ADWIRE: ADD-ON FOR WEB ITEM REVIEW SYSTEM

by

RAJESHKUMAR GANESH KANNAPALLI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2016

To my loving father Late.Ganesh Kannapalli, my mother Lalitha and my brother Rakesh.

ACKNOWLEDGEMENTS

I would like to convey my warmest gratitude to my supervising professor Dr. Gautam Das for giving me this opportunity and for his constant guidance, support and encouragement from the start until the end of my study. I am sincerely thankful to Dr. Ramez Elmasri and Dr. Vassilis Athitsos for their interest in my research and taking their valuable time to serve in my committee.

I would like to thank my PhD mentor Azade Nazi, for her constant support, patience and belief in me. I would also like to thank Ms. Camille Costabile in helping me through the final requirements of my thesis, and the department of computer science, UTA. Special thanks to all my teachers who have taught me and help me grow by providing knowledge and education.

My heartfelt thanks to my family and friends for continuous support and inspiration. Finally, I would like to thank all who have either directly or indirectly helped me in completing my thesis in time.

April 25, 2016

ABSTRACT

ADWIRE: ADD-ON FOR WEB ITEM REVIEW SYSTEM

RAJESHKUMAR GANESH KANNAPALLI, M.S.

The University of Texas at Arlington, 2016

Supervising Professor: Dr. Gautam Das

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

In the course of recent decades as purchasing choices moved on the web, the broad use and fame of online survey destinations has at the same time expanded. Users post their experience about various items and services in online sites like Yelp, TripAdvisor, Amazon, and so forth in form of reviews and ratings. Users depend upon these evaluations and surveys as the valid voice of the purchaser, guiding sales online and in-store, improve conversion rate, bounce rate and time on site. According to a survey conducted by Dimensional Research, 2013, "90% Customers Say Buying Decisions Are Influenced By Online Reviews". Such importance towards ratings and reviews has motivated business of various kinds to possess an in-house arsenal of precious user feedback for marketing and development purposes. Out of the wide variety of user feedback options, numeric scores are used in majority of areas (e.g., 5 star ratings for Laptop, average rating of a Mobile, etc.). These ratings fail to convince user as they don‘t provide information in terms of experience, feel, ease, etc., (e.g., keyboard feel of the laptop, build quality of Mobile, etc.) which can be provided by detailed reviews. Ratings also prove to be inefficient and confusing in many instances for making a decision. Consider a Product 1 with aggregate rating score of 3.0, number of ratings 100. Example., Consider Product 2 with aggregate rating score of 5.0, number of ratings 2. In this example, even tough Product 2 has higher aggregate rating than Product 1, the rating has less appreciation because number of reviews for Product 1 is much higher than Product 2. Thus, ratings alone cannot help every time the user makes a purchasing decision. The importance of detailed reviews is further enhanced in user's decision making if the product is expensive (e.g., buying expensive mobile phone). Thus, the importance of detailed reviews cannot be ignored but appreciated.

Though the 1% rule (or, the 90-9-1 rule) of Internet is presumed to be dead, a large number of users read user-generated content in the Web without contributing. According to survey conducted by Pew Internet in 2012, "Though 90% people conduct online product research, only 32% have ever posted a review online". Thus, the number of useful reviews available is far from many. This observation is a result of users tendency to overlook giving a survey as it requires time, efforts and is unrewarding. Moreover, a few sites like Hotels.com and IMDB permits users to submit feedback as ratings without any review accompaniment. Hence the eventual size of detailed review corpus available is restricted. The available text corpus further suffers from redundancy, spam, typographical and grammatical errors, etc., thus shrinking the already restricted corpus size available to make informed purchasing decisions. With such an importance for users and businesses both, the need to obtain good review corpus enhances. There exists many companies like Bazaarvoice, PowerReview, etc., which help organizations obtain reviews online for their growth and online presence. These companies encourage users to write reviews by various means such as establishing client engagement programs and providing discount coupons for writing a review, etc.

In [1] authors introduced the general TagAdvisor problem and proposed a practical solution for an organized and automated system to increase good online reviews. A good review can be considered as one which is concise, comprehensive, objective, usable and applicable. The idea being, in order to assist users write a good review, ease the task of reviewing online web item by providing a set of meaningful phrases. In a review an user can express broad opinions about the different aspects of an item which can be either positive or negative. Different opinions can also be expressed to same or different attributes of a product. Let us consider a review *Even though with a short battery life, phone has decent call quality and nice volume.* In this example, different item attributes such as Battery, Call Quality, Speakers along with their respective sentiments are independently mentioned. Hence individual attributes are either provided with positive sentiments or negative sentiments independently. Now consider another review example, *Awesome Dolby speakers for a phone,* *but has poor build quality.* In this example, positive and negative sentiment are mentioned against a single item attribute Speakers. Here different parts of the review statement express opposite sentiments for a similar attribute. In [1] proposed two coverage functions for these two styles of review writing and thereby defining two concrete problem instances, namely Independent Coverage TagAdvisor (IC-TA) and Dependent Coverage TagAdvisor (DC-TA) problems, that enable a wide range of real-world scenarios. In IC-TA, the coverage of an item attributes is independent of its sentiment, whereas is dependent on the sentiment in the DC-TA.

In this thesis, we build a system AD-WIRE which is a practical implementation for [1]. Our system identifies the top-k meaningful phrases/tags to help review the item easily for an user who wants to review an item. Using AD-WIRE the user can now compose a good review quickly by selecting phrases among a set of tags returned by the system. Thus abolishing the pain in providing time and effort required to compose a review, which is one of the main problems as mentioned earlier. Also, as phrases are selected to composes a review, the possibility for the review containing grammatical errors, etc., is decreased as the system provides an organized approach towards review writing. In order to enable a user to satisfactorily review an item, AD-WIRE considers three essential properties for the result set, —*relevance* (i.e., how well the provided set of tags describe an item to a user), *coverage* (i.e., how well the different aspects of an item are considered within the provided set of tags), and *polarity* (i.e., how well the set of tags match the opinion of the user). The design of AD-WIRE system is technically challenging for several reasons, one of reason being the solution provided in both the IC-TA and DC-TA problems is of type NP-Complete. The objective is to identify k tags that are relevant, cover maximum aspects of an item, and match the users sentiment. While the first two concerns the relationship between the item attributes and tags, the third is based on a user's personal preference. Most users provide positive comments, others tend to be critical. AD-WIRE demonstrates the result of both IC-TA and DC-TA problems, thus enabling both styles of review writing. It also visualizes the dependency of the tags to different aspects of an item so a user can make an informed decision quickly.

Going forward in coming chapters, we will initially discuss the TagAdvisor model which is provided by [1], and the coverage problems. This will be followed by the Architecture of AD-WIRE and System Demonstration. We will finally conclude evaluating the results obtained by our system.

CHAPTER 2

TAGADVISOR OVERVIEW

This chapter provides a broad overview of the TagAdvisor Problem as mentioned in [1]. We first describe the data model which is used by ADWIRE, then we will visit the problem statement along with the two different problem instances that enable a wide range of real-world scenarios.

