This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

Mining Competitors from Large Unstructured Datasets

George Valkanas, Theodoros Lappas, and Dimitrios Gunopulos,

Abstract—In any competitive business, success is based on the ability to make an item more appealing to customers than the competition. A number of questions arise in the context of this task: *how do we formalize and quantify the competitiveness between two items? Who are the main competitors of a given item? What are the features of an item that most affect its competitiveness?* Despite the impact and relevance of this problem to many domains, only a limited amount of work has been devoted toward an effective solution. In this paper, we present a formal definition of the competitiveness between two items, based on the market segments that they can both cover. Our evaluation of competitiveness utilizes customer reviews, an abundant source of information that is available in a wide range of domains. We present efficient methods for evaluating competitiveness in large review datasets and address the natural problem of finding the top-k competitors of a given item. Finally, we evaluate the quality of our results and the scalability of our approach using multiple datasets from different domains.

Index Terms—Data mining, Web mining, Information Search and Retrieval, Electronic commerce

1 Introduction

A Long line of research has demonstrated the strategic importance of identifying and monitoring a firm's competitors [1]. Motivated by this problem, the marketing and management community have focused on empirical methods for competitor identification [2], [3], [4], [5], [6], as well as on methods for analyzing known competitors [7]. Extant research on the former has focused on mining comparative expressions (e.g. "Item A is better than Item B") from the Web or other textual sources [8], [9], [10], [11], [12], [13]. Even though such expressions can indeed be indicators of competitiveness, they are absent in many domains. For instance, consider the domain of vacation packages (e.g flight-hotel-car combinations). In this case, items have no assigned name by which they can be queried or compared with each other. Further, the frequency of textual comparative evidence can vary greatly across domains. For example, when comparing brand names at the firm level (e.g. "Google vs Yahoo" or "Sony vs Panasonic"), it is indeed likely that comparative patterns can be found by simply querying the web. However, it is easy to identify mainstream domains where such evidence is extremely scarce, such as shoes, jewelery, hotels, restaurants, and furniture. Motivated by these shortcomings, we propose a new formalization of the competitiveness between two items, based on the market segments that they can both cover. Formally:

Definition 1. **[Competitiveness]:** Let $\mathcal{U}$ be the population of all possible customers in a given market. We consider that an item i covers a customer $u \in \mathcal{U}$ if it can cover all of the customer's requirements. Then, the competitiveness between two items i, j is proportional to the number of customers that they can both cover.

- *G. Valkanas and T. Lappas are with the School of Business, Stevens Institute of Technology, Hoboken, NJ, 07030. E-mails: gvalkana@stevens.edu, tlappas@stevens.edu*
- *D. Gunopulos is with the University of Athens. E-mail: dg@di.uoa.gr*

Our competitiveness paradigm is based on the following observation: *the competitiveness between two items is based on whether they compete for the attention and business of the same groups of customers (i.e. the same market segments)*. For example, two restaurants that exist in different countries are obviously not competitive, since there is no overlap between their target groups. Consider the example shown in Figure 1.

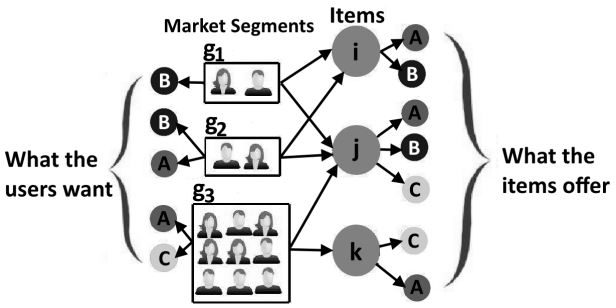


Fig. 1: A (simplified) example of our competitiveness paradigm

The figure illustrates the competitiveness between three items i, j and k. Each item is mapped to the set of features that it can offer to a customer. Three features are considered in this example: A, B and C. Even though this simple example considers only binary features (i.e. available/not available), our actual formalization accounts for a much richer space including binary, categorical and numerical features. The left side of the figure shows three groups of customers g_1, g_2, and g_3. Each group represents a different market segment. Users are grouped based on their preferences with respect to the features. For example, the customers in g_2 are only interested in features A and B. We observe that items i and k are not competitive, since they simply do not appeal to the same groups of customers. On the other hand, j competes with both i (for groups g_1 and g_2) and k (for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

g_3). Finally, an interesting observation is that j competes for 4 users with i and for 9 users with k. In other words, k is a stronger competitor for j, since it claims a much larger portion of its market share than i.

This example illustrates the ideal scenario, in which we have access to the complete set of customers in a given market, as well as to specific market segments and their requirements. In practice, however, such information is not available. In order to overcome this, we describe a method for computing all the segments in a given market based on mining large review datasets. This method allows us to operationalize our definition of competitiveness and address the problem of finding the top-k competitors of an item in any given market. As we show in our work, this problem presents significant computational challenges, especially in the presence of large datasets with hundreds or thousands of items, such as those that are often found in mainstream domains. We address these challenges via a highly scalable framework for top-k computation, including an efficient evaluation algorithm and an appropriate index.

Our work makes the following contributions:

- A formal definition of the competitiveness between two items, based on their appeal to the various customer segments in their market. Our approach overcomes the reliance of previous work on scarce comparative evidence mined from text.
- A formal methodology for the identification of the different types of customers in a given market, as well as for the estimation of the percentage of customers that belong to each type.
- A highly scalable framework for finding the top-k competitors of a given item in very large datasets.

2 Defining Competitiveness

The typical user session on a review platform, such as Yelp, Amazon or TripAdvisor, consists of the following steps:

1) Specify all required features in a query.
2) Submit the query to the website's search engine and retrieve the matching items.
3) Process the reviews of the returned items and make a purchase decision.

In this setting, items that cover the user's requirements will be included in the search engine's response and will compete for her attention. On the other hand, non-covering items will not be considered by the user and, thus, will not have a chance to compete. Next, we present an example that extends this decision-making process to a multi-user setting.

Consider a simple market with 3 hotels i, j, k and 6 binary features: *bar, breakfast, gym, parking, pool, wi-fi*. Table 1 includes the value of each hotel for each feature. In this simple example, we assume that the market includes 6 mutually exclusive customer segments (types). Each segment is represented by a query that includes the features that are of interest to the customers included in the segment. Information on each segment is provided in Table 2. For instance, the first segment includes 100 customers who are interested in parking and wi-fi, while the second segment includes 50 customers who are only interested in parking.

TABLE 1: Hotels and their Features.

Name	Bar	Breakfast	Gym	Parking	Pool	Wi-Fi
Hilton	Yes	No	Yes	Yes	Yes	Yes
Marriot	Yes	Yes	No	Yes	Yes	Yes
Westin	No	Yes	Yes	Yes	No	Yes

TABLE 2: Customer Segments

ID	Segment Size	Features of Interest
q_1	100	(*parking, wi-fi*)
q_2	50	(*parking*)
q_3	60	(*wi-fi*)
q_4	120	(*gym, wi-fi*)
q_5	250	(*breakfast, parking*)
q_6	80	(*gym, bar, breakfast*)

In order to measure the competition between any two hotels, we need to identify the number of customers that they can both satisfy. The results are shown in Table 3. The *Hilton* and the *Marriot* can cover segments q_1, q_3, and q_4. Therefore, they compete for $(100 + 50 + 60)/660 \approx 32\%$ of the entire market. We observe that this is the lowest competitiveness achieved for any pair, even though the two hotels are also the most similar. In fact, the highest competitiveness is observed between the *Marriot* and the *Westin*, that compete for 70% of the market. This is a critical observation that demonstrates that **similarity is not a good proxy for competitiveness**. The explanation is intuitive. The availability of both a pool and a bar makes the *Hilton* and the *Marriot* more similar to each other and less similar to the *Westin*. However, neither of these features has an effect on competitiveness. First, the *pool* feature is not required by any of the customers in this market. Second, even though the availability of a bar is required by segment q_6, none of the three hotels can cover all three of this segment's requirements. Therefore, none of the hotels compete for this particular segment.

Another intuitive observation is that the size of the segment has a direct effect on competitiveness. For example, even though the *Westin* shares the same number of segments (4) with the other two hotels, its competitiveness with the *Marriot* is significantly higher. This is due to the size of the q_5 segment, which is more than double the size of q_4.

TABLE 3: Common segments for restaurant pairs

Restaurant Pairs	Common Segments	Common %
Hilton, Marriot	(q_1, q_2, q_3)	32%
Hilton, Westin	(q_1, q_2, q_3, q_4)	50%
Marriot, Westin	(q_1, q_2, q_3, q_5)	70%

The above example is limited to binary features. In this simple setting, it is trivial to determine if two items can both cover a feature. However, as we discuss in detail in Section 2.1, the items in a market can have different types of features (e.g. numeric) that may be only *partially* covered by two items. Formally, let $p(q)$ be the percentage of users represented by a query q and let $V_q^{i,j}$ be the *pairwise coverage* offered by two items i and j to the space defined by the features in q. Then, we define the competitiveness between i and j in a market with a feature subset $\mathcal{F}$ as follows:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

$$C_{\mathcal{F}}(i,j) = \sum_{q \in 2^{\mathcal{F}}} p(q) \times V_{i,j}^{q}, \quad (1)$$

This definition has a clear probabilistic interpretation: *given two items i, j, their competitiveness $C_{\mathcal{F}}(i,j)$ represents the probability that the two items are included in the consideration set of a random user.* This new definition has direct implications for consumers, who often rely on recommendation systems to help them choose one of several candidate products. The ability to measure the competitiveness between two items enables the recommendation system to strategically select the order in which items should be recommended or the sets of items that should be included together in a group recommendation. For instance, if a random user u shows interest in an item i, then she is also likely to be interested in the items with the highest $C_{\mathcal{F}}(i,\cdot)$ values. Such competitive items are likely to meet the criteria satisfied by i and even cover additional parts of the feature space. In addition, as the user u rates more items and the system gains a more accurate view of her requirements, our competitiveness measure can be trivially adjusted to consider only those features from $\mathcal{F}$ (and only those value intervals within each feature) that are relevant for u. This competitiveness-based recommendation paradigm is a departure from the standard approach that adjusts the weight (relevance) of an item j for a user u based on the rating that u submits for items *similar* to j. As discussed, this approach ignores that (i) the similarity may be due to irrelevant or trivial features and (ii) for a user who likes an item i, an item j that is far superior than i with respect to the user's requirements (and thus quite different) is a better recommendation candidate than an item j' that is highly similar to i.

