DocRicher: An Automatic Annotation System for Text Documents Using Social Media

Qiang Hu [*1], Qi Liu [#2], Xiaoli Wang [*1], Anthony K.H. Tung [#2], Shubham Goyal [*1], and Jisong Yang [*1]
[*]SeSaMe Centre, [#]School of Computing, National University of Singapore
[1] {qiang.hu, dcswxl, idmsg, idmyj}@nus.edu.sg, [2] {qiliu, atung}@comp.nus.edu.sg

ABSTRACT

We demonstrate a system, *DocRicher*, to enrich a text document with social media, that implicitly reference certain passages of it. The aim is to provide an automatic annotation interface to satisfy users' information need, without cumbersome queries to traditional search engines. The system consists of four components: text analysis, query construction, data assignment, and user feedback. Through text analysis, the system decomposes a text document into appropriate topical passages, of which each is represented using detected key phrases. By submitting combinations of these phrases as queries to social media systems, the relevant results are used to suggest new annotations, that are linked to the corresponding passages. We have built a user-friendly visualization tool for users to browse automatically recommended annotations on their reading documents. Users are either allowed to rate a recommended annotation by accepting it or not; or add a new annotation by manually highlighting texts and adding personal comments. Both these annotations are regarded as the ground truth to derive new queries for retrieving more relevant contents. We also apply data fusion to merge the query results from various contexts and retain most relevant ones.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models, Relevance feedback, Clustering

General Terms

Theory and design

Keywords

Document enrichment, Social media, Ranking

1. INTRODUCTION

The popularity of e-reader applications for desktops, tablets and mobile devices, gives rise to implicit information need of readers. To enhance users' reading experience, existing systems attempt to augment text documents with supplementary materials mined from the web (e.g., [1, 6, 10, 12]). Though making much progress on linking web contents to words, phrases, and sections, they either require the document to have explicit section structure, or assume that items (words or phrases) are really important for readers. Such assumptions, however, are indeed unpractical. First, many text documents like web pages do not have obvious section structure. Second, no technique can guarantee that all readers are interested in the selected items based on their respective requirements. Others attempt to link social media to a short article (e.g., [10]). It is hard to adapt such work to support lengthy documents, as they completely ignore the reference between social media and the detailed contents.

In this paper, we focus on a new task: *given a text document, we find social media contents, which implicitly reference certain passages of it.* We use social media as the augmentation materials, instead of other web contents for two reasons. First, social media often reflects an individual's interests [11]. This presents the opportunity to amplify a document with additional information following users' interests. Second, social media often contains rich information, such as images and videos, that can enhance the text contents. This avoids the complex steps in previous work, to identify media data like images, from the results of search engines [1]. We focus on attaching social media to relevant passages of the document, instead of specific terms or the whole article. We believe that this is a better way to direct users to better understand the detailed reference between them. It poses three challenging problems to address the task: 1) How to segment the document into appropriate granularity of passages? 2) How to generate effective keyword queries to retrieve relevant social media? 3) How to efficiently assign the query results into correct passages?

We develop an automatic annotation system, denoted by *DocRicher*, by proposing techniques to solve the aforementioned problems. The main idea follows three steps in Figure 1 from (a) to (c). In Step 1, we detect the topic hierarchy of a document, to decompose it into an appropriate number of textual units. Although several topic hierarchy generation techniques have been proposed (e.g., [2, 14]), they either have no guarantee of obtaining high quality of segmentation [3], or are difficult to construct the topical tree. Consequently, we define the problem as finding a best clustering among consecutive text segments, and adapt the idea of finding the V-optimal histogram [8], to solve it. We employ a dynamic programming algorithm to find best boundaries among clus-

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2735379.

