AD-WIRE: Add-on for Web Item Reviewing System

Rajeshkumar Kannapalli[†], Azade Nazi[†], Mahashweta Das[‡*]; Gautam Das[†]
[†]University of Texas at Arlington; [‡]Hewlett Packard Labs

[†]{rajeshkumar.kannapalli@mavs, azade.nazi@mavs, gdas@cse}.uta.edu, [‡]mahashweta.das@hpe.com

ABSTRACT

Over the past few decades as purchasing options moved online, the widespread use and popularity of online review sites has simultaneously increased. In spite of the fact that a huge extent of buying choices today are driven by numeric scores (e.g., rating a product), detailed reviews play an important role for activities like purchasing an expensive DSLR camera. Since writing a detailed review for an item is usually time-consuming, the number of reviews available in the Web is far from many. In this paper, we build a system AD-WIRE that given a user and an item, our system identifies the top-k meaningful tags to help her review the item easily. AD-WIRE allows a user to compose her review by quickly selecting from among the set of returned tags or writes her own review. AD-WIRE also visualizes the dependency of the tags to different aspects of an item so a user can make an informed decision quickly. The system can be used for different type of the products. The current demonstration is built to explore review writing process for the mobile phones.

1. INTRODUCTION

Users post their experience about different products and services in online websites like Yelp, TripAdvisor, Amazon, etc., in form of numeric scores or star rating, and reviews. Out of the wide variety of user feedback options, numeric scores are used in majority of areas (e.g., 5 star ratings for restaurant, average rating of a mobile phone, etc.). But these ratings fall short in providing information such as those related to experience, (e.g., keyboard feel of the laptop, build quality of mobile phone, etc.) which can be provided by detailed reviews. The importance of detailed reviews is further enhanced in user's decision making if the product is expensive (e.g., buying high end mobile phone, expensive DSLR camera). According to Local Consumer Review Survey 2015, 92% of consumers regularly or occasionally read online reviews. Also, according to the same survey, 80% will trust reviews as much as personal recommendations. Thus, a significant proportion of consumers rely upon reviews as authentic voice of users in order to make purchasing decisions. This has motivated business of various kinds to possess an in-house arsenal of precious user feedback for marketing and development purposes. However, as mentioned by Pew Internet in 2012, only 32% users have ever contributed by providing a review. The number of useful reviews available is far from many because users tend to ignore providing a review as it is time-consuming and unrewarding.

[*]Work done as graduate student at University of Texas, Arlington

Proceedings of the VLDB Endowment, Vol. 9, No. 13
Copyright 2016 VLDB Endowment 2150-8097/16/09.

In [1] we introduced the general TagAdvisor problem and we proposed practical solution to increase detailed online reviews knowing that users may tend to avoid composing reviews. A good review should not only meet necessary requisites like conciseness, comprehensiveness, objectiveness,etc. but should also offer adequate incentive in the form of simple usability, easy applicability, etc. In this demonstration, we build a system AD-WIRE that given a user and an item, our system identifies the top-k meaningful phrases/tags to help her review the item easily. Doing so, user is now provided with option of composing her review by quickly selecting from among the set of tags returned by AD-WIRE. Hence, user is levitated from the cumbersome task of composing a good review because the user is provided with phrases she can immediately connect with the product to write a review. Our statement is supported by the use case study in [2], that 83% of the users would submit online reviews more often if they are provided a system such as AD-WIRE for reviewing items. As one of our first step, AD-WIRE system employs state-of-art text mining techniques to extract meaningful phrases or tags with sentiment attached to it from user feedback in the form of text. For example, a review statement *"The phone has a great user interface which overshadows it's poor camera."* has a positive tag (`great user interface`) and a negative tag (`poor camera`). In order to enable a user to satisfactorily review an item AD-WIRE considers three essential properties —*relevance* (i.e., how well the result set of tags describes an item to a user), *coverage* (i.e., how well the result set of tags covers the diverse aspects of an item), and *polarity* (i.e., how well sentiment is attached to the result set of tags). (Refer [2] for more details)

A user can express her broad opinion about the different aspects of an item which, in turn, can either be positive or negative. Again, a user can express both positive and negative opinion for the same attribute (or, set of attributes) of the item. Let us consider the same review *"The phone has a great user interface which overshadows it's poor camera"*.