In the following two sections we describe the computation of the two primary components of competitiveness: (1) the *pairwise coverage* $V_{i,j}^{q}$ of a query that includes binary, categorical, ordinal or numeric features, and (2) the percentage $p(q)$ of users represented by each query q.

2.1 Pairwise Coverage

We begin by defining the pairwise coverage of a single feature f. We then define the pairwise coverage of an entire *query* of features q.

Definition 2. **[Pairwise Feature Coverage]**: We define the *pairwise coverage* $V_{i,j}^{f}$ of a feature f by two items i, j as the percentage of all possible values of f that can be covered by *both* i and j. Formally, given the set of all possible values V^f for f, we define:

$$V_{i,j}^{f} = \frac{|\{v \in V^f : v \angle f[i] \wedge v \angle f[j]\}|}{|values(f)|},$$

where $v \angle f[i]$ represents that v is *covered* by the value of item i for feature f.

Next, we describe the computation of $V_{i,j}^{f}$ for different types of features.

[Binary and Categorical Features]: Categorical features take one or more values from a finite space. Examples of single-value features include the brand of a digital camera or the location of a restaurant. Examples of multi-value features include the amenities offered by a hotel or the types of cuisine offered by a restaurant. Any categorical feature can be encoded via a set of binary features, with each binary feature indicating the (lack of) coverage of one of the original feature's possible values. In this simple setting, the feature can be fully covered (if $f[i] = f[j] = 1$ or, equivalently, $f[i] \times f[j] = 1$), or not covered at all. Formally, the pairwise coverage of a binary feature f by two items i, j can be computed as follows:

$$V_{i,j}^{f} = f[i] \times f[j] \quad \text{(binary features)} \quad (2)$$

[Numeric Features]: Numeric features take values from a pre-defined range. Henceforth, without loss of generality, we consider numeric features that take values in $[0, 1]$, with higher values being preferable. The pairwise coverage of a numeric feature f by two items i and j can be easily computed as the smallest (worst) value achieved for f by either item. For instance, consider two restaurants i, j with values 0.8 and 0.5 for the feature *food quality*. Their pairwise coverage in this setting is 0.5. Conceptually, the two items will compete for any customer who accepts a quality ≤ 0.5. Customers with higher standards would eliminate restaurant j, which will never have a chance to compete for their business. Formally, the pairwise coverage of a numeric feature f by two items i, j can be computed as follows:

$$V_{i,j}^{f} = \min(f[i], f[j]) \quad \text{(numeric features)} \quad (3)$$

[Ordinal Features]: Ordinal features take values from a finite *ordered* list. A characteristic example is the popular five star scale used to evaluate the quality of a service or product. For example, consider that the values of two items i and j on the 5-star rating scale are $\star\star$ and $\star\star\star$, respectively. Customers that demand at least 4 stars will not consider either of the two items, while customers that demand at least 3 stars will only consider item j. The two items will thus compete for all customers that are willing to accept 1 or 2 stars. Therefore, as in the case of numeric features, the pairwise coverage for ordinal features is determined by the worst of the two values. In this example, given that the two items compete for 2 of the 5 levels of the ordinal scale (1 and 2 stars), their competitiveness is proportional to $2/5 = 0.4$. Formally, the pairwise coverage of an ordinal feature f by two items i, j can be computed as follows:

$$V_{i,j}^{f} = \frac{\min(f[i], f[j])}{|V^f|} \quad \text{(ordinal features)} \quad (4)$$

Pairwise coverage of a feature query: We now discuss how coverage can be extended to the query level. Figure 2 visualizes a query q that includes two numeric features f_1 and f_2. The figure also includes two competitive items i and j, positioned according to their values for the two features: $f_1[i] = 0.3, f_2[i] = 0.3, f_1[j] = 0.2$, and $f_2[j] = 0.7$. We observe that the percentage of the 2-dimensional space that each item covers is equivalent to the area of the rectangle defined by the beginning of the two axes $(0, 0)$ and the item's values for f_1 and f_2. For example, the covered area for item i is $0.3 \times 0.3 = 0.09$, equal to 9% of the entire space. Similarly, the pairwise coverage provided by both items is equal to $0.2 \times 0.3 = 0.06$ (i.e. 6% of the market).

Per our example, the pairwise coverage of a given query q by two items i, j can be measured as the volume of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

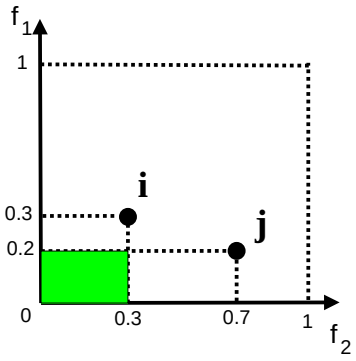


Fig. 2: Geometric interpretation of pairwise coverage

hyper-rectangle defined by the pairwise coverage provided by the two items for each feature $f \in q$. Formally:

$$V_{i,j}^{q} = \prod_{f \in q} V_{i,j}^{f} \tag{5}$$

Eq. 5 allows us to compute the pairwise coverage of any query of features, as required by the definition of competitiveness in Eq. 1.

2.2 Estimating Query Probabilities

The definition of competitiveness given in Eq. 1 considers the probability $p(q)$ that a random customer will be represented by a specific query of features q, for every possible query $q \in 2^{\mathcal{F}}$. In this section, we describe how these probabilities can be estimated from real data. Feature queries are a direct representation of user preferences. Ideally, we would have access to the query logs of the platform's (e.g. Amazon's or TripAdvisor's) search engine. In practice, however, the sensitive and proprietary nature of such information makes it very hard for firms to share publicly. Therefore, we design an estimation process that only requires access to an abundant resource: *customer reviews*. Each review includes a customer's opinions on a particular subset of features of the reviewed item. Extant research has repeatedly validated the use of reviews to estimate user preferences with respect to different features in multiple domains, such as phone apps [14], movies [15], electronics [16], and hotels [17].

A trivial approach would be to estimate the demand for each feature separately, and then aggregate the individual estimates at the subset level. However, this approach assumes *feature independence*, a strong assumption that would first have to be validated across domains. To avoid this assumption and capture possible feature correlations, we consider all the features mentioned in each review as a single query. We then compute the frequency of each query q in our review corpus $\mathcal{R}$, and divide it by the sum of the frequencies of all queries. This gives us an estimate of the probability that a random user will be interested in *exactly* the set of features includes in q. Formally:

$$p(q) = \frac{freq(q, \mathcal{R})}{\sum_{q \in 2^{\mathcal{F}}} freq(q', \mathcal{R})} \tag{6}$$

Ideally, we would have access to the set of requirements of every possible customer in existence. The maximum likelihood estimate of Eq. 6 would then compute the exact occurrence probability of any query q. While this type of global access is unrealistic, Eq. 6 can still deliver accurate estimates if the number of reviews in $\mathcal{R}$ is large enough to accurately represent the customer population. The usefulness of the estimator is thus determined by a simple question: *how many reviews do we need to achieve accurate estimates?* We address this question in Section 5.7 of the experiments, where we present our results on datasets from different domains.

2.3 Extending our Competitiveness Definition

Feature Uniformity: Our competitiveness definition assumes that user requirements are uniformly distributed within the value space of each feature. This assumption allows us to build a computational model for competitiveness, but in practice it may not always be true. For instance, the number of users demanding quality in $[0, 0.1]$ might be different than those demanding a value in $[0.4, 0.5]$. Moreover, for lack of more accurate information, it provides a conservative lower bound of our model's true effectiveness: having access to the distribution of interest within each feature could only improve the quality of our results.

If such information was indeed available, then the naive approach would be to consider all possible interest intervals combinations for all possible queries. Henceforth, we refer to these as *extended* queries. Clearly, the number of possible extended queries is exponential and renders the computational cost of any evaluation algorithm prohibitive. This limitation can be addressed by organizing the dataset into a multi-dimensional grid, where each feature represents a different dimension. Each cell in the grid represents a different extended query (i.e. a set of features and an interest interval for each feature). We can then compute the competitiveness between two items by simply counting the number of data points that fall in the cells that they can both cover.

We can also precompute the sums of each cell offline with the prefix-sum array technique [18], as well as reduce the space complexity via approximations [19], [20] or multi-dimensional histograms [21], [22]. A parameter of the grid-construction process is the cell size, with larger cells sacrificing accuracy for the sake of efficiency. In practice, this parameter will be determined by the granularity of the input data, as well as the practitioner's computational constraints.

Feature Importance: A second assumption of our competitiveness definition is that all the features in a query q are equally important. However, a user who submits the query $q = (f_1, f_2)$ may care more about f_1 than for f_2. As with the case of feature uniformity, the consideration of such weights requires the availability of appropriate data that is rarely available in practice. Nonetheless, we can address this limitation by extending our definition of pairwise coverage. For instance, consider that the feature weights are in $[0, 1]$ and that the weights for f_1 and f_2 are $w_1 = 0.8$ and $w_2 = 0.4$, respectively. We are then given two items i, j such that: $f_1[i] = 0.5, f_2[i] = 0.3, f_1[j] = 0.5, f_2[j] = 0.6$. As per our initial definition, the pairwise coverage of the 2-dimensional space by the two items is $\min(0.5, 0.5) \times \min(0.3, 0.6) = 0.5 \times 0.3 = 0.15$. If we consider the feature weights, the computation becomes: $(w_1 \times 0.5) \times (w_2 \times 0.3) = 0.048$. Formally, this extension translates to the introduction of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

feature weight as a multiplier for the right-hand side of Eq. 3. Note that, while this example includes only numeric features, the same extension for categorical and ordinal attributes trivially follows.

3 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 $\mathcal{I}$ and a set of features $\mathcal{F}$. Then, given a single item $i \in \mathcal{I}$, we want to identify the k items from $\mathcal{I}$ that maximize $\mathcal{C}_{\mathcal{F}}(i, \cdot)$.

A naive algorithm would compute the competitiveness between i and every possible candidate. The complexity of this brute force method is clearly $\Theta(2^{|\mathcal{F}|} \times n^2 \times \log K)$, 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 self-join 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 $\mathcal{I}$ and a set of features $\mathcal{F}$. Then, we say that an item $i \in I$ dominates another item $j \in I$, if $f[i] \geq f[j]$ for every feature $f \in \mathcal{F}$.

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 $\mathcal{I}$. The skyline is a well-studied 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 $\mathcal{I}$ as $Sky(\mathcal{I})$. The concept of the skyline leads to the following lemma:

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

We present the proof of Lemma 1 in Appendix B.

