Leveraging Collaborative Tagging for Web Item Design

Mahashweta Das, Gautam Das[*]
Department of Computer Science & Engineering
University of Texas at Arlington
mahashweta.das@mavs.uta.edu, gdas@uta.edu

Vagelis Hristidis[†]
School of Computing & Information Sciences
Florida International University
vagelis@cis.fiu.edu

ABSTRACT

The popularity of collaborative tagging sites has created new challenges and opportunities for designers of web items, such as electronics products, travel itineraries, popular blogs, etc. An increasing number of people are turning to online reviews and user-specified tags to choose from among competing items. This creates an opportunity for designers to build items that are likely to attract desirable tags when published. In this paper, we consider a novel optimization problem: given a training dataset of existing items with their user-submitted tags, and a query set of desirable tags, design the k best new items expected to attract the maximum number of desirable tags. We show that this problem is NP-Complete, even if simple Naive Bayes Classifiers are used for tag prediction. We present two principled algorithms for solving this problem: (a) an exact "two-tier" algorithm (based on top-k querying techniques), which performs much better than the naive brute-force algorithm and works well for moderate problem instances, and (b) a novel polynomial-time approximation algorithm with provable error bound for larger problem instances. We conduct detailed experiments on synthetic and real data crawled from the web to evaluate the efficiency and quality of our proposed algorithms.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

collaborative tagging, item design, naive bayes, optimization

[*]Partially supported by NSF grants 0812601, 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board, and grants from Microsoft Research and Nokia Research.

[†]Partially supported by NSF grants IIS-0811922, IIS-0952347, HRD-0833093 and Google Research Award.

1. INTRODUCTION

Motivation: The widespread use and popularity of online collaborative tagging sites has created new challenges and opportunities for designers of "web items" such as electronics products, travel itineraries, popular blogs, etc. Various websites today (e.g., Flickr for photos, YouTube for videos, Amazon for different products) encourage users to actively participate by assigning labels or "tags" to online resources with a purpose to promote their contents and allow users to share, discover and organize them. An increasing number of people are turning to online reviews and user-specified tags to choose from among competing items. For example, a cell phone that has been tagged `lightweight` by several users is likely to influence a prospective customer decision in its favor. This creates an opportunity for designers to build items that are likely to attract desirable tags when published. In addition to traditional marketplaces like electronics, autos or apparel, tag desirability also extends to other diverse domains. For example, music websites such as `Last.fm` use social tags to guide their listeners in browsing through artists and music. An artist creating a new musical piece can leverage the tags that users have selected, in order to select the piece's attributes (e.g. acoustic and audio features) that will increase its chances of becoming popular. Similarly, a blogger can select a topic based on the tags that other popular topics have received.

Our paper investigates this novel tag maximization problem, i.e., how to decide the attribute values of new items and to return the top-k "best" items that are likely to attract the maximum number of desirable tags. We provide more details as follows.

Tag Maximization Problem: Assume we are given a set of items, each having a set of attributes and a set of user-submitted tags (e.g., cell phones on Amazon's website, each described by a set of attributes, and associated user tags). From this training data, for each distinct tag, we assume a classifier has been constructed for predicting the tag given the attributes. Tag prediction is a recent area of research (see Section 6 for discussion of related work), and the existence of such classifiers is a key assumption in our work. In addition to the item's explicitly specified attributes, other implicit factors also influence tagging behavior, such as the perceived utility and quality of an item to the user, the tagging behavior of the user's friends, etc. However, pure "content-based" tag prediction approaches are often quite effective – e.g., in the context of laptops, attributes such as smaller dimensions and the absence of a built-in DVD drive may attract tags such as `portable`.

Given a query consisting of a subset of tags that are considered "desirable", our task is to suggest a new item (i.e., a combination of attribute values) such that the expected number of desirable tags for this potential item is maximized. This can be extended to the top-k version, where the task is to suggest the k potential items with the highest expected number of desirable tags.

This information can assist web item designers in designing new items (or fine-tuning existing web content) to make them more attractive to users. Moreover, designers can explore the dataset in an interactive manner by picking and choosing different sets of desirable tags to get insight on how to build new items that target different user populations – e.g., in the context of cell phones, tags such as `lightweight` and `powerful` target professionals, whereas tags such as `cheap`, `cool` target younger users.

Novelty, Technical Challenges and Approaches: The dynamics of social tagging has been an active research area in recent years. However related literature primarily focuses on the problems of tag prediction, including cold-start recommendation to facilitate web-based activities. To our best knowledge, tags have not been studied in the context of web item design before. Of course, real-world item design is a complex task, and is an area that has been heavily studied in economics, marketing, industrial engineering and more recently in computer science. Many factors like the cost and return on investment are currently considered. We argue that the user feedback (in the form of tags of existing competing items) should be taken into consideration in the design process, especially since online user tagging is extremely widespread and offers unprecedented opportunities for understanding the collective opinion and preferences of a huge consumer base. We envision user tags to be one of the several factors in item design that can be used in conjunction with more traditional factors - e.g., our algorithms return k potential new items that maximize the number of desirable tags; and this information can assist web item designers, who can then further post-process the returned results using additional constraints such as profitability, price, resource constraints, item diversity, etc.

Solving the tag maximization problem is technically challenging. In most item sets, complex dependencies exist among the tags and items, and it is difficult to determine a combination of attribute values that maximizes the expected number of desirable tags. In this paper we consider the very popular Naive Bayes Classifier for tag prediction, although our problem framework also extends to other classifiers. As one of our first results, we show that even for this classifier (with its simplistic assumption of conditional independence), the tag maximization problem is NP-Complete. Given this intractability result, it is important to develop algorithms that work well in practice. A highlight of our paper is that we have avoided resorting to heuristics, and instead have developed principled algorithms that are practical *and at the same time* possess compelling theoretical characteristics.

Our first algorithm is a novel exact top-k algorithm **ETT** (Exact Two-Tier Top-k algorithm) that performs significantly better than the naive brute-force algorithm (which simply builds all possible items and determines the best ones), for moderate problem instances. Our algorithm is based on nontrivial adaptations of top-k query processing techniques (e.g., [4]), and has an interesting two-tier architecture. At the bottom tier, we develop a sub-system for each distinct tag, such that each sub-system has the ability to compute on demand a stream of items in order of decreasing probability of attracting the corresponding tag, *without having to pre-compute* all possible items in advance. This is achieved by partitioning the set of attributes into smaller groups (thus, each group represents a "partial" item), and running a separate merge algorithm over all the groups. The top tier considers the items retrieved from each sub-system in a round-robin manner, computes the expected number of desirable tags for each retrieved item, and stops when a threshold condition is reached. Although in the worst case this algorithm can take exponential time, for many datasets with strong correlations between attributes and tags, the stopping condition is reached much earlier.

