Efficient and Domain-Invariant Competitor Mining

Theodoros Lappas
Boston University
tlappas@cs.bu.edu

George Valkanas
University of Athens
gvalk@di.uoa.gr

Dimitrios Gunopulos
University of Athens
dg@di.uoa.gr

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 relationship between two items? Who are the true 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. We present efficient methods for evaluating competitiveness in large datasets and address the natural problem of finding the top-k competitors of a given item. Our methodology is evaluated against strong baselines via a user study and experiments on multiple datasets from different domains.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data Mining*; H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

competitor mining, competitors, domain-invariant

1. INTRODUCTION

Competitiveness is a challenge that every product or service provider has to face, regardless of the application domain. A significant amount of relevant work has demonstrated the strategic importance of identifying and monitoring an entity's competitors [19]. In fact, a long line of research from the marketing and management community has been devoted to empirical managerial methods for competitor identification [8, 7, 10, 3, 18], as well as to methods for analyzing competitors [5], defending against competitive incursions, and devising appropriate response strategies [11, 6]. Our own work focuses on competitor identification, a key step for any competitiveness-driven study or application. Contrary to the significant amount of available work by the marketing community, the problem has been largely overlooked by computer scientists. For the latter, the challenge is to propose formalizations and competitor-identification algorithms that can utilize the vast amounts of rich data that is nowadays available on the web and other digital sources. Some progress toward this direction has been made by the information systems community [14, 15, 13, 1, 16, 27]. While the proposed approaches help motivate the problem, they present significant shortcomings. These include the lack of a formal definition of competitiveness, as well as the existence of assumptions that limit the applicability of these approaches. Specifically, these techniques are based on mining comparative expressions (e.g. "Item A is better than Item B") from the web or other textual sources. Even though such expressions can be indicators of competitiveness, they are absent in many domains. For example, consider the domain of vacation packages (e.g flight-hotel-car combinations). In this case, the 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 from the domain of technology (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 trivial to consider other mainstream domains where such findings are extremely scarce, if not non-existent(e.g. shoes, jewelery, hotels, restaurants, furniture). Finally, even in domains where such approaches are applicable, they cannot actually evaluate the competitiveness relationship between *any* two items. Instead, they can only identify a *subset* of the competitors, based on the available evidence. Our own work overcomes these drawbacks, by providing a formal definition of competitiveness that is applicable across domains. On a high-level, the fundamental problem we address in our work is the following:

PROBLEM 1. *We are given a set of items $\mathcal{I}$, defined within the feature space $\mathcal{F}$ of a particular domain. Then, given any pair of items I, I' from $\mathcal{I}$ we want to define a function $C_{\mathcal{F}}(I, I')$ that computes the competitiveness between the two in the context of the domain.*

As mentioned in the statement of the problem, our notion of competitiveness is based on the feature-space $\mathcal{F}$ of the corresponding domain. 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 group of users (i.e. the same market share), and to what extent. For example, two restaurants that exist in different countries are obviously not competitive to each other, since there is no overlap between their target groups. In the ideal scenario, we would have access to the complete set of users that could be interested in a given item. Then, given any two items, we could trivially compute their competitiveness based on the overlap of their respective sets. In practice, however, this is clearly not an option. Taking this into consideration, we formalize the competitiveness between two items based on their respective *features*. For example, if the considered items are MP3 Players, $\mathcal{F}$ could consist of the features *price, sound quality, battery life, connectivity, capacity* and *design*. Our motivation is that, regardless of the domain, users compare and evaluate items based on their features. Therefore, by attaching our formalization to the feature space, we ensure the availability of a consistent and informative resource for competitiveness evaluation. We provide a (simplified) overview of our approach in Figure 1.

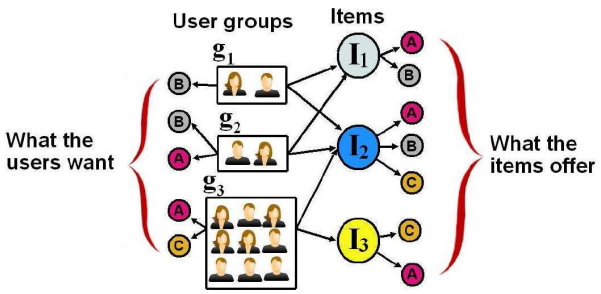


Figure 1: A (simplified) example of our competitiveness paradigm

The figure illustrates the competitiveness between three different items I_1, I_2 and I_3. Each item is mapped to the set of features that it can offer to the users. Three distinct features are considered in this example: A, B and C. Note that, for this simple example, we only consider binary features (i.e. available/not available). Our actual formalization accounts for a much richer space of binary, categorical and numerical features. The left side of the figure shows three groups of users (g_1, g_2, g_3). The example assumes that these are the only groups in existence. Users are grouped based on their preferences with respect to the features. For example, the users in group g_2 are only interested in features A and B. As can be seen by the figure, items I_1 and I_3 are not competitive to each other, since they simply do not appeal to the same groups of users. On the other hand, I_2 is in competition with both I_1 (for groups g_1 and g_2) and I_3 (for g_3). Finally, another interesting observation is that I_2 competes with I_1 for a total of 4 users, and with I_3 for a total of 9 users. In other words, I_3 is a stronger competitor for I_2, since it claims a much larger portion of I_2's market-share than I_1. In our work, we propose ways to deduce these user-groups from sources such as query logs and customer reviews, and describe methods to estimate the size of the market share that they represent. Our work is the first to utilize the opinions expressed in customer reviews as a resource for mining competitiveness.

