TWITTER NEWS: HARNESSING TWITTER TO BUILD AN ARTICLE RECOMMENDATION SYSTEM

Arman Suleimenov

THESIS
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF MASTER OF SCIENCE IN
ENGINEERING

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF COMPUTER
SCIENCE

Adviser: Andrea S. LaPaugh

June 2012

1 Abstract

With more than 140 million active users and 340 million tweets a day (as of March 2012), Twitter presents a great source of recommendation knowledge for articles shared on the platform. In this work, we analyze 836 Twitter users from the technology and entrepreneurship domain with 78,508 links shared by them. We explore and evaluate different (existing and novel) techniques for a recommender system for articles including the following ones: vector-to-vector similarity where the user vector is constructed from the text of the tweets produced, topic-modeling based approach where we learn the topic distribution for each article, as well as the novel hybrid technique which is based on piecewise article vector representation, content-boosted collaborative filtering with pseudo user-ratings as well as the relatedness function which depends on relevance, novelty, connection size and transition smoothness between articles.

2 Introduction

According to the study conducted at UCSD [1], an average American consumes about 11 hours of information per day and 13.99% of this is spent on web browsing and email. With an abundance of information, computers should help us in minimizing the time to find the useful content. Search engines are great when there is a clear user intent - you know exactly what you are looking for and formulate that in a search query. However, often we don't know specifically what we are looking for and just want to read 'something interesting'. As a result, users rely on the social links from communities like reddit and Hacker News, the feeds on Twitter and Facebook, RSS readers like Google Reader. However, with noisy and intensive stream of new content on these sites, users face a 'needle in a haystack' challenge when they want to find the most interesting articles in least amount of time. This thesis addresses this challenge by analyzing algorithms for recommending URLs based on URLs shared by users on Twitter.

Filtering and discovery are essential for the content recommendation system. Users want to narrow down the full list of items they are exposed to in order to get the most of the time invested in the attention scarcity age. On the other hand, it is useful to see interesting content outside of your network to expand your horizons and stay on top of the most relevant articles. To help users filter and discover the web's best content, our approach is to make the list of articles to read personalized. With about 140 million active users and more than 2 billion tweets posted every week, Twitter is a wonderful platform to choose as the source of links. Twitter users are not just passive consumers of content, they also produce (tweet) and the user-generated content can be used in our personalization algorithms. The content on Twitter is recent and that is essential to being considered interesting. Unlike other research projects which use Twitter, we ignore the social interaction between users in order to limit our analysis to a computationally feasible set. From personal experience as well as the previous research [2] which showed that sharing URLs is one of the common uses of Twitter, we know that by choosing this platform it is certain that we get a significant set of links ready for personalization. Another important reason to choose Twitter is the fact that the company provides a set of public APIs letting us download the users' public timelines[1].

3 Problem formulation

Formally, the article recommendation problem can be formulated as follows. Given a set of all users $U = \{u_1, u_2, \dots, u_N\}$ and a set of all articles $C = \{c_1, c_2, \dots, c_D\}$ shared by those N users, recommend to a user M articles that she might be interested in reading. Let $f(u, c)$ be a utility function that measures the desirability of item c to user u, $f : U \times C \to R$ (where R is totally ordered set of user-item scores). Then for each user u, the goal of our system is to choose M articles from C which maximize the user utility. Let $r_{ij} \epsilon \{0, 1\}$ denote whether user u_i shared article c_j on Twitter. The fact that it was shared means that user u_i is interested in article c_j. It is worth noting the distinction with other recommender systems like Netflix or Amazon where users explicitly rate items, in the Twitter case $r_{ij} = 0$ can be interpreted in two ways: either user u_i is not aware of the existence of article c_j or she is not interested in it.

The goal of our recommender system is to recommend articles which are not in a user's library (i.e. have not been shared by her). There are two types of articles to recommend: out-of-matrix and in-matrix. In out-of-matrix prediction also known as "cold start recommendation" we are dealing with articles which have never been rated (i.e. shared by none of the users). Since these articles are not associated with any ratings, collaborative filtering techniques, which use other users' ratings, can not make predictions on these items. Despite the importance of out-of-matrix prediction in recommending newly published articles, the focus of this work is to investigate the articles of the second type. In in-matrix prediction, as the name suggests, the recommendation system operates on the articles rated

[1]The main challenge of using Twitter API as of May 2012 is the strict rate limits: 350 requests per hour, and up to 3,200 of a user timeline's most recent statuses an hour.

(i.e. shared) by at least one user in the system. We will only consider the in-matrix prediction.

4 Related Work

There is a great amount of research in news recommender systems as it constitutes a problem-rich research area with numerous applications which help users to efficiently sift through noise aiming to find interesting content. Recommendation engines are usually classified into the following three categories:

- collaborative filtering: the user is recommended items (links to articles in our case) that users with similar tastes and preferences liked in the past;
- content-based filtering: the user is recommended items similar to the ones she liked in the past (based on the content of the articles in our case);
- hybrid systems: the user is recommended items based on methods which combine collaborative filtering and content-based filtering.

4.1 Collaborative filtering

Collaborative filtering works by collecting ratings for items within the dataset and matching users based on the similarity of information needs or preferences.