2.1 Data Model

As each item consists of item attributes we can consider an item has a well defined schema $I_A = \{a_1, a_2, ..., a_m\}$ and each item i as a tuple $\{a.v_1, a.v_2, ..., a.v_m\}$ with I_A as schema, where $a.v_y$ is the value of item attribute a_y; e.g., In <brand=Samsung, model=Galaxy Note 3>, <brand, model> describes the item schema and <Samsung, Galaxy Note 3> describes the item as a tuple. Let n be the total number of tags in T such that, $T = \{t_1, t_2, ..., t_n\}$. The data D in a tuple $< I, T >$, will thus represent a set of items, and the tag vocabulary respectively. Each tag which is associated with an item action can be represented as $< i, \mathtt{T} >$ where $i \in I$, and $\mathtt{T} \in T$. Table 2.1 shows an example of the mobile data in the data store. As mentioned in [1], to predict the rules describing dependency between attributes and tags we use existing world class classifiers and Rule learning models. These models provide rules along with it‘s probability, if there are several rules for a tag t_x and an item i having attributes values $\{a.v_1, a.v_2, ...a.v_m\}$,the one with highest probability p will be selected. Our objective will then be to identify the top-k tags $T^* = \{t_1, t_2, ..., t_k\}$ for a item $i \in I$ such that a user can find the set of tags T^* “meaningful”, and can provide review for an item i with ease by selecting tags from T^*.

Table 2.1: An example mobile phone review data as $< I, T >$

Items (I)								**Tags (T)**	
Item (i)	Color (a_1)	Secondary Camera (a_2)	Primary Camera (a_3)	Touch screen (a_4)	Screen Size (a_5)	RAM (a_6)	Screen Type (a_7)	Positive Tags (T^+)	Negative Tags (T^-)
i_1	rose gold	5mp	12mp	True	4.7	3GB	capacitive	stylish, lightweight	poor battery
i_2	silver black	5mp	16mp	True	5.1	3GB	corning gorilla	super cool	poor battery, gimmicky touchscreen

Table 2.2: Set of rules for the item i_1 in Table 2.1

Attributes	Tags	p
Primary Camera = 16mp, Touchscreen=true, Screen Size = 5.1"	super cool	0.25
Color=rose gold, Secondary Camera = 5mp, Touchscreen=true	stylish	0.3
Secondary Camera = 5mp, RAM = 3GB, Touchscreen=true	poor battery	0.2
Touchscreen=true	gimmicky touchscreen	0.25

The Table 2.2 can be represented in a bipartite graph model as shown in Fig: 2.1. The top nodes consists of positive tag T^+ and negative tags T^- whereas the bottom nodes consists of item attribute values.

2.2 Properties of Tag

To term the result set of tags meaningful, [1] defines three essential properties of tags. The properties are as follows:

1. **Relevance**: Relevance can be defined as how well a set of tags (T^*) where $T^* = \{t_1, t_2, ..., t_k\}$ and $T^* \in T$ describe the item i to the user. So, given item i and tag vocabulary T, the relevance of a tag $t_x \in T^*$ denotes how well t_x describes i. Mathe-

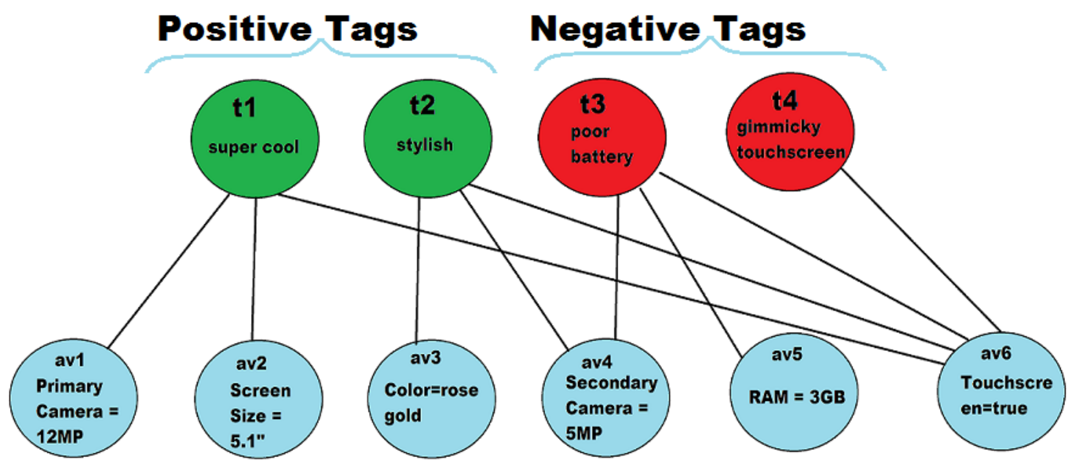

Figure 2.1: TagAdvisor Bipartite Graph Model

matically, it is the probability of obtaining t_x given i, i.e., $\textsc{rel}(t_x, i) = Pr(t_x|i)$.This score can be computed by employing a probabilistic classifier, that models the relationship between item attributes and tags. Thus, $\textsc{rel}(T^*) = \sum_{t_x \in T^*} \Big(\textsc{rel}(t_x, i)\Big)$.

2. **Coverage**: Coverage can be defined as for a given item i, tag vocabulary T, and a set of associated rules $\Re = \{\{a.v\} \to t_x\}$, the coverage of a tag $t_x \in T^*$ for i is the set of distinct item attribute values have been covered by it. We say t_x covers the attribute value $a.v_y$ if $a.v_y \in \{a.v\}$, i.e., $\textsc{cov}(t_x, i) = \{a.v\}$. For the example in Table 2.1, for item i_1 if we consider $T^* = \{$`stylish`, `gimmicky touchscreen`$\}$, the independently covered attributes will be Color=rose gold, Secondary Camera = 5mp, Touchscreen=true.

3. **Polarity**: Polarity of a set of tags can be defined as how good the sentiment of tags are distributed for an item i, and tag vocabulary T. Thus polarity of T^* is measured as the ratio of the number of the positive tags to the number of the negative tags, i.e., $\textsc{pol}(T^*) = \frac{|T^{*+}|}{|T^{*-}|}$. For the example in Table 2.1, for item i_1 if we consider T^* $= \{$`stylish`, `gimmicky touchscreen`$\}$, the $\textsc{pol}(T^*) = \frac{1}{1} = 1$. Thus implying the polarity is evenly distributed among positive and negative sentiments.