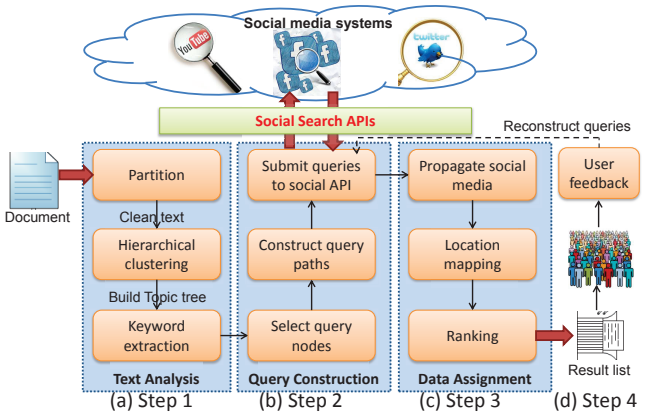

Figure 1: System workflow

ters, and use the clustering results in each iteration to build each level of topic hierarchy. Step 2 uses the topic hierarchy to generate a limited number of keyword queries. Each query is constructed from a subtree in the hierarchy, containing combinations of selective phrases in the tree nodes within the subtree. The number of queries can be flexibly controlled by the rate limit in social media systems[1]. In Step 3, the returned results are only propagated to the leaf nodes in the corresponding subtree. We assign each result in a greedy way, by computing its Overlap to each tree node. Within a leaf node, we map most relevant results into the location of a segment based on the cosine similarity.

Based on the above methodology, the system automatically crawls social media systems, and assigns the query results to relevant passages in a document. For each passage, the assigned results are used to generate annotations for recommendation. As shown in Figure 1 (d), the system also implements a user-friendly interface to visualize the suggested annotations, allowing users to make feedback by accepting them or not. Moreover, users are allowed to add manual annotations by attaching personal comments into highlighted texts. We take the accepted results of highest ranks and the manual annotations of most popular as new contexts to build new keyword queries, and use effective data analytics and fusion tools to refine the query results.

Compared to existing systems, we provide a more elegant platform to augment a document with social media. First, social contents present the opportunity to amplify a document with information following the user's interests. Second, displaying annotations by highlighting the corresponding passages are more user-friendly for users to understand a document. This avoids the complex term selection techniques required by existing work, and overcomes the limitations on previous work that can be only applied to documents with explicit section structures.

2. TECHNICAL FOUNDATION

2.1 Text Analysis

Given a document, we analyze its text contents to construct a topic hierarchy, aiming to decompose it into an appropriate number of textual units. This paper defines the problem as finding a best clustering among consecutive text segments based on the term vector model [4].

[1]http://en.wikipedia.org/wiki/Rate_limiting

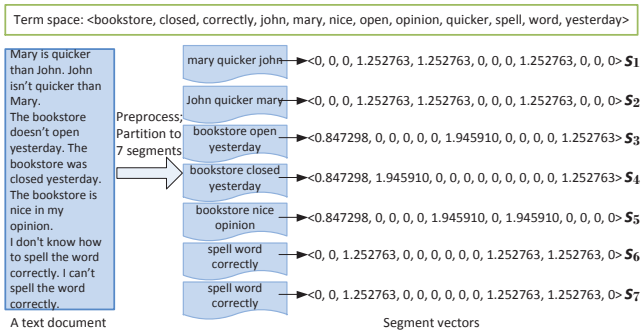

Figure 2: Segment vectors from a text document

Let d be a text document, t be a term representing a word, and $|d| = M$ be the total number of unique terms in a document. Suppose that a document is partitioned into N segments and each segment is represented as a weight vector in the term space. Let $\{\boldsymbol{s_1}, \ldots, \boldsymbol{s_N}\}$ be the set of N segment vectors, where each $\boldsymbol{s_i} \in R^M$ for $1 \leq i \leq N$, i.e., $\boldsymbol{s_i} = (w_{1,i}, \ldots, w_{M,i})$. For $1 \leq j \leq M$, we have

$$w_{j,i} = tf(t_j, \boldsymbol{s_i}) \times isf(t_j) = tf(t_j, \boldsymbol{s_i}) \times \ln \frac{N}{sf(t_j)},$$

where $tf(t_j, \boldsymbol{s_i})$ is the frequency of t_j in $\boldsymbol{s_i}$, and $sf(t_j)$ is the number of segments containing t_j. The weight evaluates a term based on its frequency within a segment and its distribution across all segments. We have $isf(t_j) = 0$ if t_j appears in every segment. This helps to avoid terms that may not be useful for identifying segment boundaries.

