LARGE UNSTRUCTURED DATASETS FROM MINING COMPETITORS

[1]SHAIK ABDUL JABBAR , M.Tech, Assistant Professor
[2]A.EMMANUEL RAJU, M.Tech, Assistant Professor
[3]H ATEEQ AHMED, M.Tech, Assistant Professor
DEPARTMENT OF CSE
DR K V SUBBA REDDY INSTITUTE OF TECHNOLOGY, DUPADU, KURNOOL

Abstract— Now-a-days in any business field we are hearing about the word _competition‘. So, by competitive analysis we can analyze the competitors and can assess the strengths and weakness of a competitor. Along line of research has demonstrated the strategic importance of identifying and monitoring a firm’s competitors. Motivated by this problem, the marketing and management community have focused on empirical methods for competitor identification as well as on methods for analyzing known competitors. Extant research on the former has focused on mining comparative expressions (e.g. Item A is better than Item) from the Web or other textual sources. Even though such expressions can indeed be indicators of competitiveness, they are absent in many domains. There are so many efficient methods for addressing the problem of finding top-k competitors in terms of scalability, accuracy.

Keywords— Data Mining, Competitor Mining, Competitors, Information search and retrieval

I. INTRODUCTION

Users often have difficulties in expressing their web search needs; they may not know the keywords that can retrieve the information they require [1]. Keyword suggestion (also known as query suggestion), which has become one of the most fundamental features of commercial Web search engines, helps in this direction. After submitting a keyword query, the user may not be satisfied with the results, so the keyword suggestion module of the search engine recommends a set of m keyword queries that are most likely to refine the user’s search in the right direction. Effective keyword suggestion methods are based on click information from query logs [2], [3], [4], [5], [6], [7], [8] and query session data [9] or query topic models. New keyword suggestions can be determined according to their semantic relevance to the original keyword query. The semantic relevance between two keyword queries can be determined (i) based on the overlap of their clicked URLs in a query log (ii) by their proximity in a bipartite graph that connects keyword queries and their clicked URLs in the query log [5], [6], [7], [8], (iii) according to their co occurrences in query sessions [13], and (iv) based on their similarity in the topic distribution space [12]. However, none of the existing methods provide location aware keyword query suggestion, such that the suggested keyword queries can retrieve documents not only related to the user information needs but also located near the user location. This requirement emerges due to the popularity of spatial keyword search that takes a user location and user-supplied keyword query as arguments and returns objects that are spatially close and textually\ relevant to these arguments. Google processed a daily average of 4.7 billion queries in 20111, a substantial fraction of which have local intent and target spatial web objects (i.e., points of interest with a web presence having locations as well as text descriptions) or geo-documents (i.e., documents associated with geo-locations). Furthermore, 53% of Bing’s mobile searches in 2011 were found to have a local intent.2 To fill this gap, we propose a Location-aware Keyword query Suggestion (LKS) framework. We illustrate the benefit of LKS using a toy example. Consider five geo-documents d1–d5 as listed in Figure 1(a). Each document di is associated with a location di:_ as shown in Figure 1(b). Assume that a user issues a keyword query kq = \seafood" at location _q, shown in Figure 1(b). Note that the relevant documents d1–d3 (containing \seafood") are far from _q. A location aware suggestion is \lobster", which can retrieve nearby documents d4 and d5 that are also relevant to the user’s original search intention. Previous keyword query suggestion models (e.g., [6]) ignore the user location and would_sh", which again fails to retrieve nearby relevant documents. Note that LKS has a different goal and therefore differs from other location-aware recommendation methods .

The first challenge of our LKS framework is how to effectively measure keyword query similarity while capturing the spatial distance factor. In accordance to previous query suggestion approaches LKS constructs and uses a keyword-document bipartite graph (KD-graph for short), which connects the keyword queries with their relevant documents as shown in Figure 1(c). Different to all previous approaches which ignore locations, LKS adjusts the weights on edges in the KD-graph to capture not only the semantic relevance between keyword queries, but also the spatial distance between the document locations and the query issuer’s location _q. We apply a random walk with restart (RWR) process [22] on the KD-graph, starting from the user supplied query kq, to find the set of m key- word queries with the highest semantic relevance to kq and spatial proximity to the user

location. RWR on a KD-graph has been considered superior to alternative approaches [7] and has been a standard technique employed in various (location-independent) keyword suggestion studies.