Table 1: Example: mobile phone review data $< I, T >$

Items (I)							Tags (T)	
Item (i)	OS Type (a_1)	Secondary Camera (a_2)	Screen Resolution (a_3)	Auto focus (a_4)	Screen PPI (a_5)	HDR (a_6)	Positive Tags (T^+)	Negative Tags (T^-)
i_1	Blackberry	2mp	518400	True	294	True	`great user interface`, `good battery life`	`poor camera`, `unresponsive`
i_2	Android	2mp	629500	True	572	True	great selfies	`battery drain`

Table 2: Set of rules for the item i_1 in Table 1

Attributes	Tags	p
OS Type=Blackberry	`great user interface`	0.5
Auto focus=True, Secondary Camera=2mp, HDR=true	`great selfies`	0.3
Screen Resolution=518400	`good battery life`	0.1
HDR=true	`battery drain`	0.1
Secondary Camera=2mp, Screen Resolution=518400, Screen PPI=294	`poor camera`	0.3
OS Type=Blackberry	`unresponsive`	0.4

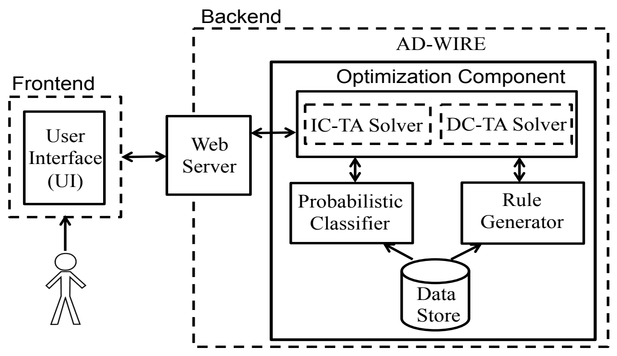


Figure 1: AD-WIRE Architecture.

In this example, different item attributes such as User Experience and Camera 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, *"The phone has a poor camera, but manages to provide great selfies"*. In this example, positive and negative sentiment are mentioned against a single item attribute Camera. Here different parts of the review statement express opposite sentiments for a similar attribute. In [1] we proposed two coverage functions 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, and dependent on the sentiment in the DC-TA. AD-WIRE demonstrates the result of both IC-TA and DC-TA problems and visualizes the dependency of the tags to different aspects of an item so a user can make an informed decision quickly.

We next explain AD-WIRE architecture and we provide examples to demonstrate how our system work.

2. AD-WIRE ARCHITECTURE

Figure 1 shows an end to end architecture of the AD-WIRE. We next describe the role of each module.

User Interface and Web Server: The key task of the Web Server is to provide the user access to the User Interface (UI). Thus, user can interact and provide various inputs to the AD-WIRE system such as, selecting an item from a list of products, rate the item, articulate review, and submit review. On providing inputs, the user is able to retrieve output in a fraction of a time. The UI visualizes the dependency of the suggested tags to the item attribute and it informs the user with aspect of the item covered by her selected tags. We discuss the detail of the UI in Section 3.2.

Data Store: Table 1 shows an example of the mobile data in the data store. We model data D in an online review site as $< I, T >$, which represents a set of items and tag vocabulary respectively. Each tagging action by a user is represented as $< i, \mathtt{T} >$ where $i \in I$, and $\mathtt{T} \in T$. Every item $i \in I$ is associated with a well-defined schema $I_A = \{a_1, a_2, ..., a_m\}$ and each item i is 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. Since each tag is a user feedback for an item, it describes the item positively or negatively. Therefore, we partition T into T^+ and T^-.

Rule Generator: This module is responsible to find the complex dependencies that exist between item attributes and tags. Table 2 shows extracted rules for the item i_1 in Table 1. For example, the first row of the Table 2 indicates that if a mobile phone has BlackBerry OS, then with probability of 0.5 it is responsible for the receiving the tag **`great user interface`**. Note that, our work is not influenced by or biased towards any brand. This module uses the rule based classifier [3] to find the rules in our dataset. Rule Generator provides the dependencies between the tags and item attributes. It then sends the rules to Optimization Component.

Probabilistic Classifier: This module is 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. Mathematically, it is measured as the probability of obtaining t_x given i, i.e., $\text{REL}(t_x, i) = Pr(t_x|i)$ which can be computed using existing probabilistic classifiers. This module uses the rule based classifier [3] to find the relevance score.