However, although the exact algorithm performs well for moderate problem sizes, it did not easily scale to much larger sized datasets. Heuristic techniques like hill-climbing, branch and bound, etc. does not guarantee any sort of worst case behavior, either in running time or in item quality and thus we also develop a novel approximation algorithm **PA** (Poly-Time Approximation algorithm) that runs in worst case polynomial time, *and* also guarantees a provable bound on the approximation factor in item quality. The principal idea is to group the desirable tags into *constant-sized* groups, find the top-k items for each sub-group, and output the overall top-k items from among these computed items. For each sub-problem thus created, we show that it can be solved by a polynomial time approximation scheme (PTAS) given any user-defined approximation factor. The algorithm's overall running time is exponential only in the (constant) size of the groups, thus giving overall a polynomial time complexity.

We experiment with synthetic as well as real datasets crawled from the web to compare our algorithms. User study on the real dataset demonstrates that the items suggested by our algorithms appear to be meaningful. With regard to efficiency, the exact algorithm performs well on moderate problem instances, whereas the approximation algorithm scaled very well for larger datasets.

In summary, we make the following main contributions:

- We introduce the novel problem of top-k item design based on user-submitted tags and show that this problem is NP-complete, even if tag prediction is modeled using simple Naive Bayes Classifiers.
- We develop an exact two-tier algorithm (ETT) to compute the top-k best items that works well for moderate problem instances.
- We also design an approximation algorithm (PA) based on a polynomial time approximation scheme (PTAS), with provable error bounds, for larger datasets.
- We perform detailed experiments on synthetic and real datasets crawled from the web to demonstrate the effectiveness of our developed algorithms.

2. PROBLEM FRAMEWORK

Let $\mathbb{D} = \{o_1, o_2, ..., o_n\}$ be a collection of n items, where each item entry is defined over the attribute set $A = \{A_1, A_2, ..., A_m\}$ and the tag dictionary space $T = \{T_1, T_2, ..., T_r\}$. Each attribute A_i can take one of several values a_i from a multi-valued categorical domain D_i, or one of two values 0, 1 if a boolean dataset is considered.[1] A tag T_j is

[1]Our framework allows numeric attributes, but as is com-

a bit where a 0 implies the absence of a tag and a 1 implies the presence of a tag for item o. Each item is thus a vector of size $(m + r)$, where the first m positions correspond to a vector of attribute values, and the remaining r positions correspond to a boolean vector.[2]

We assume such a dataset has been used as a training set to build Naive Bayes Classifiers (NBC), that classify tags given the attribute values (one classifier per tag). The classifier for tag T_j defines the probability that a new item o is annotated by the tag T_j:

$$\begin{aligned} Pr(T_j \mid o) &= Pr(T_j \mid a_1, a_2, ..., a_m) \\ &= \frac{Pr(T_j).\Pi_{i=1}^{m} Pr(a_i \mid T_j)}{Pr(a_1, a_2, ..., a_m)} \end{aligned} \quad (1)$$

where a_i is the value of o for attribute A_i, $a_i \epsilon D_i$. The probabilities $Pr(a_i \mid T_j)$ are computed using the dataset. In particular, $Pr(a_i \mid T_j)$ is the proportion[3] of items tagged by T_j that have $A_i = a_i$. $Pr(T_j)$ is the proportion of items in the dataset that has T_j.

Similarly, we compute the probability $Pr(T_j' \mid o)$ of an item o not having tag T_j:

$$Pr(T_j' \mid o) = \frac{Pr(T_j').\Pi_{i=1}^{m} Pr(a_i \mid T_j')}{Pr(a_1, a_2, ..., a_m)} \quad (2)$$

We know that $Pr(T_j \mid o) + Pr(T_j' \mid o) = 1$; hence from Equations 1, 2 we get :

$$\begin{aligned} Pr(a_1, a_2, ..., a_m) &= Pr(T_j).\Pi_{i=1}^{m} Pr(a_i \mid T_j) + \\ &\quad Pr(T_j').\Pi_{i=1}^{m} Pr(a_i \mid T_j') \end{aligned} \quad (3)$$

From Equations 1, 3,

$$\begin{aligned} Pr(T_j \mid o) &= Pr(T_j \mid a_1, a_2, ..., a_m) \\ &= \frac{Pr(T_j).\Pi_{i=1}^{m} Pr(a_i \mid T_j)}{Pr(T_j).\Pi_{i=1}^{m} Pr(a_i \mid T_j) + Pr(T_j').\Pi_{i=1}^{m} Pr(a_i \mid T_j')} \\ &= \frac{1}{1 + \frac{Pr(T_j')}{Pr(T_j)} \Pi_{i=1}^{m} \frac{Pr(a_i \mid T_j')}{Pr(a_i \mid T_j)}} \end{aligned}$$

For convenience we use the notation

$$R_j = \frac{Pr(T_j')}{Pr(T_j)} \Pi_{i=1}^{m} \frac{Pr(a_i \mid T_j')}{Pr(a_i \mid T_j)} \quad (4)$$

Consider a query which picks a set of desirable tags $T^d = \{T_1, \ldots, T_z\} \subseteq T$.

The expected number of desirable tags $T_j \in T^d$ that a new item o, characterized by $(a_1, a_2, ..., a_m) \in A$ is annotated with, is given by:

$$\mathbb{E}(o, T^d) = \Sigma_{j=1}^{z} \frac{1}{1 + R_j} \quad (5)$$

We are now ready to formally define the main problem.

TAG MAXIMIZATION PROBLEM *Given a dataset of tagged items $\mathbb{D} = \{o_1, o_2, ..., o_n\}$, and a query T^d, design k new items that have the highest expected number of desirable tags they are likely to receive, given by Equation 5.*

mon with Naive Bayes Classifiers, we assume that they have been appropriately binned into discrete ranges.

[2]A more complex framework which leverages the frequencies of tags is left for future work.

[3]The observed probabilities are smoothened using the Bayesian m-estimate method [3]. We note that more sophisticated Bayesian methods that use an informative prior may be employed instead.

For the rest of the paper, we explain our algorithms in a boolean framework, which can be readily generalized to handle the case of categorical data. We also assume that all tags are of equal "weight"− if tags are of varying importance, Equation 5 can be re-written as a weighted sum, and all our proposed algorithms can be modified accordingly.