The formal definition of the competitiveness $C_{\mathcal{F}}(I_i, I_j)$ between two items I_i and I_j, in the context of their domain's feature-space $\mathcal{F}$, is the first contribution of our work. As we demonstrate in our experiments, the evaluation of competitiveness can be a major computational challenge when dealing with real datasets of hundreds or even thousands of items. Motivated by this, we propose an algorithm for the natural problem of finding the top-k competitors of a given item. The proposed framework is geared toward scalability and efficiency, which makes it applicable to domains with large populations of items.

Roadmap
In Section 2 we introduce our competitiveness paradigm. In Sections 3 and 4 we present an efficient framework for finding the top-k competitors of a given item. The experimental evaluation of our work is presented in Section 5. In Section 6 we discuss previous related work. Finally, we conclude the paper in Section 7.

2. FORMALIZING COMPETITIVENESS

In this section, we describe how we can formalize and measure the competitiveness between any two items within a given domain. This formalization serves as the building block of our framework. Note that our definition can be easily extended to handle more than two items at a time.

2.1 Competitiveness via Coverage

In order to synthesize a competitor-mining method that works across domains, we need a formalization of competitiveness that is both accurate and flexible. Motivated by this, we build upon a crucial factor that remains consistent across domains: *user preferences*. In every market, the ultimate goal is to convert users into customers by meeting their individual requirements. Consider a single user u, interested in a specific domain (e.g. restaurants). While the domain may contain numerous items, the user will ultimately choose only one to spend his money on. In a typical scenario, the user follows the following steps:

1. Encode requirements and preferences in a query.
2. Submit the query to a search engine and retrieve the matching items.
3. Process matching items and make the final choice.

We observe that the items that do not match the user's criteria are never considered. In other words, they never get the chance to *compete* for his attention. As far this single user is concerned, the set of competitors consists of the matching items retrieved by the search engine. Consider the following motivating example:

Example: *A user is trying to pick a restaurant for dinner. He has a very limited budget and is only interested in Italian restaurants in the Boston Area. Only the restaurants that satisfy these criteria will compete for the user's attention. On the other hand, Chinese restaurants, restaurants from New York, and expensive establishments are not truly competitors with respect to this particular user, since they are outside the boundaries of his personal requirements and thus never had a chance to be chosen.*

In this example, the user is interested in the features {*price range, location, food type*}. The respective values that encode the user's requirements are {*Cheap, Boston, Italian*}.

Clearly, any other assignment of values could be specified for the same query (e.g. {*Cheap, Chicago, Chinese*}). In fact, each possible value-assignment represents the preferences of a different user. Formally, given a subset of features $\mathcal{F}'$, let $\mathcal{V}_{\mathcal{F}'}$ be the complete space of all possible value assignments over the features in $\mathcal{F}'$. We observe that every item covers a portion of the entire space $\mathcal{V}_{\mathcal{F}'}$, and, hence, covers the corresponding population of users. For example, a cheap restaurant in Boston that serves both Italian and American food *covers* a user who is interesting in cheap Italian food in Boston, as well as a user who is interested in cheap American food in the same city.

In order to evaluate the competitiveness of two given items I_i, I_j in the context of a subset of features $\mathcal{F}'$, we need to compute the number of possible value assignments over $\mathcal{F}'$ that are satisfied by *both* items. Formally, we define pairwise coverage as follows:

DEFINITION 1. [**Pairwise Coverage**] *Given the complete set of features $\mathcal{F}$ in a given domain of interest, let $\mathcal{V}_{\mathcal{F}'}$ be the complete space of all possible value-assignments over the features in a subset $\mathcal{F}' \subseteq \mathcal{F}$. Then, the* coverage $cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_k)$ *of a pair of items I_i and I_j with respect to $\mathcal{V}_{\mathcal{F}'}$ is defined as the portion of $\mathcal{V}_{\mathcal{F}'}$ that is covered by both items.*

Considering the above definition, we observe that the coverage of each dimension (i.e. each feature $F \in \mathcal{F}'$) is independent of the others. Therefore, we first compute the percentage of each dimension that is covered by the pair. We can then optimally compute the coverage of the entire space $\mathcal{V}_{\mathcal{F}'}$ as the product of the respective coverage values $\mathcal{V}_{\{F\}}$ for every $F \in \mathcal{F}'$. Formally:

$$cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) = \prod_{F \in \mathcal{F}'} cov(\mathcal{V}_{\{F\}}, I_i, I_j) \quad (1)$$

This computation has a clear geometric interpretation: The portion of the space $\mathcal{V}_{\mathcal{F}'}$ that is covered by a pair of items can be represented as a hyper-rectangle in $|\mathcal{F}'|$-dimensional space. For each dimension F, $cov(\mathcal{V}_{\{F\}}, I_i, I_j)$ gives us the portion of the dimension that is covered by the two items. Finally, by multiplying the individual coverage values, we are essentially computing the volume of the hyper-rectangle that represents the entire space $\mathcal{V}_{\mathcal{F}'}$.