Broadly, collaborative filtering can be categorized into two categories: memory-based, and model-based. In memory-based approach, the predicted rating is calculated as the weighted average of ratings from other users. The weight is proportional to the similarity between users (Pearson correlation coefficient and cosine similarity are among the most common ways to define this user-to-user similarity). The user-to-user similarities are computed offline and assuming the ratings are binary the predicted rating for article c by user u_i is given by:

$r_{u_i,c} = \sum_{j \neq i} I(u_j, c) * sim(u_i, u_j)$

where $I(u_j, c)$ is the indicator variable which is 1 if user u_j shared an article c on Twitter and 0 otherwise, $sim(u_i, u_j)$ is similarity between users u_i and u_j. We also can introduce a set of binary variables p_{ij} to indicate a preference of user u_i to an article c_j:

$$p_{ij} = \begin{cases} 1, & \text{if } r_{u_i,c_j} > \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

Neighborhood model approach to collaborative filtering is memory-based and usually comes in two variations: user-oriented, and item-oriented. In user-oriented method unknown ratings are predicted based on the actual ratings from similar users. In item-oriented approach a rating is calculated on the basis of ratings given by the current user to similar items. Since the number of items in recommendation systems (books on Amazon, movies on Netflix) is usually less than the number of users, the item-oriented approach offers more scalability and more accuracy. Taking the item-oriented approach the equation above becomes

$r_{u_i,c} = \frac{\sum_{j \in S_{ci}} sim(c,c_j) I(u_i,c_j)}{\sum_{j \in S_{ci}} sim(c,c_j)}$

where S_{ci} is the set of k items shared by user u_i that are most similar to article c. The formula above when dealing with explicit feedback (liking, upvoting, star ratings) is usually modified to correct for biases caused by different mean ratings from different users. These enhancements are not applicable in implicit feedback model with binary ratings.

In model-based approach to collaborative filtering, the unknown ratings are predicted by models learnt from user's past ratings. The modern model-based methods categorize a user into multiple classes as users might have interests in multiple topics. Such models include latent semantic indexing (LSI) and its probabilistic modification (PLSI), Bayesian clustering, Latent Dirichlet Allocation, Markov decision process and multiplicative factor models. Latent factor models provide more accurate recommendation results than neighborhood methods, and Netflix challenge showcased that very clearly [27]. Let's describe the model induced by matrix factorization. In matrix factorization every ith user is associated with a latent user-factors vector $u_i \in R^K$ and each jth article is associated with an item-factors vector $c_j \in R^K$ where K is the dimension of the latent low-dimensional space both users and items are represented in. The prediction is made by taking an inner product of user and item vectors: $r_{ij} = u_i^T c_j$. How should we estimate the parameters? We model the observed ratings directly and avoid overfitting by minimizing the regularized square error loss with respect to matrices $U = (u_i)_{i=1}^N$ and $C = (c_j)_{j=1}^D$:

$min_{U,C} \sum_{i,j} (r_{ij} - u_i^T c_j)^2 + \lambda_u \|u_i\|^2 + \lambda_c \|c_j\|^2$

where $\lambda_u$and λ_c are used to regularize the model and are usually learnt through stochastic gradient descent.

Collaborative filtering has several advantages over content-based approaches. First, it can support for recommendation of items that are not easily analyzable by computers (movies, images, people) since the relevance is solely determined by the users of the system. Second, collaborative filtering is capable of analyzing items beyond the dimensions discovered by pure content-based methods: the interestingness of an article can be based on its quality or user's taste. Third, collaborative filtering methods are able to make serendipitous recommendations - items whose content do not match with the information the system has gathered about users, but which in fact is of great interest to them. Hu et al. studied latent-factor based collaborative filtering on datasets with implicit feedback and found that implicit user-item observations should be decoupled into two parts: an estimate whether the user would like/dislike the item (preference) and a confidence level. Das et al. propose a content-agnostic approach to personalize news articles on Google News based on click history which linearly combines recommendations from the following three approaches: collaborative filtering using MinHash clustering, Probabilistic Latent Semantic Indexing (PLSI) and covisitation counts [4].

4.2 Content-based filtering

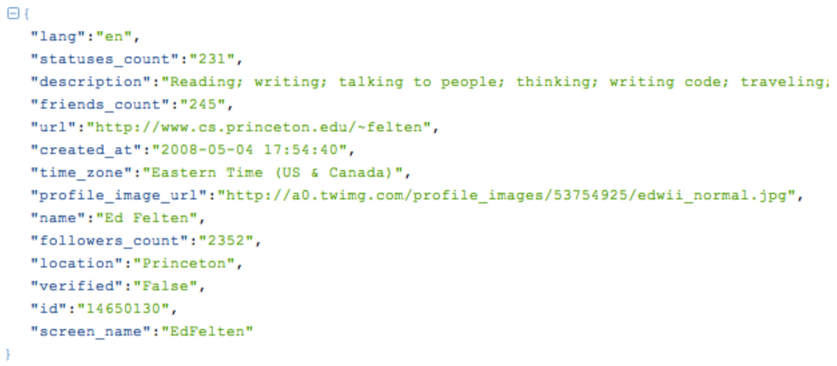
The content-based filtering is strongly connected to and has its roots in information retrieval research. Both users and articles are represented as sets of attributes characterizing them. The candidate articles are compared with the articles shared by the user and the most relevant are recommended. Kompan and Bielikova propose a short vector representation of an article for similarity search and content-based recommendation which consists of the following parts: title, term-frequency of title words in the article content, names and places, keywords, category and CLI index, which measures the understandability of the text [6]. Chen et al. show their study of link recommendation on Twitter with 12 recommendation algorithms in the design space with the following dimensions: candidates for content sources, topic interest models for users, and ranking using social voting [5]. In [16] in order to recommend unrated items to users with unique interests, authors describe a content-based book recommendation engine which utilizes information extraction and a simple Bayesian learning algorithm for text categorization.