3. COMPLEXITY ANALYSIS AND EXACT ALGORITHMS

Our first result is on the NP-completeness of the Tag Maximization problem.

THEOREM 1. *The Tag Maximization problem is NP-Complete even for boolean datasets and for $k = 1$.*

Proof: The membership of the decision version of the problem in NP is obvious. To verify NP-hardness, we reduce the 3SAT problem to the decision version of our problem. We first reduce the 3SAT problem to the minimization version of the optimization problem, represented as $\mathbb{E}^{min}(o, T^d)$ and then reduce $\mathbb{E}^{min}(o, T^d)$ to $\mathbb{E}(A, T^d)$.

Reduction of 3SAT to decision version of $\mathbb{E}^{min}(o, T^d)$:

3SAT is the popular NP-complete boolean satisfiability problem in computational complexity theory, an instance of which concerns a boolean expression in conjunctive normal form, where each clause contains exactly 3 literals. Each clause C_j is mapped to a tag T_j in the instance of $\mathbb{E}^{min}(o, T^d)$ and each variable x_i is mapped to attribute value a_i. We make the following assignments so that if there is a boolean assignment vector $\vec{a} = [a_1, ..., a_m]$ that satisfies 3SAT, then $\mathbb{E}^{min}(o, T^d)$ equals zero (and if $\vec{a}$ does not satisfy 3SAT, then $\mathbb{E}^{min}(o, T^d)$ has a non-zero sum).

- For a variable x_i specified as positive literal in 3SAT, set $\Pr(a_i = 0 \mid T_j) = 1$
- For a variable x_i specified as negative literal in 3SAT, set $\Pr(a_i = 1 \mid T_j) = 1$
- For a particular clause and for the unspecified attributes (variables), set $\Pr(a_i = 0 \mid T_j) = \Pr(a_i = 1 \mid T_j) = 1$

For example, consider 3SAT instance $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$. For each tag, we create two products. For instance for the first tag, x_1 (that corresponds to A_1) is negative and hence for both the first and second product it is $A_1 = 1$. x_4 is missing from the first tag, hence for the first product it is $A_4 = 0$ and for the second it is $A_4 = 1$.

Table 1: Table of boolean attributes and tags

Attributes				Tags	
A_1	A_2	A_3	A_4	T_1	T_2
1	0	1	0	1	0
1	0	1	1	1	0
0	1	0	1	0	1
0	1	1	1	0	1

Reduction of $\mathbb{E}^{min}(A, T^d)$ to $\mathbb{E}(A, T^d)$:

If we have a boolean assignment vector $\vec{a} = [a_1, ..., a_m]$ that minimizes the expected number of tags being present, we have the corresponding $\Pr(T_j' \mid a_1, a_2, ..., a_m)$. Hence, we get $\Pr(T_j \mid a_1, a_2, ..., a_m) = 1 - \Pr(T_j' \mid a_1, a_2, ..., a_m)$ that maximizes the expected number of tags being present. □

A brute-force exhaustive approach (henceforth, referred to as **Naive**) to solve the Tag Maximization problem requires us to design all possible 2^m number of items and compute $\mathbb{E}(o, T^d)$ for each possible item. Note that the number of items in the dataset is not important for the execution cost, since an initialization step can calculate all the conditional tag-attribute probabilities by a single scan of the dataset. Although general purpose pruning-based optimization techniques (such as branch-and-bound algorithms) can be used to solve the problem more efficiently than Naive, such approaches are only limited to constructing the top-1 item, and it is not clear how they can be easily extended for $k > 1$.

In the following subsection, we propose a novel exact algorithm for any k based on interesting and nontrivial adaptations of top-k query processing techniques. This algorithm is shown in practice to explore far fewer item candidates than Naive, and works well for moderate problem instances.

3.1 Exact Two-Tier Top-k Algorithm

We develop an exact *two tier* top-k algorithm (*ETT*) for the Tag Maximization problem. For simplicity, henceforth we refer to desirable tags as just tags. The main idea of *ETT* is to determine the "best" items for each individual tag in tier-1 and then match these items in tier-2 to compute the globally best items (across all tags). Both tiers use pipelined techniques to minimize the amount of accesses, as shown in Figure 1. The output of tier-1 is z unbounded buffers (one for each tag) of complete items, ordered by decreasing probability for the corresponding tag. These buffers are not fully materialized, but may be considered as "sub-systems" that can be accessed on demand in a pipelined manner.

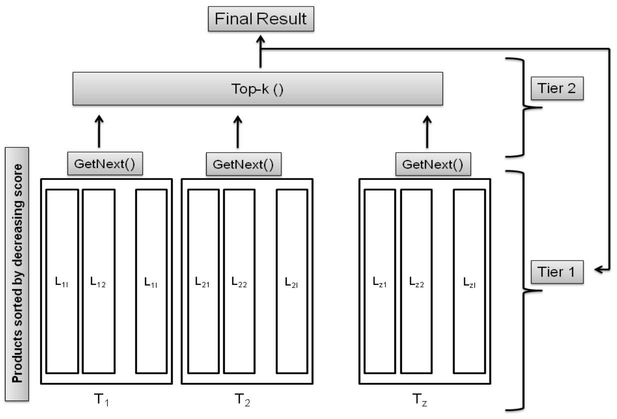

Figure 1: Two-Tier Top-K Algorithm Framework

In tier-2, the top items from the z buffers are combined in a pipelined manner to produce the global top-k items, akin the *Threshold Algorithm* (TA) [4]. In turn, tier-2 makes GetNext() requests (see Figure 1) to various buffers in tier-1 in round-robin manner. In tier-1, for each specific tag, we partition the set of attributes into subsets, and for each subset of attributes we precompute a list of all possible partial attribute value assignments, ordered by their "score" for the specific tag (the score will be defined later). The partial items are then scanned and joined, leveraging results from Rank-Join algorithms [9] that support top-k ranked join queries in relational databases, in order to feed information to tier-2. We provide more details below.

3.1.1 Tier-1

Suppose we partition the m attributes into l subsets, where each subset has $m' = \frac{m}{l}$ attributes as follows: $\{a_1, \ldots, a_{m'}\}$, $\{a_{m'+1}, \ldots, a_{2m'}\}$, ..., $\{a_{m-m'+1}, \ldots, a_m\}$. We create partial item lists $L_{j1}, \ldots, L_{jl}$ for each tag T_j. Each list L_{ji} has $2^{m'}$ entries (partial items). Consider the first list L_{j1}. The *score* of a partial item $o^p \in L_{j1}$ with attribute values $a_1, \ldots, a_{m'}$ for T_j is

$$\mathbb{E}_{partial}(o^p, \{T_j\}) \quad = \quad \sqrt[l]{P_j}.\Pi_{i=1}^{m'} \frac{Pr(a_i \mid T_j')}{Pr(a_i \mid T_j)} \qquad (6)$$

where $P_j = \frac{Pr(T_j')}{Pr(T_j)}$. Note that the l-th root of P_j is used in order to distribute the effect of P_j from Equation 4 to the l lists, such that when they are combined using multiplication, we get P_j.