Definition 1 allows us to evaluate the coverage provided by a pair of items to (the value space of) any subset of features $\mathcal{F}'$. Conceptually, $\mathcal{F}'$ captures the fraction of the population that is interested in the features included in $\mathcal{F}'$. In practice, the size of the corresponding population varies across subsets. For example, in the domain of restaurants, the subset {*food quality, price range*} is arguably of interest to more users than the subset {*Wi-Fi availability, delivery options*}. To account for this in our definition of competitiveness, we attach a popularity weight $w(\mathcal{F}')$ to each feature subset. We revisit the computation of these weights in Section 4, where we discuss practical methods for learning the weights from sources such as *query logs* and *customer reviews*. For the remaining of our analysis, we assume that the weights are provided as part of the input. Further, we define $\mathcal{Q}$ to be the collection of subsets with a non-zero weight. Formally: $\mathcal{Q} = \{\mathcal{F}' \in 2^{\mathcal{F}} : w(\mathcal{F}') > 0\}$. Taking the above into consideration, we formally define the competitiveness of two items I_i, I_i as follows:

DEFINITION 2. [**Competitiveness**] *Given the complete set of features $\mathcal{F}$ of a domain of interest, let $\mathcal{Q}$ be the set of all subsets of $\mathcal{F}$ that have a non-zero popularity weight. Then, the competitiveness of two given items I_i and I_i is defined as:*

$$C_{\mathcal{F}}(I_i, I_j) = \sum_{\mathcal{F}' \in \mathcal{Q}} w(\mathcal{F}') \times cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) \quad (2)$$

where $cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j)$ is the portion of $\mathcal{V}_{\mathcal{F}'}$ that is covered by both I_i and I_j.

2.2 Computing Coverage

Our definition of competitiveness between two given items is based on the pairwise coverage that they provide to the value space of the different subsets of features. We observe that the value space is complex, since it can contain different types of features. By supporting virtually every reasonable feature-type (numeric, ordinal, boolean, categorical), our framework guarantees the flexibility required to encode the requirements of virtually *any* potential customer. Next, we discuss the different types of features that we consider in our work, and show how coverage is defined for each of them.

Categorical Features: In our work, we identify two subtypes of categorical features: single-value and multi-value. For a single-value feature F, each item assumes exactly one value from the respective value-space $\mathcal{V}_{\{F\}}$, e.g. the brand of a digital camera. Clearly, boolean features are simply a special case of this group, assuming values from {*YES, NO*}. The pairwise coverage of two items I_i, I_j, given a single-value feature F, is defined as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \begin{cases} 1 & \text{if } I_i[F] = I_j[F] \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

For a multi-value feature F, each item can be mapped to any *subset* of values from the respective value-space $\mathcal{V}_{\{F\}}$. Assume, for example, the feature *parking* from Table 1, referring to the parking facilities of a restaurant. Since a restaurant can in fact provide any number of these options (even all of them), *parking* is a multi-value categorical feature. We define the pairwise coverage of two items I_i, I_j for a multi-value feature F, as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \frac{|I_i[F] \cap I_j[F]|}{|\mathcal{V}_{\{F\}}|} \quad (4)$$

Conceptually, the covered portion is defined as the overlap between the sets of values that are mapped to each item, divided by the total number of possible values for F. Clearly, the dividend is always a value in $[0, 1]$.

[Ordinal Features]: The value space $\mathcal{V}_{\{F\}}$ of an ordinal feature F assumes values from a finite *ordered* list: $\mathcal{V}_{\{F\}} = \{v_1, v_2, v_3, ...\}$. Depending on the nature of the feature, higher or lower states may be preferable. For example, for the feature *price range*, lower values are preferable. On the other hand, for the feature *stars rating*, higher values are better.

First, let us introduce two functions that will aid us in our definition of coverage in the context of ordinal features. Given an ordinal feature F and two values $v_1, v_2 \in \mathcal{V}_{\{F\}}$, let $loser(F, v_1, v_2)$ return the least preferable between the two values. In addition, let $weq(F, v_1)$ return the set of values that are worse or equal to v_1. For example, for the *price range* feature discussed above, $weq(F, \$\$\$) = \{\$\$\$, \$\$\$\$\}$. Then, the pairwise coverage of two given items to the value space is defined as follows:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = \frac{|weq(loser(F, I_i[F], I_j[F]))|}{|\mathcal{V}_{\{F\}}|} \quad (5)$$

As in the case of categorical features, $cov(\mathcal{V}_{\{F\}}, I_i, I_j)$ takes values in $[0, 1]$.

[Numeric Features]: A numeric feature F takes values from a continuous pre-defined range. Without loss of generality, we assume that all numeric features are normalized to take values in $[0, 1]$. Higher or lower values may be preferable, depending on the nature of the feature. As in the case of ordinal features, we define $loser(F, v_1, v_2)$ to return the least preferable of two given values for a feature F. Then, given two items I_i, I_j, the pairwise coverage of the value space $\mathcal{V}_{\{F\}}$ is defined as:

$$cov(\mathcal{V}_{\{F\}}, I_i, I_j) = loser(F, I_i, I_j) \quad (6)$$

Conceptually, the value space that is commonly covered by the two items is bounded by the one with the least preferable value. Consider the following example.

Example: Consider the subset of features $\mathcal{F}'$ shown in Table 1. The respective representations for two items I_i, I_j are $\{\$\$\$, Boston, \{Street,\ Valet\}, 0.8\}$ and $\{\$\$, Boston, \{Street,\ Priv.\ Lot\}, 0.6\}$. Then, following Eq. 1, the pairwise coverage of the two items of is computed as follows:

$$cov(\mathcal{V}_{\mathcal{F}'}, I_i, I_j) = \frac{2}{4} \times 1 \times \frac{1}{3} \times 0.6 = 0.1$$

Table 1: Feature-subsets and their respective value-spaces.