4.3 Hybrid systems

The third (and the most successful) type of link recommendation engines are hybrid systems which combine the collaborative filtering and content-based components in order to avoid the limitations associated with each of them. One way to combine these methods is to implement each of them separately and then combine the predictions. Another approach is to incorporate the characteristics of the content-based filtering into collaborative filtering and vice versa. Finally, some recommendation systems [3] present the unified model of two techniques. Wang and Blei combine collaborative filtering based on latent factor models and content analysis based on probabilistic topic modeling to recommend scientific papers from CiteULike [3]. Using click data on Google News, Liu et al. [9] developed a Bayesian framework for predicting users' current news interests from their past activities and news trends overall. They combine the content-based recommendation score based on the predicted user's interest and topic category with the collaborative filtering score from [4] to rank the candidate articles. Melville et al. propose an elegant framework which combines neighborhood-based collaborative filtering with content-based method (bag-of-words Naive Bayes text classifier) for movie recommendation [23].

4.4 Miscellaneous

Lv et al. study the problem of post-click news recommendation on Yahoo! News and propose to identify related news articles based on four aspects: relevance, novelty, connection clarity, and transition smoothness (the latter two concepts are described in detail in the Algorithms section) [7]. Li et al. model personalized news article recommendation as a contextual-bandit problem where an algorithm sequentially selects articles based on contextual information about users and articles while adapting its selection strategy to maximize the total number of clicks [20]. Wei et al. propose different scalable stochastic algorithms to optimize social network topology in order to build a recommendation model that deals with no explicit news ratings by users [8]. Mannern et al. [14] build a global user profile across many existing services, and enrich news meta-data via smart indexing and linking available datasets in order to provide an open news recommendation platform. Abel et al. introduce a user modeling framework based on hashtags, entities (events, persons, products, etc.), or topics and compare these user models in the context of a news recommendation system [15]. Hannon et al. [17] propose a recommender of Twitter users to follow by modeling the users by tweets and relationships of their social graphs. Phelan et al. explored four different strategies for filtering news articles from personal and community RSS indexes by using the personal Twitter feed as well as the tweets from a user's Twitter social graph [24]. Guy et al. study the personalization on the enterprise activity stream of Lotus Connections (social media application suite for organizations), which consists of status updates and other social media activity, by building the user profile on three dimensions (people, terms, and places) and analyzing their effect on accuracy and throughput [10]. In LOGO authors study the evolution of user interests and propose a news recommender which integrates the long-term and short-term reading preferences.



```
{
  "lang":"en",
  "statuses_count":"231",
  "description":"Reading; writing; talking to people; thinking; writing code; traveling;
  "friends_count":"245",
  "url":"http://www.cs.princeton.edu/~felten",
  "created_at":"2008-05-04 17:54:40",
  "time_zone":"Eastern Time (US & Canada)",
  "profile_image_url":"http://a0.twimg.com/profile_images/53754925/edwii_normal.jpg",
  "name":"Ed Felten",
  "followers_count":"2352",
  "location":"Princeton",
  "verified":"False",
  "id":"14650130",
  "screen_name":"EdFelten"
}
```

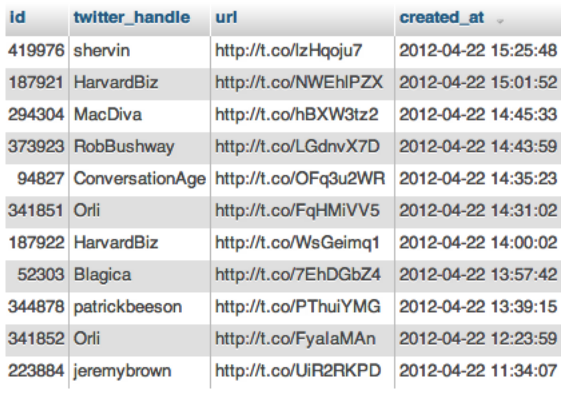
Figure 1: The Twitter profile information for Ed Felten

5 Data

We created a list of 2,385 hand-picked Twitter users in technology and entrepreneurship domains. This was done by scraping 21 lists curated by various influential bloggers as well as the people in the space I follow personally. The lists include web innovators, influential hackers, top technology bloggers, entrepreneurs, companies and software engineers. To make the data collection feasible[2], only 836 of these 2,385 were considered for analysis. For each of these users, I collected the profile information which included the username, number of tweets, number of followees/followers, language, etc. In figure 1, you can see the profile associated with Ed Felten, Professor of Computer Science and Public Affairs at the Princeton University.