Lemma 1 verifies that we do not need to consider the entire set of candidates in order to find the top-k competitors. This motivates us to construct the *skyline pyramid*, a structure that greatly reduces the number of items that need to be considered. We refer to the algorithm used to construct the skyline pyramid as `PyramidFinder`. The input to `PyramidFinder` is the set of items $\mathcal{I}$. The output is the skyline pyramid $\mathcal{D}_{\mathcal{I}}$. The algorithm relies on the extraction of the skyline layers of the dataset, using a modified version of BBS [25], [26]. Each item from the i_{th} skyline layer is then assigned an inlink from all items of the $(i-1)_{th}$ level that dominate it. We present the complete pseudocode and complexity analysis of the algorithm in Appendix C.

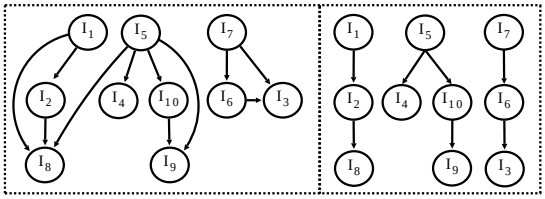


Fig. 3: The left side shows the dominance graph for a set of items. An edge $I_i \rightarrow I_j$ means that I_i dominates I_j. The right side of the figure shows the skyline pyramid.

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 $\mathcal{I}$, the set of features $\mathcal{F}$, the item of interest i, the number k of top competitors to retrieve, the set $\mathcal{Q}$ of queries and their probabilities, and the skyline pyramid $\mathcal{D}_{\mathcal{I}}$. 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 top-k set (line 6) and is used to prune candidates. In line 7, we initialize the set of candidates $\mathcal{X}$ 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$(TopK, \mathcal{D}_{\mathcal{I}})$. In every iteration of lines 8-17, CMiner feeds the set of candidates $\mathcal{X}$ 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 $TopK \cup \mathcal{X}$. This can be achieved in linear time, since both $\mathcal{X}$ and $TopK$ are sorted. In line 13, the pruning threshold LB is set to the worst (lowest) score among the new $TopK$. Finally, GETSLAVES() is used to expand the set of candidates by including items that are dominated by those in $\mathcal{X}$.

Discussion of UPDATETOPK(): This routine processes the candidates in $\mathcal{X}$ and finds at most k candidates with the highest competitiveness with i. The routine utilizes a data structure $localTopK$, implemented as an associative array: the score of each candidate serves as the key, while its id serves as the value. The array is key-sorted, to facilitate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

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$(localTopK)$

the computation of the k best items. The structure is automatically truncated so that it always contains at most k items. In lines 21-22 we initialize the lower and upper bounds. For every item $j \in \mathcal{X}$, $low(j)$ maintains the current competitiveness score of j as new queries are considered, and serves as a lower bound to the candidate's actual score. Each lower bound $low(j)$ starts from 0, and after the completion of UPDATETOPK(), it includes the true competitiveness score $C_{\mathcal{F}}(i, j)$ of candidate j with the focal item i. On the other hand, $up(j)$ is an optimistic upper bound on j's competitiveness score. Initially, $up(j)$ is set to the maximum possible score (line 22). This is equal to $\sum_{q \in \mathcal{Q}} p(q) \times V^q_{i,i}$, where $V^q_{i,i}$ is simply the coverage provided exclusively by i to q. It is then incrementally reduced toward the true $C_{\mathcal{F}}(i, j)$ value as follows. For every query $q \in \mathcal{Q}$, $maxV$ holds the maximum possible competitiveness between item i and any other item for that query, which is in fact the coverage of i with respect to q. Then, for each candidate $j \in \mathcal{X}$, we subtract $maxV$ from $up(j)$ and then add to it the actual competitiveness between i and j for query q. If the upper bound $up(j)$ of a candidate j becomes lower than the pruning threshold LB, then j can be safely disqualified (lines 27-29). Otherwise, $low(j)$ is updated and j remains in consideration (lines 30-31). After each update, the value of LB is set to the worst score in $localTopK$ (lines 32-33), to employ stricter pruning in future iterations.

If the number of candidates $|\mathcal{X}|$ becomes less or equal to k (line 37), the loop over the queries comes to a halt. This is an early-stopping criterion: since our goal is to retrieve the best k candidates in $\mathcal{X}$, having $|\mathcal{X}| <= k$ means that all remaining candidates should be returned. In lines 41-46 we complete the competitiveness computation of the remaining candidates and update $localTopk$ accordingly. This takes place after the completion of the first loop, in order to avoid unnecessary bound-checking and improve performance.

Complexity: If the item of interest i is dominated by at least k items, then these will be returned by $masters(i)$. This step can be done in $O(k)$, by iteratively retrieving k items that dominate i. Otherwise, the complexity of CMiner is controlled by UPDATETOPK(), which depends on the number of items in the candidate set $\mathcal{X}$. In its simplest form, in the k-th call of the method, the candidate set contains the entire k-th skyline layer, $\mathcal{D}_{\mathcal{I}}[k]$. According to Bentley et al. [27], for n uniformly-distributed d-dimensional data points (items), the expected size of the skyline (1st layer) is $|\mathcal{D}_{\mathcal{I}}[0]| = \Theta(\frac{ln^{d-1}n}{(d-1)!})$. UPDATETOPK() will be called at most k times, each time fetching (at least) 1 new item, meaning that we will evaluate $O(k * \frac{ln^{d-1}n}{(d-1)!})$ items. For each candidate, we need to iterate over the $|\mathcal{Q}|$ queries and update the $TopK$ structure with the new score, which takes $O(logk)$ time using a Red-Black tree, for a total complexity of $O(|\mathcal{Q}| * k * logk * \frac{ln^{d-1}n}{(d-1)!})$. However, as we discuss next, this is a pessimistic analysis based on the naive assumption that each of the k layers will be considered entirely.

In practice, with the exception of the first layer, we only need to check a small fraction of the candidates in the skyline layers. For instance, in a uniform distribution with consecutive layers of similar size, the number of points to be considered will be in the order of k, since links will be evenly distributed among the skyline points. As we only expand the top-k items in each step, approximately k new items will be evaluated next, making the cost of UPDATETOPK() in subsequent calls $O(|\mathcal{Q}| * k * logk)$. Given that this cost is paid for each of the (at most) $k-1$ iterations after the first one, the total cost becomes $O(|\mathcal{Q}| * (k^2 + \frac{ln^{d-1}n}{(d-1)!}) * logk)$. As we show in our experiments, the actual distributions found in real datasets allow for much faster computations. In the following section, we describe several speed-ups that can achieve significant savings in practice.

In terms of space, the UPDATETOPK() method accepts $|\mathcal{X}|$ items as input and operates on that set alone, resulting in $O(|\mathcal{X}|)$ space. For each item in $\mathcal{X}$, we maintain its lower and upper bound, which is still $O(|\mathcal{X}|)$. As we iterate over the queries, we update those values and discard items, reducing the required space, bringing it closer to $O(k)$. Since

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

the $TopK$ structure always contains k entries, the space of CMiner is determined by $\mathcal{X}$, which is at its maximum when we retrieve the first skyline layer (line 7). Our assumption that the primary skyline fits in memory is reasonable and shared by prior works on skyline algorithms [24].

4 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 $\mathcal{Q}$ 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:

$$low(j) \leftarrow low(j) + p(q) \times V^q_{i,j} \tag{7}$$

$$up(j) \leftarrow up(j) - p(q) \times V^q_{i,i} + p(q) \times V^q_{i,j} \tag{8}$$

By expanding the sequences and using the initial values $low(j) = 0$ and $up(j) = C_{\mathcal{F}}(i,i)$, we can re-write the bounds:

$$low^m(j) = \sum_1^m p(q_m) \times V^{q_m}_{i,j}$$

$$up^m(j) = C_{\mathcal{F}}(i,i) - \sum_1^m p(q_m) \times V^{q_m}_{i,i} + \sum_1^m p(q_m) \times V^{q_m}_{i,j},$$

where $low^m(j)$ and $up^m(j)$ are the values of the bounds after considering the m_{th} query q_m. We can then define a recursive function $T(j) = up(j) - low(j)$ as follows:

$$T(j) \leftarrow T(j) - p(q) \times V^q_{i,i} \tag{9}$$

$T(j)$ captures the margin of error for the competitiveness between the item of interest i and a candidate j. As more queries are evaluated and the two bounds are updated, the margin decreases. Finally, it becomes equal to zero when we have the final $C_{\mathcal{F}}(i,j)$ score. We hypothesize that the ability to minimize this margin faster can increase the number of pruned candidates due to the existence of stricter bounds in early iterations. Given Eq. 7 and 8, the value of $T(j)$ after considering m queries can be re-written as follows:

$$T^m(j) = C_{\mathcal{F}}(i,i) - \sum_{\ell=1}^m p(q_\ell) \times V^{q_\ell}_{i,i}, \tag{10}$$

where q_ℓ is the ℓ_{th} query processed by the algorithm. Given Eq. 10, it is clear that we can optimally minimize the margin between the lower and upper bounds on the competitiveness of a candidate by processing queries in decreasing order of their $p(q) \times V^q_{i,i}$ values. We refer to this ordering scheme as COV. We evaluate the computational savings achieved by COV in Section 5.4 of our experiments, where we also compare it with alternative approaches.

4.2 Improving UPDATETOPK() and GETSLAVES()

In this section we describe several improvements to the CMiner's two main routines. We implement all of these improvements into an enhanced algorithm, which we refer to as CMiner++. We include this version in our experimental evaluation, where we compare its efficiency with that of CMiner, as well as to that of other baselines.