Feature	Type	Value-Set $\mathcal{V}_{\{\mathcal{F}\}}$
price range	ordinal	{\$, \$\$, \$\$\$, \$\$\$\$}
location	categorical (single)	{*Boston, New York*}
parking	categorical (multi)	{*Street, Priv. Lot, Valet*}
food quality	numeric	$[0, 1]$

3. FINDING THE TOP-K COMPETITORS

In the previous section we presented a formal definition of the competitiveness between any two items. Given this definition, we study the natural problem of finding the top-k competitors of a given item. Formally, the problem is defined as follows:

PROBLEM 2. *We are given a set of items $\mathcal{I}$, defined within the feature space $\mathcal{F}$ of a domain. Then, given a single item $I \in \mathcal{I}$, we want to identify the k items from $\mathcal{I} \setminus \{I\}$, that maximize the pairwise competitiveness with I:*

$$I^* = \underset{I' \in \mathcal{I} \setminus \{I\}}{\operatorname{argmax}} C_{\mathcal{F}}(I, I') \quad (7)$$

A naive algorithm for this problem would iterate over all items in $I' \in \mathcal{I} \setminus \{I\}$. For each such item I', it would compute $w(\mathcal{F}') \times cov(\mathcal{V}_{\mathcal{F}'}, I, I')$ for every subset $\mathcal{F}' \in \mathcal{Q}$, where $\mathcal{Q}$ is the collection subsets with a non-zero weight. It would then be trivial to obtain the top-k competitors for the given item. However, considering that $\mathcal{I}$ can contain thousands of items, the computational cost can be overwhelming. We demonstrate this in our experiments, where we compare our own CMiner technique with the naive approach.

3.1 The CMiner Algorithm

Motivated by the inefficiency of the naive approach, we present CMiner, a new algorithm for Problem 2. Our approach combines scalability with the ability to handle the online arrival of new items. The latter is crucial in many mainstream domains. As an example, consider the case when items are vacation packages. In such a domain, an arbitrary number of new packages can be introduced at any point in time. Hence, we would like to preprocess the data in a way that allows us to compute the competitors of a new package without having to repeat the entire computation effort.

First, we define the concept of *item dominance*, which will aid us in our further analysis:

DEFINITION 3. **[Item Dominance]:** *Given two items I_i, I_j from a set $\mathcal{I}$ defined within a feature-space $\mathcal{F}$, we say that an item I_i dominates an item I_j if both of the following conditions are true:*

1. *$I_j[F] \subseteq I_i[F]$, for every multi-value categorical feature $F \in \mathcal{F}$*
2. *$I_i[F] \geq I_j[F]$, for every ordinal, numerical, or single-categorical feature $F \in \mathcal{F}$.*

Conceptually, an item dominates another if it has better values for all features of the considered space $\mathcal{F}$. Clearly, if I_i dominates I_j, then it is also more competitive with respect to any other item from $\mathcal{I}$ (since it covers at least as much coverage to any possible sub-space as I_j). This observation 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 set that are not dominated by any other point in the set [4]. 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 $I_i \in \mathcal{I}$, let $\mathcal{Y}$ contain the k items with the highest $\mathcal{C}_{\mathcal{F}}(\cdot, I_i)$ values from $Sky(\mathcal{I})$. Then, an item $I_j \in \mathcal{I}$ can only be in the top-k competitors of I_i, if $I_j \in \mathcal{Y}$ or I_j is dominated by one of the items in $\mathcal{Y}$.*

The proof is by contradiction, given the definition of item dominance and the skyline concept. We omit it for lack of space.

By applying Lemma 1, we do not need to consider the entire set of items in order to find the top-k competitors of a given item I^*. Instead, it is sufficient to recursively check for the items that are dominated by the current top-k items from the upper levels. In order to fully utilize this observation, we construct a structure that greatly reduces the number of items that need to be considered for the computation of the top-k competitors set. We refer to this structure as the *skyline pyramid*. This structure is similar to the *dominant graph*, presented by Zou and Chen [29], except that it does not allow multiple parents for each item. The pyramid can be constructed by recursively computing the skyline and removing the skyline points from the current set, until the entire collection of items has been exhausted. Standard techniques can be used for computing the skyline on each iteration [17], as well as for updating the pyramid in case new items are introduced [12]. Each item from the i_{th} layer of the skyline is assigned an inlink from the item

from i_{th} level that dominates it. If multiple such dominators exist, we simply choose one randomly. This is simply done to avoid re-checking the dominated item during the competitor-finding process, and does not affect the optimality of the result. A formal proof is trivial and omitted for lack of space.An example of the skyline pyramid is shown in Figure 2.

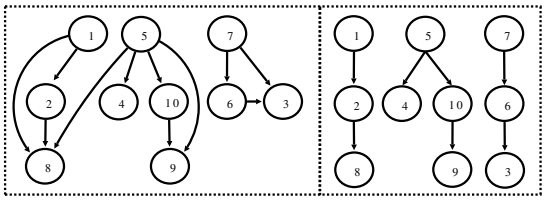


Figure 2: The left side of the figure shows the complete dominance graph for a given set of items. An edge $I_i \rightarrow I_j$ means that I_i dominates I_j. The right side of the figure shows the skyline domination pyramid.

The CMiner Algorithm: Next, we present CMiner, an optimal algorithm for finding the top-k competitors of any given item. Our algorithm makes use of the skyline pyramid described earlier in this section, in order to reduce the number of items that need to be considered and minimize the number of required coverage computation. The intuition is that, since 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 to the algorithm 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 collection $\mathcal{Q}$ of feature-subsets with non-zero weights, and the skyline pyramid $\mathcal{D}_\mathcal{I}$.

In lines 1-4, the algorithm uses $masters(I^*)$ to retrieve the set of items that *dominate* I^*. Note that this set can be easily pre-computed for all the items during the pyramid-construction phase. These items are guaranteed to have the maximum possible competitiveness with I^*. If at least k such items exist, we can just report them and conclude. Otherwise, we append them to the final result and decrement our budget of k accordingly. The LB variable maintains the lowest lower bound from the current top-k set. This is used as pruning threshold for the candidates. In lines 6-7 we initialize the upper and lower bounds for each candidate. In line 8 we initialize the set of candidates $\mathcal{X}$ as the union of the items in the first layer of the pyramid and the items dominated by those in the *TopK*. The latter is returned via calling `getSlaves`$(TopK, \mathcal{D}_\mathcal{I})$.