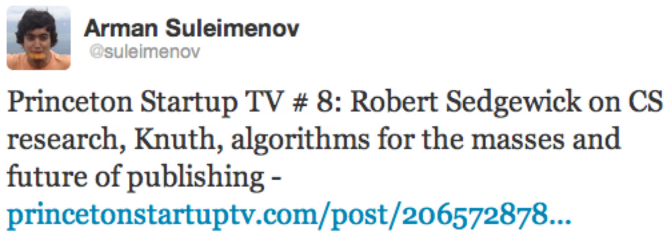
For each user, we collected the tweets from his or her timeline in batches of 50 tweets per API request. To limit the number of tweets from one microblogger, we put a cap of 10,000 tweets per user. In the database we stored only three pieces of information per each status update: the text of the tweet, Twitter handle and the date. All other data such as the number of retweets, and the score according to Topsy (for which we used the API of the real-time search engine for the social web, Topsy) relevant to each tweet was ignored. This data might be used in the future to incorporate the social signals in the recommendation algorithms. Then we found all the tweets which contain URLs ignoring the tweets which start with '@' (mentions) as they usually contain the links relevant only to the user to whom they're directed. We saved in the database the following triple: (Twitter handle, URL, date of posting). This resulted in 514,739 unique URLs from 836 users. To construct the pool of URLs

[2]To fight the strict Twitter rate limits for REST API, I wrote different daemons in Python which were periodically making API calls and collecting data using 6 Twitter developer accounts across 6 machines: all four cycles servers from the Princeton Computer Science Department (soak.cs.princeton.edu, wash.cs.princeton.edu, rinse.cs.princeton.edu, spin.cs.princeton.edu), my 2Gb RAM node on Linode as well as my laptop.



id	twitter_handle	url	created_at
419976	shervin	http://t.co/lzHqoju7	2012-04-22 15:25:48
187921	HarvardBiz	http://t.co/NWEhlPZX	2012-04-22 15:01:52
294304	MacDiva	http://t.co/hBXW3tz2	2012-04-22 14:45:33
373923	RobBushway	http://t.co/LGdnvX7D	2012-04-22 14:43:59
94827	ConversationAge	http://t.co/OFq3u2WR	2012-04-22 14:35:23
341851	Orli	http://t.co/FqHMiVV5	2012-04-22 14:31:02
187922	HarvardBiz	http://t.co/WsGeimq1	2012-04-22 14:00:02
52303	Blagica	http://t.co/7EhDGbZ4	2012-04-22 13:57:42
344878	patrickbeeson	http://t.co/PThuiYMG	2012-04-22 13:39:15
341852	Orli	http://t.co/FyalaMAn	2012-04-22 12:23:59
223884	jeremybrown	http://t.co/UiR2RKPD	2012-04-22 11:34:07

Figure 2: The format of the links corpus used for analysis



Arman Suleimenov
@suleimenov

Princeton Startup TV # 8: Robert Sedgewick on CS research, Knuth, algorithms for the masses and future of publishing - princetonstartuptv.com/post/206572878...

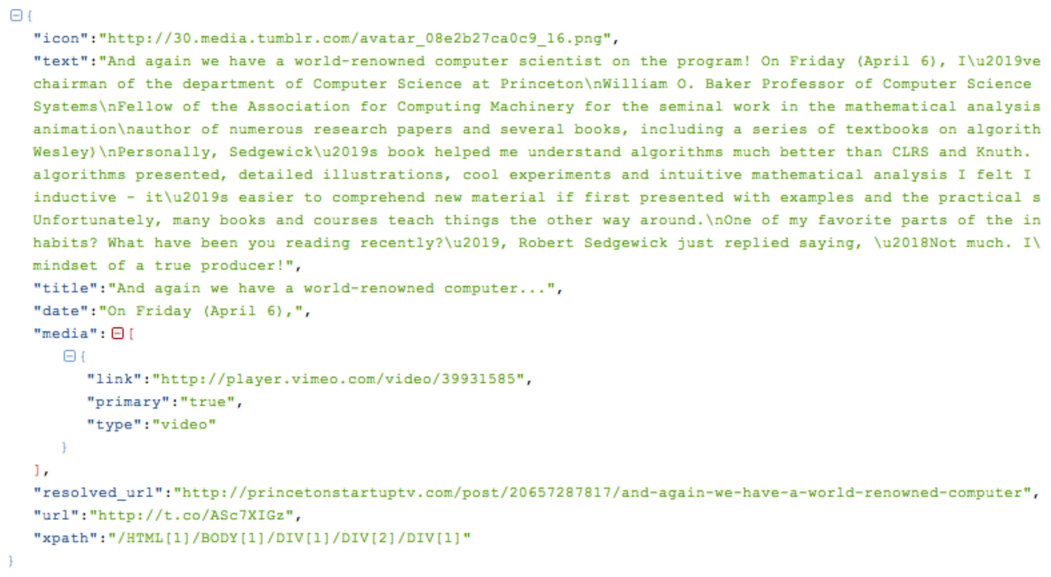
Figure 3: Example tweet which contains a link to an article

we used for analysis, we sorted all the URLs by the date of creation (the most recent first) and only take the maximum of 100 URLs per user. The reason we only looked at the most recent URLs was to increase the overlap of articles shared by users (most people usually post links to new articles) and, hence, reduce the sparsity of our users-articles matrix. The resulting pool which was ultimately used for content-based recommendations contains 78,508 links. To provide context, in figure 2 we displayed the format of the links corpus.