EXAMPLE 1. *In Figure 2, a text document is preprocessed and partitioned into* 7 *segments, i.e.,* $N = 7$. *The term space has totally* 12 *unique terms, i.e.,* $M = 12$. *For each segment vector* $\boldsymbol{s_i}$ *with* $1 \leq i \leq 7$, *we compute its weight value from* $w_{1,i}$ *to* $w_{12,i}$. *For example,* $\boldsymbol{s_3}$ *is computed as* $<0.847298, 0, 0, 0, 0, 0, 1.945910, 0, 0, 0, 0, 1.252763>$. *The weight value of* $w_{1,3} = tf_{1,3} \times isf_3 = 1 \times \ln \frac{7}{3} = 0.847298$, *as there are three segments containing the term of "bookstore".*

We preprocess the text document before using the term vector model. There are typically three steps for document preprocessing: sentence boundary identification, top-word elimination, and stemming [7]. In general, the punctuation of dot is used as the sentence boundary, and words without any semantic information are eliminated. For the purpose of stemming, we use a popular open source tool [9].

As segments may have different lengths, we normalize a segment vector $\boldsymbol{s_i}$ to a unit segment vector $\hat{\boldsymbol{s_i}}$, i.e.,

$$\hat{\boldsymbol{s_i}} = \frac{\boldsymbol{s_i}}{\|\boldsymbol{s_i}\|},$$

where $\|\boldsymbol{s_i}\|$ is the L^2 norm of $\boldsymbol{s_i}$. It can be computed as $\sqrt{w_{1,i}^2 + w_{2,i}^2 + \ldots + w_{M,i}^2}$. Intuitively, this normalization helps to identify similar segments that have different lengths but deal with the same topic. A normalized segment vector $\hat{\boldsymbol{s}}$ lies on the unit sphere in R^M. For example, the segment vector $\boldsymbol{s_3}$ in Figure 2 can be normalized into a unit vector $\hat{\boldsymbol{s_3}}$ of $<0.343797, 0, 0, 0, 0, 0, 0.789567, 0, 0, 0, 0, 0, 0.508318>$ as shown in Figure 4. Given two such unit vectors $\hat{\boldsymbol{s_1}}$ and $\hat{\boldsymbol{s_2}}$, let $\theta(\hat{\boldsymbol{s_1}}, \hat{\boldsymbol{s_2}})$ denote the angle between them. The cosine similarity is computed as the inner product, i.e.,

$$cos(\theta(\hat{\boldsymbol{s_1}}, \hat{\boldsymbol{s_2}})) = \hat{\boldsymbol{s_1}}^T \hat{\boldsymbol{s_2}}.$$

In this paper, we use the cosine similarity to evaluate the similarity between two segments. Intuitively, two segments with higher cosine value are more similar. That means that they may have more similar term distributions, and may follow the same subtopic. Accordingly, we formulate the problem of finding a best clustering among consecutive text segments as follows.

PROBLEM 1. *Given a text document with its normalized segment vectors, a limit k on the number of clusters, and an objective function called "MaxQuality", find a clustering that maximizes the objective function and follows the constraint that each cluster contains continuous segments.*

The objective function, is the sum of cosine similarities between all segment centres and their neighborhoods in the same cluster. In each cluster denoted by $\boldsymbol{I_i}$, the segment centre is defined as $\boldsymbol{c_i} = \frac{1}{|\boldsymbol{I_i}|}\sum_{\hat{\boldsymbol{s}}\in\boldsymbol{I_i}} \hat{\boldsymbol{s}}$, where $|\boldsymbol{I_i}|$ is the number of segments in the cluster $\boldsymbol{I_i}$. We normalize it as $\hat{\boldsymbol{c_i}} = \frac{\boldsymbol{c_i}}{\|\boldsymbol{c_i}\|}$. Then the *MaxQuality* of $\boldsymbol{I_i}$ is computed as

$$\sum_{\hat{\boldsymbol{s}}\in\boldsymbol{I_i}} \hat{\boldsymbol{s}}^T\,\hat{\boldsymbol{c_i}} = \|\sum_{\hat{\boldsymbol{s}}\in\boldsymbol{I_i}} \hat{\boldsymbol{s}}\|.$$

We develop an optimal algorithm to solve problem 1. The solution is similar to finding the V-optimal histogram using dynamic programming [8], that attempts to find the best boundaries among k clusters by maximizing the objective function. The optimal algorithm uses a systematic search to solve the problem. In Figure 3, the optimal clustering of k clusters can be reduced to the optimal clustering of k-1 clusters by enumerating all possible boundaries for the last k^{th} cluster. Therefore, the *MaxQuality* of the optimal clustering can be computed using dynamic programming.