In every iteration of lines 9-15, the algorithm does the following: (i) it feeds the set of candidates $\mathcal{X}$ routine, which prunes items based on the LB threshold, (ii) updates the *TopK* set via the $merge(\cdot)$ function, (iii) updates the pruning threshold LB, (iv) expands the set of items by including the items that they dominate.

Discussion of `UpdateTopK()`: This routine processes the items in $\mathcal{X}$ and finds at most k with the highest competitiveness scores among $\mathcal{X}$, subject to the condition that this score is higher than the global pruning threshold LB. The approach uses two bounds *low* and *up*, for every $I \in \mathcal{X}$.

Algorithm 1 CMiner

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

1: $TopK \leftarrow masters(I^*)$
2: **if** ($k \leq |TopK|$) **then**
3: **return** $TopK$
4: $k \leftarrow k - |TopK|$
5: $LB \leftarrow -1$
6: $low(I) \leftarrow 0, \forall I \in \mathcal{X}.$
7: $up(I) \leftarrow \sum_{\mathcal{F}' \in \mathcal{Q}} w(\mathcal{F}') \times (cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)), \forall I \in \mathcal{X}.$
8: $\mathcal{X} \leftarrow$ `getSlaves`$(TopK, \mathcal{D}_\mathcal{I}) \cup \mathcal{D}_\mathcal{I}[0]$
9: **while** ($|\mathcal{X}|$!= 0) **do**
10: $\mathcal{X} \leftarrow$ `updateTopK`$(k, LB, \mathcal{X})$
11: **if** ($|\mathcal{X}|$!= 0) **then**
12: $TopK \leftarrow merge(TopK, \mathcal{X})$
13: **if** ($|TopK| = k$) **then**
14: $LB \leftarrow TopK[k]$
15: $\mathcal{X} \leftarrow$ `getSlaves`$(\mathcal{X}, \mathcal{D}_\mathcal{I})$
16: **return** $TopK$

17: **Procedure** `updateTopK`*(k, LB, $\mathcal{X}$)*
18: $localTopK \leftarrow \emptyset$
19: **for** every $\mathcal{F}' \in \mathcal{Q}$, in sorted order **do**
20: $sc \leftarrow w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)$
21: $localTopK \leftarrow \emptyset$
22: **for** every item $I \in \mathcal{X}$ **do**
23: $up(I) \leftarrow up(I) - sc + w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I)$
24: **if** ($up(I) < LB$) **then**
25: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{I\}$
26: **else**
27: $low(I) \leftarrow low(I) + w(\mathcal{F}') \times (cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I))$
28: $localTopK.add(I, low(I))$
29: **if** ($|localTopK| = k$) **then**
30: $LB \leftarrow localTopK[k]$
31: **if** ($|\mathcal{X}| \leq k$) **then**
32: **break**
33: **for** every item $I \in \mathcal{X}$ **do**
34: **for** every remaining $\mathcal{F}' \in \mathcal{Q}$ **do**
35: $low(I) \leftarrow low(I) + w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I)$
36: $localTopK.add(I, low(I))$
37: **return** $localTopK$

$low(I)$ maintains the competitiveness score of item I, as new feature subsets are considered. $up(I)$ is an optimistic upper bound on I's competitiveness score. Therefore, *up* begins with the maximum possible competitiveness score, $\mathcal{C}_\mathcal{F}(I^*, I^*)$.

For every feature subset, we examine all items in $\mathcal{X}$ and update their *up* value. If at any point $up(I) < LB$ (line 24), item I can be safely removed from the $\mathcal{X}$. If, at any point, $|\mathcal{X}|$ becomes less or equal to k, the loop over the subsets comes to a halt. In lines 33-36 we update the lower bounds of the remaining items in $\mathcal{X}$. We do this outside the loop, in order to avoid unnecessary bound checking and improve performance. Observe that the routine processes subsets in *sorted order*. In Section 4, we elaborate on the impact of the ordering on the performance of CMiner.

Complexity: The complexity of CMiner depends on the number of points in each layer of $\mathcal{D}_\mathcal{I}$. According to Bentley et al. [2], for n uniformly-distributed d-dimensional data points (items), the expected size of the skyline is $\Theta(\frac{ln^{d-1}n}{(d-1)!})$. Since we need to examine at most k skyline layers to find the top-k result, this value is upper-bounded by $\Theta(k * \frac{ln^{d-1}n}{(d-1)!})$. This bound naively assumes that each layer should be considered entirely. In practice, however, we only need to check

a small fraction of items that are dominated by the items considered in the previous layer. For instance, for a uniform distribution with consecutive skyline layers of similar sizes, 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, at most k new items will be introduced. Therefore, for small values of k, the complexity of CMiner is O ($|\mathcal{I}| * |\mathcal{Q}| * k^2$), where $\mathcal{Q}$ is the set of feature subsets with non-zero weights.

4. WEIGHT ESTIMATION

Our analysis has assumed that the weight $w(\mathcal{F}')$ of each subset of features $\mathcal{F}'$ is provided as input. In this section, we discuss methods for computing these weights.