The next step was to extract the title and the clean text of an article (with ad blocks, comments, html tags removed) for every URL. This is a challenging part in itself as we haven't limited ourselves to articles from a small number of well-formatted sites like The New York Times or the Atlantic. It requires to learn the structure and layout of every page to find the location of the headline, name of an author, keywords/tags and the actual content of an article. To classify the components of every webpage and get the relevant semantic metadata I used tools like Diffbot [3] and Readability [4].

An example of tweet and the JSON object associated with a link shared within that tweet are displayed in figure 3 and figure 4 respectively.

[3]http://www.diffbot.com
[4]http://pypi.python.org/pypi/readability-api/



```
{
  "icon":"http://30.media.tumblr.com/avatar_08e2b27ca0c9_16.png",
  "text":"And again we have a world-renowned computer scientist on the program! On Friday (April 6), I\u2019ve
  chairman of the department of Computer Science at Princeton\nWilliam O. Baker Professor of Computer Science
  Systems\nFellow of the Association for Computing Machinery for the seminal work in the mathematical analysis
  animation\nauthor of numerous research papers and several books, including a series of textbooks on algorith
  Wesley)\nPersonally, Sedgewick\u2019s book helped me understand algorithms much better than CLRS and Knuth.
  algorithms presented, detailed illustrations, cool experiments and intuitive mathematical analysis I felt I
  inductive - it\u2019s easier to comprehend new material if first presented with examples and the practical s
  Unfortunately, many books and courses teach things the other way around.\nOne of my favorite parts of the in
  habits? What have been you reading recently?\u2019, Robert Sedgewick just replied saying, \u2018Not much. I\
  mindset of a true producer!",
  "title":"And again we have a world-renowned computer...",
  "date":"On Friday (April 6),",
  "media":[
     {
        "link":"http://player.vimeo.com/video/39931585",
        "primary":"true",
        "type":"video"
     }
  ],
  "resolved_url":"http://princetonstartuptv.com/post/20657287817/and-again-we-have-a-world-renowned-computer",
  "url":"http://t.co/ASc7XIGz",
  "xpath":"/HTML[1]/BODY[1]/DIV[1]/DIV[2]/DIV[1]"
}
```

Figure 4: The JSON object associated with a single article

Number of users	836
Number of users with no fewer than 10 links	821
Number of user-url pairs	519,754
Number of unique URLs	514,739
Sparsity of the user-item matrix	99.879%
Minimum number of links shared by the user	14
Maximum number of links shared by the user	3,139
Average number of links in user's library	633

Table 1: Dataset stats

5.1 Statistics for the dataset

Let's present some statistics on the analyzed dataset in the table 1.

6 Algorithms

6.1 TF-IDF

In this approach we take all the tweets from the timeline of user u, remove all the stop words, apply stemming[5] and then create the following profile vector $Profile(u) = (v_{u1}, \ldots, v_{um})$ where m is the total number of unique words in u's tweets. The value of v_{ui} is calculated using the classic TF or TF-IDF (term frequency / inverse document frequency) measures. In the first approach (referred to

[5]I used Porter stemmer from the Natural Language Toolkit: http://www.nltk.org/

as the TF approach throughout the paper), TF weight for v_{ui} term in u's tweets is defined as

$v_{ui} = TF_u(w_i) = 1 + log_{10}(freq_{w_i})$

where $freq_{w_i}$ - number of occurrences of word w_i in u's tweets. In the second approach (referred to as TF-IDF approach), TF-IDF weight v_{ui} for term w_i in u's tweets is then defined as $v_{ui} = TF_u(w_i) * IDF_u(w_i)$. Here,

$TF_u(w_i) = freq_{w_i}/max(freq_{w_z})$

where maximum is computed over all frequencies of keywords w_z which appeared in user u's tweets.

$IDF_u(w_i) = log((N + D)/n_i)$

where N, D are the total number of users and articles in the dataset respectively, n_i is number of users who used the term w_i at least once. The intuition behind these measures is trivial. High values of TF indicate that the user frequently uses these keywords, thus indicating higher interest. Higher values of IDF mean that few other users use this term in their tweets which signify the uniqueness of the user and can better distinguish between different users. Similarly, we define TF and TF-IDF vectors for articles. Then the utility $f(u, c)$ can be computed using the cosine similarity measure:

$f(u, c) = cos(Profile(u), Content(c)) = \frac{v_u \cdot v_c}{|v_u||v_c|} = \frac{\sum_{i=1}^{K} v_{ui} v_{ci}}{\sqrt{\sum_{i=1}^{K} v_{ui}^2}\sqrt{\sum_{i=1}^{K} v_{ci}^2}}$

where K is the total number of words in the dictionary.