Even though CMiner can effectively prune low quality candidates, a major bottleneck within the UPDATETOPK() function is the computation of the final competitiveness score between each candidate and the item of interest i (lines 41-46). Speeding up this computation can have a tremendous impact on the efficiency of our algorithm. Next, We illustrate this with an example. Assume that items are defined in a 4-dimensional space with features f_1, f_2, f_3, f_4. Without loss of generality, we assume that all features are numeric. We also consider 3 queries $q_1 = (f_1, f_2, f_3)$, $q_2 = (f_2, f_3, f_4)$ and $q_3 = (f_2, f_4)$, with probabilities $w(q_1)$, $w(q_2)$, and $w(q_3)$, respectively. In order to compute the competitiveness between two items i and j, we need to consider all queries and, according to Eq. 5, compute $V^{q_1}_{i,j} = V^{f_1}_{i,j} \times V^{f_2}_{i,j} \times V^{f_3}_{i,j}$, $V^{q_2}_{i,j} = V^{f_2}_{i,j} \times V^{f_3}_{i,j} \times V^{f_4}_{i,j}$, and $V^{q_3}_{i,j} = V^{f_2}_{i,j} \times V^{f_4}_{i,j}$. Given that the three items include common sequences of factors, we would like to avoid repeating their computation, when possible. First, we sort all features according to their frequency in the given set of queries. In our example, the order is: f_2, f_3, f_4, f_1. In this order, (f_2, f_3) becomes a common *prefix* for q_1 and q_2, whereas f_2 is a common *prefix* for *all* 3 queries. We then build a *prefix-tree* to ensure that the computation of such common prefixes is only completed once. For instance, the computation of $V^{f_2}_{i,j} \times V^{f_3}_{i,j}$ is done only once and used for both q_1 and q_2. The tree is used in lines 41-46 of CMiner to expedite the computation of the competitiveness between the item of interest and the remaining candidates in $\mathcal{X}$. This improvement is inspired by Huffman encoding, whereby frequent symbols (features in our case) are closer to the root, so that they are encoded with fewer bits. Note that Huffman encoding is optimal if the symbols independent of each other, as is the case in our own setting.

The GETSLAVES() method is used to extend the set of candidates by including the items that are dominated by those in a provided set (lines 7 and 15). Henceforth, we refer to this as the *dominator set*. A naive implementation would include all items that are dominated by at least one item in the dominator set. However, as stated in Lemma 1, if an item j is dominated by an item j', then the competitiveness of j with any item of interest cannot be higher than that of j'. This implies that items that are dominated by the k-th best item of the given set will have a competitiveness score lower than the current k-th score and will thus not be included in the final result. Therefore, we only need to expand the top $k-1$ items and only those that have not been expanded already during a previous iteration. In addition, the GETSLAVES() method can be further improved by using the lower bound LB (the score of the k-th best candidate) as follows: instead of returning all the items that are dominated by those in the dominator set, we only have to consider a dominated item j if $C_{\mathcal{F}}(j,j) > LB$. This is due to the fact that the competitiveness between i and j is upper-bounded by the minimum coverage achieved by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

either of the two items (over all queries), i.e., $C_{\mathcal{F}}(i,j) \leq \min(C_{\mathcal{F}}(i,i), C_{\mathcal{F}}(j,j))$. Therefore, an item with a coverage $\leq LB$ cannot replace any of the items in the current $TopK$.

5 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.

5.1 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 in Section 2.2 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.

TABLE 4: Dataset Statistics

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

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

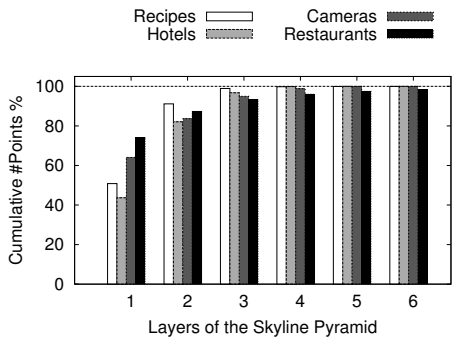


Fig. 4: Cumulative distribution of items across the first 6 layers of the skyline pyramid.

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.

Baselines: We compare CMiner with two baselines. The *Naive* basline, is the brute-force approach described in Section 3. The second is a clustering-based approach that first iterates over every query q and identifies the set of items that have the same value assignment for the features in q and places them in the same group. The algorithm then iterates over the reported groups and updates the pairwise coverage $\mathcal{V}^q_{i,j}$ for the item of interest i and an arbitrary item j from each group (it can be any item, since they all have the same values with respect to q). The computed coverage is then used to update the competitiveness of all the items in the group. The process continues until the final competitiveness scores for all items have been computed. Assuming that we have a collection of items $\mathcal{I}$, a set of queries $\mathcal{Q}$, and at most M groups per query, the complexity is O($|\mathcal{I}|$ * M * $|\mathcal{Q}|$).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

Obviously, when each group is a singleton, the algorithm is equivalent to the brute-force approach. We refer to this technique as *GMiner*. We also evaluate our enhanced CMiner++ algorithm, that implements the speedups of Section 4.2. Finally, unless stated otherwise, we always experiment with $k \in \{3, 10, 50, 150, 300\}$.

5.2 Evaluating comparative methods

Previous work on competitor mining has been based on comparative evidence between two items, found in different types of text data. However, these approaches are based on the assumption that such comparative evidence can be found in abundance in the available data. In this experiment, we evaluate this assumption on our four datasets. For every pair of items in each dataset, we report (1) the number of reviews that mention both items and (2) the number of reviews that include a direct comparison between the two items. We extract such comparative evidence based on the union of "competitive evidence" lexicons used by previous work [8], [9], [10], [11], [12], [13]. Given two items i and j, the lexicon includes the following comparative patterns: *i in contrast to j, i unlike j, i compared with j, compare i to j, i beat(s) j, i exceeds j, i outperform(s) j, prefer i to j, i than j , i same as j, i similar to j, i superior to j, i better than j, i worse than, i more than j, i less than j, i vs j*. We present the results in Table 5, in which we report the average number of findings for each pair of items in each dataset.

TABLE 5: Evidence on Comparative Methods

	Co-occurrence	Comparative
Cameras	1.7	1.2
Hotels	0.06	0.02
Restaurants	0.09	0.04
Recipes	0	0

The results verify that methods based on comparative evidence are completely ineffective in many domains. In fact, even for CAMERAS, the dataset with the largest count, evidence was limited to a very small number of pairs. Specifically, the expected number of times that any two specific cameras appear together in the same review is 1.7. In addition, only 1.2 of these co-occurrence were actually comparative, a number that is far too low to allow for a confident evaluation of competitiveness. This demonstrates the sparseness of comparative evidence in real data, which greatly limits the applicability of any approach that is based on such evidence. These findings further motivate our work, which has no need for this type of information.

5.3 Computational Time

In this experiment we compare the speed of CMiner with that of the two baselines (Naive and GMiner), as well as with that of the enhanced CMiner++ algorithm. Specifically, we use each algorithm to compute the set of top-k competitors for each item in our datasets. The results are shown in Figure. 5. Each plot reports the average time, in seconds, per item (y-axis) against the various k values (x-axis).

The figures motivate some interesting observations. First, the Naive algorithm consistently reports the same computational time regardless of k, since it naively computes the competitiveness of every single item in the corpus with respect to the target item. Thus, any trivial variations in the required time are due to the process of maintaining the top-k set. In general, Naive is outperformed by the two other algorithms, and is only competitive for very large values of k for the HOTELS dataset. The latter case can be attributed to the small number of queries and items included in this dataset, which limit the ability of more sophisticated algorithms to significantly prune the space when the number of required competitors is very large.

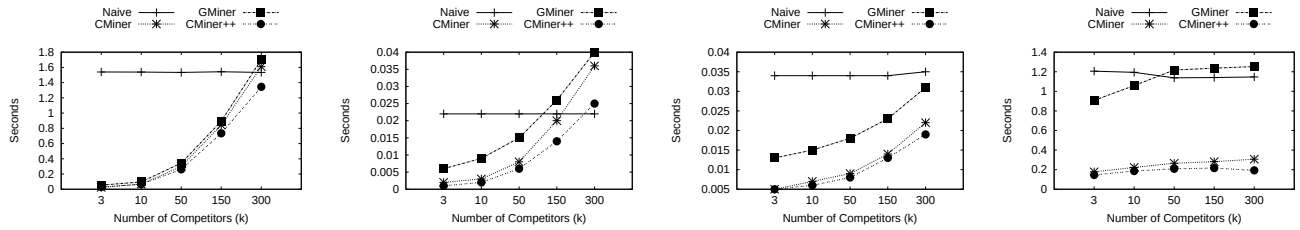
For the CAMERAS dataset, CMiner and GMiner, exhibit almost identical running times. This is due to (1) the very large number of distinct queries for this dataset (14779), which serves as a computational bottleneck for CMiner and (2) the highly clustered structure of the item population, which includes 579 items. A deeper analysis reveals that GMiner identifies and average of 443.63 item groups (i.e. groups of identical items) per query. This means that the algorithm saves (on expectation) a total of $(579 - 443) \times 14779 = 2009944$ coverage computations per query, allowing it to be competitive to the otherwise superior CMiner. In fact, for the other datasets, CMiner displays a clear advantage. This advantage is maximized for the RECIPES dataset, which is the most populous of the four in terms of included items. The experiment on this dataset also illustrates the scalability of the approach with respect to k. For the HOTELS and RESTAURANTS datasets, even though the computational time of CMiner appears to rise as k increases for the other three datasets, it never goes above 0.035 seconds. For the CAMERAS dataset, the large number of considered queries has an adverse of the scalability of CMiner, since it results in larger number of required computations for larger values of k. This finding motivates us to consider pruning the set of queries by eliminating those that have a low probability. We explore this direction in the experiment presented in Section 5.6.

Finally, we observe that the enhanced CMiner++ algorithm consistently outperformed all the other approaches, across datasets and values of k. The advantage of CMiner++ is increased for larger values of k, which allow the algorithm to benefit from its improved pruning. This verifies the utility of the improvements described in Section 4.2 and demonstrates that effective pruning can lead to a performance that far exceeds the worst-case complexity analysis of CMiner.

5.4 Ordering Efficiency

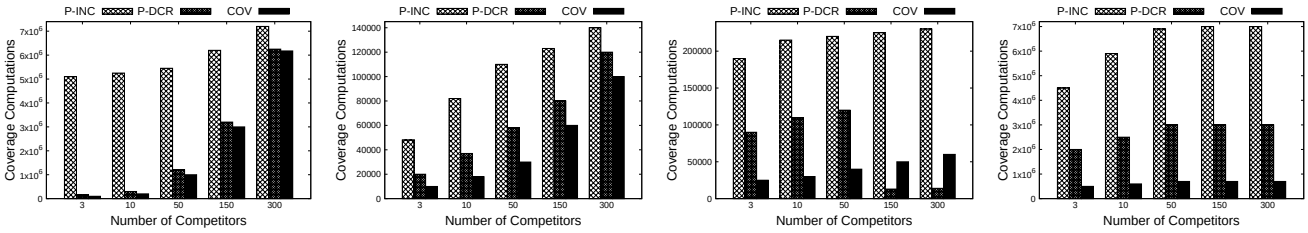
In Section 4.1 we introduced the COV ordering scheme, which determines the processing order of queries by CMiner. Next, we demonstrate COV's superiority over the P-INC and P-DCR oredring schemes, which process queries in increasing and decreasing probability order, respectively. For each approach, we compute (1) the number of pairwise query coverages $V^q_{i,j}$ that need to be computed (line 25 of Algorithm 1) and (2) the number of distinct queries that need to be processed (line 22 of Algorithm 1) to compute the top-k competitors of each item. The results are shown in Figures 6 and 13 (Appendix D). The x-axis of each plot holds the value of k, while the y-axis holds the average number of processed queries / coverages.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 5: Average time (per item) to compute top-k competitors for each dataset.