The motivation of assigning a different weight to each subset stems from the observation that not all features are equally important to users. Based on this, a straightforward approach is to consider the weight of each individual feature separately, and then aggregate to the subset level. This aggregation could be achieved by selecting the sum, average, median, maximum or minimum over all the individual features in a set. This approach assumes independence among the features of an item. This assumption, however, is not always valid. For example, it may be the case that people who are interested in the screen resolution of a laptop computer are also more likely to be interested in the included graphics card. This motivates an approach that considers the popularity of *feature-subsets* instead of individual features. We identify two sources from which we can learn the popularity of a subset: *query logs* and *customer reviews*.

Query logs: The first source is the query logs of the search engine on the website where the items are hosted. Regardless of the interface through which the user encodes his preferences in a query, the set of selected feature is always recorded in a dedicated log. Assuming the existence of a large enough user-base, we can simply estimate the popularity of a feature-subset based on the number of times that it was queried upon by the users.

Customer reviews: In cases when query logs are unavailable or inadequate, the weights of the subsets can be estimated by considering the *reviews* that are available for the items in the domain. As an example of such a dataset, consider the union of the review sets that are available for all the digital cameras offered on amazon.com. Each of these reviews comments on a particular subset of attributes from the digital-camera domain. Hence, the review corpus serves as an intuitive way to access user preferences. For example, a user who is greatly interested in the wheelchair-accessibility of a restaurant is more likely to discuss this feature in his review. We implement and employ review mining as a means for estimating the weights of feature-subsets in our experiments. In practice, one can choose to ignore subsets that appear less frequently than a set threshold. In our own experiments, we consider all subsets that appear at least once.

4.1 Subset Ordering

Given an item of interest I^*, CMiner iterates over the given set of subsets and computes the coverage provided by I^* and each candidate item to the value-space that corresponds to each subset. Given our definition of competitiveness, we next consider **IC**, an ordering scheme that processes subsets in descending order by $w(\mathcal{F}') \times cov(\mathcal{V}'_{\mathcal{F}'}, I^*, I^*)$, where I^* is the item of interest. As stated in the following lemma, **IC** achieves the optimal *convergence rate* (i.e. there exists no ordering that leads to a faster convergence).

LEMMA 2. [IC Convergence Rate]: Given two items I_i, I_j, the ordering imposed by the **IC** scheme results in the fastest possible convergence to the target-value $C_{\mathcal{F}}(I_i, I_j)$ (i.e. the true competitiveness between the two items)

PROOF. Assume that we want to compute the competitiveness between the target item I^* and a candidate I'. Let l_i and u_i be the lower and upper competitiveness bounds, after checking $\mathcal{F}'_i$, the i-th feature subset imposed by the **IC** scheme. For ease of notation, we use $C_{\mathcal{F}'_i}(I_1, I_2) = w(\mathcal{F}'_i) \times cov(\mathcal{V}_{\mathcal{F}'_i}, I_1, I_2)$, for any two items. Every time a new subset is considered, l_i and u_i are updated, until finally all the subsets have been evaluated and both variables converge to the actual competitiveness score $C_{\mathcal{F}}(I^*, I')$. We now define $T_i = u_i - l_i$. Since both l_i and u_i ultimately converge to $C_{\mathcal{F}}(I^*, I')$, T_i converges to 0. Also, $T_i \geq 0, \forall i$. The convergence rate of T_i is:

$$\frac{T_i}{T_{i-1}} = 1 - \frac{C_{\mathcal{F}'_i}(I^*, I^*)}{u_{i-1} - l_{i-1}} \tag{8}$$

Also, we know that: $u_i = C_{\mathcal{F}}(I^*, I^*) - \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I^*) + \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I^*)$ and $l_i = \sum_{j=1}^{i} C_{\mathcal{F}'_j}(I^*, I')$

By immediate replacement in Eq. 8, the convergence rate becomes:

$$\frac{T_i}{T_{i-1}} = 1 - \frac{C_{\mathcal{F}'_i}(I^*, I^*)}{C_{\mathcal{F}}(I^*, I^*) - \sum_{j=1}^{i-1} C_{\mathcal{F}'_j}(I^*, I^*)} \tag{9}$$

As it can be seen by Eq. 9, the convergence rate depends only on the score of the target item I^*. The **IC** ordering scheme processes subsets in decreasing order of $C_{\mathcal{F}'_j}(I^*, I^*)$, which is the maximum possible coverage that any item can jointly achieve with I^*. Thus, the numerator is equal to the i_{th} maximum possible value among all feature-subsets. Similarly, the difference in the denominator is minimized, since the subtracted sum maintains the highest possible value (and $C_{\mathcal{F}}(I^*, I^*)$ is constant). □

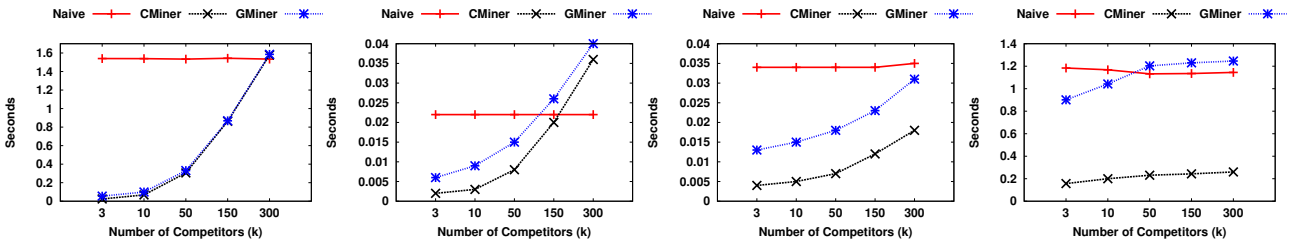
5. EXPERIMENTAL EVALUATION

For our evaluation, we compiled the following datasets:

`Digital Cameras` from `Amazon.com`: The features of this domain include the objective attributes of each camera (e.g. *price, number of megapixels*), as well as numeric attributes representing the opinions of the users on the item's different characteristics (e.g. *photo quality, video quality*). These were extracted via the method by Ding et al. [9], which assigns a numeric opinion-value to each feature of an item, given the corpus of reviews. All scores were normalized to be in $[0, 1]$, with higher scores being preferable. The same method was also used for the datasets from `Booking.com` and `TripAdvisor.com`.