2.3 TagAdvisor Problem

Given a set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$ *for an item* $i = \{a.v_1, a.v_2, ...\}$ *and* $t_x \in T$, *non-negative integer budget* k, *relevance parameter* β *(*$0 \leq \beta \leq 1$*), and user factor* α *(*$0 \leq \alpha \leq 1$*), find a subset of* $T^* \subseteq T$ *such that:*

- $|T^*| \leq k$;
- $\text{POL}(T^*) = \frac{\alpha}{1-\alpha}$;
- $\text{REL}(T^*) \geq \beta \times \text{REL}_{max}^{T,k}$;
- $\text{COV}(T^*)$ *is maximized,*

where $\text{POL}(T^*)$ is the sentiment in opinion by tags in T^*, i.e., the number of positive tags ($k\alpha$) to the number of negative tags($k - k\alpha$), $\text{REL}(T^*)$ is the combined relevance of tags in T^*, $\text{REL}_{max}^{T,k}$ is the maximum combined relevance for k tags from T such that the k selected tags provide the same sentiment in opinion, and $\text{COV}(T^*)$ is the total number of item attributes covered by tags in T^*. The relevance parameter β ensures that $\text{REL}(T^*)$, the relevance score of tags in T^* matches the relevance score $\text{REL}_{max}^{T,k}$ as much possible. The user factor α denotes the ratio of positive and negative tags preferred by a user.

2.4 Types of Coverage

By definition of coverage mentioned in [1] , we consider reviews to be broadly categorized into two types. First let us consider a review *Even though with a short battery life, phone has decent call quality and nice volume* In this example, different attributes such as Battery, Call Quality, Speakers along with their respective sentiments are independently mentioned. Hence individual attributes are either provided with positive sentiments or negative sentiments independently. Now consider another review example, *Awesome Dolby speakers for a phone, but has poor build quality.* In this example, positive and negative sentiment are mentioned against a single attribute Speakers. Thus, different parts of the

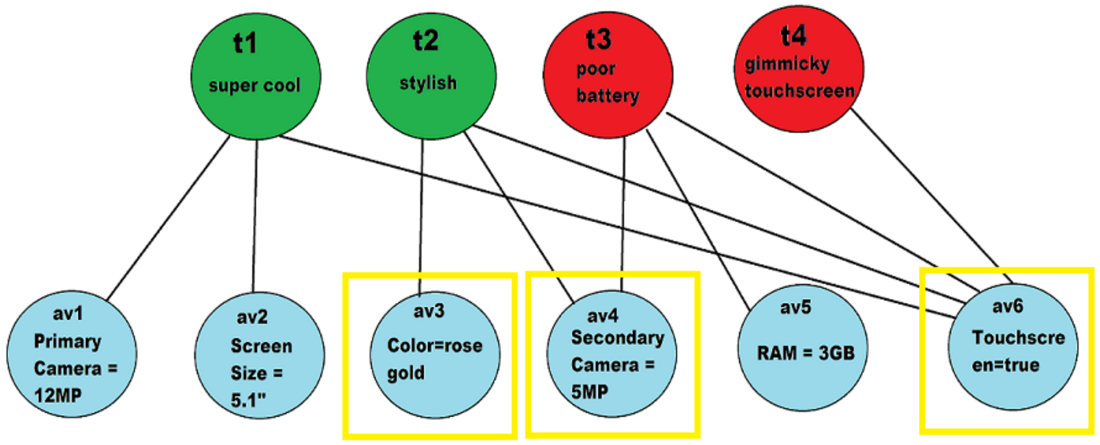

Figure 2.2: Independent Coverage TagAdvisor Bipartite Graph

review statement express opposite sentiments for a similar attribute. The TagAdvisor problem requires to maximize coverage, with two types of review writing styles we define two types of TagAdvisor Problems as mentioned below.

2.4.1 Independent Coverage TagAdvisor

Independent Coverage TagAdvisor Problem is a specialized version of TagAdvisor Problem where the coverage function to maximize $\text{cov}_{IC}(T^*)$ is defined as the size of set of item attribute values covered by the tags in T^*, independent of their sentiment. We say, T^* covers an attribute value $a.v_y$ for an attribute a_y of an item i if there exists a tag t_x covering $a.v_y$. Given a set of tags T^*, Independent-Coverage of T^* is defined as:

$$\text{cov}_{IC}(T^*) = |\bigcup_{t_x \in T^*} \text{cov}(t_x, i)| \tag{2.1}$$

For the example in Table 2.1, for item i_1 if we consider $T^* = \{$`stylish`, `gimmicky touchscreen`$\}$ IC-TA will cover 3 item attribute values, i.e., Color=rose gold, Secondary Camera = 5mp, Touchscreen=true. This can be verified by observing the Fig: 2.2.

2.4.2 Dependent Coverage TagAdvisor

Dependent Coverage TagAdvisor Problem is a specialized version of TagAdvisor Problem with a variation in the coverage function to maximize $\textsc{cov}_{DC}(T^*)$. In Dependent Coverage TagAdvisor, an attribute value for a given item i and attribute a_y, $a.v_y$ depends on the sentiment of its associated tags. An attribute value $a.v_y$ is covered, according to [2] if one of the following holds:

- $a.v_y$ is covered by both positive and negative tags, and atleast one of its tag from both the sentiments belong to T^*. Formally, $\exists t_x^+ \in T^*, \exists t_w^- \in T^{-^*}$ such that $a.v_y \in \textsc{cov}(t_x^+, i) \cap a.v_y \in \textsc{cov}(t_w^-, i)$

- $a.v_y$ is covered by positive tags only, and atleast one of its positive tags belongs to T^*. Formally, $\exists t_x^+ \in T^*, \forall t_w^- \in T^*$ such that $a.v_y \in \textsc{cov}(t_x^+, i) \;\cap\; a.v_y \notin \;\; \textsc{cov}(t_w^-, i)$

- $a.v_y$ is covered by negative tags only, and atleast one of its negative tags belongs to T^*. Formally, $\forall t_x^+ \in T^{+^*}, \exists t_w^- \in T^{-^*}$ such that $a.v_y \notin \textsc{cov}(t_x^+, i) \cap a.v_y \in \; \textsc{cov}(t_w^-, i)$

Given a set of tags T^*, Dependent-Coverage of T^* is formally defined as:

$$\begin{aligned}
\textsc{cov}_{DC}(T^*) &= |(\bigcup_{t_x^+ \in T^*} \textsc{cov}(t_x^+, i)) \bigcap (\bigcup_{t_w^- \in T^*} \textsc{cov}(t_w^-, i))| \\
&+ |\bigcup_{t_x^+ \in T^*} \textsc{cov}(t_x^+, i) \setminus \bigcup_{t_w^- \in T^*} \textsc{cov}(t_w^-, i)| \\
&+ |\bigcup_{t_w^- \in T^*} \textsc{cov}(t_w^-, i) \setminus \bigcup_{t_x^+ \in T^*} \textsc{cov}(t_x^+, i)|
\end{aligned} \tag{2.2}$$

For the example in Table 2.1, for item i_1 let us consider $T^* = \{$`stylish`, `gimmicky touchscreen`$\}$. DC-TA will cover 2 item attribute values i.e., Color=rose gold and Touchscreen=true. Note that even though Secondary Camera = 5mp is associated with the `stylish`, it is not covered because there exist a negative tag `Poor battery` which depends on this item attribute value but it is not in T^*. Fig: 2.3 can be referred for detailed understanding.