6.2 Topic Modeling

In this approach we use the topic modeling algorithm known as latent Dirichlet allocation (LDA) [28] to discover a set of topics from a collection of all articles. In LDA every document (in our case - news article or the document which consists of all the tweets from a single user) is represented as a mixture of topics where a topic is a distribution over terms which is biased around those associated with a single theme. LDA assumes that a document is produced in the following fashion:

1. Decide on the number of words the document c_j will have.

2. Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of K topics): $\theta_j \sim Dirichlet(\alpha)$. For example, $\theta_j = (0.25, 0.4, 0.35)$ would mean that document c_j is 0.25 about topic 1, 0.4 about topic 2, and 0.35 about topic 3.

3. Generate each word n by:

    (a) Picking a topic (according to the multinomial distribution sampled above): $z_{jn} \sim Mult(\theta_j)$

(b) Generating the actual word (according to the topic's multinomial distribution): $w_{jn} \sim Mult(\beta_{z_{jn}})$

Given this generative model, LDA attempts to backtrack from the documents to find a set of topics that are most likely have generated the collection. Compared to a clustering model which assigns every document to a single cluster (topic), LDA lets articles exhibit multiple topics.

In the first technique based on LDA (let's refer to this method as LDA1), our document collection consists of $N + D$ documents where N, D are the total number of users and articles respectively. The document representing every user is the union of all tweets from her timeline. We run LDA to discover topic distributions for every document. Given K topics, every document j is represented by a K-vector $(\theta_{j1}, \theta_{j2}, \dots, \theta_{jK})$ where θ_{ji} is the topic proportion of topic i in the document j. Then the utility $f(u, c)$ where u and c are both K-vectors outlined above representing a user and an article respectively can be again calculated as a cosine similarity of topic vectors.

In the second technique based on LDA (we'll refer to this algorithm as LDA2), our document collection consists of only D articles. In other words, we are ignoring the text of the tweets posted. Then the same way as in the first technique we run the LDA and, based on the per-document topic distributions, every document j is represented by a K-vector $\theta_j = (\theta_{j1}, \theta_{j2}, \dots, \theta_{jK})$. How do we represent the user as a vector? We look at all the articles user u shared on Twitter and sum the vectors associated with those articles component-wise. More formally, $Profile(u) = \sum_z \theta_z$ where z runs over all articles z from user u's library. The scoring $f(u, c)$ is again based on cosine similarity.

6.3 Hybrid

Let's present a novel hybrid recommendation technique. The evaluation of this technique requires editorial judgement and we will leave it for future work. We will combine the scores from each of the algorithms described below as $\sum_A w_A r_c^A$ where w_A is the weight given to algorithm A, r_c^A - score given by algorithm A to article c [4]. For algorithms (1, 3) where we define the item-to-item similarity measure, r_c^A is defined as the average of all values $sim(c, t)$ where t runs through all articles shared by the active user. Top M articles are chosen as recommendations. Weights w_A's should be learned by picking the best parameter combination from a pre-selected discrete parameter space (different combinations of weights). Once this is done, for more accurate results we should consider using SVM with linear kernel to learn weights w_A's. Let's describe every individual algorithm comprising this hybrid technique.

6.3.1 Piecewise Article Vector Representation

In this approach we will construct the recommendations in the following manner extending the idea presented by Kompan and Bielikova [6]. For every article shared by the user we find the top 10 most similar articles (according to the similarity measure defined below), merge the results, remove the duplicates and present to the user M articles with the highest score overall. Here we decided to go beyond simplistic representation of an article as a TF/TF-IDF of all the words it contains. The article vector will consist of the following parts (we refer to each part as the partial vector):

1. **Title**. This part contains all the words within the title (stemmed, with stopwords ignored). The number associated with every word is its frequency proportion within the title. For example, if the title is "The Long Grind Before You Become an Overnight Success", the corresponding partial vector is $(\text{"}long\text{"} : 0.2, \text{"}grind\text{"} : 0.2, \text{"}become\text{"} : 0.2, \text{"}overnight\text{"} : 0.2, \text{"}success\text{"} : 0.2)$.

2. **Term frequencies of title words in the article**. This part contains of lemmatized title words (same as in part 1 - Title), but numbers associated with them are defined differently. Here, it is the frequency across the entire article: $TF(word_i) = count(word_i)/\sum_k count(word_k)$ where $count(word_i)$ is the number of occurrences of ith word in the document and k runs over all words of an article.

3. **Term frequencies of top 20 most frequent words in the article**.

4. **Topic proportions of the K topics found by topic modeling algorithm on the entire article corpus.**

5. **Coleman-Liau Index**. This readability test index measures the understandability of a text. Its result approximates the US grade level (US grade level 1 corresponds to ages 6 to 8) necessary to comprehend the text. This part is obviously not crucial for the item-to-item similarity, however, it is needed to rank the results based on the hypothesis that a Twitter user wants to read articles of a similar difficulty level.