Lists L_{jl} are ordered by descending $\frac{1}{E_{partial}}$, since R_j appears on the denominator of Equation 5. The l lists are accessed in round-robin fashion and for every combination of partial items from the lists, we join them to build a complete item and resolve its exact score by Equation 5.

An item is returned as a result of GetNext() to tier-2 if its score is higher than the MPFS (*Maximum Possible Future Score*), which is the upper bound on the score of an unseen item. To compute MPFS, we assume that the current entry from a list is joined with the top entries from all other lists:

$$MPFS = \frac{1}{1 + max((s_{j1}.h_{j2}.. \cdot h_{jl}), (h_{j1}.s_{j2}.. \cdot h_{jl}), .., (h_{j1}.h_{j2}.. \cdot s_{jl}))} \quad (7)$$

where s_{ji} and h_{ji} are the last seen and top entries from list L_{ji} respectively.

3.1.2 Tier-2

In this tier, the z unbounded buffers, one for each tag, are combined using the summation function, as shown in Equation 5. Each item from one buffer matches exactly one entry (the identical item) from each of the other buffers. Items are retrieved from each buffer using GetNext() operations, and once retrieved we directly compute its score for all other tags by running each Naive Bayes classifier, without using the process of tier-1. An item is output if its score is higher than the threshold, which is the sum of the last seen scores from all z buffers. A bounded buffer with the k best results so far is maintained. On termination, this buffer is returned as the top-k items.
The pseudocode of ETT is shown in Algorithm 1.

Table 2: Example tagged items dataset

	Attribute				Tag	
ID	A_1	A_2	A_3	A_4	T_1	T_1
1	0	0	0	1	0	0
2	0	1	0	0	0	1
3	0	1	0	1	0	0
4	0	1	1	1	1	1
5	1	0	0	0	1	0
6	1	0	0	1	0	1
7	1	0	1	1	1	1
8	1	1	0	1	0	1

EXAMPLE 1. Consider the boolean dataset of 10 objects, each entry having 4 attributes and 2 tags in Table 2. We partition the 4 attributes into groups of 2 attributes: (A_1, A_2) form list L_{j1} and (A_3, A_4) form list L_{j2}. We run NBC

Algorithm 1 **ETT (Naive Bayes probabilities, attributes per group m', k)**: top-k exact items

```
  //Main Algorithm
 1: Top-$k$-Buffer $\leftarrow \{\}$
 2: for $j = 1$ to $z$ do
 3:   $B_j \leftarrow \{\}$ // unbounded buffer of candidate results-
      items per tag
 4:   for $i = 1$ to $l$ do
 5:     $s_{ji}, h_{ji} \leftarrow$ top entry from list $L_{ji}$
 6:   end for
 7: end for
 8: Call Threshold()

  //Method Threshold() – Tier-2
 1: while true do
 2:   for $j = 1$ to $z$ do
 3:     $(o_j, score_j(o_j)) \leftarrow$ GetNext($j$)
 4:     ExactScore($o_j$) $\leftarrow$ Compute for $o_j$ by Equation 5
 5:   end for
 6:   Update Top-$k$-Buffer with new items if necessary
 7:   MinK $\leftarrow$ lowest score in Top-$k$ buffer
 8:   $\alpha \leftarrow \sum_j score_j(o_j)$ // Threshold
 9:   if MinK $\geq \alpha$ then
10:     return top-$k$ items
11:   end if
12: end while

  //Method GetNext($j$) : $(o_j, score_j(o_j))$ – Tier-1
 1: while true do
 2:   Compute MPFS by Equation 7
 3:   // $score_j(o)$ for item $o$ is defined as $1/(1 + R_j)$ ($R_j$
      defined by Equation 4)
 4:   if $B_j$ has an item $o$ with $score_j(o) > MPFS$ then
 5:     return ($o$, $score_j(o)$) AND remove it from $B_j$
 6:   end if
 7:   Retrieve next entry $o^p$ from a list $L_{ji}$ in round robin
      and advance $s_{ji}$
 8:   Join $o^p$ with all combinations of partial items from
      other lists and create all items $NewItems$
 9:   Add $NewItems$ to buffer $B_j$ of candidate results-
      items
10: end while
```

and calculate all conditional tag-attribute probabilities. The algorithm framework for the running example is presented in Figure 2. List L_{11} and L_{12} under tag T_1 is sorted in decreasing order of $\frac{1}{E_{partial}}$, given by Equation 6 (and, similarly for L_{21} and L_{22} under tag T_2).

In iteration 1, call to Threshold() in tier-2 calls GetNext() for T_1 and T_2 respectively in tier-1. During GetNext(T_1), join-1 builds item 1010, whose $score_1(1010) = 0.95$ and MPFS(1010) =0.95. Since $score_1 \geq$ MPFS, (1010,0.95) is returned to tier-2. GetNext(T_2) returns (1111,0.93) to tier-2. In tier-2, call to Threshold() returns ExactScore(1010) =1.70, ExactScore(1111) =1.75. Now, top-k-buffer gets 1111; MinK=1.75 and α=1.88. Since MinK $\leq \alpha$, we continue to iteration 2. In iteration 2, we proceed similarly. GetNext(T_1) returns (1011, 0.92) and GetNext(T_2) returns (1110,0.88) to tier-2. Call to Threshold() in tier-2 gives ExactScore(1011) =1.76, ExactScore(1110) =1.77. The top-k-buffer is updated to 1110; MinK=1.77 and α=1.79. Since MinK $\leq \alpha$, we continue to iteration 3. In tier-1 of iteration 3, GetNext(T_1) returns (0010,0.89) and GetNext(T_2) returns (0111,0.84) to tier-2. Then Threshold() is called and we get ExactScore(0010) =1.76, ExactScore(0111) =1.77. The Bounded Buffer continues to be 1110; MinK=1.77 and α=1.74. We see that MinK $\geq \alpha$. Hence, ETT terminates and returns 1110 as the top-1 item. Thus, ETT returns the best item by just looking up 6 items, instead of 16 items (as in Naive algorithm). □