The second challenge is to compute the suggestions efficiently on a large dynamic graph. Performing keyword\ suggestion instantly is important for the applicability of LKS in practice. However, RWR search has a high computational cost on large graphs. Previous work on scaling up RWR search require pre-computation and/or graph segmentation part of the required RWR scores are materialized under the assumption that the transition probabilities between nodes (i.e., the edge weights) are known beforehand. In addition, RWR search algorithms that do not rely on pre-computation accelerate the computation by pruning nodes based on their lower or upper bound scores and also require the full transition probabilities. However, the edge weights of our KD-graph are unknown in advance, hindering the application of all these approaches. To the best of our knowledge, no existing technique can accelerate RWR when edge weights are unknown a priori (or they are dynamic). To address this issue, we present a novel partition-based algorithm (PA) that greatly reduces the cost of RWR search on such a dynamic bipartite graph. In a nutshell, our proposal divides the keyword queries and the documents into partitions and adopts a lazy mechanism that accelerates RWR search. Pam and the lazy mechanism are generic techniques for RWR search, orthogonal to LKS, therefore they can be applied to speed up RWR search in other large graphs. In summary, the contributions of this paper are: _ We design a Location-aware Keyword query Suggestion (LKS) framework, which provides suggestions that are relevant to the user's information needs and can retrieve relevant documents close to the query issuer's location. _ We extend the state-of-the-art Bookmark Coloring Algorithm (BCA) [28] for RWR search to compute the location-aware suggestions.

II. LITERATURE SURVEY

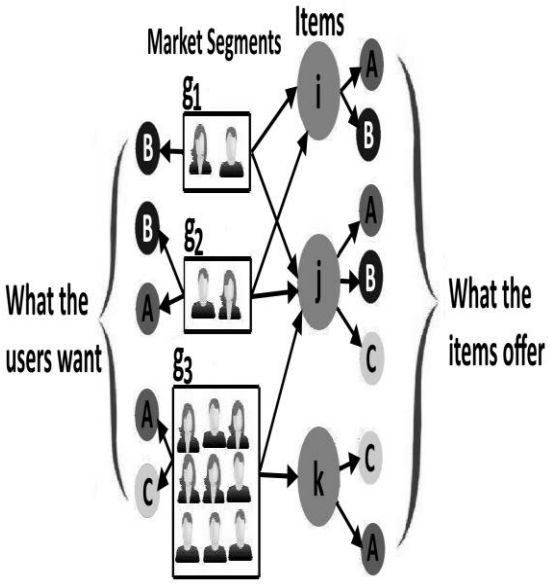
developed an automatic system that discovers companies which are in competition from public information sources. In this the data is extracted and also uses transformation learning techniques to get appropriate data normalization which combines structured and unstructured sources uses probabilistic models to represent the unlinked data and succeeds in discovering competitors. The paper also introduced iterative graph reconstruction process and also used machine learning algorithms for finding competitors. But this technique has a problem of finding market demands

presented a formal definition of competitiveness between two items. In this authors have used many domains and also handled the problems in pre vious approaches. In this author consider the items are positioned in multi-dimensional feature space and also considers the opinions and preferences of users. However, this technique has addressed the problem of finding top-k competitors of a given items.

verifies that competing products are likely to have similar web footprints a phenomenon that refers to online isomorphism. In this they consider different types of isomorphism between two firms such as overlap between the in-link and out-link of respective websites. But the need for isomorphism feature limits its applicability to products and makes it unsuitable for items and domains where such features are not available (or) extremely sparse.

accomplishes a task for mining competitors with respect to an entity. Here entity refers to person, product (or) a company. The paper proposed an algorithm called ―CoMiner‖ which first extracts the comparative items of input entity and rank them according to comparability. But CoMiner was developed for supporting a specific domain and effort for further domains is still challenging.