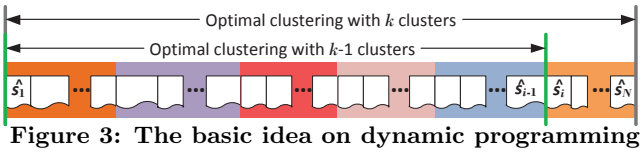

Figure 3: The basic idea on dynamic programming

Continuing with Example 1, suppose we normalized the 7 obtained segment vectors into unit vectors. We then present how to build a V-optimal histogram with $k = 3$ clusters on these unit vectors using dynamic programming.

EXAMPLE 2. *Consider the input of 7 unit segment vectors in Figure 4 (a). Figure 4 (b) shows three iterations to run the optimal algorithm. Iteration 1 first constructs the V-optimal histogram with 1 cluster. The MaxQuality is computed for every possible interval. For example, MaxQuality[1..3] is computed as the MaxQuality in the interval of "1..3" (= $\{\hat{\boldsymbol{s_1}}, \hat{\boldsymbol{s_2}}, \hat{\boldsymbol{s_3}}\}$), i.e., MaxQuality[1..3] = 2.23. Iteration 2 constructs the histogram of 2 clusters using the results from Iteration 1. Figure 4 (b) shows an example to calculate MaxQuality[1..7]. There are six possible clustering. By computing the MaxQuality for each clustering, the second or fifth one is the best one that has a maximum value. Our algorithm randomly pick the fifth one as the best clustering. Thus, we have MaxQuality[1..7] = 4.86. Similarly, Iteration 3 computes the histogram of 3 clusters. Finally, the optimal clustering for 7 given vectors is found with MaxQuality[1..7] = 6.04.*

We use the output of the optimal algorithm to produce the topic hierarchy in a top-down approach. See the running example in Figure 4. We output the best clustering in each iteration of the optimal algorithm and retain the hierarchical relationships between clusters as shown in Figure 4 (c). Then, the topic hierarchy can be naturally generated by representing each node using the selective keywords or phrases from the segment centre in each cluster as shown in Figure 4 (d). The only problem is how to extract informative keywords to label the tree nodes. We use a new ranking score, denoted by *ctf-dtf*, for selecting the keywords based on the term frequency. Given a cluster $\boldsymbol{I_i}$ and a term t. The $ctf(t, \boldsymbol{I_i})$ is the number of occurrences of t in $\boldsymbol{I_i}$. The $dtf(t)$ is the number of occurrences of t in the whole document d.

$$\text{ctf-dtf}(t, \boldsymbol{I_i}) = \alpha \times \frac{ctf(t, \boldsymbol{I_i})}{|\boldsymbol{I_i}|} + (1-\alpha) \times \frac{dtf(t)}{|d|}$$

Here, α is a predefined weight with $0 \le \alpha \le 1$. In our implementation, we use $\alpha = 0.5$. A term occurring both frequently in the cluster and in the whole document will have a high ranking score. Consequently, it will be a strong candidate for being a representative keyword. Consider the cluster in the first level in Figure 4 (c). Terms "bookstore, ..., word" of higher scores are selected, and used as labelled keywords for the root tree node as shown in Figure 4 (d). To detect key phrases containing more useful context information, we use an adjacency list to index each item, storing all its adjacent words in the document. For any two keywords, we use the adjacency list to check whether they are adjacency or not, and return the count they appear together in a passage. If the count exceeds a given threshold, we combine them into one phrase. In Figure 4 (d), two keywords "closed" and "yesterday" are combined into one phrase.

In our optimal algorithm, the parameter k is flexible to control the granularity of segmentation, i.e., the node number: how many textual units are to be identified at each level. With dynamic programming, we can expand the level number by incrementally computation from its parent level, with guarantee on the quality of clustering, i.e., the quality of segmentation in each level. As far as we know, no previous technique can satisfy all these requirements.

2.2 Query Construction

The topic hierarchy can represent a text document in a top-down fashion. That is to say, the lower-level nodes can summarize topical passages more specifically than the upper-level ones, by reserving the context. Based on the hierarchy, it is easy to derive multiple queries for retrieving relevant social media contents. The naive approach is to construct one query for each root-to-leaf path, using the combination of selective phrases in the leaf node. As shown in Figure 4 (e), three queries are generated using all the root-to-leaf paths of the topic tree in Figure 4 (d). For each query path, the selected keywords or phrases are submitted to the API of social media systems like Twitter Search API using the OR search[2]. Clearly, the query count is equal to the leaf node number. Note that a phrase including multiple keywords is regarded as AND operators in the search.