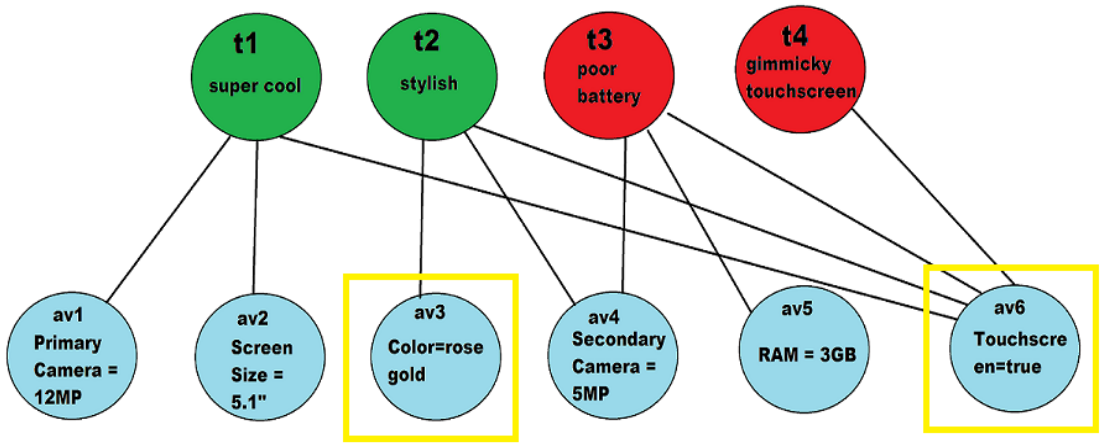

Figure 2.3: Dependent Coverage TagAdvisor Bipartite Graph

2.5 Algorithms

In this section we provide a brief overview of algorithms mentioned in [2] to solve the IC-TA and DC-TA problems. As mentioned in previous section Independent Coverage TagAdvisor Problem, can be defined as a instance of TagAdvisor Problem (TA) with same inputs and constraints except that the objective:

- $\text{cov}_{IC}(T^*)$ (given by Equation 2.1) is maximized

On the same lines Dependent Coverage TagAdvisor Problem, can be defined as instance of TagAdvisor Problem (TA) with same inputs and constraints except that the objective:

- $\text{cov}_{DC}(T^*)$ (given by Equation 2.2) is maximized

Both these problems are mapped as NP-Complete problems and we hence prefer Approximation Algorithms in place of exact algorithms for the implementation of AD-WIRE. (refer [2] for details).

2.5.1 Approximation Algorithm (A-IC-TA)

To verify NP-Completeness, we try to reduce an already existing NP-Complete problem to IC-TA and argue that a solution exists to it if and only if a solution exists to IC-TA problem. For IC-TA we consider Max-Coverage problem with group budget constraints

(MCG) [3] and reduce it to IC-TA problem. We argue that if and only if, a solution to IC-TA problem exists does a solution to MCG exists.In Max-Coverage problem with group budget constraints (MCG) problem our goal is to pick k sets from S, given $S = \{S_1, S_2, ...\}$ as a collection of S_i, where each S_i is a subset of a ground set $\mathcal{X}$ and S is partitioned into groups $G_1, G_2, ..., G_m$. The constraint of the problem is at most k_i be picked from each group G_i and cardinality of their union is maximum. AD-WIRE implements a greedy solution proposed by authors in [3] for IC-TA.

In IC-TA the set S is the set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$ which is partitioned into positive tags group and negative tags group. The greedy approach iteratively picks those relevant unpicked tags that cover the maximum number of uncovered item attribute values. It checks if the sentiment associated with the tag along with the relevance of the result set, if all conditions pass, the tag is chosen part of the result set.

Algorithm 1 is the pseudo code for our algorithm, denoted as **A-IC-TA**. The A-IC-TA algorithm iteratively picks tags from T that cover the maximum number of uncovered item attribute values such that the number of positive and negative tags are $k_1 = \lceil \alpha k \rceil$, $k_2 = k - k_1$ and $\text{REL}(T^*) \geq \beta \cdot \text{REL}_{max}^{T,k}$. $\text{REL}_{max}^{T,k}$ is the summation of the first k_1 positive tags and k_2 negative tags found in T^+ and T^- respectively where both are sorted by relevance. Example: In our example, if $\alpha = 0.5$, $\beta = 0.5$ and $k = 2$ for item i_1. The algorithm will first consider sort both the lists $T^+ = \{$`stylish`, `super cool`$\}$ and $T^- = \{$ `gimmicky touchscreen`,`poor battery`$\}$. It will then select `stylish` into its result set as `stylish` provides relevance of 0.3 which is higher than $\text{REL}(T^*) \geq \beta \cdot \text{REL}_{max}^{T,k} = 0.5{*}0.3 = 0.15$. Thus `stylish` is selected in the result set. Later, `super cool` is iteratively picked but since $\alpha = 0.5$ and $k = 2$, implies only one positive tag is allowed in the result set. Thus `super cool` is discarded. Similarly `gimmicky touchscreen` is allowed to be a part of the result set discarding `poor battery`. Hence, the result set $T^*= \{$`stylish`, `gimmicky touchscreen`$\}$.

Algorithm 1: IC-TA Algorithm (A-IC-TA)

```
Input  : Tag vocabulary $T$, set of rules $\Re = \{\{a.v\} \rightarrow t_x\}$, budget $k > 0$,
         relevance parameter $0 < \beta \leq 1$, user factor $0 < \alpha \leq 1$
Output: set of tags $T^* \subseteq T$ of size $k$
1  $k_1 = \lceil k \cdot \alpha \rceil$; $k_2 = k - k_1$;
2  $T^* = \emptyset$;
3  for $x = 1$ to $k$ do
4      for $t_y \in T \setminus T^*$ do
5          if $(t_y \in T^+ \ and \ |T^{+^*}| < k_1) \ or \ (t_y \in T^- \ and \ |T^{-^*}| < k_2)$ then
6              if $\textsc{rel}(T^* \cup t_y) \geq \beta \cdot \textsc{rel}^{T,x}_{max}$ then Compute($\textsc{cov}_{IC}(T^* \cup t_y)$);
7          end
8      end
9      $t_y = \underset{t_y \in T \setminus T^*}{\text{argmax}} \, \textsc{cov}_{IC}(T^* \cup t_y)$;
10     $T^* = T^* \cup t_y$;
11 end
12 return $T^*$
```

2.5.2 Approximation Algorithm (A-DC-TA)

In paper [1], the DC-TA problem is defined as a NP-Complete problem. To verify NP-Completeness, it reduces the MAX-SUM Facility Dispersion problem [4, 5, 6] a problem already defined to be NP-Complete, to DC-TA problem. It argues that if and only if, a solution to DC-TA problem exists does a solution to MAX-SUM Facility Dispersion exists. In MAX-SUM Facility Dispersion problem, we are given a set $V = \{v_1, v_2, ..., v_n\}$ where each v_i is a node of graph. Let their be n nodes, a non-negative distance $w(v_i, v_j)$ for each pair of nodes v_i, v_j, and an integer p which is smaller than n, the goal is to find a subset of p nodes $P = \{v_{i_1}, v_{i_2}, ..., v_{i_p}\}$ of V, such that sum of distances are maximized. ADWIRE implements the Approximation Algorithm as a solution for the DC-TA Problem.