III. SYSTEM DESIGN AND ANALYSIS



IV. FINDING THE TOP-K COMPETITORS

Given the definition of the competitiveness in Eq. 1, we study the natural problem of finding the top-k competitors of a given item. Formally: Problem 1. [Top-k Competitors Problem]: We are presented with a market with a set of n items I and a set of features F. Then, given a single item i ∈ I, we want to identify the k items from I that maximize CF (i, ·). A naive algorithm would compute the competitiveness between i and every possible candidate. The complexity of this brute force method is clearly Θ(2|F | × n 2 × log K), which can be easily dominated by the powerset factor and, as we demonstrate in our experiments, is impractical for large datasets. One option could be to perform the naive computation in a distributed fashion. Even in this case, however, we would need one thread for each of the n 2 pairs. This is far from trivial, if one considers that n could measure in the tens of thousands. In addition, a naive MapReduce implementation would face the bottleneck of passing everything through the reducer to account for the self-join included in the computation. In practice, the selfjoin would have to be implemented via a customized technique for reduce-side joins, which is a non-trivial and highly expensive operation [23]. These issues motivate us to introduce CMiner, an efficient exact algorithm for Problem 1. Except for the creation of our indexing mechanism, every other aspect of CMiner can also be incorporated in a parallel solution. First, we define the concept of item dominance, which will aid us in our analysis:

Definition 3. [Item Dominance]: Consider a market with a set of items I and a set of features F. Then, we say that an item i ∈ I dominates another item j ∈ I, if f[i] ≥ f[j] for every feature f ∈ F.

Conceptually, an item dominates another if it has better or equal values across features. We observe that, per Eq. 1, any item i that dominates j also achieves the maximum possible competitiveness with j, since it can cover the requirements of any customer covered by j. This motivates us to utilize the skyline of the entire set of items I. The skyline is a wellstudied concept that represents the subset of points in a population that are not dominated by any other point [24]. We refer to the skyline of a set of items I as Sky(I). The concept of the skyline leads to the following lemma:

Lemma 1. Given the skyline Sky(I) of a set of items I and an item i ∈ I, let Y contain the k items from Sky(I) that are most competitive with i. Then, an item j ∈ I can only be in the top-k competitors of i, if j ∈ Y or if j is dominated by one of the items in Y.

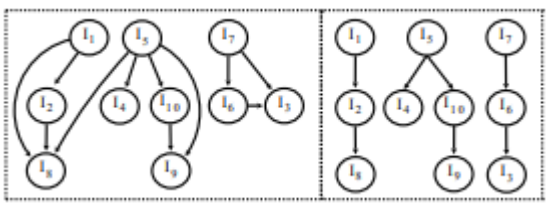


FIG1: The left side shows the dominance graph for a set of items. An edge Ii → Ij means that Ii dominates Ij . The right side of the figure shows the skyline pyramid.

The CMiner Algorithm: Next, we present CMiner, an exact algorithm for finding the top-k competitors of a given item. Our algorithm makes use of the skyline pyramid in order to reduce the number of items that need to be considered. Given that we only care about the top-k competitors, we can incrementally compute the score of each candidate and stop when it is guaranteed that the top-k have emerged. The pseudocode is given in Algorithm 1.

Discussion of CMiner: The input includes the set of items I, the set of features F, the item of interest i, the number k of top competitors to retrieve, the set Q of queries and their probabilities, and the skyline pyramid DI. The algorithm first retrieves the items that dominate i, via masters(i) (line 1). These items have the maximum possible competitiveness with i. If at least k such items exist, we report those and conclude (lines 2-4). Otherwise, we add them to TopK and decrement our budget of k accordingly (line 5). The variable LB maintains the lowest lower bound from the current topk set (line 6) and is used to prune candidates. In line 7, we initialize the set of candidates X as the union of items in the first layer of the pyramid and the set of items dominated by those already in the TopK. This is achieved via calling GETSLAVES(T opK, DI). In every iteration of lines 8-17, CMiner feeds the set of candidates X to the UPDATETOPK() routine, which prunes items based on the LB threshold. It then updates the TopK set via the MERGE() function, which identifies the items with the highest competitiveness from T opK ∪ X . This can be achieved in linear time, since both X and T opK are sorted. In line 13, the pruning threshold LB is set to the worst (lowest) score among the new T opK. Finally, GETSLAVES() is used to expand the set of candidates by including items that are dominated by those in X .