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 6: Average number of pairwise coverage computations under different ordering schemes.

We observe a consistent advantage for COV, across datasets and values of k. This verifies that COV increases the efficiency of CMiner by rapidly eliminating candidates that fail to cover important queries. We observe that this advantage is more profound for RECIPES and CAMERAS. This is a reasonable outcome, as the computational bottleneck of CMiner lies with the nested loop structure of the UPDATETOPK() routine, which iterates over all queries and over all items for each query (lines 22-39 of Algorithm 1). Therefore, datasets with a large number of items (RECIPES) or queries (CAMERAS) can benefit the most from the ability of CMiner to quickly eliminate candidates.

5.5 Pruning Efficiency

Much of CMiner's efficiency stems from its ability to discard or selectively evaluate candidates. We illustrate this in Figure 7. The figure includes one set of bars for each dataset, with each bar representing a different value of k ($k \in \{3, 10, 50, 150, 300\}$, in the order shown).

The white portion of each bar (post-pruned) represents the average number of items pruned within UPDATETOPK() (line 28). There, an item is pruned if, as we go over the set of queries Q, its upper bound reaches a value that is lower than LB (the lowest competitor in the current top-K). The black portion of each bar (pre-pruned) represents the average number of items that were never added to the candidate set $\mathcal{X}$ because their best-case scenario (self coverage) was apriori worse than LB. Therefore, they can be eliminated and we do not have to consider their competitiveness in the context of the queries. We explain this mechanism in detail in the final paragraph of Section 4.2. Finally, the pattern-filled portion (unpruned) at the top of each bar refers to the average number of items that were fully evaluated in their entirety (i.e. for all queries). We observe that the vast majority of candidates is eliminated by one of the two types of pruning that we consider here. The high number of pre-pruned queries is particularly encouraging, as it implies the highest computational savings. Finally, it is important to note that these findings are consistent across datasets.

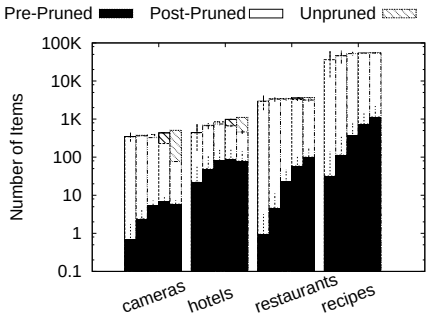


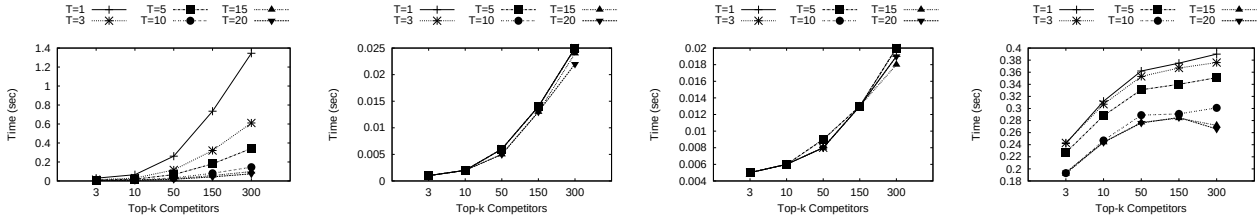
Fig. 7: Pruning Effectiveness

5.6 Reducing the Number of Considered Queries

By default, CMiner considers all queries with a non-zero probability, with each query representing a different market segment. As larger segments are more likely to contribute to the competitiveness of two items, an intuitive way to speed up the algorithm is to ignore low-probability queries, i.e. queries with frequency lower than a threshold T. We repeat our experiment with $T \in \{1, 3, 5, 10, 15, 20\}$ and record the time required for each combination of (k, T). The results are shown in Figure 8, which includes a plot for each dataset. The x-axis of each plot holds the different values of k, while the y-axis holds the required computational time in seconds. The plot includes one line for each value of T. We also evaluate the quality of the results, using Kendall's τ coefficient [29] to compare the produced top-k lists with the respective exact solution (i.e. $T = 1$). The results are shown in Figure 9. For the plots in this figure, the y-axis holds the computed Kendall τ values. For all plots, we report the average values computed over all the items in each dataset.

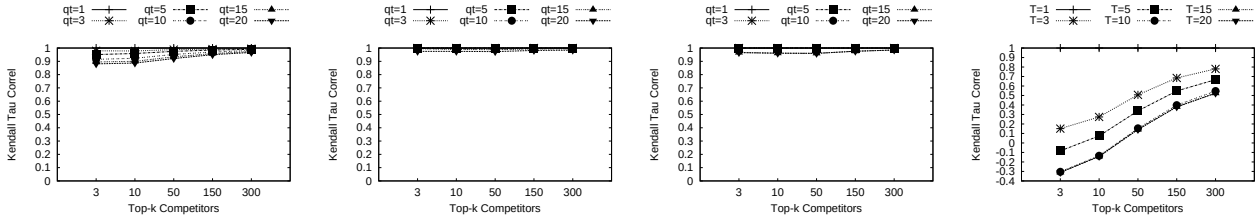
For hotels and restaurants, query elimination does not yield significant gains. On the other hand, for cameras, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 8: Computational times of CMiner for different values of the k and T parameters.



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 9: Kendall Tau values achieved by CMiner for different values of the k and T parameters.

runtime of CMiner drops as T increases. In fact, the algorithm achieves increasingly higher savings as k increases. For recipes, we observe significant savings even for small values of T and k. A deeper study of the data reveals that these discrepancies can be attributed to the number of queries that they include. Specifically, as shown in Table 4, the number of queries for the cameras dataset is two orders of magnitude larger than that for the hotels and restaurants datasets. As a result, CMiner has a low computational cost for restaurants and hotels, even when all queries are considered. However, for cameras, the large number of queries serves as a significant computational bottleneck, which is relieved as the value of T becomes higher and less queries need to be considered. For the recipes dataset, the improvement can be attributed to the number of items (100K): By reducing the number of queries, processed in the external loop of CMiner's UPDATETOPK() routine (line 23 Algorithm 1), we also significantly reduce the executions of the inner loop (line 25), which iterates over the large set of items in this dataset.

The second observation is that, for all datasets except recipes, CMiner achieves near-perfect results even for larger values of T. This is based on the observed values of the Kendall τ coefficient, which was consistently above 0.9 for all evaluated combinations of the k and T parameters. This is an encouraging finding, since it reveals a highly appealing and practical tradeoff between the computational efficiency and quality of CMiner. In addition, it is important to note that the practice of reducing the size of number of considered queries does not require any modifications to the algorithm itself and can thus be applied with minimum effort. A careful examination of the recipes dataset reveals that the low correlation values can be attributed to the fact that most queries have a low frequency and, in fact, their frequency distribution is nearly uniform. As a result, even a low value for the T threshold eliminates a large number of queries and prevents CMiner from computing the exact solution to the top-k problem. This finding reveals that, by studying the frequency distribution of the queries in a given dataset, we can make an informed decision on whether or not eliminating low-frequency queries is advisable, as well as on what the value of the threshold should be.

5.7 Convergence of Query Probabilities

In Section 2.2 we described the process of estimating the probability of each query by mining large datasets of customer reviews. The validity of this approach is based on the assumption that the number of available reviews is sufficient to allow for confident estimates. Next, we evaluate this assumption as follows. First, we merge all the reviews in each dataset into a single set, sort them by their submission date, and split the sorted sequence into fixed-size segments. We then iteratively append segments to the review corpus $\mathcal{R}$ considered by Eq. 6 and re-compute the probability of each query in the extended corpus. The vector of probabilities from the i_{th} iteration is then compared with that from the $(i-1)_{th}$ iteration via the L1 distance: the sum of the absolute differences of corresponding entries (i.e. the two estimates for the same query in both vectors). We apply the process for segments of 25 reviews. The results are shown in Figure 10. The x-axis of each plot includes the number of reviews, while the y-axis is the respective L1 distance.

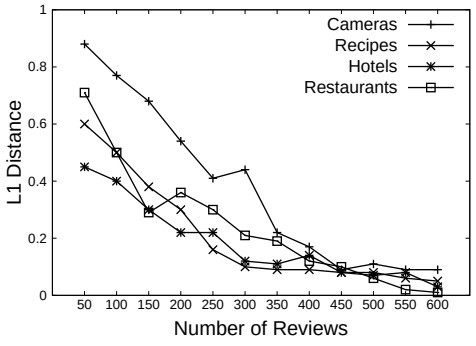


Fig. 10: Convergence of query probabilities.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

Based on our results, we see that all the datasets exhibited near identical trends. This is an encouraging finding with useful implications, as it informs us that any conclusions we draw about the convergence of the computed probabilities will be applicable across domains. Second, the figures clearly demonstrate the convergence of the computed probabilities, with the reported L1 distance dropping rapidly to trivial levels below 0.2, after the consideration of less than 500 reviews. The convergence of the probabilities is an especially encouraging outcome that (i) reveals a stable categorical distribution for the preferences of the users over the various queries, and (ii) demonstrates that only a small seed of reviews, that is orders of magnitude smaller than the thousands of reviews available in each dataset, is sufficient to achieve an accurate estimation of the probabilities.

5.8 A User Study

In order to validate our definition of competitiveness, we conduct a user study as follows. First, we select 10 random item from each of our 4 datasets. We refer to these 10 items as the *seed*. For each item i in the seed, we compute its competitiveness with every other item in the corpus, according to our definition. As a baseline, we also rank all the items in the corpus based on their distance to i within the feature-space of each dataset. The intuition behind this approach is that highly similar items are also likely to be competitive. The L1 distance was used for numeric and ordinal features, and the Jaccard distance was used for categorical attributes. We refer to this as the NN approach (i.e. Nearest Neighbor). We then chose the two items with the highest score, the two items with the lowest score, and two items from the middle of the ranked list. This was repeated for both approaches, for a total of 12 candidates per item in the seed (6 per approach).