In practice, the naive solution sometimes fails to satisfy our requirements, as the social media system like Twitter

[2]https://dev.twitter.com/rest/public/search

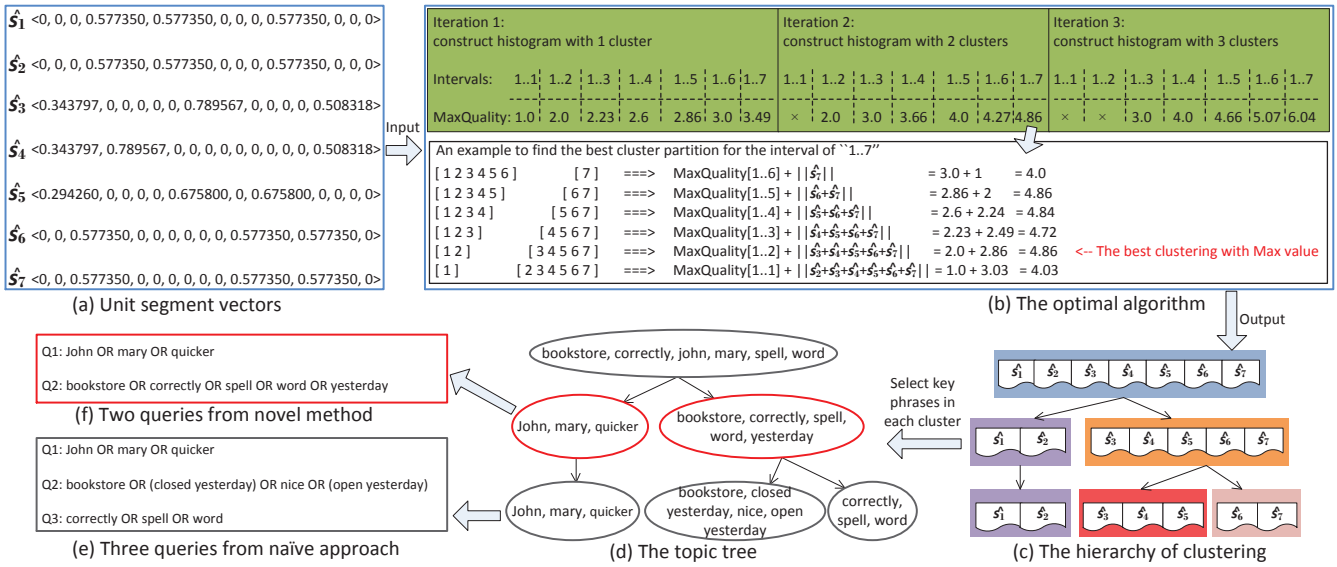

Figure 4: A running example

has the rate limit problem[3]. The need arises to construct a limited number of queries, especially when the allowed request number is less than the leaf node number. So far, no work takes this need into account.

We propose a bottom-up aggregation method to reduce the query number. The intuition is to group some query paths together if their leaf nodes contain the lowest common ancestor[4] (LCA). Based on the property of topic tree, each level only has one pair of nodes derived from a LCA in one step higher level. If we aggregate two query paths containing this pair of nodes into one subtree from the bottom level, we can reduce the query number one by one until the cardinality of query set obeys the rate limit. For each query subtree, we can construct one query using the selected phrases of its LCA, as an OR search to retrieve social media systems. For a query subtree having only one branch, we can use any descendant as the selected LCA. Continuing with the example in Figure 4 (e), Q2 and Q3 are grouped into one query using their LCA in Figure 4 (f). The LCAs are presented in the red circles in Figure 4 (d).