Algorithm 1 CMiner

Input: Set of items $\mathcal{I}$, Item of interest $i \in \mathcal{I}$, feature space $\mathcal{F}$, Collection $\mathcal{Q} \in 2^{\mathcal{F}}$ of queries with non-zero weights, skyline pyramid $\mathcal{D}_{\mathcal{I}}$, int k
Output: Set of top-k competitors for i

1: $TopK \leftarrow masters(i)$
2: **if** ($k \leq |TopK|$) **then**
3: **return** $TopK$
4: **end if**
5: $k \leftarrow k - |TopK|$
6: $LB \leftarrow -1$
7: $\mathcal{X} \leftarrow$GETSLAVES$(TopK, \mathcal{D}_{\mathcal{I}}) \cup \mathcal{D}_{\mathcal{I}}[0]$
8: **while** ($|\mathcal{X}|$!= 0) **do**
9: $\mathcal{X} \leftarrow$ UPDATETOPK$(k, LB, \mathcal{X})$
10: **if** ($|\mathcal{X}|$!= 0) **then**
11: $TopK \leftarrow$MERGE$(TopK, \mathcal{X})$
12: **if** ($|TopK| = k$) **then**
13: $LB \leftarrow$WORSTIN$(TopK)$
14: **end if**
15: $\mathcal{X} \leftarrow$GETSLAVES$(\mathcal{X}, \mathcal{D}_{\mathcal{I}})$
16: **end if**
17: **end while**
18: **return** $TopK$

19: **Routine** UPDATETOPK$(k, LB, \mathcal{X})$
20: $localTopK \leftarrow \emptyset$
21: $low(j) \leftarrow 0, \forall j \in \mathcal{X}.$
22: $up(j) \leftarrow \sum_{q \in \mathcal{Q}} p(q) \times V^q_{j,j}, \forall j \in \mathcal{X}.$
23: **for every** $q \in Q$ **do**
24: $maxV \leftarrow p(q) \times V^q_{i,i}$
25: **for every item** $j \in \mathcal{X}$ **do**
26: $up(j) \leftarrow up(j) - maxV + p(q) \times V^q_{i,j}$
27: **if** ($up(j) < LB$) **then**
28: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{j\}$
29: **else**
30: $low(j) \leftarrow low(j) + p(q) \times V^q_{i,j}$
31: $localTopK.update(j, low(j))$
32: **if** ($|localTopK| \geq k$) **then**
33: $LB \leftarrow$WORSTIN$(localTopK)$
34: **end if**
35: **end if**
36: **end for**
37: **if** ($|\mathcal{X}| \leq k$) **then**
38: **break**
39: **end if**
40: **end for**
41: **for every item** $j \in \mathcal{X}$ **do**
42: **for every remaining** $q \in \mathcal{Q}$ **do**
43: $low(j) \leftarrow low(j) + p(q) \times V^q_{i,j}$
44: **end for**
45: $localTopK.update(j, low(j))$
46: **end for**
47: **return** TOPK$(localTopK)$

V. BOOSTING THE CMINER ALGORITHM

Next, we describe several improvements that we have applied to CMiner in order to achieve computational savings while maintaining the exact nature of the algorithm. 4.1 Query Ordering Our complexity analysis is based on the premise that CMiner evaluates all queries Q for each candidate item j. However, this assumption naively ignores the algorithm's pruning ability, which is based on using lower and upper bounds on competitiveness scores to eliminate candidates early. Next, we show how to greatly improve the algorithm's pruning effectiveness by strategically selecting the processing order of queries (line 23 of CMiner).

