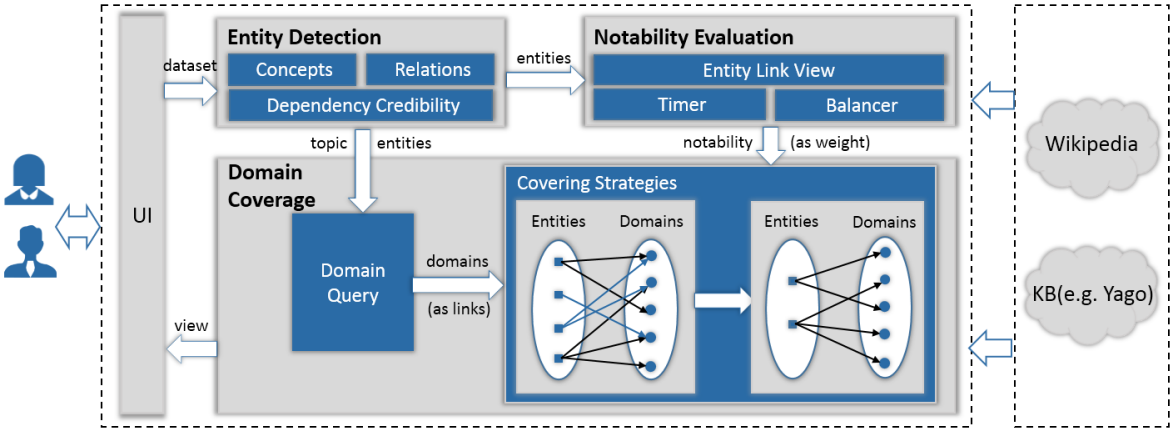


Fig. 1. The Architecture of PocketView

a summarization of instances in a table instead of schema elements. In addition, [6] proposed a K-means algorithm to cluster similar tuples in a table based on Euclidean distance and prompt workers with tuples that are nearest to each of k cluster centers. However, it just shrinks the table while those selected tuples may not be notable. Thus, it can not help to quickly understand the table.

We develop a system called PocketView that helps us easily create a high quality pocket view of a large table. It has the following features: (1) PocketView can automatically identify the primary entity column without the user's guidance; (2) PocketView is equipped with a time-sensitive evaluation module that takes the popularity factor into consideration to pick out notable entities exactly; (3) PocketView will return us a concise and informative summarization of the target table consisting of notable entities and covering all domains.

II. System Overview

Fig. I shows the building blocks of PocketView. It consists of three core modules.

Entity Detection Module. This module is to identify the primary entity column of the table. Intuitively, the primary entity column is the one that can determine the values of most other columns. Thus, after obtaining the candidate concepts of each column and the candidate relations between two columns with the help of knowledge bases, this module utilizes the Pointwise Mutual Information [7] to prune some incoherence concepts and relations, and further calculate the dependency strength between each pair of related columns based on the Probase [8], a probabilistic knowledge base that contains a large number of concepts, entities, isA relationships, etc. The column that has the highest dependency strength is the primary entity column.

Notability Evaluation Module. A time-sensitive notability evaluation algorithm is integrated in this module. It consists of three components: EntityLinkView (ELV), Balancer and Timer. ELV maps entities to Wikipedia articles and constructs a directed graph where each vertex is an entity and each edge means that two entities have a hyperlink in Wikipedia pages. The entity links may be sparse which will affect the performance of our algorithm. Therefore, Balancer provides two functions Expander and Classifier to avoid this problem. Expander aims to find some other related entities from Wikipedia but not in T to add more links in the graph, which will be treated as virtual vertices. Classifier is to train a classifier model to identify the entities that are obviously innominate and remove the corresponding vertices. Timer is designed to obtain the page views of each entity within a certain period. More details will be given in Section IV-A.

Domain Coverage Module. This module takes on two jobs. The first one is to obtain the domains of each entity with SPARQL queries. The second one is to build a weighted bipartite graph to represent the notability information of each entity and the coverage relationship between entities and domains, and then find a solution to the MWSNCP problem in graph model to return a pocket view that consists of notable and broad-coverage entities. More details can be found in Section IV-B.