In order to map DC-TA as MAX-SUM Facility Dispersion problem, we modify our DC-TA problem approach. Consider a graph $G_{DC-TA} = (V_T, E)$ as DC-TA model where V_T is tag represented as node in graph and $E \subseteq (V_T x V_T)$, relevance parameter β, and user factor α, the goal is to select $k_1 = \lceil \alpha k \rceil$ positive tags and $k_2 = k - k_1$ negative tags such that $\text{REL}(T^*) \geq \beta \cdot \text{REL}^{T,k}_{max}$ and $\vartheta_{DC}(T^*)$ is minimum.

AD-WIRE implements the greedy algorithm 2 which produces a solution with constant factor approximation of the optimal. In A-DC-TA Algorithm consider the user factor α which implies k_1 positive and k_2 negative tags. On each iteration of the first iteration cycle of the algorithm iteratively picks the cross-edges $(v_{t_x}, v_{t_y}), t_x \in T^+, t_y \in T^-$ under two constraints, first, the relevance score of result set including edge is greater than equal to $\beta \cdot \text{REL}^{T,k'}_{max}$ and secondly, the edge adds minimum weight to $\vartheta_{DC}(T^*)$ as compared to all other edges. Under all these scenarios algorithm adds those tags to the T^* until the number of selected positive or negative tags be k_1 or k_2 respectively. Once the quota of a particular sentiment is reached, the other iterative cycle begins. In the other iterative cycles either the number of positive or negative tags is not equal to k_1 or k_2 respectively. Accordingly either a positive or negative iterative cycle is executed and remaining tags from same sentiment are selected in result set. Let us assume k_2 negative tags are selected, thus the algorithm needs to find positive tags only for the result set. It finds the new tag $t_y \in T^+ \setminus T^*$ with the relevance score of atleast $\beta \cdot \text{REL}^{T,k'}_{max}$, which add minimum weight to $\vartheta_{DC}(T^*)$. Similarly, let us assume k_1 positive tags are selected, thus the algorithm needs to find negative tags only for the result set. It finds the new tag $t_y \in T^- \setminus T^*$ with the relevance score of atleast $\beta \cdot \text{REL}^{T,k'}_{max}$, which add minimum weight to $\vartheta_{DC}(T^*)$.

Algorithm 2: DC-TA Algorithm (A-DC-TA)

```
1  $k_1 = \lceil k\alpha \rceil$; $k_2 = k - k_1$; $T^* = \emptyset$;
2  while ($k_1 > 0$ and $k_2 > 0$) do
3      $k' = |T^*| + 2$ ;
4      for $e = (t_x, t_y), (t_x \in T^+ \setminus T^*, t_y \in T^- \setminus T^*)$ do
5          if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,k'}$ then
               Compute($\vartheta_{DC}(T^* \cup \{t_x, t_y\})$);
6      $T^* = T^* \cup \operatorname{argmin}_{t_x \in T^+ \setminus T^*, t_y \in T^- \setminus T^*} \vartheta_{DC}(T^* \cup \{t_x, t_y\})$ ;
7      $k_1 = k_1 - 1$; $k_2 = k_2 - 1$;
8  while ($|T^*| < k$) do
9      if ($k_1 > 0$) then
10         $k' = |T^*| + 1$ ;
11         for $e = (t_x, t_y), (t_x \in {T^*}^+, t_y \in T^+ \setminus T^*)$ do
12             if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,k'}$ then
                   Compute($\vartheta_{DC}(T^* \cup \{t_x, t_y\})$);
13         $T^* = T^* \cup \operatorname{argmin}_{t_x \in {T^*}^+, t_y \in T^+ \setminus T^*} \vartheta_{DC}(T^* \cup \{t_x, t_y\})$ ;
14     if ($k_2 > 0$) then
15         $k' = |T^*| + 1$;
16         for $e = (t_x, t_y), (t_x \in {T^*}^-, t_y \in T^- \setminus T^*)$ do
17             if $\text{REL}(T^* \cup \{t_x, t_y\}) \geq \beta \cdot \text{REL}_{max}^{T,k'}$ then
                   Compute($\vartheta_{DC}(T^* \cup \{t_x, t_y\})$);
18         $T^* = T^* \cup \operatorname{argmin}_{t_x \in {T^*}^-, t_y \in T^- \setminus T^*} \vartheta_{DC}(T^* \cup \{t_x, t_y\})$ ;
19 return $T^*$
```

CHAPTER 3

ADWIRE ARCHITECTURE

In previous chapters we have discussed about the TagAdvisor model, various coverage problems and how TagAdvisor can be used to solve user problem of review writing. ADWIRE, is a practical implementation of TagAdvisor and is built to review mobile products. When a user uses ADWIRE to review a mobile product he should expect to find top-k meaningful tags describing the mobile item user intends to review. In this chapter we will discuss in detail about the building and functionality of each block of modules that together form ADWIRE.

3.1 Data Store

The Data Store is an important module as it contains both Table 2.1 and Table 2.2. The main functionality of data store is to provide instant access of data to the incoming request by other modules as seen in Fig 3.1. To obtain a smooth system, it is critical the response time of the data store is minimal. As the number of products might be vary, it is required to index items efficiently without much overhead for the user to maintain them. Also, the data store is expected to get simple read queries with where clauses. Considering these aforementioned criteria having MongoDB based as the data store proves beneficial. MongoDB offers great indexing, easy store, easy retrieve, fast access to data and minimal overhead. The data is stored in JSON format and is easily retrieved.

3.2 Rule Generator

This module is responsible to find the complex dependencies that exist between item attributes and tags. Table 2.2 shows extracted rules for the item i_1 in Table 2.1. For

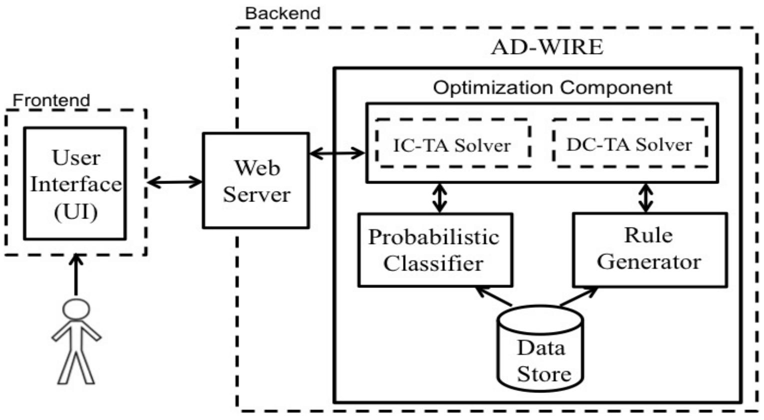

Figure 3.1: AD-WIRE Architecture.

example, the first row of the Table 2.2 indicates that if a mobile phone has 5mp primary camera, screen size of 5.1" and is touchscreen, then with probability of 0.3 it is responsible for the receiving the tag `super cool`. ADWIRE aims to discover small distinguished set of rules so as to classify if a set of attribute values provide a tag as class variable. For these reasons our module uses the rule based classifier [7] to find the rules in our data set. Given a user and an item Rule Generator provides the dependencies between the tags and item attributes. The item attributes can be called rule antecedent while the tags as rule consequent. The item attributes are connected logically via AND operator. These rules are then send to Optimization Component.