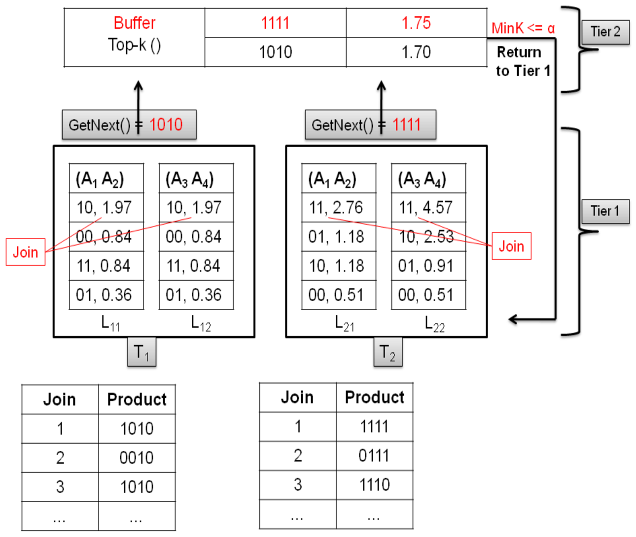

Figure 2: Iteration 1: Exact Two-Tier Top-K Algorithm for Example in Table 2

4. APPROXIMATION ALGORITHM

The exact algorithm of Section 3.1 is feasible only for moderate instances of the Tag Maximization problem. For larger problem instances, in this section we discuss a principled approximation algorithm (**PA**, or *polynomial time approximation algorithm*) that provides guarantee in the quality of the top-k results *as well as* running time.

The main idea is to group the desirable tags into *constant-sized* groups of z' tags each, find the top-k items for each subgroup, and output the overall top-k items from among these computed items.[4] For each sub-problem thus created, we show that it can be solved by a *polynomial time approximation scheme* (PTAS) [5], i.e., can be solved in polynomial time given any user-defined approximation factor. The overall running time of the algorithm is exponential only in the (constant) size of the groups, thus giving overall a polynomial time complexity.

We now consider a sub-problem consisting of only a constant number of tags, z'. We also restrict our discussion to the case $k = 1$ (more general values of k are discussed later). We shall design a polynomial time approximation scheme (PTAS) for this sub-problem. A PTAS is defined as follows. Let $\epsilon > 0$ be any user-defined parameter. Given any instance of the sub-problem, let PTAS return the item o_a.

[4]Interestingly, we note that in this algorithm we create (z/z') sub-problems by grouping tags; in contrast in our exact ETT algorithm we create sub-problems (i.e., subsystems) by grouping attributes.

Let the optimal item be o_g. The PTAS should run in polynomial time, and $ExactScore(o_a) \geq (1-\epsilon)ExactScore(o_g)$.

In describing the PTAS, we first discuss a simple exponential time exact top-1 algorithm for the sub-problem, and then show how it can be modified to the PTAS. Given m boolean attributes and z' tags, the exponential time algorithm makes m iterations as follows: As an initial step, it produces the set S_0 consisting of the single item $\{0^m\}$ along with its z' scores, one for each tag. In the first iteration, it produces the set containing two items $S_1 = \{0^m, 10^{m-1}\}$ each accompanied by its z' scores, one for each tag. More generally, in the ith iteration, it produces the set of items $S_i = \{\{0,1\}^i \times 0^{m-1}\}$ along with their z' scores, one for each tag. Each set can be easily derived from the set computed in the previous iteration. Once m iterations have been completed, the final set S_m contains all 2^m items along with their exact scores, from which the top-1 item can be returned, which is that product for which the sum of the z' scores is the largest. However, this algorithm takes exponential time, as in each iteration the sets double in size.

The main idea of the PTAS is to not allow the sets to become exponential in size. This is done by *compressing* each set S_i produced by each iteration to another smaller set S'_i, so that they remain polynomial in size. Each item entry in S_i can be viewed as points in a z'-dimensional space,whose z' co-ordinates correspond to the item scores for z' individual tags respectively, by Equation 5. Essentially, we use a clustering algorithm in z'-dimensional space, and for each cluster, only the representative item is retained, while all other items in the cluster are deleted. The clustering has to be done in a careful way so as to guarantee that for the items that are deleted, the representative item's exact score should be close to the deleted item's exact score. Thus when the top-1 item of the final compressed set S'_m is returned, its exact score should not be too different from exact score of the top-1 item assuming no compression was done.

The pseudocode of PA is shown in Algorithm 2.

Example 2. We execute PA on the example in Table 2 without any grouping of tags (i.e., $z = z' = 2$): Let the compression factor σ be 0.5. We start with $S'_0 = \{0000\}$. In iteration 1, $S_1 = \{0000, 1000\}$ with each item having two-dimensional co-ordinates (0.31, 0.20) and (0.51, 0.38) respectively. After compression, we get $S'_1 = \{1000\}$. In iteration 2, $S_2 = \{1000, 1100\}$ with two-dimensional co-ordinates (0.51, 0.38) and (0.31, 0.58) respectively. After compression we get $S'_2 = \{1000, 1100\}$. In iteration 3, $S_3 = \{1000, 1100, 1010, 1110\}$ with co-ordinates (0.51, 0.38), (0.31, 0.58), (0.95, 0.75) and (0.89, 0.88) respectively. After compression we get $S'_3 = \{1000, 1100, 1110\}$. In the final iteration 4, $S_4 = \{1000, 1100, 1110, 1001, 1101, 1111\}$ with co-ordinates (0.51, 0.38), (0.31, 0.58), (0.89, 0.88), (0.37, 0.52), (0.20, 0.72) and (0.82, 0.93). After compression we get $S'_4 = \{1000, 1100, 1111\}$. The top-1 approximate item is 1111 with score 1.75 while the optimal item is 1110 with score 1.77. Figure 3 shows the compression in the four iterations. The boolean items in red font are the cluster representatives.$\square$

Theorem 2. *Given a user defined approximation factor ϵ, if we group the tags into z/z' groups of z' tags per group, and set the compression factor $\sigma = \epsilon/2m$ then*

1. *The output of PA has an exact score that is at least $\frac{z'}{z(1+\epsilon)}$ times the exact score of the optimal item*