We then created a user study on the online survey-site kwiksurveys.com/. In total, the survey was taken by 20 different annotators. Each of the 10 seed items was paired with each of its 12 corresponding candidates, for a total of 120 different pairs. The pairs were shown to the annotators in a randomized order. Annotators had access to a table with the values of each item for every feature, as well as a link to the original source. For each pair, each annotator was asked whether she would consider buying the candidate instead of the seed item. The possible answers were "YES", "NO" and "NOT SURE". Table 8 in Appendix F shows an example of the pairs that were shown to the annotators.

Figure 11 contains the results of the user study. The figure shows 3 pairs of bars. The left bar of each pair corresponds to our approach, while the right bar to the NN approach. The leftmost pair represents the user responses to the top-ranked candidates for each approach. The pair in the middle represents the responses to the candidates ranked in the middle, and, finally, the pair on the right represents the responses to the bottom-ranked candidates.

Each bar in Figure 11 captures the fraction of each of the three possible responses. The lower, middle, and upper part of the bar represent the "YES", "NO" and "NOT SURE" responses, respectively. For example, the first bar on the left reveals that about 90% of the annotators would consider our top-ranked candidates as a replacement for the seed item. The remaining 10% was evenly divided between the "NO" and "NOT SURE" responses.

The figure motivates multiple relevant observations. First, we observe that the vast majority of the top-rank competitors proposed by our approach were verified as likely replacements for the seed item. These are thus verified as strong competitors that could deprive the seed item from potential customers and decrease its market share. On the other hand, the top-ranked candidates of NN were often rejected by the users, who did not consider these items to be competitive. Both approaches exhibited their worst results for the RECIPES dataset, even though the "YES" percentage of the top-ranked items by our method was almost twice that of NN. The difficulty of the recipes domain is intuitive, as users are less used to consider recipes in a competitive setting. The middle-ranked candidates of our approach attracted mixed responses from the annotators, indicating that it was not trivial to determine whether the item is indeed competitive or not. An interesting observation is that, for some of our datasets, the middle-ranked candidates of NN were more popular than its top-ranked ones, which implies that this approach fails to emulate the way the users perceive the competitiveness between two items. The bottom-ranked candidates of our approach were consistently rejected, verifying their lack of competitiveness to the seed item. The bottom-ranked items by the NN approach were also frequently rejected, indicating that it is easier to identify items that are *not* competitive to the target.

Finally, to further illustrate the difference between our competitiveness model and the similarity-based approach, we conducted the following quantitative experiment. For each item i in a dataset, we retrieve its 300 top-ranked competitors, as ordered by each of the two methods. We then compute the Kendall τ and overlap of the two lists. We report the average of these two quantities over all items in the dataset. The results in Table 6 demonstrate that the rankings of the two techniques are significantly different both in their ordering and in the items that they contain.

In conclusion, the survey and our qualitative analysis validate our definition of the competitiveness and demonstrates that similarity is not an appropriate proxy for competitiveness. These results complement the discussion that we presented in Section 2, which includes a realistic example of the shortcomings of the similarity-based paradigm.

TABLE 6: Comparing Competitiveness and NN.

	Kendall τ		#Commons	
	Avg	**StdDev**	**Avg**	**StdDev**
Cameras	-0.0658	0.281	181.886	49.922
Hotels	0.0438	0.2074	185.786	32.7872
Restaurants	-0.36	0.1532	82.523	32.737
Recipes	-0.642	0.133	14.524	30.804

6 Related Work

This paper builds on and significantly extends our preliminary work on the evaluation of competitiveness [30]. To the best of our knowledge, our work is the first to address the evaluation of competitiveness via the analysis of large unstructured datasets, without the need for direct comparative evidence. Nonetheless, our work has ties to previous work from various domains.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

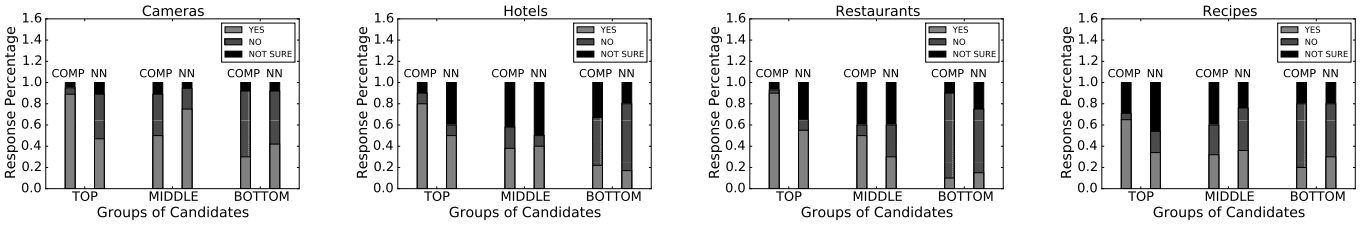


Fig. 11: Results of the user study comparing our competitiveness paradigm with the Nearest-Neighbor approach.

Managerial Competitor Identification: The management literature is rich with works that focus on how managers can *manually* identify competitors. Some of these works model competitor identification as a mental categorization process in which managers develop mental representations of competitors and use them to classify candidate firms [3], [6], [31]. Other manual categorization methods are based on market- and resource-based similarities between a firm and candidate competitors [1], [5], [7]. Finally, managerial competitor identification has also been presented as a sense-making process in which competitors are identified based on their potential to threaten an organizations identity [4].

Competitor Mining Algorithms: Zheng et al. [32] identify key competitive measures (e.g. market share, share of wallet) and showed how a firm can infer the values of these measures for its competitors by mining (i) its own detailed customer transaction data and (ii) aggregate data for each competitor. Contrary to our own methodology, this approach is not appropriate for evaluating the competitiveness between any two items or firms in a given market. Instead, the authors assume that the set of competitors is given and, thus, their goal is to compute the value of the chosen measures for each competitor. In addition, the dependency on transactional data is a limitation we do not have.

Doan et al. explore user visitation data, such as the geo-coded data from location-based social networks, as a potential resource for competitor mining [33]. While they report promising results, the dependence on visitation data limits the set of domains that can benefit from this approach.

Pant and Sheng hypothesize and verify that competing firms are likely to have similar web footprints, a phenomenon that they refer to as *online isomorphism* [34]. Their study considers different types of isomorphism between two firms, such as the overlap between the in-links and out-links of their respective websites, as well as the number of times that they appear together online (e.g. in search results or new articles). Similar to our own methodology, their approach is geared toward pairwise competitiveness. However, the need for isomorphism features limits its applicability to firms and make it unsuitable for items and domains where such features are either not available or extremely sparse, as is typically the case with co-occurrence data. In fact, the sparsity of co-occurrence data is a serious limitation of a significant body of work [8], [10], [11], [35] that focuses on mining competitors based on comparative expressions found in web results and other textual corpora. The intuition is that the frequency of expressions like "Item A is better than Item B" "or item A Vs. Item B" is indicative of their competitiveness. However, as we have already discussed in the introduction, such evidence is typically scarce or even non-existent in many mainstream domains. As a result, the applicability of such approaches is greatly limited. We provide empirical evidence on the sparsity of co-occurrence information in our experimental evaluation.

Finding Competitive Products: Recent work [36], [37], [38] has explored competitiveness in the context of *product design*. The first step in these approaches is the definition of a dominance function that represents the value of a product. The goal is then to use this function to create items that are not dominated by other, or maximize items with the maximum possible dominance value. A similar line of work [39], [40] represents items as points in a multidimensional space and looks for subspaces where the appeal of the item is maximized. While relevant, the above projects have a completely different focus from our own, and hence the proposed approaches are not applicable in our setting.

Skyline computation: Our work leverages concepts and techniques from the extensive literature on skyline computation [24], [25], [41]. These include the dominance concept among items, as well as the construction of the skyline pyramid used by our CMiner algorithm. Our work also has ties to the recent publications in *reverse skyline* queries [42], [43]. Even though the focus of our work is different, we intend to utilize the advances in this field to improve our framework in future work.

7 Conclusion

We presented a formal definition of competitiveness between two items, which we validated both quantitatively and qualitatively. Our formalization is applicable across domains, overcoming the shortcomings of previous approaches. We consider a number of factors that have been largely overlooked in the past, such as the position of the items in the multi-dimensional feature space and the preferences and opinions of the users. Our work introduces an end-to-end methodology for mining such information from large datasets of customer reviews. Based on our competitiveness definition, we addressed the computationally challenging problem of finding the top-k competitors of a given item. The proposed framework is efficient and applicable to domains with very large populations of items. The efficiency of our methodology was verified via an experimental evaluation on real datasets from different domains. Our experiments also revealed that only a small number of reviews is sufficient to confidently estimate the different types of users in a given market, as well the number of users that belong to each type.

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 network-based 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.

[11] S. Bao, R. Li, Y. Yu, and Y. Cao, "Competitor mining with the web," *IEEE Trans. Knowl. Data Eng.*, 2008.

[12] G. Pant and O. R. L. Sheng, "Avoiding the blind spots: Competitor identification using web text and linkage structure," in *ICIS*, 2009.

[13] D. Zelenko and O. Semin, "Automatic competitor identification from public information sources," *International Journal of Computational Intelligence and Applications*, 2002.

[14] R. Decker and M. Trusov, "Estimating aggregate consumer preferences from online product reviews," *International Journal of Research in Marketing*, vol. 27, no. 4, pp. 293–307, 2010.

[15] C. W.-K. Leung, S. C.-F. Chan, F.-L. Chung, and G. Ngai, "A probabilistic rating inference framework for mining user preferences from reviews," *World Wide Web*, vol. 14, no. 2, pp. 187–215, 2011.

[16] K. Lerman, S. Blair-Goldensohn, and R. McDonald, "Sentiment summarization: evaluating and learning user preferences," in *ACL*, 2009, pp. 514–522.

[17] E. Marrese-Taylor, J. D. Velásquez, F. Bravo-Marquez, and Y. Matsuo, "Identifying customer preferences about tourism products using an aspect-based opinion mining approach," *Procedia Computer Science*, vol. 22, pp. 182–191, 2013.

[18] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant, "Range queries in olap data cubes," in *SIGMOD*, 1997, pp. 73–88.

[19] Y.-L. Wu, D. Agrawal, and A. El Abbadi, "Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes," in *CIKM*, ser. CIKM '00, 2000, pp. 414–421.

[20] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi, "Approximating multi-dimensional aggregate range queries over real attributes," in *SIGMOD*, 2000, pp. 463–474.

[21] M. Muralikrishna and D. J. DeWitt, "Equi-depth histograms for estimating selectivity factors for multi-dimensional queries," in *SIGMOD*, 1988, pp. 28–36.