3.3 Probabilistic Classifier

The rules obtained from the Rule Generator may contain similar tags for different set of item attribute values. We need these set of rules to have a minimum support and minimum confidence throughout the dataset. Thus, we apply probabilistic classifiers which are responsible to find the relevance score of a tag to an item. Given item i and tag vocabulary T, the relevance of a tag $t_x \in T^*$ denotes how well t_x describes i which can be computed using existing probabilistic classifiers. This module uses the rule based classifier [7] to find the relevance score.

3.4 Optimization Component

Optimization Component is responsible for solving the ADWIRE optimization problem. As shown in Fig 3.1, this module lies in the core of the system and is very important for functioning of the module. TagAdvisor Problem as explained in previous chapters is implemented in this module. The Optimization Component requires user input parameters such as item i, relevance parameter β , user factor α and number of required tags k. On providing these inputs the component provides the user with two sets of tags. One tag set for independent coverage style of writing whereas the other for a dependent style of writing. Thus, the module consists of both the IC-TA Solver and DC-TA Solver to obtain their individual result sets. The component also integrates interfacing the web server, the probabilistic classifier and rule generator. We refer readers to[1],[7]and [2] for technical details of the problems.

3.5 User Interface and Web Server

The main functionality of the User Interface (UI) is to provide an access to the system abstracting the complexity of a system. It should allow user to provide user input parameters such as item i, relevance parameter β , user factor α and number of required tags k. On provision of these input parameters, it should display the user with a list of top-k meaningful tags for both styles of review writing. The UI should be able to explain the user about the attributes he is reviewing when he selects any tag. Most importantly the UI should be smooth and provide easy access. The built User Interface interacts with the ADWIRE system via a Web Server on which it hosts. The basic functionality of the web server is to provide access to the UI, interact with the Optimization Component. As only a single system module exists on the server with simple UI, the selected server should be simple enough to provide static files over the web browser without much overhead of maintaining the server up.

CHAPTER 4

SYSTEM DEMONSTRATION

Using the basic architecture as described in the previous chapter we develop AD-WIRE. In this chapter we will discuss the actual software tools, utilities and configurations used for exact implementation of each of these modules.

4.1 System Implementation

AD-WIRE is developed on a Intel Quad Core 2.5 Ghz machine running Ubuntu with 16GB RAM. In order to establish the data store, we first employ a state-of-art text mining tool to obtain keywords and their sentiments from a text corpus of reviews for each item. For example, a review statement *"Even though with a short battery life, phone has decent call quality and nice volume"* when processed by the text mining tool should provide positive (`decent call quality`,`nice volume`) and a negative tag (`short battery life`). We use IBM Watson‘s AlchemyAPI from "www.alchemyapi.com" as the text mining tool, it provides the required data for each item reviews. This data is stored in our data store which is made of MongoDB. There are different collections in place for each data tables and reference tables. Now in order to obtain complex dependencies between item attributes and keywords, we use Rule generator as mentioned in Fig 3.1. These rules are then utilized by the probabilistic classifier to obtain relevance of the tags. In AD-WIRE we utilize Weka‘s JRIPPER rule based classifier [8], [9], as our Rule Generator and Probabilistic Classifier, different other available models like Decision Trees, SVM Classifiers, etc., could also be used. We employ JRIPPER to extract the set of rules that shows the dependency between item attributes and tags. The optimization component is built on Python 2.7.9 as it offers smooth integration with MongoDB, weka and the web server. BottlePy 0.12 is used as a

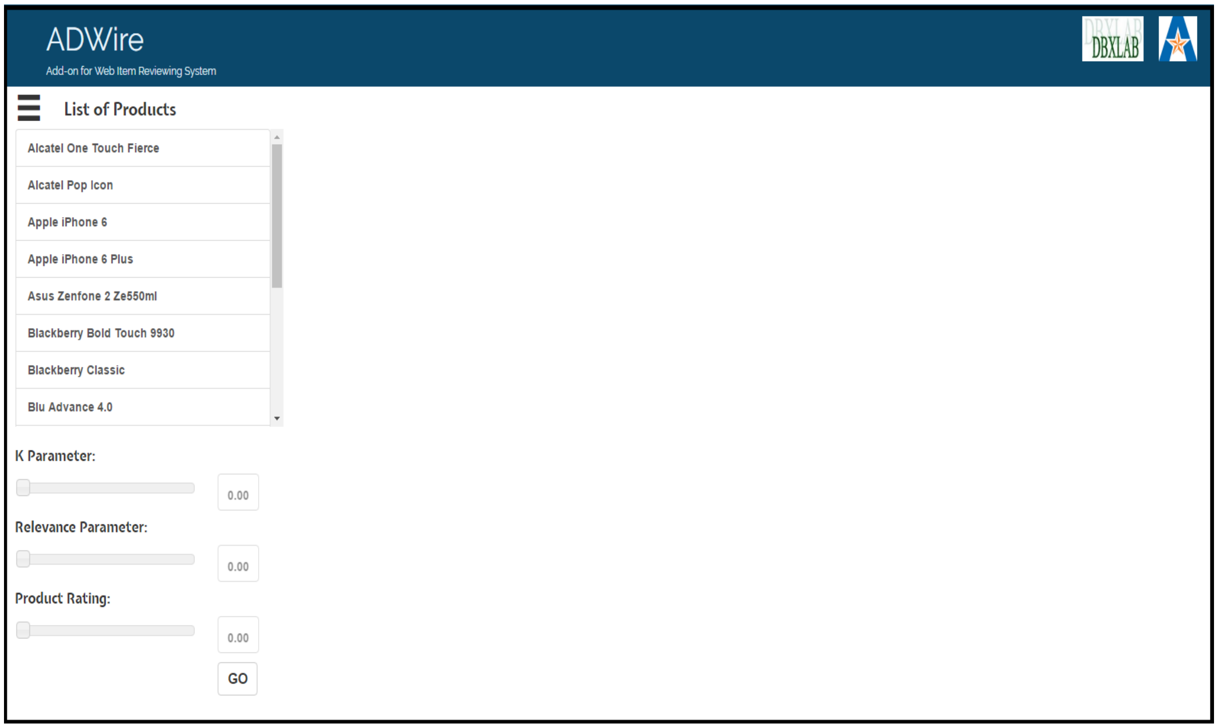

Figure 4.1: AD-WIRE User Interface 1.

web server to provide web access to the system via the User Interface. The User Interface is developed in HTML, JavaScript, Jquery to enable seamless web browsing and great user experience.