The article-to-article similarity is defined as an extension of a standard cosine similarity (let two articles be represented by vectors s and t):

$$similarityScore(s,t) = \frac{\sum_{i=1}^{5}\sum_{j=1}^{n} s_{ij} t_{ij}}{\sqrt{\sum_{i=1}^{5}\sum_{j=1}^{n} s_{ij}^2}\sqrt{\sum_{i=1}^{5}\sum_{j=1}^{n} t_{ij}^2}}$$

where n is the cardinality of each of the 5 parts of the entire article vector. To make sure that cardinalities of vectors whose inner product is taken match, we assume sparse representation of each of the partial vectors. As expected, when taking the inner product of two vectors, we multiply the values associated with the same keys component-wise: (“*peter*” : 0.2, “*thiel*” : 0.2, “*palantir*” : 0.2, “*ycombinator*” : $0.4)^T$(“*paul*” : 0.2, “*graham*” : 0.2, “*ycombinator*” : 0.25, “*essays*” : 0.35) $= 0.4 \cdot 0.25 = 0.1$.

6.3.2 Content-Boosted Collaborative Filtering

Since the user-article matrix is highly sparse (with sparsity of 99.879%), we will replace 0’s in the matrix (corresponding to articles not shared by users) with pseudo-ratings obtained from content-based predictor. For content-based predictions we will use a bag-of-words Naive Bayes text classifier. Every article is turned into features: title, text of an article, author of an article, domain of the URL to indicate the publication name, the presence of media (video). To build the model that classifies articles, Twitter users used in the dataset should judge some articles as either interesting and worth clicking or not. The articles, users shared on Twitter, are labeled as interesting by default. Then we will user 5-fold cross validation to find the best split of all the labeled articles into a training set and a test set (80/20 split). From the set of judged links to articles, the classifier learns a user profile and predicts ratings for unlabeled links. The pseudo rating r_{ik} of article c_k by user u_i is defined as 1 if the rating is available (u_i shared the link to c_k in her Twitter feed) and as item rating from the Naive Bayes classifier othwerise:

$$r_{ik} = \begin{cases} 1, & \text{if user } u_i \text{ has article } c_k \text{ in his library} \\ nbc_{ik}, & \text{otherwise} \end{cases}$$

where nbc_{ik} is the the rating predicted by Naive Bayes classifier.

Now when pseudo-ratings are created, let’s describe the collaborative filtering component of the algorithm which is based on neighborhood model. First, we weight all users in the dataset on the basis of their similarity with the active user. If we use the Pearson correlation coefficient for this measure, then the similarity between user u_i and u_j will be defined by the following formula:

$$P_{ij} = \frac{\sum_{k=1}^{D}(r_{ik}-\overline{r_i})(r_{jk}-\overline{r_j})}{\sqrt{\sum_{k=1}^{D}(r_{ik}-\overline{r_i})^2 \sum_{k=1}^{D}(r_{jk}-\overline{r_j})^2}}$$

where D is the total number of articles, r_{ik} is the pseudo-rating of article c_k from user u_i, $\overline{r_i}$ is the mean rating by the user u_i. Finally, we compute the content-boosted collaborative filtering prediction $pred_{ik}$ for the user u_i and article c_k as the weighted combination of n users most similar to user u_i according to Pearson correlation coefficient:

$$pred_{ik} = \overline{r_i} + \frac{\sum_{j=1}^{n}(r_{jk}-\overline{r_j})P_{ij}}{\sum_{j=1}^{n} P_{ij}}$$

If a user rates very few items in the training stage of the Naive Bayes classifier, the accuracy of pure content-based ratings and, hence, pseudo-ratings will not be as good. The inaccuracies in pseudo-ratings lead to misleadingly high correlations between users. Therefore, as the potential improvement, we can incorporate confidence in correlations by multiplying the Pearson correlation coefficient in the formula for $pred_{ik}$ by Harmonic Mean weighting factor [23].

6.3.3 Technique based on relatedness of articles

Borrowing an idea from Lv et al. [7], let's show how we can define the relatedness between a pair of articles. The relatedness is based on the four aspects: relevance and novelty, connection clarity and transition smoothness. Given the relevance and novelty, connection size and transition smoothness scores as well as the editorial judgement, we can learn the relatedness function $f(s,t)$ from the general boosting method GBRank [31]. Let's describe each aspect of the relatedness in detail.

Relevance and novelty On the one hand, if two articles are too dissimilar, they are not really related. On the other hand, if two articles are too similar, it is likely that a user has no interest in the other as it does not contain any new information when compared to a story she has already read. There are standard information retrieval functions which capture both relevance and novelty in one score. Apart from well-known cosine similarity, they are Okapi BM25 [29], language models with Dirichlet prior smoothing [30], and language models with Jelinek-Mercer smoothing [30].