III. Demonstration Overview

Next, we will show how one can easily use PocketView to generate a concise and informative data summarization.

Datasets Specification. The user can upload a relational table by clicking the "UPLOAD" button in Fig 2. No label like the table header is needed. We have also prepared a real-world table fetched from Wikipedia, which contains 229 nameless or famous companies and covers most of domains under the topic of company.

Pocket View Generation. PocketView will begin to work with the default settings after clicking the "POCKET" button in Fig 2. The user can also manually set the running parameters in the setting bar, such as "View Scale" to control the size of summarization T^s, "Time Sensitivity Weight" to control the weight of time factor in measuring notability and also "Start" and "End" to set a certain time period for evaluating entity notability. Fig. 2 presents a pocket view of our prepared table, composed of 15 representative entities. Compared with the entire dataset as the background in Fig. 2, our pocket view

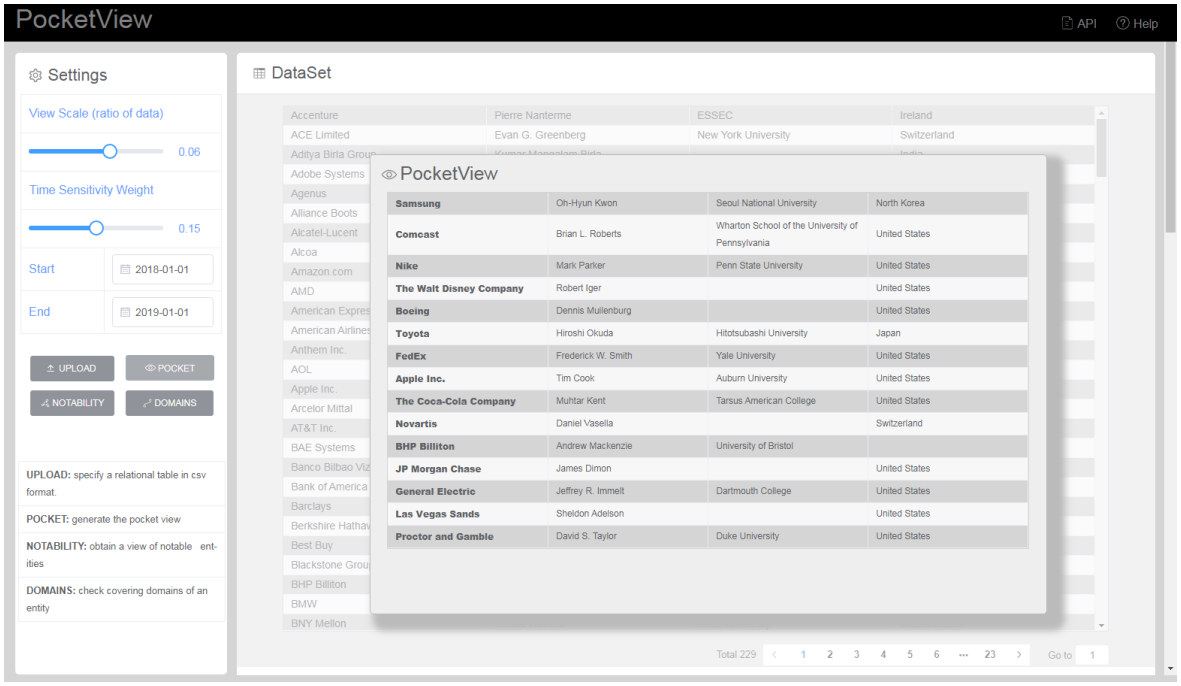


Fig. 2. A running example of `PocketView`

only display notable entities while not losing any domain. Furthermore, it also uses the boldface to highlight the primary entity column.

`PocketView` can also provide some data information to tell stories of a table, which includes

Notability View. During the workflow of creating a pocket view, our system will also provide the user a view of the notability information as shown in Fig. 3(a) by clicking the "NOTABILITY" button. The user can easily pick out the notable entities in it since the size of the vertex is proportional to its notability. He/She can also check the links of an entity as shown in Fig. 3(b) after selecting it with the mouse.