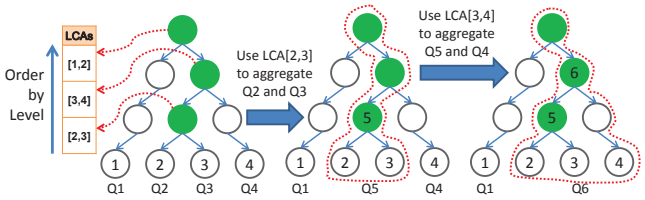

Figure 5: An example to store LCAs

To efficiently support query aggregation, the need arises to pre-compute LCAs for each pair of consecutive nodes in the topic tree. We adapt Tarjan's off-line lowest common ancestors algorithm [5], to store LCAs in each level of the tree. See Figure 5. Given a four-level topic tree, we find LCAs for each pair of consecutive leaf nodes, which are indexed in a list. For example, the LCA of leaf nodes 2 and 3 is denoted by "[2,3]" in the list of LCAs. Clearly, each level contains only one LCA, and the LCA list is ordered by level number from bottom to top. With this index, it is easy for us to apply the bottom-up query aggregation. Suppose the limit only allows us to submit two queries. Given four root-to-leaf paths, we have four queries of Q1, Q2, Q3, and Q4, which are constructed using the combination of phrases from four leaf nodes. We show how to reduce the query number using the LCA index. We scan the list to first access the LCA of "[2,3]". Then, the query paths containing leaf nodes of 2 and 3 are aggregated into one query subtree, as shown in the red dotted region in the second three in the figure. Then a new query Q5 is generated by using the LCA of node 5. Similarly, we can further integrate two queries of Q5 and Q4 into one query of Q6 by using the LCA of "[3,4]". Thus, the two queries of Q1 and Q6 are finally constructed and submitted to the social search API.

2.3 Data Assignment

While obtaining relevant social media contents, we employ a propagation based method to assign the query results to relevant passages in two phases.

1. For each query, we propagate the returned results down to the descendant nodes of the query subtree using a greedy algorithm. For each result, we compute its Overlap to all the descendant nodes, and assign it to the one with the highest Overlap value. All results are finally assigned into the leaf nodes of the topic tree.

2. For each leaf node, we use the cosine similarity to rank the results and map each result into the appropriate segment (passage). For each segment, we only retain most relevant results and discard those of very low ranking scores. Moreover, if there is no result in a segment, we submit the selected phrases of it as an AND search to retrieve its relevant contents.

We use the Overlap to evaluate the similarity between a result and a tree node, that is defined as the size of the intersection between two term vectors. To finally rank the assigned results for a passage, we use cosine similarity as it is a most common measure [4].

[3]https://dev.twitter.com/rest/public/rate-limiting
[4]http://en.wikipedia.org/wiki/Lowest_common_ancestor

2.4 User Feedback

As described above, the query results are used to automatically suggest annotations that are attached to appropriate passages. We further improve the quality of enrichment by adopting the idea of crowdsourcing[5]. That is, we implement a visualization tool that allows users to browse social media annotations for a document and make feedback. Users can either rate the relevance of a suggested annotation by accept or reject it, or manually add an annotation with personal comments by highlighting texts. For a suggested annotation, we assume that it is more relevant if more users accept it and vice versa; while for a manual annotation, we consider it is more popular with more likes and replies (users can like or comment a manual annotation). According to user feedback, we use the most relevant or popular annotations as the ground truth, to generate new queries for retrieving social media. See Figure 6. The crowdsourcing results from users are re-used as the new contexts for constructing queries. We also apply standard data fusion approaches [13], to merge query results from various contexts and retain most relevant social contents.

2.5 Technical Advantages

The topic hierarchy generation algorithm is the main techniques in our *DocRicher* system. It contains obvious benefits that are three-fold: 1) we propose a global optimal solution with guarantee on the quality of clustering in each level of the topic hierarchy; 2) we provide a flexible way to generate a given number of keyword queries using the hierarchical relationship, overcoming the rate limit problem of social APIs; 3) we use the propagate strategy to assign the query results to avoid unnecessary cross-passage similarity computations.

3. DEMONSTRATION

We demonstrate the *DocRicher* system, with all characteristics described above, such that users can have better understanding of how *DocRicher* employs social media contents like Tweets to augment a document.

Figure 7 presents a snapshot of the ebook reader interface after automatic enrichment on a guide book[6]. The right column exhibits the ebook reading panel; while the left panel marked as (A) presents the "Top Related Tweets", which are suggested annotations automatically returned from our *DocRicher* system. Users can conveniently browse new incoming tweets by clicking the "Enrich" button on the top. When users move the mouse cursor to a specific tweet, such as the one marked as (B), the ebook reading panel automatically show up the highlighted texts of the corresponding passage, marked as (C), that is associated with the tweet. Meanwhile, users can interact with the system to judge that whether a suggested tweet is relevant to the highlighting passage or not, by clicking the "Accept" or "Reject" button inside the tweet. Users are also allowed to do a manual annotation by highlighting a passage and writing the comment. The region marked as (D) lists such manual annotations.