CMiner uses the following update rules for the lower and upper bounds for a candidate j:

$$low(j) \leftarrow low(j) + p(q) \times V^q_{i,j}$$
$$up(j) \leftarrow up(j) - p(q) \times V^q_{i,i} + p(q) \times V^q_{i,j}$$

VI. EXPERIMENTAL EVALUATION

In this section we describe the experiments that we conducted to evaluate our methodology. All experiments were completed on an desktop with a Quad-Core 3.5GHz Processor and 2GB RAM.

Datasets and Baselines

Our experiments include four datasets, which were collected for the purposes of this project. The datasets were intentionally selected from different domains to portray the cross-domain applicability of our approach. In addition to the full information on each item in our datasets, we also collected the full set of reviews that were available on the source website. These reviews were used to (1) estimate queries probabilities, as described and (2) extract the opinions of reviewers on specific features.The highly-cited method by Ding et al. [28] is used to convert each review to a vector of opinions, where each opinion is defined as a feature-polarity combination (e.g. service+, food-).

The percentage of reviews on an item that express a positive opinion on a specific feature is used as the feature's numeric value for that item. We refer to these as opinion features. Table 4 includes descriptive statistics for each dataset, while a detailed description is provided below. CAMERAS: This dataset includes 579 digital cameras from Amazon.com. We collected the full set of reviews for each camera, for a total of 147192 reviews. The set of features includes the resolution (in MP), shutter speed (in seconds), zoom (e.g. 4x), and price. It also includes opinion features on manual, photos, video, design, flash, focus, menu options, lcd screen, size, features, lens, warranty, colors, stabilization, battery life, resolution, and cost. HOTELS: This dataset includes 80799 reviews on 1283 hotels from Booking.com. The set of features includes the facilities,activities, and services offered by the hotel. All three of these multi-categorical features are available on the website. The dataset also includes opinion features on location, services, cleanliness, staff, and comfort. RESTAURANTS: This dataset includes 30821 reviews on 4622 New York City restaurants from TripAdvisor.com. The set of features for this dataset includes the cuisine types and meal types (e.g. lunch, dinner) offered by the restaurant, as well as the activity types (e.g. drinks, parties) that it is good for. All three of these multi-categorical features are available on the website. The dataset also includes opinion features on food, service, value-for-money, atmosphere, and price. RECIPES: This dataset includes 100000 recipes from Sparkrecipes.com. It also includes the full set of reviews on each recipe, for a total of 21685 reviews. The set of features for each recipe includes the number of calories, as well as the following nutritional information, measured in grams: fat, cholesterol, sodium, potassium, carb, fiber, protein, vitamin A, vitamin B12, vitamin C, vitamin E, calcium, copper, folate, magnesium, niasin, phosphorus, riboflavin, selenium, thiamin, zinc. All information is openly available on the website.

Dataset	#Items	#Feats.	#Subsets	Skyline Layers
CAMERAS	579	21	14779	5
HOTELS	1283	8	127	5
RESTAURANTS	4622	8	64	12
RECIPES	100000	22	133	22

Table1: Dataset Statistics

For each dataset, the 2nd, 3rd, 4th and 5th columns include the number of items, the number of features, the number of distinct queries, and the number of layers in the respective skyline pyramid, respectively. In order to conclude the description of

our datasets, we present some statistics on the skyline-pyramid structure constructed for each corpus. Figure 4 shows the distribution of items in the first 6 skyline layers of each dataset. We observe that, for all datasets, nearly 99% of the items can be found within the first 4 layers, with the majority of those falling within the first 2 layers. This is due to the large dimensionality of the feature space, which makes it difficult for items to dominate one another. As we show in our experiments, the skyline pyramid enables CMiner to clearly outperform the baselines with respect to computational cost. This is despite the high concentration of items within the first layers, since CMiner can effectively traverse the pyramid and consider only a small fraction of these items.