Domain View. The user can easily check the covering domains of an entity in `PocketView` by clicking the "DOMAINS" button. The domain coverage of the 15 representative entities in the pocket view is shown in Fig. 3(c) where domains involved are distributed on both sides of the entities. Besides, the coverage rate is provided in the bottom right, and the detected topics of the table is given in the top right.

IV. Core Techniques

In this section, we will introduce some technical details of `PocketView` including the time-sensitive notability evaluation and the solution to domain coverage.

A. Time-sensitive Notability Evaluation

The evaluation of notability may not perform well if we only consider the entity links since the popularity of some entities may vary greatly in different periods. To solve this problem, we propose a time-sensitive notability evaluation algorithm. Given the ELV, the significance of an entity depends on its connectivity in the graph. Recall that popularity is sensitive to time. Then with a time period t, the notability of an entity e_i, denoted as $NB\left(e_i, t\right)$, could be calculated using the following iterative formula:

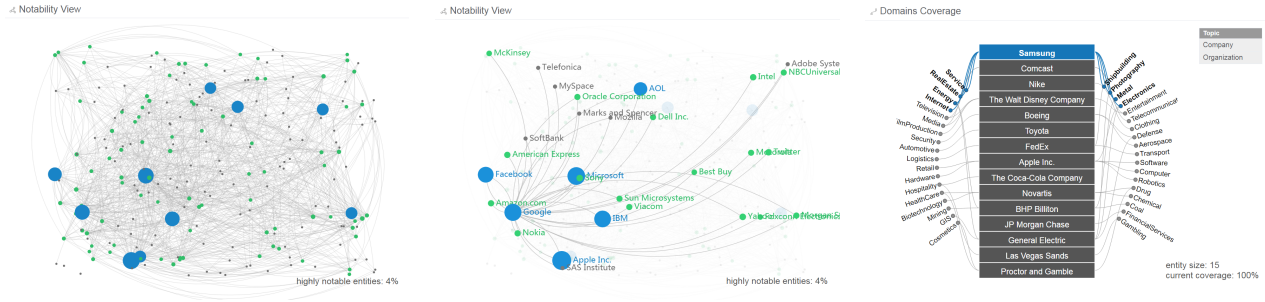
$$NB^{k}\left(e_i, t\right) = (1-\alpha) \cdot \sum_{e_j \in I_{e_i}} \frac{NB^{k-1}\left(e_j, t\right)}{O(e_j)} + \alpha \cdot TW\left(e_i, t\right)$$

where k denotes the number of iteration; I_{e_i} denotes the set of inlinks to e_i; $O(e_j)$ denotes the number of outlinks from e_j; $TW\left(e_i, t\right)$ denotes the popularity degree of e_i within period t; and α is a time damping factor. The larger α is, the more sensitive that notability would be affected by t.

B. A Heuristic Solution to Domain Coverage

Recall that we model our data summarization problem into a weighted bipartite graph model $G(U, V, E)$ where U, V denote the vertex set of entities and domains respectively and an edge $(u, v) \in E$ means that v is one of the domains of entity u. When the weight $w(u)$ denotes u's notability, our problem is to find a subset of U with no more than n items that can cover all vertices in V and has the maximum total vertex weight (MWSNCP problem).

We denote the minimal number of entities that can cover V as min_e. Many algorithms such as [9] have been proposed to calculate min_e and we just use them as the tool. In this paper, we focus on the situation where $n \geq min_e$, which means that there certainly exist a n-size subset of U that can cover all vertices in V. A simple but time consuming method is enumerating all n-size vertex set from U and finding out the one with maximum weight. During the process, some effective pruning techniques can be utilized to reduce the search space. For example, assume that the current optimal result is T_n with



(a) (b) (c)

Fig. 3. Notability View and Domain View

n items covering all vertices in V, and the next candidate path is S_m with m items while $m < n$. We can prune S_m if there not exist a subset $D_{(n-m)}$ in $U - S_m$ that can cover the left vertices in V or the total weight of $(S_m + D_{n-m})$ can not exceed T_n's weight.