DocRicher is equipped with an automatic recommendation system based on knowledge discovery on a text document and its annotations. We run a periodical task on the background to process each text document. Given a page of "Tourist Guide Sydney" as an example, we first analyze its text contents to generate a three-level topic tree, as shown in the region marked as (E), with each node consisting of extracted phrases. For example, the second leaf node has phrases of *tourist*, *Sydney Opera House* and *Opera*, which are summarized from the passage marked as (C). We construct a query with combinations of these phrases to search the Twitter API, and visualize most relevant results for users to understand the underlying discoveries in an interactive manner. For example, the first three tweets in the region marked as (A) are the top-3 results. If the suggested annotation marked as (B) is accepted as a relevant result by many users, then we save this as a permanent annotation, that will be late displayed in the region marked as (D). We further adopt those permanent annotations with highest relevant scores to construct new queries for retrieving social media, and do data fusion for both old and new results. The top results for each passage are updated, and will be displayed in the region marked as (A) when users refresh current "Tourist Guide Sydney" page or click the "Enrich" button.

[5]http://en.wikipedia.org/wiki/Crowdsourcing

[6]The book is from http://www.australiatourism.com.

Acknowledgment

This work was supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO, at the SeSaMe Centre, under the project of Readpeer.

4. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Kannan, and K. Kenthapadi. Similarity search using concept graphs. In *CIKM*. ACM Association for Computing Machinery, November 2014.

[2] R. Angheluta, R. D. Busser, and M.-F. Moens. The use of topic segmentation for automatic summarization. In *Workshop on Text Summarization in Conjunction with the ACL*, pages 11–12, 2002.

[3] L. Carroll. Evaluating hierarchical discourse segmentation. *ACL*, 2010.

[4] I. S. Dhillon, Y. Guan, and J. Kogan. Refining clusters in high dimensional text data. In *SIAM*, pages 71–82, 2002.

[5] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *STOC*, pages 246–251, 1983.

[6] N. Gandhi, V. Gaikwad, P. Kasat, N. Garg, A. Doke, V. Kumar, S. Karande, V. Banahatti, and N. Pedanekar. Pustack: Towards an augmented, scalable and personalized interface for paper textbooks. In *APCHI*, pages 174–177, 2013.

[7] V. Gupta and G. S. Lehal. A survey of text summarization extractive techniques. *JETWI*, 2(3):258–268, 2010.

[8] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[9] A. G. Jivani. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[10] W. Kang, A. K. H. Tung, W. Chen, X. Li, S. Qiyue, C. Zhang, F. Zhao, and X. Zhou. Trendspedia: An internet observatory for analyzing and visualizing the evolving web. In *ICDE*, 2014.

[11] M. Michelson and S. A. Macskassy. Discovering users' topics of interest on twitter: A first look. In *AND*, pages 73–80, 2010.

[12] R. Mihalcea and A. Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *CIKM*, CIKM '07, pages 233–242, 2007.

[13] J. A. Shaw, E. A. Fox, J. A. Shaw, and E. A. Fox. Combination of multiple searches. In *TREC*, pages 243–252, 1994.

[14] Y. Yaari. Segmentation of expository texts by hierarchical agglomerative clustering. *RANLP*, 1997.