`Hotels` **from** `Booking.com`**:** The feature-set for this domain consists of objective features (e.g. *price, location*) and the opinion values extracted from the relevant reviews on different attributes (e.g. *cleanliness, service quality*).

`Restaurants` **in New York from** `TripAdvisor.com`**:** The



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

Figure 3: Average time (per item) to compute top-k competitors for the various datasets

feature-set for this domain consists of objective features (e.g. *type of food served*) and the opinion values extracted from the relevant reviews on different attributes (e.g. *food quality,service quality*).

`Recipes` from `Sparkrecipes.com`: The feature-set for each recipe consists of the different nutritional values (e.g. *grams of protein and carbohydrates*), as listed on the website.

The datasets were selected from different domains to portray the cross-domain applicability of our approach. Table 5 summarizes some basic statistics for each dataset.

Table 2: 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

The second, third, fourth and fifth columns include the number of items, the number of features, the number of distinct feature-subsets, and the number of layers in the respective (complete) skyline pyramid, respectively. The feature subsets were extracted from the set of reviews that is available for each dataset; the frequency of each subset of features is equal to the number of times they were included together in a review. For all four datasets, nearly 99% of the items can be found within the first 4 layers of the pyramid, with the majority of those falling within the first 2 layers. This is due to the large dimensionality of the datasets, which makes domination unlikely. As we show in our evaluation, the skyline pyramid helps CMiner clearly outperform the baselines with respect to the 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 the included items.

Baselines: We compare our CMiner algorithm with two baselines. The first is the Naive approach described in Section 3. The second is a clustering-based approach that works as follows. First, it iterates over the considered feature-subsets. For each subset $\mathcal{F}'$, it identifies the set of items that have the same value assignment for the features in $\mathcal{F}'$, and places them in the same group. Thus, $\mathcal{F}'$ is mapped to different groups of items with the save value-assignments over its features. The algorithm then iterates over the reported groups. For each group, it updates the pairwise coverage provided to $\mathcal{V}'_{\mathcal{F}}$ by the target item I^* and an arbitrary item from the group (it can be any item, since they all have the same values with respect to $\mathcal{F}'$). The computed coverage is then used to update the competitiveness of all the items in the group. The process continues until the optimal competitiveness scores for all items have been computed. Assuming there are at most M groups per feature-subset, the algorithm's complexity is O($|\mathcal{I}|$ * M * $|\mathcal{Q}|$). Obviously, when each group is a singleton, the algorithm degrades to the Naive case. We refer to this technique as *GMiner*.

Unless otherwise stated, we use the **IC** ordering scheme in our experiments, as described in Section 4.1. All experiments were run on a desktop with a Quad-Core 3.5GHz Processor and 2GB RAM.

5.1 Computational Time

In this experiment we compare CMiner with the two baselines (Naive and GMiner), in terms of computational time. First, we use each of the three algorithms to compute the set of top-k competitors for each item in the four datasets. The process is repeated for $k \in \{3, 10, 50, 150, 300\}$. The results for the four datasets are shown in Figures. 3(a-d). The x-axis holds the different values of k. The y-axis holds the respective computational times (in seconds). We report the average time for each item.

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`.

For the `Cameras` dataset, CMiner and GMiner exhibit almost identical running times. This is due to the very large number of distinct feature-subsets for this dataset, which favors GMiner. In particular, this dataset has 14779 different subsets and GMiner identifies, on average, 443.63 groups per subset. This means that the algorithm saves roughly a total of $(579 - 443) \times 14779 = 2009944$ coverage computations per item, 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.

5.2 Feature Subsets

In Section 4.1, we described the **IC** ordering scheme, which determines the order in which subsets are processed by CMiner. Two alternatives to this scheme are processing the subsets in descending (**W-DSC**) and ascending (**W-ASC**) order by weight. Next, we evaluate the impact of the three schemes on the efficiency of CMiner. We only show the results for the `Recipes` dataset, which was by far the largest in terms of both size and dimensionality. The findings for the other corpora were identical and are omitted for lack of space. Figure 4 shows the total number of subset-processings

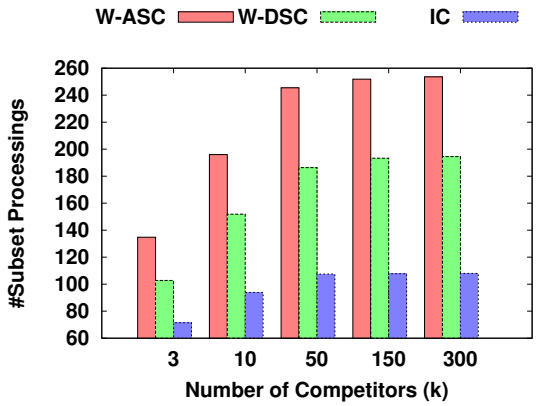


Figure 4: Number of subset processings that required under different ordering schemes.

(i.e. the total number of times the for-loop in line 19 of Algorithm 1 is executed) required by each ordering scheme until CMiner converges to the top-k competitors, for different values of k. We report the average over all items in the corpus. The results demonstrate that **IC** clearly outperforms the two baselines, consistently requiring far less subsets for all values of k. We also obeserve that no more than 110 subsets had to be processed, for all considered values of k. This allows CMiner to quickly converge to the optimal top-k set.