Since the MWSNCP problem is NP-hard, we also propose a greedy algorithm to improve the response speed of our system. The basic idea is, if n is large enough, we prefer the entity in each step whose weight is high since it is easily to cover all vertices in V. On the contrary, if n is close to min_e, we prefer the entity whose coverage is broad since the coverage requirement becomes hard to meet.

Specifically, when n is large enough, we first change the vertex weight from the notability of an entity to the potential loss of notability from its neighbour set, which means if an entity u is selected, the probability of choosing the entities that share at least one domain with u will decrease for their coverings have lost part of effect. Then we can switch our problem to the existing MWSCP problem to obtain the approximate optimal result, which is to cover all domains with no vertex size limitation while achieving minimal total vertex weight. Thus, we propose a loss function for each entity to calculate the new weight. After the switch, we could get our preliminary result L_m, which has covered all domains.

The next work is to ensure the size limitation of L_m. If $m < n$, we continue to add vertices to L_m until $m = n$. Since all domains have been covered, the vertex in $U - L_m$ whose notability is highest will be the first choice in each step. However, if $m > n$, we have to reduce $m - n$ items from L_m while not losing any domain. There are two ways of reducing the size: replacing or cutting. Replacing is the strategy that replaces a set of vertices (size > 1) in L_m with one in $U - L_m$ while keeping the same coverage. Each replacement will be evaluated by the average loss of losing the weight of one vertex. Note not every vertex in $U - L_m$ can be a candidate, it must contains more than one child vertex in L_m (child vertex of u: whose coverage is the subset of u). Recall that the MWSCP problem does not consider the size limitation, it means there may exist unnecessary vertices in L_m, whose coverage is not unique. These vertices can be cut safely but may lose much weight. Thus, we need to make a trade-off between replacing and cutting.

We compare the average loss of each case and choose the smallest one at each step until the size of L_m decreases to n. If there are no vertices to cut or replace while the size is still larger than n, we can take the randomized basin hopping strategy to get out of the deadlock. The worst case is to use the result set of the MSCP problem to replace the same number of vertices in L_m and then return to the cutting step. It is the worst because the MSCP problem will not consider the total vertex weight, but it is an insurance of a reliable solution to cover all domains under the size limitation n. Similarly, if n is closer to min_e, we can switch our problem to the MSCP problem and improve the result by adding or replacing vertices.

Discussion. If $n < min_e$, there is no solution under the size limitation n. In this situation, we can only try to cover the vertices in V as many as possible. A simple but effective strategy is to select the vertex in U with the largest degree in each step which also has the largest weight if there exist other vertices that have the same degree value until n vertices are chosen.

ACKNOWLEDGMENT

This work is supported by the National Key R & D Program of China (2018YFC0809800) and the National Natural Science Foundation of China (61370060, 61902017).

REFERENCES

[1] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang and Y. Ye. "KATARA: a data cleaning system powered by knowledge bases and crowdsourcing". In SIGMOD, 2015.
[2] Y. Pochampally and K. Karlapalem. "Notability Determination for Wikipedia." In WWW, 2017.
[3] S. Brin and L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". Computer Networks and ISDN Systems, 30(1–7):107–117, 1998.
[4] J. Brank, G. Leban and M. Grobelnik. "Annotating documents with relevant Wikipedia concepts". In SiKDD, 2017.
[5] C. Yu and H. V. Jagadish. "Schema Summarization". In VLDB, 2006.
[6] N. Wang and H. Liu. "Annotating web tables with the crowd". Computing and Informatics, 37(4):969-991, 2018.
[7] K. W. Church and P. Hanks. "Word association norms, mutual information, and lexicography". Comput. Linguist., 16(1):22-29, 1990.
[8] W. Wu, H. Li, H. Wang, and K. Zhu. "Probase: A probabilistic taxonomyfor text understanding". In SIGMOD, 2012.
[9] Z. Ren, Z. Feng, L. Ke and Z. Zhang. "New ideas for applying ant colony optimization to the set covering problem". Computers & Industrial Engineering, 58(4):774-784, 2010.