[22] N. Thaper, S. Guha, P. Indyk, and N. Koudas, "Dynamic multidimensional histograms," in *SIGMOD*, 2002, pp. 428–439.

[23] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *AcM sIGMoD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[24] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[25] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," ser. SIGMOD '03.

[26] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos, "Skyline ranking à la IR," in *ExploreDB*, 2014, pp. 182–187.

[27] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson, "On the average number of maxima in a set of vectors and applications," *J. ACM*, 1978.

[28] X. Ding, B. Liu, and P. S. Yu, "A holistic lexicon-based approach to opinion mining," ser. WSDM '08.

[29] A. Agresti, *Analysis of ordinal categorical data*. John Wiley & Sons, 2010, vol. 656.

[30] T. Lappas, G. Valkanas, and D. Gunopulos, "Efficient and domain-invariant competitor mining," in *SIGKDD*, 2012, pp. 408–416.

[31] J. F. Porac and H. Thomas, "Taxonomic mental models in competitor definition," *Academy of Management Review*, vol. 15, no. 2, pp. 224–240, 1990.

[32] Z. Zheng, P. Fader, and B. Padmanabhan, "From business intelligence to competitive intelligence: Inferring competitive measures using augmented site-centric data," *Information Systems Research*, vol. 23, no. 3-part-1, pp. 698–720, 2012.

[33] T.-N. Doan, F. C. T. Chua, and E.-P. Lim, "Mining business competitiveness from user visitation data," in *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*. Springer, 2015, pp. 283–289.

[34] G. Pant and O. R. Sheng, "Web footprints of firms: Using online isomorphism for competitor identification," *Information Systems Research*, vol. 26, no. 1, pp. 188–209, 2015.

[35] K. Xu, S. S. Liao, J. Li, and Y. Song, "Mining comparative opinions from customer reviews for competitive intelligence," *Decis. Support Syst.*, 2011.

[36] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating competitive products," *PVLDB*, vol. 2, no. 1, pp. 898–909, 2009.

[37] Q. Wan, R. C.-W. Wong, and Y. Peng, "Finding top-k profitable products," in *ICDE*, 2011.

[38] Z. Zhang, L. V. S. Lakshmanan, and A. K. H. Tung, "On domination game analysis for microeconomic data mining," *ACM Trans. Knowl. Discov. Data*, 2009.

[39] T. Wu, D. Xin, Q. Mei, and J. Han, "Promotion analysis in multi-dimensional space," *PVLDB*, 2009.

[40] T. Wu, Y. Sun, C. Li, and J. Han, "Region-based online promotion analysis," in *EDBT*, 2010.

[41] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," ser. VLDB, 2002.

[42] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse top-k queries," in *ICDE*, 2010.

[43] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis, "Identifying the most influential data objects with reverse top-k queries," *PVLDB*, 2010.

[44] K. Hose and A. Vlachou, "A survey of skyline processing in highly distributed environments," *The VLDB Journal*, vol. 21, no. 3, pp. 359–384, 2012.

Georgios Valkanas George Valkanas received his PhD in Computer Science from the University of Athens, Greece in 2014. Currently, he is a Research Engineer at Detetica, NYC, NY. Prior to that, he was a Post-Doctoral Researcher in the School of Business at Stevens Institute of Technology. His research interests include web and text mining, information retrieval and extraction, with a high interest in user preferences and behavior.

Theodoros Lappas Theodoros Lappas received his PhD in Computer Science from the University of California, Riverside in 2011. Currently, he is a an assistant professor in the School of Business at Stevens Institute of Technology. He joined Stevens after completing his post-doctoral study in the Computer Science Department of Boston University. His research interests include data mining and machine learning, with a particular focus on scalable algorithms for mining text and graphs. He is a member of the IEEE.

Dimitrios Gunopulos Dimitrios Gunopulos received his PhD degree from Princeton University in 1995. Currently, he is a Professor in the Department of Informatics and Telecommunications, University of Athens, Greece. He has held positions as a postoctoral fellow at MPII, Germany, research associate at IBM Almaden, and assistant, associate, and professor of computer science and engineering at UC-Riverside. His research interests include data mining, knowledge discovery in databases, databases, sensor networks, and peer-to-peer systems. He is a member of the IEEE.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

APPENDIX A
EXTENSION OF PREVIOUS WORK

This manuscript builds on our previous preliminary work on the evaluation of competitiveness, published in the ACM SIGKDD 2012 conference. In addition to including a significantly improved write-up and a much more extensive discussion of both our methods and results, the submitted manuscript extends our previous work by introducing:

1) An updated motivation and description of the problems addressed by our work, including new informative examples and improved notation (Sections 1 and 2).
2) An extended section on related work, including a discussion of papers from management and marketing science, as well as recent papers published after the conference version of this manuscript (Section 6)
3) A formal proof of Lemma 1, which is critical for the proposed methodology and was not included in the conference version (Section 3).
4) An extended discussion of the time and space complexity of the CMiner algorithm (Section 3).
5) A new algorithm, titled CMiner++. This algorithm, which is now included in our experimental evaluation, implements a number of computational speedups that increase the search and pruning capabilities of the previously proposed CMiner algorithm and increase its scalability (Section 4.2).
6) A new experiment that includes the results of the COV ordering scheme (named IC in the conference version) on all four datasets considered by our evaluation, as well as a comparison with other alternatives. The experiment evaluates the scheme in terms of both the number of required coverage computations and the number of queries that need to be considered, for a total of 8 plots. The conference version only included the results for the coverage computations for a single dataset (Section 5.4).
7) A new experiment that evaluates the ability of the proposed algorithm to achieve high-quality results, even when only a subset of the queries is considered. The experiment includes an evaluation of both the quality of the results and the computational savings that are recorded in such cases (Section 5.6).
8) A new experiment that evaluates the ability of the proposed algorithm to achieve high-quality results, even when only a subset of the candidate competitors is considered. The experiment includes an evaluation of both the quality of the results and the computational savings that are recorded in such cases (Section E).
9) A new experiment that evaluated the convergence of the query probabilities required by our methodology, for all four datasets (Section 5.7).
10) A comparison with previous work on mining competitors based on competitive evidence from text (Section 5.2).
11) An extended user study that includes results on all 4 datasets included in our evaluation: RECIPES, CAMERAS, HOTELS, and RESTAURANTS (Section 5.8).
12) A new experiment that quantitatively evaluates the pruning capability of the CMiner algorithm (Section 5.5).
13) A new experiment that illustrates the difference between competitiveness and similarity (added to Section 5.8).
14) An extension of our definition of competitiveness that considers (i) the non-uniformity of the distribution of a user's interest within the value space of each feature, and (ii) the differences in importance among the features included in the same user query (Section 2.3)

APPENDIX B
PROOF OF LEMMA 1

Proof We will prove this by contradiction. Let j be an item that is not included in $\mathcal{Y}$ and has a competitiveness $C_{\mathcal{F}}(i,j)$ with i that is higher or equal to at least one of the items in $\mathcal{Y}$. According to the lemma, we assume that j is not dominated by any of the items in $\mathcal{Y}$. Observe that j cannot be in the skyline, since its competitiveness would then have been enough to also include it in $\mathcal{Y}$. Hence, it is guaranteed to be dominated by at least one of the items in the skyline. This means that there is an item $j' \in Sky(\mathcal{I})$ such that $C_{\mathcal{F}}(i,j') \geq C_{\mathcal{F}}(i,j)$. However, since j has a greater or equal competitiveness to that of at least one of the items in $\mathcal{Y}$, j' will also be included in $\mathcal{Y}$. This leads to a contradiction, since we assumed that none of the items in $\mathcal{Y}$ dominate j.

APPENDIX C
PSEUDOCODE AND COMPLEXITY ANALYSIS FOR `PyramidFinder`

Algorithm 2 `PyramidFinder`

Input: Set of items $\mathcal{I}$
Output: Dominance Pyramid $\mathcal{D}_{\mathcal{I}}$

$\mathcal{D}_{\mathcal{I}}[0] \leftarrow$ *Sky*$(\mathcal{I})$
$\mathcal{Z} \leftarrow \mathcal{I} \setminus$ *Skyline*$(\mathcal{I})$
$level \leftarrow 1$.
while $\mathcal{Z}$ is not empty **do**
  $\mathcal{D}_{\mathcal{I}}[level] \leftarrow$ *Sky*$(\mathcal{Z})$
  for every item $j \in \mathcal{D}_{\mathcal{I}}[level]$ **do**
    for every item $i \in \mathcal{D}_{\mathcal{I}}[level-1]$ **do**
      if i dominates j **then**
        Add a link $i \rightarrow j$
        break
      end if
    end for
  end for
  $\mathcal{Z} \leftarrow \mathcal{Z} \setminus$ *skyline*$(\mathcal{Z})$
  $level \leftarrow level + 1$
end while

Space Complexity: Algorithm 2 describes the creation of the pyramid indexing structure. In principle, the algorithm simply rearranges the original data in skyline layers. Therefore, any additional space required by the structure is due to the edges added between data points of consecutive

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

skyline layers (line 9). As is typical with skyline-related problems [44], this depends greatly on the underlying data distribution. Assuming n data points overall, if there is only one skyline layer, then no additional information is stored. In the worst case, we will need n^2 additional space to store the edges. This is extremely unlikely even in artificially generated distributions [24]. On the other hand, if the data were fully correlated, we would require $\Theta(n)$ space for the edges. This is also unrealistic, as it implies a monopolistic market.

Figure 12 provides an empirical analysis of the number of edges stored per dataset, contrasting it to the theoretical bounds of the worst possible and fully correlated cases, discussed earlier. The space required to store the pyramid structure is well below the limit of the correlated dataset. This is related to both the high-dimensionality of the space and the fact that we are dealing with non-monopolistic settings.

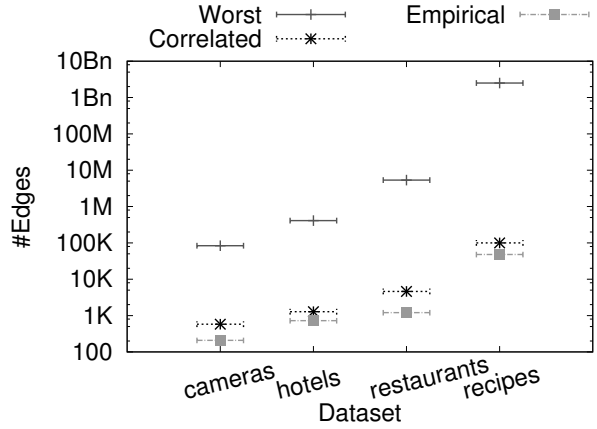


Fig. 12: Comparative

Time Complexity: Computationally, the algorithm has two major parts: (i) finding the skyline layers and (ii) adding the edges. The complexity is $level$ times the complexity of any state-of-the-art algorithm that the practitioner chooses for skyline computation (we use BBS [25]), where $level$ is the number of skyline layers.