5.3 A User Study

In order to validate our competitiveness paradigm, we conduct a user study as follows. First, we select 10 random items from the `Cameras` corpus. 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. In addition to our own CMiner approach, we also rank all the items in the corpus based on their distance to I^* in the feature-space. The L_1 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`. The survey was taken by 20 different human 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 in a randomized order to the annotators, who were also given access to a table including the values

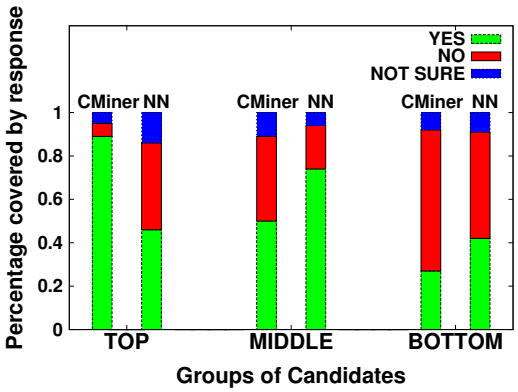


Figure 5: Results of the user study comparing our competitiveness paradigm with the Nearest-Neighbor approach.

of each item for every feature. For each pair, the annotator was asked whether he would consider buying the candidate instead of the seed item. The possible answers were "YES", "NO" and "NOT SURE". The results are shown in Figure 5.

The y-axis holds the percentage covered by each approach. The figure shows 3 pairs of bars. The left bar of each pair corresponds to our CMiner approach, while the right bar to the `NN` approach. The first two bars from the left represent the responses of the users to the top-ranked candidates for each approach. The two bars in the middle represent the responses to the candidates ranked in the middle, and, finally, the two bars on the right represent the responses to the bottom-ranked candidates. Each bar 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 some interesting observations. First, we observe that the vast majority of the top-ranked items by our method were identified by the annotators as possible replacements for the seed item. These are thus verified as strong competitors that could deprive the seed item from potential customers. On the other hand, the top-ranked candidates of the NN approach were often rejected by the users, who did not consider these items to be competitive.

The middle-ranked candidates of our approach attracted mixed responses from the annotators, indicating that it was not straightforward to determine whether the item is indeed competitive or not. An interesting observation is that the middle-ranked candidates of the NN approach were more popular than its top-ranked ones. The interpretation is that this approach fails to emulate the way the users perceive the competitiveness between two items.

Finally, the bottom-ranked candidates of our approach were consistently rejected by the annotators, verifying their lack of competitiveness. 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. In conclusion, the survey demonstrated the ability of our paradigm to capture the competitiveness between two items. Further, our approach consistently outperformed an intuitive baseline, indicating that the task is non-trivial.

6. RELATED WORK

While our work is the first to consider domain-invariant competitor mining, it has ties to existing relevant literature.

Competitor Mining: Previous work [14, 13, 1, 26] focuses on mining competitors based on comparative expressions found in web results and other textual corpora. The intuition is that the occurrence of expressions like "Item A is better than Item B" "or item A Vs. Item B" is indicative of the competitiveness relationship between the two items. However, as we have already discussed in the introduction, such comparative evidence are typically scarce, or even non-existent in many mainstream domains. As a result, the applicability of such approaches is greatly limited.

Finding Competitive Products: Recent work [22, 23, 28] 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. This can measure domination of other items or potential customers. The goal is then to use this function to create non-dominated items, or items with the maximum possible dominance value. A similar line of work [25, 24] 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 (and vice versa).

Skyline computation: Our work leverages concepts and techniques from the extensive literature on skyline computation [4, 12, 17]. 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 [20, 21]. Even though the focus of our work is different, we intend to study the advances in this field and evaluate their potential to improve the efficiency of our framework in future work.

7. CONCLUSION

In this work, we presented a formal definition of the competitiveness between two items. Our formalization is applicable across domains, overcoming the shortcomings of previous approaches. We consider a number of factors that have been overlooked by previous approaches, such as the position of the items in the multi-dimensional feature space, and the preferences and opinions of the users. A user study was conducted to verify the validity of our notion of competitiveness. Based on our competitiveness paradigm, we addressed the problem of finding the top-k competitors of a given item. Our framework is designed to be efficient and scalable, in order to be applicable to large populations of items. Our methodology was evaluated via an experimental evaluation on real datasets from different domains.

Acknowledgments

This work was supported by the MODAP and DISFER projects.

8. REFERENCES

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

[2] 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.

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

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

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

[6] M.-J. Chen and I. C. MacMillan. Nonresponse and delayed response to competitive moves: The roles of competitor dependence and action irreversibility. 1992.

[7] B. H. Clark and D. B. Montgomery. Managerial Identification of Competitors. *Journal of Marketing*, 1999.

[8] R. Deshpandé and H. Gatingon. Competitive analysis. *Marketing Letters*, 1994.

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

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

[11] H. Gatignon, E. Anderson, and K. Helsen. Competitive reactions to market entry: Explaining interfirm differences. *Journal of Marketing Research*, 1989.

[12] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. VLDB '02, 2002.

[13] R. Li, S. Bao, J. Wang, Y. Liu, and Y. Yu. Web scale competitor discovery using mutual information. In *ADMA*, 2006.

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

[15] 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.

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

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

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

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

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

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

[22] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products.

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

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

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

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

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

[28] 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.

[29] L. Zou and L. Chen. Pareto-based dominant graph: An efficient indexing structure to answer top-k queries. *IEEE Trans. Knowl. Data Eng.*, 23(5):727–741, 2011.