Optimization Component: This module is responsible for solving the AD-WIRE optimization problem which is centered around three properties - relevance, coverage, and polarity. In [1, 2] we formalize the various ways of reviewing an item by proposing two coverage functions 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. While we refer readers to [1, 2] for technical details of the problems, here we provide a brief summary of each problem in order to present a more comprehensive picture of the system implementation.

IC-TA Solver: In this module, coverage is defined as the total number of item attribute values covered by the tags in T^*, independent of their sentiment. Specifically, an attribute value $a.v_y$ for an attribute a_y of an item i is covered by T^* if there exists a tag t_x covering $a.v_y$, independent of its sentiment. Given a set of tags T^*, INDEPENDENT-COVERAGE of T^* is defined as:

$$\text{COV}_{IC}(T^*) = |\bigcup_{t_x \in T^*} \text{COV}(t_x, i)| \tag{1}$$

In Table 1, for item i_1 if we consider $T^* = \{$`great selfies`, `poor camera`$\}$ IC-TA will cover 5 item attribute values, i.e., Secondary Camera = 2mp, Screen Resolution = 518400, HDR = True, Auto focus = True and Screen PPI = 294.

IC-TA Solver inputs an item and a list of rules pertaining to the item. This module aims to maximize the coverage of T^* given in Equation 1 under constraints provided by user in terms of user rating and the relevance parameter. The IC-TA optimization problem is solved using A-IC-TA [1]. Intuitively, the algorithm iteratively picks relevant unpicked tags from T that cover the maximum number of uncovered item attribute values such that the ratio of the number of the positive tags to the number of the negative tags satisfies the user rating.

DC-TA Solver: In this module, coverage of $a.v_y$ depends on the sentiment of its associated tags. Given a set of tags T^*, DEPENDENT-COVERAGE of T^* is formally defined as:

$$\begin{aligned} \text{COV}_{DC}(T^*) &= |(\bigcup_{t_x^+ \in T^*} \text{COV}(t_x^+, i)) \bigcap (\bigcup_{t_w^- \in T^*} \text{COV}(t_w^-, i))| \\ &+ |\bigcup_{t_x^+ \in T^*} \text{COV}(t_x^+, i) \setminus \bigcup_{t_w^- \in T^*} \text{COV}(t_w^-, i)| \\ &+ |\bigcup_{t_w^- \in T^*} \text{COV}(t_w^-, i) \setminus \bigcup_{t_x^+ \in T^*} \text{COV}(t_x^+, i)| \end{aligned} \tag{2}$$

An attribute value $a.v_y$ for an attribute a_y of an item i is covered if one of the following holds:

- $a.v_y$ is covered by both positive and negative tags, and atleast one of its positive and atleast one of its negative tags belong to T^*. Formally, $\exists t_x^+ \in T^*, \exists t_w^- \in T^{-^*}$ such that $a.v_y \in \text{COV}(t_x^+, i) \cap a.v_y \in \text{COV}(t_w^-, i)$
- $a.v_y$ is covered only by positive tags and not negative tags, and atleast one of its positive tags belongs to T^*. Formally, $\exists t_x^+ \in T^*, \forall t_w^- \in T^*$ such that $a.v_y \in \text{COV}(t_x^+, i) \cap a.v_y \notin \text{COV}(t_w^-, i)$
- $a.v_y$ is covered only by negative tags and not positive tags, and atleast one of its negative tags belongs to T^*. Formally, $\forall t_x^+ \in T^{+^*}, \exists t_w^- \in T^{-^*}$ such that $a.v_y \notin \text{COV}(t_x^+, i) \cap a.v_y \in \text{COV}(t_w^-, i)$

For the example in Table 1, for item i_1 let us consider $T^* = \{$`great selfies`, `poor camera`$\}$. DC-TA will cover 3 item attribute values i.e., Secondary Camera = 2mp, Auto focus = True and Screen PPI = 294. Note that even though Screen Resolution = 518400 is associated with the `poor camera`, it is not covered because there exist a positive tag `good battery life` which depends on this item attribute value but it is not in T^*, similarly Auto focus = True is also not covered.