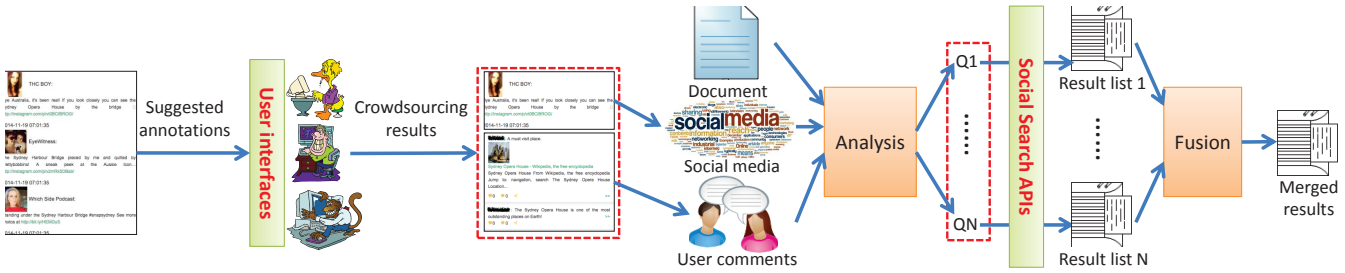

Figure 6: The architectural implementation of user feedback

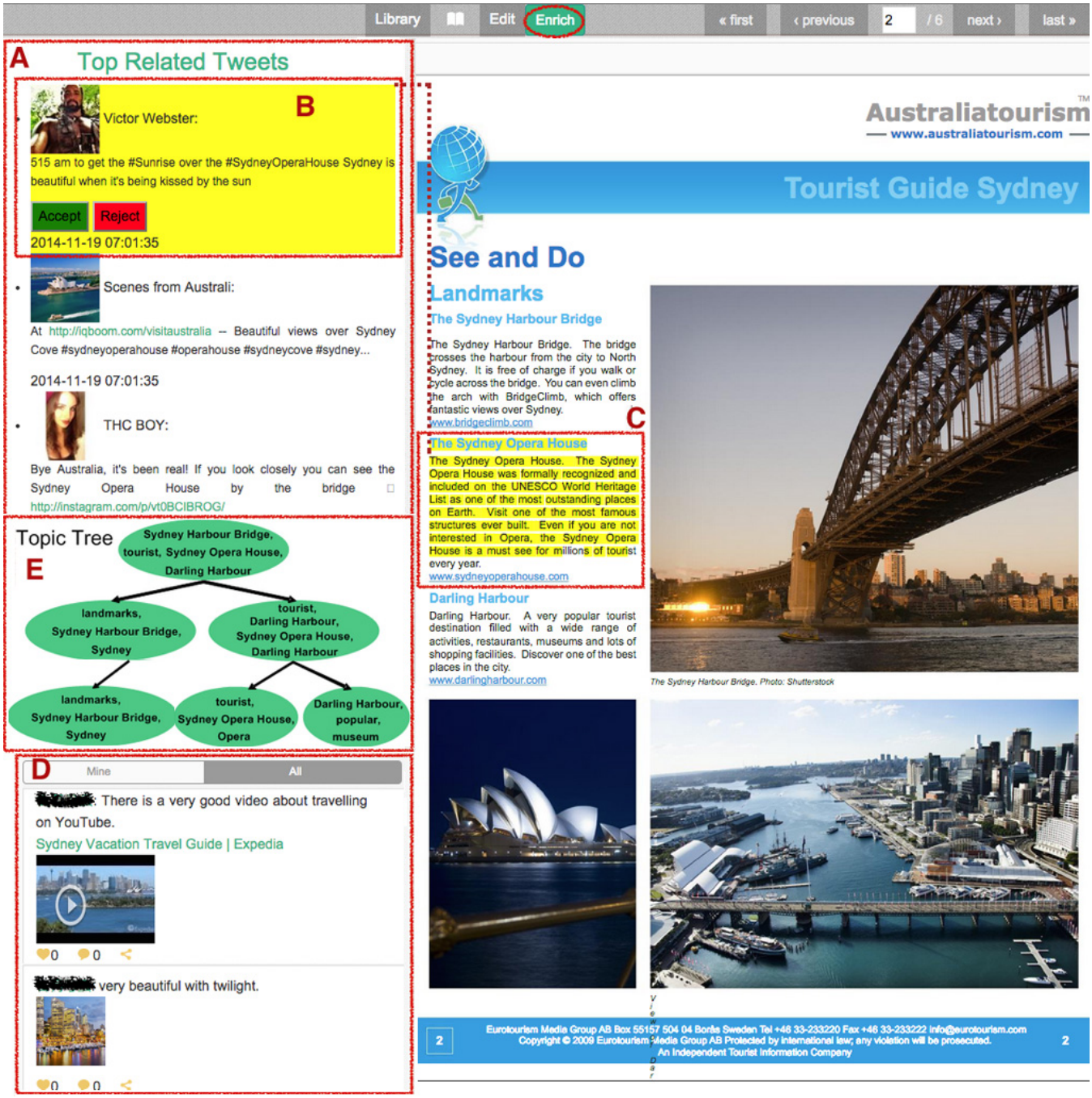

Figure 7: Current ebook reader with enriched tweet annotations