4.2 User Interface

1. **Input Interface:** In [2], the TagAdvisor Problem uses k, α and β as input parameters for item selected in order to provide the top-k meaningful tags. In AD-WIRE the user is expected to provide these inputs via The Input Parameter area as shown in Figure 4.1. On accessing the site, AD-WIRE displays the list of products available for reviewing. After selecting the review item, user selects the following items. Firstly, k parameter which is a non-negative integer value, it defines the number of top-k tags to be displayed. A *Relevance Parameter* which implies, how well the result set of tags describes the item. Finally, user selects the Product Rating which quantifies

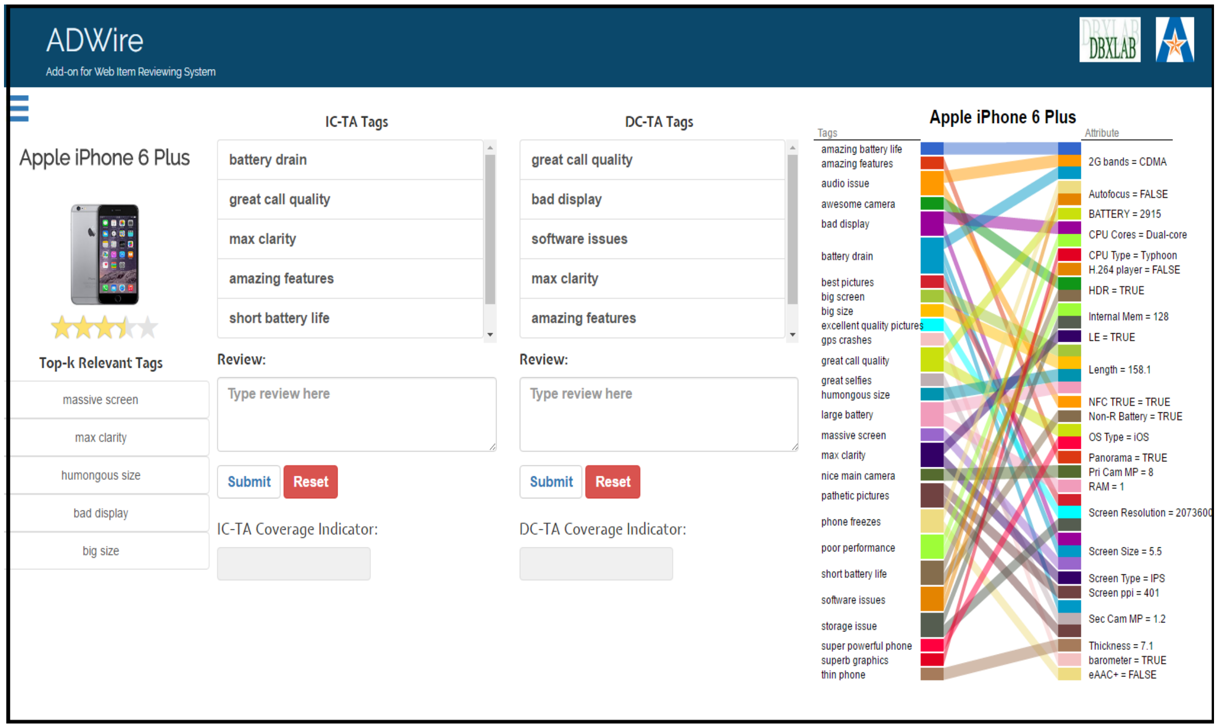

Figure 4.2: AD-WIRE User Interface 2.

user satisfaction towards the product. Together these parameters are submitted as a form to fetch the result tags.

2. **Reviewing:** After the user selects the required inputs and hits "GO", the browser displays a new page as shown in Figure 4.2. This interface is where the user writes the review. The interface can be divided into columns.

   (a) Product Information: This column displays the item the user selected, the image of the product followed by the product rating user provided in previous page. We also display top-k relevant tags to signify the need for a top-k meaningful tags and how top-k relevant tags fail to connect with the users sentiment and critical experience.

   (b) Independent Review: This column displays top-k tags obtained from IC-TA Solver. The user can select any tag he connects with and the corresponding tag will appear in the review text. Below the review writing section the user can find IC-TA Coverage Indicator which will display the user a list of attribute values covered by tags the user selected in review. Thus, with the help of a set of meaningful tags and an overview of item attributes being covered, the user can easily articulate the review.

(c) Dependent Review: This column displays top-k tags obtained from DC-TA Solver. The functionality and display of this column is exactly the same as Column 2.

(d) Bipartite Graph: This column consists of a bipartite graph for display as shown in Figure 4.2. The graph is displayed using a data visualization javascript library. The left side of the graph consists of tags related to the product. These tags are associated with item attribute values which are displayed on the right side of the graph.

CHAPTER 5

EXPERIMENTAL EVALUATION

This chapter presents the results of extensive experimental analysis performed by providing various queries to AD-WIRE. The experimental results reinforce our promise that AD-WIRE provide excellent quality of tags for any given circumstances. We proceed to evaluate the performance of both A-IC-TA and A-DC-TA in terms of quality by varying different user inputs.

5.1 Experimental Setup

All experiments were conducted on an Intel Quad Core 2.5 Ghz machine running Ubuntu with 16GB RAM. The points displayed in graph are numbers obtained as the average over all products. We conduct a comprehensive set of experiments using real data crawled from the web to evaluate efficiency and quality of our proposed algorithms. We crawled Amazon.com and GSMAreana.com for building a mobile dataset.

5.2 Coverage Analysis

In this section we evaluate the quality of results returned by the implemented A-IC-TA Solver and A-DC-TA Solver of AD-WIRE. We measure the proportion of tags covered by the result set of k tags in T^* for both the A-IC-TA and A-DC-TA algorithms implemented in the respective solvers. We conduct our experiments with different set of constraint conditions, i.e., user factor (α) and k.

1. **K Parameter Vs Coverage** In this evaluation, we set $\alpha = 0.5$, $\beta = 1.0$, and vary k from 2 to 10. So ideally, when we increase the number of K tags to be returned the proportion of attribute space covered should also increase. That is exactly the