Given an item and a set of rules pertaining to the item. This module solves the DC-TA Problem by maximizing the coverage of T^* given in Equation 2 under constraints provided by user in terms of user rating and the relevance parameter. The DC-TA optimization problem is solved using A-DC-TA [2]. At each step *DC-TA Solver* computes the coverage score of adding two tags to T^* using the Equation 2. It then picks tags that have maximum relevance score such that the ratio of the number of the positive tags to the number of the negative tags satisfies the user rating.

3. SYSTEM DEMONSTRATION

In this section, we first describe details of our system implementation. Next, we illustrate the user interface and how audience can interact with our system.

3.1 System Implementation:

AD-WIRE has been implemented on a Intel Quad Core 2.5 Ghz machine running Ubuntu with 16GB RAM. It uses BottlePy 0.12 to provide web access to the system. The front end is developed in HTML, JavaScript, Jquery to enable seamless web browsing. The back end of the system is built in Python 2.7.9. AD-WIRE uses MongoDb, for fast and easy access to data. The data is crawled from different websites. The mobile specifications are obtained from GSMAreana.com and the reviews are collected from amazon.com. Total of near 3000 mobile phone specifications were collected with 160k raw reviews. We process the reviews to identify a set of positive and negative tags using the keyword extraction toolkit AlchemyAPI[1]. We employ RIPPER implemented in [4] to extract the set of rules that shows the dependency between item attributes and tags.

3.2 User Interface

Input Interface: This interface, as shown in Figure 2, consists of fields required to be filled by the user. As our current demonstration is customized for mobile phone products, AD-WIRE displays a list of phones available for review. After selecting an item, user provides following parameters. *k Parameter* is a non-negative integer value which 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, *Product Rating* which quantifies user satisfaction towards the product. These parameters are submitted as a form to fetch the tags. For example, here the user selects "Blackberry Classic" as the mobile product and requests 2 tags with 0.5 relevance and product rating. This implies, the result set must contain one positive tag and one negative tag with relevance of atleast 0.5 or higher. Based upon these inputs, AD-WIRE provides relevant tags for user to articulate and review mobile phones easily.

Product Information: When user submits the form in Input Interface, AD-WIRE displays product details along with the user's product ratings. It also shows $top - k$ result of a tags by only considering the relevance and polarity (shown in Figure 3). Based on user inputs, one positive tag `great user interface` and one negative tag `unresponsive` are displayed. These tags have the highest relevance in their sentiment groups (Table 2). One can notice that both of

[1]www.alchemyapi.com

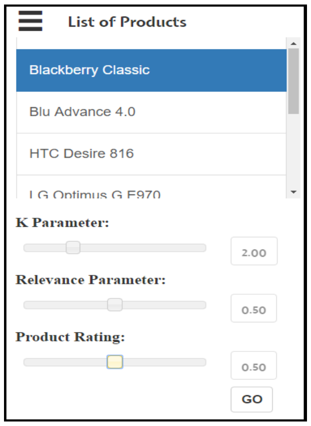


Figure 2: AD-WIRE UI: Input Interface.

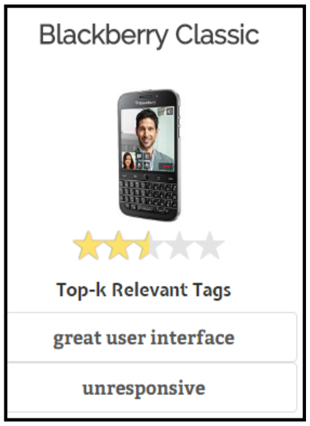
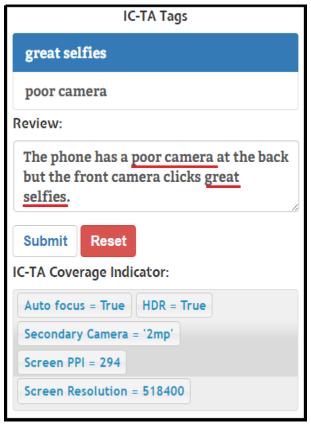
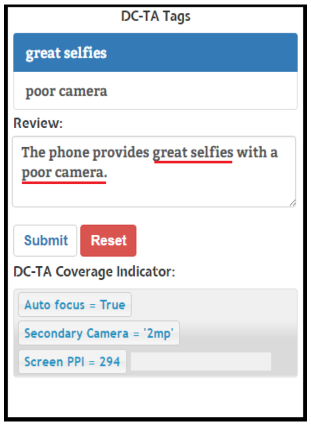


Figure 3: AD-WIRE UI: Product Information.