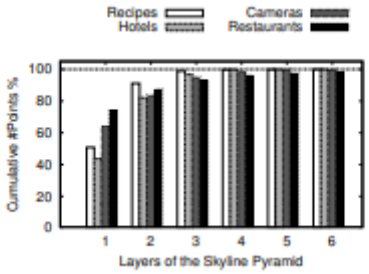


Fig2: Cumulative distribution of items across the first 6 layers of the skyline pyramid.

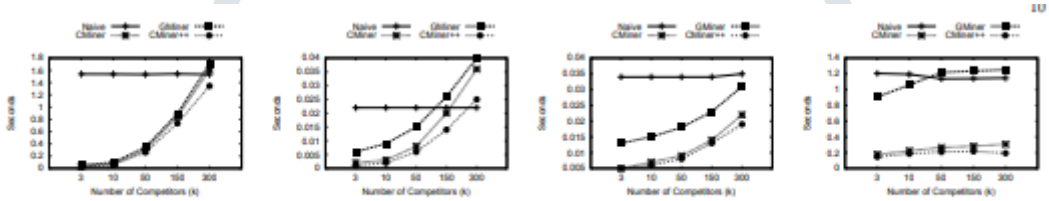


Fig3: Average time (per item) to compute top-k competitors for each dataset.

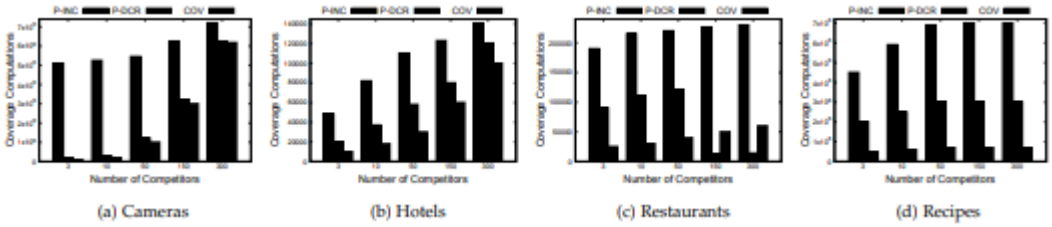


Fig4: Average number of pairwise coverage computations under different ordering schemes.

VII. CONCLUSION

Data mining has importance regarding finding the patterns, forecasting, discovery of knowledge etc., in different business domains. Machine learning algorithms are widely used in various applications. Every business related application uses data mining techniques. To improve such business or providing appropriate competitors for the business to the user need the support of web mining techniques. The competitor mining is one such a way to analyze competitors for the selected items. In this paper, we gave a comprehensive analysis of the competitor mining algorithms with its advantages and drawbacks. Finally, the CMiner++ yielded least computation time when comparing others. The most important features and process are not considered in the all baseline algorithms. This can be improved in the further researches

REFERENCES

[1] M.E.Porter, Competitive Strategy: Techniques for Analyzing Industries and Competitors. Free Press, 1980.

[2] R. Deshpand and H. Gatingon, “Competitive analysis,” Marketing Letters, 1994.

[3] B. H. Clark and D. B. Montgomery, “Managerial Identification of Competitors,” Journal of Marketing, 1999.

[4] W. T. Few, “Managerial competitor identification: Integrating the categorization, economic and organizational identity perspectives,” Doctoral Dissertaion, 2007.

[5] M. Bergen and M. A. Peteraf, "Competitor identification and competitor analysis: a broad-based managerial approach," Managerial and Decision Economics, 2002.

[6] J. F. Porac and H. Thomas, "Taxonomic mental models in competitor definition," The Academy of Management Review, 2008.

[7] M.-J. Chen, "Competitor analysis and interfirm rivalry: Toward a theoretical integration," Academy of Management Review, 1996.

[8] R. Li, S. Bao, J. Wang, Y. Yu, and Y. Cao, "Cominer: An effective algorithm for mining competitors from the web," in ICDM, 2006.

[9] Z. Ma, G. Pant, and O. R. L. Sheng, "Mining competitor relationships from online news: A networkbased approach," Electronic Commerce Research and Applications, 2011.

[10] R. Li, S. Bao, J. Wang, Y. Liu, and Y. Yu, "Web scale competitor discovery using mutual information," in ADMA, 2006.