Algorithm 2 PA (Naive Bayes probabilities, attributes per group z', compression factor σ): top-1 approximate item in polynomial time

```
 //Main Algorithm
 1: Partition tags $T$ into $z/z'$ groups $T_1, \ldots, T_{z/z'}$
 2: for $r = 1$ to $\frac{z}{z'}$ do
 3:     $o_r \leftarrow$ PTAS($T_r$)
 4:     Compute ExactScore($o_r$) by Equation 5
 5: end for
 6: return $o_r$ with max ExactScore

 //Method $PTAS(T_r) : o$)
 1: $S'_0 \leftarrow \{0^m\}$ // boolean vector of size $m$ with all 0's
 2: for $i = 1$ to $m$ do
 3:     $S_i = S'_{i-1} \cup S''_{i-1}$ // $S''_{i-1}$ : $S'_{i-1}$ with $i$th bit set to 1
 4:     // Compress $S_i$ to $S'_i$ using compression factor $\sigma$
 5:     $S'_i \leftarrow \{\}$
 6:     repeat
 7:         $o \leftarrow$ any selected item in $S'_{i-1}$
 8:         $S'_i \leftarrow S'_i \cup \{o\}$
 9:         Delete from $S_i$ all items $o'$ such that $\forall T_j \in T_r$,
            $|\mathbb{E}(o, \{T_j\}) - \mathbb{E}(o', \{T_j\})| \leq \sigma\mathbb{E}(o, \{T_j\})$
10:     until $S_i$ is empty
11: end for
12: return item $o$ in $S'_m$ with largest $|\mathbb{E}(o, T_r)|$
```

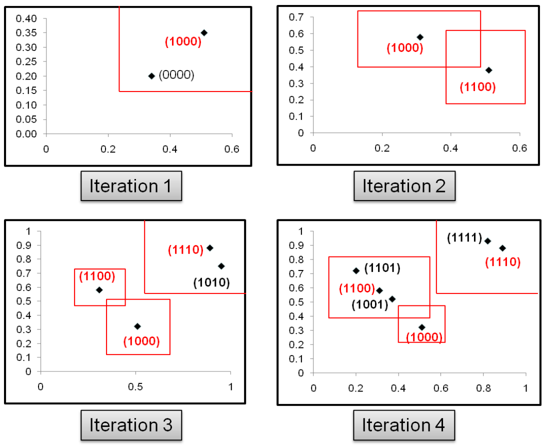

Figure 3: Compression in PA Algorithm for Example Dataset of Two Tags in Table 2

2. *PA runs in polynomial time*

Proof of Part 1: Consider any tag group T_r, and let o^{OPT} be the optimal item for this group, and o^{APP} be the item returned by PTAS. For every item o in the set S_m (assuming no compression was used in any iterations), there is an item o_a in the compressed set S'_m that satisfies

$$\mathbb{E}(o, T_r) \leq (1+\sigma)^m \mathbb{E}(o_a, T_r) \tag{8}$$

In particular, the following holds

$$\mathbb{E}(o^{OPT}, T_r) \leq (1+\sigma)^m \mathbb{E}(o^{APP}, T_r) \tag{9}$$

Since $\sigma = \epsilon/2m$, some algebraic simplifications results in:

$$\mathbb{E}(o^{OPT}, T_r) \leq (1+\epsilon) \mathbb{E}(o^{APP}, T_r) \tag{10}$$

The above analysis is for a single tag group. Since there are z/z' tag groups, it is easy to see that this introduces an additional factor of z'/z to the overall approximation factor.

Proof of Part 2 (sketch): To show that PA is a polynomial time algorithm, the main task is to show that the compressed lists are always polynomial in length. We first observe that probability quantities such as $Pr(a_i \mid T_j)$ are rational numbers, where both the numerator as well as the denominator are integers bounded by n (number of items in the dataset). If we then consider Equation 5, we can conclude that the score of any item for any single tag can be represented as a rational number, where the numerator and denominator are integers bounded by $O(n^m)$. Thus, we can normalize each such score into an integer by multiplying it with $O(n^m)$.

Next, consider a z'-dimensional cube with each side of length $L = O(n^m)$. We partition the cube into z'-dimensional *cells* as follows: Along each axis, start with the furthest value L, and then proceed towards the origin by marking the points $L/(1+\sigma)$, $L/(1+\sigma)^2$, and so on. The number of points marked along each axis is $\log_{(1+\sigma)} L = O(m \log_{(1+\sigma)} n)$ which is a polynomial in m and n. Then at each marked point we pass $(z'-1)$-dimensional hyperplanes perpendicular to the corresponding axis. Their intersections creates $O(poly(m,n)^{z'})$ cells within cube $L^{z'}$.

Due to this skewed method of partitioning cube into cells, we see that the cells that are further away from the origin are larger. Consider the ith iteration of the PTAS algorithm. Each item in S_i may be represented as a point in this cube. Though within any cell there may be several points corresponding to items of S_i, after compression *there can be at most only one* point corresponding to an item of S'_r, because two or more points could not have survived the compression process as the score between them is too small.

The length of any compressed list in the PTAS algorithm is at most $O(poly(m,n)^{z'})$. When z' is a constant, this translates to an overall polynomial running time for PA. □

Extending from Top-1 to Top-k: Our PA algorithm can be modified to return top-k items instead of just the best item. For the tag group T_r, once a set of items S_i is built, we compress to form the set S'_i. However, every time a cluster representative is selected, instead of deleting all the remaining points in the cluster, we remember $k-1$ items within the cluster and associate them with the cluster representative (and if the cluster has less than k items, we remember and associate all the items with the cluster representative).

When all the m iterations are completed, we can return the top-k items as follows: we first return the best item of S'_m along with the $k-1$ items associated with it. If the number of associated items are less than $k-1$, the second best cluster representative of S'_m and the set of items associated with it are returned, and so on.

When the approximate top-k items from all tag groups have been returned, the main algorithm returns the overall best top-k items from among them. It can be shown that this approach guarantees an approximation factor for the score of the top-k items returned.

5. EXPERIMENTS

We conduct a set of comprehensive experiments using both synthetic and real datasets for quantitative and qualitative analysis of our proposed algorithms. Our quantitative performance indicators are (a) *efficiency* of the proposed exact and approximation algorithm, and (b) *approximation factor* of results produced by the approximation algorithm. The efficiency of our algorithms is measured by the overall execution time and the number of items that are considered from the pool of all possible items, whereas approximation factor is measured as the ratio of the acquired approximate result score to the actual optimal result score. We also conduct a user study through Amazon Mechanical Turk study to qualitatively assess the results of our algorithms.

System configuration: Our prototype system is implemented in Java with JDK 5.0. All experiments were conducted on an Windows XP machine with 3.0Ghz Intel Xeon processor and 2GB RAM. The JVM size is set to 512MB. All numbers are obtained as the average over three runs.