The phone has a poor camera at the back but the front camera clicks great selfies.

Figure 4: AD-WIRE UI: IC-TA Review.



The phone provides great selfies with a poor camera.

Figure 5: AD-WIRE UI: DC-TA Review.

these tags are associated with a single attribute value OS Type = Blackberry. Thus even with maximum relevance, result set fails to provide user with a broad feature space to articulate a detailed review. AD-WIRE considers three properties mentioned in [2] to address this problem.

IC-TA Review: User is provided with two result set of tags and can select either IC-TA Tags or DC-TA Tags based upon his style of review writing. The IC-TA tags as shown in Figure 4, are obtained by IC-TA Solver of the Optimization Component explained in Section 2. These tags collectively provide a relevance score higher than the threshold provided by user, i.e., *Relevance Parameter*. On selection of any tag from the list, the tag get displayed in the text review area. The attributes associated with each selected IC-TA tags are displayed in IC-TA Coverage Indicator. It allows user to understand different item attributes related to her review. In our example, IC-TA tags consists of a positive tag `great selfies` and negative tag `poor camera`. User selects both tags to articulate and write a detailed review easily. On selection, corresponding attributes Auto focus = True, Secondary Camera = 2mp, HDR=True,Screen PPI = 294 and Screen Resolution = 518400 get displayed in the IC-TA Coverage Indicator in accordance to Equation 1.

DC-TA Review: User can select DC-TA tab to utilize DC-TA tags. These tags are obtained by DC-TA Solver of the Optimization Component explained in Section 2. The GUI of DC-TA as shown in Figure 5, is similar to IC-TA. On selection of any tag from the list, the tag get displayed in the text review area. The attributes associated with each selected DC-TA tags are displayed in DC-TA Coverage Indicator. In our example, DC-TA tags consists of one positive tag `great selfies` and one negative tag `poor camera`., which get displayed in the text review area. User selects both the tags to articulate and write a detailed review easily. On selection, Secondary Camera = 2mp, HDR=True and Screen PPI = 294 get displayed in DC-TA Coverage Indicator in accordance to Equation 2.

D3 Visualization: AD-WIRE displays bipartite graph visualization, to explain the complicated and important dependency between tags and item attributes values. Figure 6 shows the bipartite graph for the rules in Table 2. Tags are found to the left side of the graph and item attribute values to the right. On hover of any node, the visualization highlights the hovered node, its edges and corresponding nodes. User can have a quick glance at the total feature space and it's corresponding tags, thus enabling user to better understand the product under review.

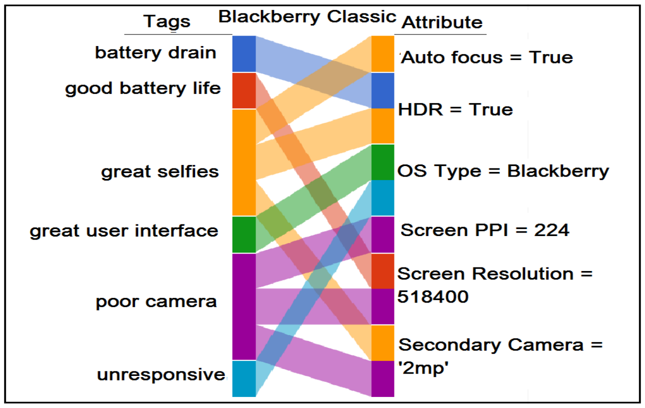


Figure 6: AD-WIRE UI: D3 Visualization.

4. ACKNOWLEDGMENT

The work of Azade Nazi and Gautam Das was partially supported by National Science Foundation under grant 1344152, Army Research Office under grant W911NF-15-1-0020 and a grant from Microsoft Research. Any findings expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors listed above.

5. REFERENCES

[1] A. Nazi, M. Das, and G. Das, "The TagAdvisor: Luring the lurkers to review web items," in *Proceedings 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," *arXiv preprint arXiv:1602.06454*, 2016.
[3] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *KDD*, 1998, pp. 80–86.
[4] 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.