The standard approach to adding the edges is $O(d * n^2)$, where d is the dimensionality of the data, due to the dominance checks. Lets assume that m_i points exist in the skyline of the i-th layer extracted by BBS, for a total of λ layers. Adding edges between consecutive layers yields a total runtime of $\sum^{\lambda-1} m_i * m_{i+1}$. From the Cauchy-Schwarz inequality, we know that

$$\sum^{\lambda-1}(m_i * m_{i+1}) \leq \sqrt{(\sum^{\lambda-1} m_i^2) * (\sum^{\lambda-1} m_{i+1}^2)}$$

The left hand side is bounded and the upper bound is achieved when all m_i are equal. Given that m_i's are integers, the true bound may not be always attainable, yet it still holds that the maximum value is obtained when $m_i = \frac{n}{\lambda}$, or as best equally distributed as possible. By plugging this result to the original cost function, the worst runtime becomes a function of the number of layers, given by

$$f(\lambda) = \sum^{\lambda-1}(\frac{n}{\lambda})^2 = n^2 \frac{\lambda - 1}{\lambda^2}$$

The first derivative of this function is $f'(\lambda) = n^2 * (\frac{2-\lambda}{\lambda^3})$, which has a global maximum for $\lambda = 2$, i.e. 2 layers. In that case, $m_1 = m_2 = \frac{n}{2}$@, and the worst runtime becomes $(\frac{n}{2})^2 = O(n^2)$. As each comparison takes time $O(d)$, the overall cost is $O(d * n^2)$. We note that, even though the cost could be improved with an in-memory R-Tree, this might degenerate to a linear scan for very high-dimensional data.

As we show in Table 7, the pyramid construction is not responsible for the main cost of computing the competitors. The runtime was only substantial (67 minutes) for our biggest dataset (100K items), for which we had to use the naive approach for edge addition, due to the high dimensionality ($d = 22$). It was negligible for all the other cases. However, the structure offered significant runtime savings during search, especially for the competitors at the top of the list. Instead, the brute force method that computes competitiveness between all possible item pairs requires $\Theta(2^{|\mathcal{F}|} * n^2 * logk)$, which can be easily dominated by the powerset factor.

TABLE 7: Runtime for creating the Pyramid structure.

	Cameras	Hotels	Restaurants	Recipes
Runtime (sec)	0.087	0.152	0.338	3999.322

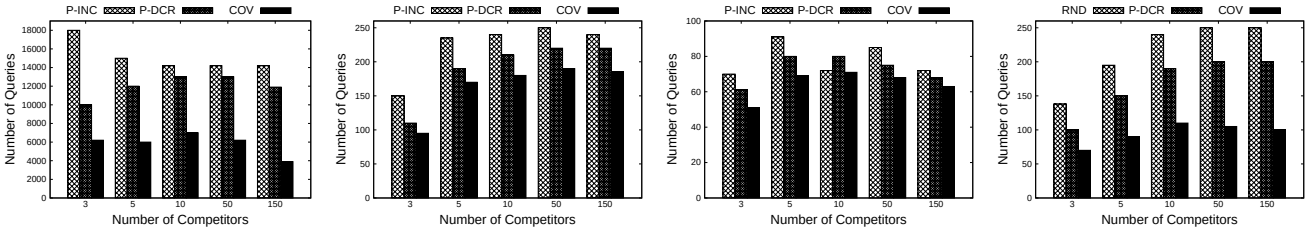
APPENDIX D
REDUCING THE NUMBER OF CONSIDERED QUERIES

We have already discussed, in Section 5.4, how CMiner is able to evaluate only a fraction of the total pairwise coverages, by using the COV ordering scheme introduced in Section 4.1. Figure 13 portrays the average number of queries that we must evaluate for each candidate item *before* we start computing the exact competitiveness score of the items in the $localTopK$ structure (lines 23-40 of Algorithm 1)

Once again, the CAMERAS and RECIPES dataset benefit the most from this ordering scheme, even for large values of k, with a 50% less queries being evaluated. This result also sheds some light to our earlier finding regarding the computed pairwise coverages results. Interestingly, for those two datasets, the number of evaluated queries seems to remain rather constant, despite an increasing k value. The difference in the number of evaluated queries is also visible for the remaining two datasets, although not as prominent, and slightly increasing as k increases too.

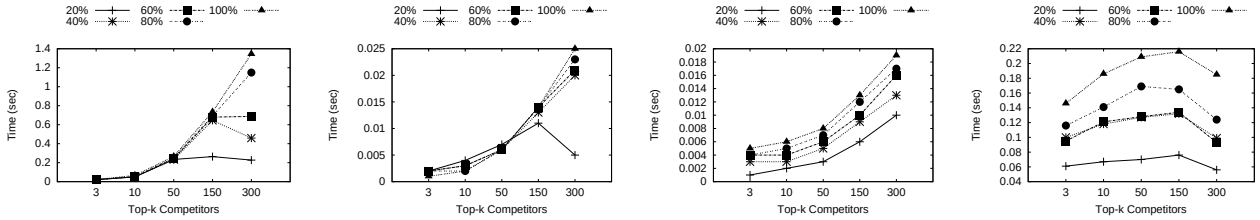
Given those results, it is undeniable that the COV scheme allows us to perform far better pruning, without sacrificing the correctness of the result. Furthermore, all of those results, in addition to the ones in Figure 6, validate our theoretical work regarding the convergence of the approach, discussed in detail in Section 4.1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering



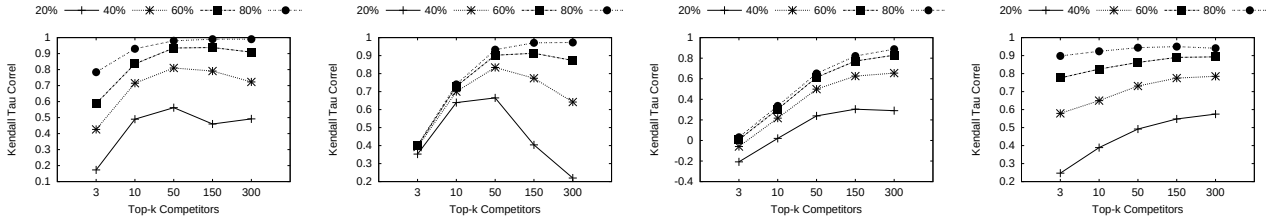
(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 13: Average number of query considerations under different ordering schemes.



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 14: Computational cost of CMiner on just the top $x\%$ of candidates, according to coverage.



(a) Cameras (b) Hotels (c) Restaurants (d) Recipes

Fig. 15: Results of using CMiner on just the top $x\%$ of candidates, according to coverage.

APPENDIX E
REDUCING THE NUMBER OF CANDIDATES

Despite the scalability that CMiner demonstrated in our experiments, it is still a fairly complex approach that considers the entire space of candidates. In this experiment, we verify the need for such an algorithm by evaluating a simpler and much faster approximation approach.

According to the definition given in Eq. 1, the competitiveness between two items is based on the percentage of customers that they can both cover. Intuitively it would be reasonable to expect that candidates with a high coverage are more likely to be included in the top-k competitors of a random item. Thus, we hypothesize that, by focusing exclusively on high-coverage items, we could significantly reduce the number of candidates that need to be considered. We evaluate this hypothesis by comparing the top-k sets retrieved by the CMiner on the full dataset with those retrieved when focusing on just the top $x\%$ of the candidates, as ranked by coverage. The two top-k lists are compared using Kendall's tau correlation coefficient. We repeat the experiment for all four datasets, for $x \in \{20, 40, 60, 80\}$, and for $k \in \{3, 10, 50, 150, 300\}$. We report the average value of the coefficient for each parameter configuration. The results are shown in Figure 15. We also report the computational cost of each configuration in Figure 14.

The figures reveal a number of informative findings. First, as anticipated, higher values of x yield results that are closer to the exact solution. However, for $x \leq 40\%$, the achieved coefficient values were consistently low For larger values of x, the algorithm delivers results closer to the correct output, but only for larger values of k. For instance, for hotels and restaurants, even $x = 80\%$ was not enough to achieve high correlation values for $k > 10$. With respect to the computational cost, we observe that significant savings are only achieved for large values of k (> 50), even if small percentages of the entire population of candidates are considered. This is a testament to CMiner's ability to efficiently evaluate and prune the search space, while also computing an exact solution. The results demonstrate that the a-priori elimination of a percentage of the search space can have a significant adverse effect on the quality of the results, while only delivering computational savings in limited cases. This provides further motivation for our own approach, which focuses on considering the entire space of candidates and only eliminates candidates when we have enough information to guarantee that they are not in the top-k.

APPENDIX F
EXAMPLE PAIRS OF THE USER STUDY

As an example of the annotation setup, we provide some sample pairs in Table 8 from the restaurants dataset. The top-ranked (most competitive) restaurant is better than the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2705101, IEEE Transactions on Knowledge and Data Engineering

TABLE 8: Examples of top-, middle-, and bottom-ranked competitors from the user study on RESTAURANTS.

ID	Food	Service	Value	Atm.	Price	Meal Types	Good For	Cuisines
Focal Item	0.83	0.79	0.74	0.78	0.96	Dinner, Lunch	Families	Italian
Top	0.9	0.9	0.9	0.80	0.97	Dinner, Lunch	Families, Bar scene, Budget, Large groups Special occ.	Italian, Pizza
Middle	0.80	0.80	0.70	0.80	0.91	Dinner, Late Night	Bar scene, Large groups Business, Special occ.	Mediterranean, Middle Eastern
Bottom	0.6	0.80	0.60	0.80	0.67	Dinner	Large Groups, Bar scene	American

focal (seed) item for all numeric features and covers more values for all categorical features. On the other hand, the middle-ranked item has better values than the focal item for some numeric features (e.g. service) and worse for others (e.g. price). That being said, the gaps in the values of the two items are consistently small across numeric features. In addition, the two items have no overlap with respect to 2 of the 3 categorical features (Good For, Cuisines), a fact that demonstrates that they compete for different market segments and lowers the competitiveness. Finally, the bottom ranked item has no overlap with the focal item for any of the categorical features. This item also has significantly lower values for many of the numeric features, a fact that further contributes to its low competitiveness.