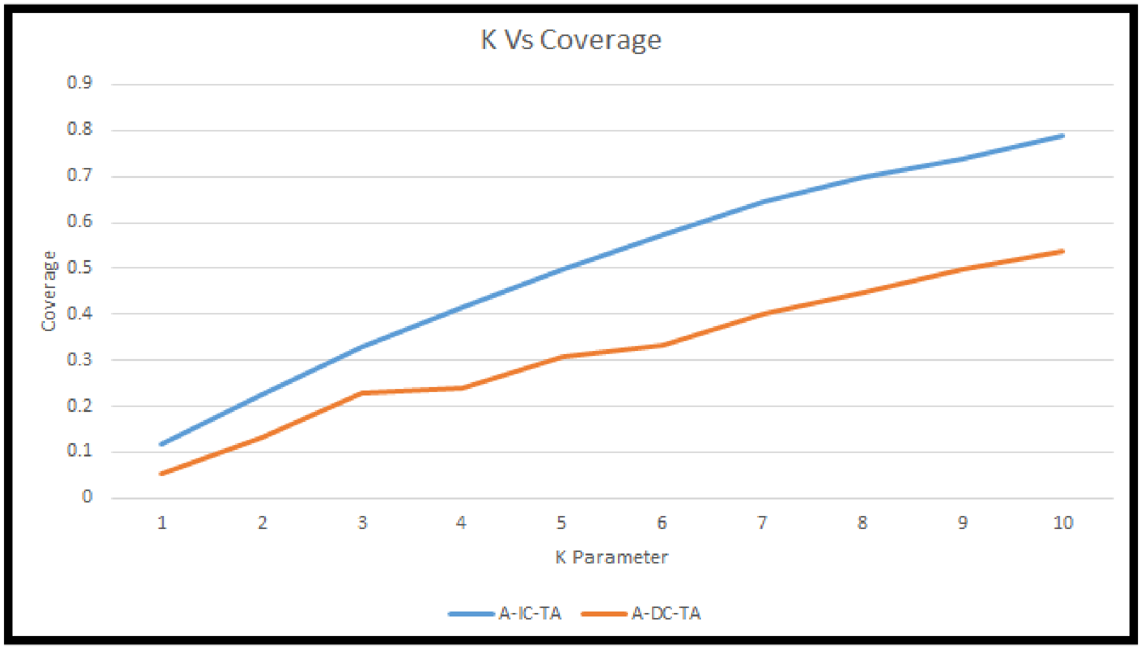

Figure 5.1: Quality of A-IC-TA and A-DC-TA algorithms by varying k, $\alpha = 0.5$, $\beta = 0.5$

behavior as shown in Figure 5.1. As one can also find from observing the plots is A-IC-TA provides a better coverage compared to A-DC-TA, which makes sense as the coverage function of A-IC-TA as mentioned in 2.1 is unbound whereas the coverage function of A-DC-TA as mentioned in 2.2 is bound with sentiment associated in tags, thus reducing the proportion of covered attributes.

2. **User Factor Vs Coverage** In the second evaluation, we set $k = 10$, $\beta = 1.0$, and vary α from 0.1 to 1.0. When we change α, we are changing the proportion of positive and negative tags. Thus, when we increase α from 0.1 to 1.0, we are increasing the positive tags present in the result set, thereby decreasing the negative tags. The behavior of coverage under such circumstances is as shown in Figure 5.2. As one can also find from observing the plots is A-IC-TA again provides a better coverage compared to A-DC-TA, which makes sense as the coverage function of A-IC-TA as mentioned in 2.1 is unbound whereas the coverage function of A-DC-TA as mentioned in 2.2 is bound with sentiment associated in tags, thus reducing the proportion of

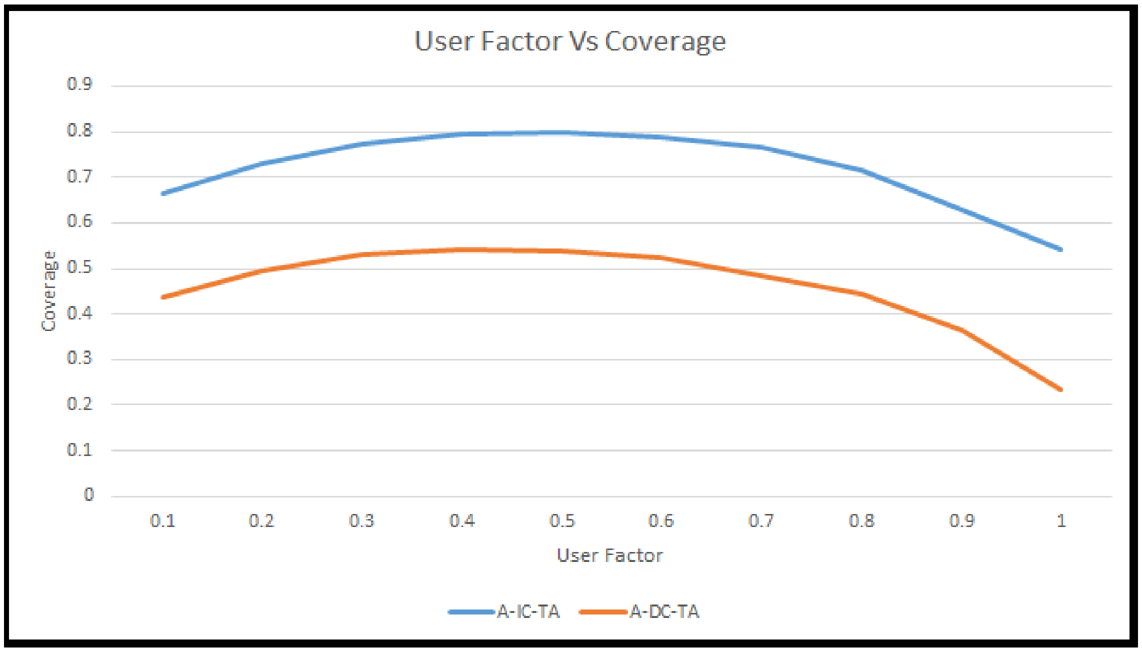

Figure 5.2: Quality of A-IC-TA and A-DC-TA algorithms by varying *alpha*, $k = 10$, $\beta = 0.5$

covered attributes. But both the A-IC-TA and A-DC-TA provide less coverage at the extreme ends compared to the middle. This implies that when user factor is at extremes the attribute space available to incoporate k positive or k negative tags decreases.

In both the experimental evaluations our results match with the exact algorithms mentioned in [2]. This guarantees that AD-WIRE provides a set of top-k meaningful tags under various constraints provided by user, thus being able to help the user review item easily.

REFERENCES

[1] A. Nazi, M. Das, and G. Das, “The tagadvisor: Luring the lurkers to review web items,” in *Proc. of the ACM SIGMOD*, 2015, pp. 531–543.

[2] A. Nazi, M. Das, and G. Das, “Web item reviewing made easy by leveraging available user feedback,” in *arXiv preprint arXiv:1602.06454*, 2015.

[3] C. Chekuri and A. Kumar, “Maximum coverage problem with group budget constraints and applications,” in *Proc. of Approx*, 2004.

[4] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, “Heuristic and special case algorithms for dispersion problems,” *Operations Research*, vol. 42, no. 2, pp. 299–310, 1994.

[5] E. Erkut, “The discrete p-dispersion problem,” *European Journal of Operational Research*, vol. 46, no. 1, pp. 48 – 60, 1990.

[6] P. Hansen and I. D. Moon, “Dispersion facilities on a network,” *Presentation at the TIMS/ORSA Joint National Meeting*, vol. 46, 1988.

[7] B. Liu, W. Hsu, and Y. Ma, “Integrating classification and association rule mining,” in *KDD*, 1998, pp. 80–86.

[8] W. W. Cohen, “Fast effective rule induction,” in *12th ICML*, 1995.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: An update,” *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

BIOGRAPHICAL STATEMENT