Real Camera Dataset: We crawl a real dataset of 100 cameras[5] listed at Amazon (http://www.amazon.com). The products contain technical details (attributes), besides the tags customers associate with each product. The tags are cleaned by domain experts to remove synonyms, unintelligent and undesirable tags such as `nikon coolpix`, `quali`, `bad`, etc. Since the camera information crawled from Amazon lacks well-defined attributes, we look up Google Products (http://www.google.com/products) to retrieve a rich collection of technical specifications for each product. Each product has 40 boolean attributes, such as `self-timer, face-detection, red-eye fix`, etc; while the tag dictionary includes 40 unique keywords like `lightweight`, `advanced`, `easy`, etc.

Synthetic Dataset: We generate a large boolean matrix of dimension 10,000 (items)×100 (50 attributes + 50 tags) and randomly choose submatrices of varying sizes, based on our experimental setting. We split the 50 independent and identically distributed attributes into four groups, where the value is set to 1 with probabilities of 0.75, 0.15, 0.10 and 0.05 respectively. For each of the 50 tags, we pre-define relations by randomly picking a set of attributes that are correlated to it. A tag is set to 1 with a probability p if majority of the attributes in its pre-defined relation have boolean value 1.

We use the synthetic datasets for quantitative experiments, while the real dataset is used in the user study.

5.1 Quantitative Results: Performance

Exact Algorithm: We first compare the Naive approach with our ETT. Since the Naive algorithm can only work for small problem instances, we a pick a subset from the synthetic dataset having 1000 items, 16 attributes and 8 tags. Figures 4 and 5 compare the execution time and the number of candidate items considered, for Naive and ETT respectively, when the number of attributes (m) varies (number of items = 1000, number of tags = 8). The Naive algorithm considers all 2^m items. We used as number of attributes per group $m' = 2,2,4,5,4,7,4,6$ for $m = 4,6,8,10,12,14,16,18$ respectively in ETT (more analysis of m' in Figure 6). As can be seen, Naive is orders of magnitude slower than ETT.

Next, we study the behavior of attribute groupings on ETT. For a sub-sample picked from our synthetic dataset having 20 attributes, 1000 items and 8 tags, we experiment with different possible attribute groupings, m' = 1, 2, 4, 5, 10, 20. Figure 6 shows the effect of m' on the performance of ETT. The execution time and number of items considered

[5]As discussed earlier, the number of items in the dataset is not important for the execution cost; analysis in Figure 8.

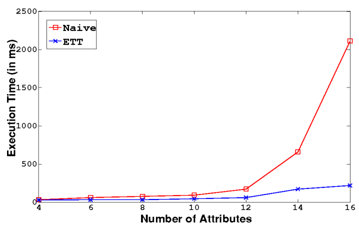

Figure 4: Execution time for varying m (Synthetic data)

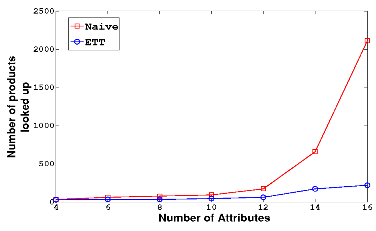

Figure 5: Number of items built for varying m (Synthetic data)

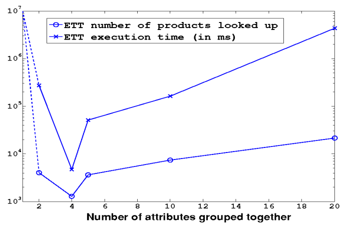

Figure 6: ETT behavior for varying m' (Synthetic data)

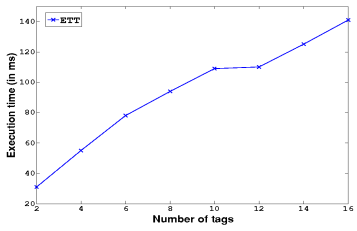

Figure 7: Execution time for varying z (Synthetic data)

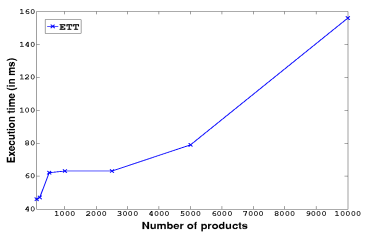

Figure 8: Execution time for varying n (Synthetic data)

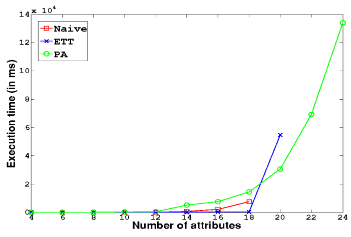

Figure 9: Execution time for varying m (Synthetic data)

for $m' = 1$ is not reported in Figure 6 as it was too slow. The trade-off of choosing m' is: a small m' means there are many short lists in tier-1, so that the cost of joining the lists is high. In contrast, a large m' indicates fewer but longer lists in tier-1 resulting in increased cost of creating the lists. We observe that the best balance is struck when $m' = 4$ attributes forming 5 lists, each having 2^4=16 items.

We also vary the number of tags z and number of items n in the dataset to study the behavior of ETT. We pick a subset from the synthetic dataset having 1000 items, 16 attributes and 16 tags, and consider further subsets of this dataset. Figure 7 reflects the change in execution time with increasing number of tags for the synthetic data (number of items = 1000, number of attributes = 12, attribute grouping = 3). The increase in number of tags increases the number of GetNext() operations in ETT, and hence the running time rises steadily. Figure 8 depicts how an increase in the number of items in the dataset (number of attributes = 12, number of tags = 8, attribute grouping = 3) barely affects the running time of ETT since an initialization step calculates all conditional tag-attribute probabilities.

Table 3: PA Performance (Synthetic data)

User-Defined Approx Factor(ϵ)	Execution Time (in ms)	Obtained Approx Factor
0.5	406.0	0.93
0.4	749.0	0.94
0.3	2796.0	0.97
0.2	41094.0	0.98

Approximation Algorithm: We observe in Figure 4 that the execution time of ETT outperforms that of Naive, for moderate data instances. Figure 9 reveals that while ETT is extremely slow beyond number of attributes (m) = 16, PA with an approximation factor ϵ=0.5, continues to return guaranteed results in reasonable time with increasing number of attributes m for a subset of the synthetic data (number of items = 1000, number of tags = 8). Therefore, PA performs better than ETT for larger datasets having many attributes and tags.

Table 3 depicts how the execution time and the obtained approximation factor vary with change in the user-defined approximation factor ϵ, for a subset of the synthetic dataset having 1000 items, 20 attributes and 8 tags . Note that, in this experimental set-up, z=z'=8 so that the number of tag groups = 1. The obtained approximation factor is the ratio of the score for top-1 item obtained by PA to the score of the optimal *item* obtained by an extended run of ETT.

5.2 Qualitative Results: User Study

We now validate how designers can leverage existing item information to design new items catering different groups of people in a user study conducted on Amazon Mechanical Turk (https://www.mturk.com) on the real camera dataset. We also consult DPreview (http://www.dpreview.com), a website about digital cameras and digital photography. There are two parts to our user study. Each part of the study involves thirty independent single-user tasks. Each task is conducted in two phases: *User Knowledge Phase* where we estimate the users' background and *User Judgment Phase* where we collect users' responses to our questions.

In the first part of our study, we build four new cameras (two digital compact and two digital slr) using our PA algorithm with an approximation factor ϵ=0.5, by considering tag sets corresponding to compact cameras and slr cameras respectively. We present these four new cameras along with four existing popular cameras (presented anonymously) and observe that 65% of users choose the new cameras, over the existing ones. For example, users overwhelmingly prefer our new compact digital camera over Nikon Coolpix L22 because the former supports both automatic and manual focus while the latter does not, thus validating how our techniques can benefit designers.

The second part of the study concerns six new cameras designed for three groups of people: young students, old retired and professional photographers. Domain experts identify and label three overlapping sets of tags from the camera dataset's complete tag vocabulary, one set for each group and we then build two potential new cameras for each of the three groups. For each of the six new cameras thus built, we ask users to assign at least five tags by looking up the complete camera tag vocabulary, provided to them. We observe that majority of the users rightly classify the six cameras into the three groups. The correctness of the classification is validated by comparing the tags received for a camera to the three tag sets identified by domain experts; we also validate the correctness by consulting data available in Dpreview. As an example, the cameras designed by leveraging tags corresponding to professional photographers draw tags like `advanced`, `high iso`, etc. while cameras designed by leveraging tags corresponding to old retired draw tags like `lightweight`, `easy`, etc. Figure 10 shows the percentage of users classifying the six cameras correctly. Thus, our technique can benefit designers build new items that are likely to attract desirable tags from different groups of people.

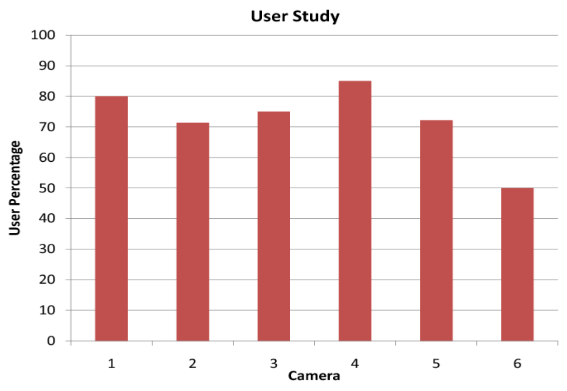

Figure 10: Users Classify Cameras Correctly

6. RELATED WORK

The dynamics of social tagging has been an active research area in recent years, with several papers focusing on the tag prediction problem. A recent work [16] proposes a probabilistic model for personalized tag prediction and employs the Naive Bayes classifier. Related research in text mining [12] found that the Naive Bayes classifier performs better than SVM and CRF in classifying blog sentiments. Another study that indirectly supports the use of Naive Bayes for tag prediction is done by Heymann et al. [7], who found that tag-based association rules can produce very high-precision predictions. The process of collaborative tagging has been studied in [6]. Other related work investigates tag suggestion, usually from a collaborative filtering and UI perspective; for example with URLs [15] and blog posts [11].

The problem of item design has been studied by many disciplines including economics, industrial engineering and computer science [13]. Optimal item design or positioning is a well studied problem in Operations Research and Marketing. Shocker *et al.* [14] first represented products and consumer preferences as points in a joint attribute space. Later, several techniques [1, 2] were developed to design/position a new item. Work in this domain requires direct involvement of consumers, who choose preferences from a set of existing alternative products. Miah et al. [10] study the problem of selecting product snippets given a user query log, in order for the designed snippet to be returned by the maximum number of queries. However, none of these works has studied the problem of item design in relation to social tagging.

Our top-k pipelined algorithm is inspired by the rich work on top-k algorithms ([4, 9]). A recent survey by Ilyas et al. [8] covers many of the important results in this area.

7. CONCLUSIONS

In this paper we consider the novel problem of leveraging online collaborative tagging in product design. We formally define the Tag Maximization problem, investigate its computational complexity, and propose several principled algorithms that are shown to work well in practice. Our work is a preliminary look at a very novel area of research, and there appear to be many exciting directions of future research. Our immediate focus is to extend our work to include tag prediction using other classifiers, such as decision trees, SVMs, and regression trees (the latter is applicable when we wish to predict the frequency of occurrence of desirable tags attracted by products). We also intend to evaluate the applicability of our proposed framework to other novel applications, e.g., guide recommender systems recommend better vacation travel itineraries by tracking tag history, help online authors write better blogs, and others.

8. REFERENCES

[1] S. Albers and K. Brockhoff. Optimal product attributes in single choice models. *Journal of the Operational Research Society, 31, 647-655*, 1980.

[2] M. D. Albritton and P. R. McMullen. Optimal product design using a colony of virtual ants. *European Journal of Operational Research*, 176(1):498–520, January 2007.

[3] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *ECAI*, pages 147–149, 1990.

[4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[6] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *J. Inf. Sci.*, 32(2):198–208, 2006.

[7] P. Heymann, D. Ramage, and H. Garcia-Molina. Social tag prediction. In *SIGIR*, pages 531–538, 2008.

[8] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[9] I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Rank-join algorithms for search computing. In *SeCO Workshop*, pages 211–224, 2009.

[10] M. Miah, G. Das, V. Hristidis, and H. Mannila. Determining attributes to maximize visibility of objects. *IEEE Transactions on Knowledge and Data Engineering*, 21:959–973, 2009.

[11] G. Mishne. Autotag: a collaborative approach to automated tag assignment for weblog posts. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 953–954, New York, NY, USA, 2006. ACM Press.

[12] A. Pak and P. Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, 2010.

[13] T. Selkar and W. Burlesson. Context-aware design and interaction in computer systems. *IBM Syst. J.*, 39:880–891, July 2000.

[14] A. D. Shocker and V. Srinivasan. A Consumer-Based Methodology for the Identification of New Product Ideas. *MANAGEMENT SCIENCE*, 20(6):921–937, 1974.

[15] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006*, Edinburgh, Scotland, 2006.

[16] D. Yin, Z. Xue, L. Hong, and B. D. Davison. A probabilistic model for personalized tag prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 959–968, New York, NY, USA, 2010. ACM.