Connection size All the functions which measure relevance and novelty are essentially estimating the size of the pure word overlap between two documents. This, of course, overlooks the fact that there might be otherwise unrelated articles that share the same words. Connection size is an attempt to deal with this by modeling the cohesiveness of topics in two articles (topic overlap as opposed to simple word overlap). By taking probabilistic topic modeling approach (e.g. LDA), let's first identify topics associated with each document in the pair. Then the connection size between article s and t can be calculated as the negative Kullback-Leibler divergence (also known as information gain):

$connection(s,t) = -\sum_{i=1}^{K} P(i|s) log \frac{P(i|s)}{P(i|t)}$

where K is the total number of topics, $P(i|s)$ is the probability associated with topic i in the article s. The more topics two articles share, the higher the score. This measure can be further improved to decrease the effects of topics associated with many articles and to deal with the case when two articles share many loosely correlated topics, but do not share the same discriminative topic with high probability [7].

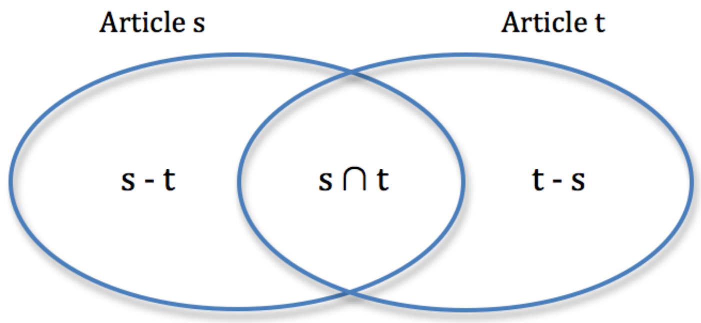


Figure 5: Venn diagram of content intersection of two articles

Transition smoothness Transition smoothness measures the continuity between articles s and t. Two articles can be called transition-smooth if the article formed by combining them is coherent. As shown in Venn diagram in figure 5, $s \cap t$ is shared among articles s and t and should be ignored when considering transition smoothness. Rather, we want to know the transition between $s - t$ and $t - s$. How can we represent $s - t$ and $t - s$? Let's represent each by the document-based context vector $\vec{context}(s - t)$ whose cardinality is D - the number of articles in the entire pool. The weight of the ith component $\vec{context}(s - t)_i$ corresponding to article c_i is the relevance score $f(s - t, c_i)$ defined according to Okapi BM25 ranking function [29]. The more relevant the article c_i to $s - t$, the more significant role it will play in the vector $\vec{context}(s - t)$. To put everything together, the transition smoothness score between articles s and t is computed as the cosine similarity of vectors $\vec{context}(s - t)$ and $\vec{context}(t - s)$:

$$transition(s,t) = \frac{\vec{context}(s-t)\vec{context}(t-s)}{\|\vec{context}(s-t)\|\|\vec{context}(t-s)\|}$$

Alternatively, transition smoothness can be defined by modeling the word-based context [7].

7 Evaluation

We evaluate different algorithms on the set of all articles from the dataset (constructed as the union of the users' libraries) and individual user libraries. The goal of our recommender system is to find M specific articles which are supposed to be the most interesting to the user. This task is known in the literature as the top-M recommendation task [50]. Common error metrics such as RMSE (root mean squared error) or MAE (mean absolute error) which measure the average error between actual and predicted ratings are not applicable for evaluation as they are not designed to measure the top-M performance. Plus, the ratings we are dealing with have only two values (0 or 1) and the zero ratings

	Recommended	Not recommended
Shared	tp (true positive)	fn (false negative)
Not shared	fp (false positive)	tn (true negative)

Table 2: Classification of four scenarios when an item is recommended to the user

are uncertain (the user is either not interested in the article or is unaware of it).

7.1 Recall

Let's focus on the accuracy metrics such as precision and recall. There are four possibilities (table 2) when a user is presented with an article the system considers appealing to her.

Then precision is defined to be $\frac{tp}{tp+fp}$ - the fraction of recommended articles which were shared. The recall = $\frac{tp}{tp+fn}$ - the fraction of shared articles which were recommended. However, in the calculation of precision we assume that articles not shared by the user are uninteresting. This is erroneous assumption as users might be interested in it, but just don't know about its existence. Thus, the number of false positives is overestimated. Therefore, we decided to focus on recall (where one-ratings are known to be true positives). Note the trivial assumption we are making here is that if a user shares an article, it is interesting for her. It is possible that the text of the tweet (ignored by our system) which accompanies a link contains a negative comment, but knowing that the opposite of love is not hatred, but indifference, we are safe to assume that articles a user didn't like, but cared enough to share are interesting to her. We measured $recall@M = \frac{sharedInTop}{totalShared}$where $sharedInTop$ is the number of links shared by the user which appear in top M, $totalShared$ is the total number of links shared by the user. The $recall@M$ of the whole system is defined as the average of recalls of individual users. In figure 6 we present the performance of different algorithms for varying values of M (the number of recommended articles in the list): 25, 50, 75, ..., 275 on the subset of system with 20 users and 1,692 articles total. TF approach performs the poorest, and LDA2 performs slightly better than LDA. The baseline is the random model in which a user sees a random recommended set of M articles. The expected recall for the baseline method for a pool of D articles is just M/D. We could not find recall measures on the pool of URLs in other research papers for comparison. The values of recall for the model that only uses LDA-like features on the pool of scientific articles [3] are higher, but their LDA model for in-matrix prediction is also different than ours (different number of topics, different vocabulary, different maximum number of iterations to train the model, etc.).

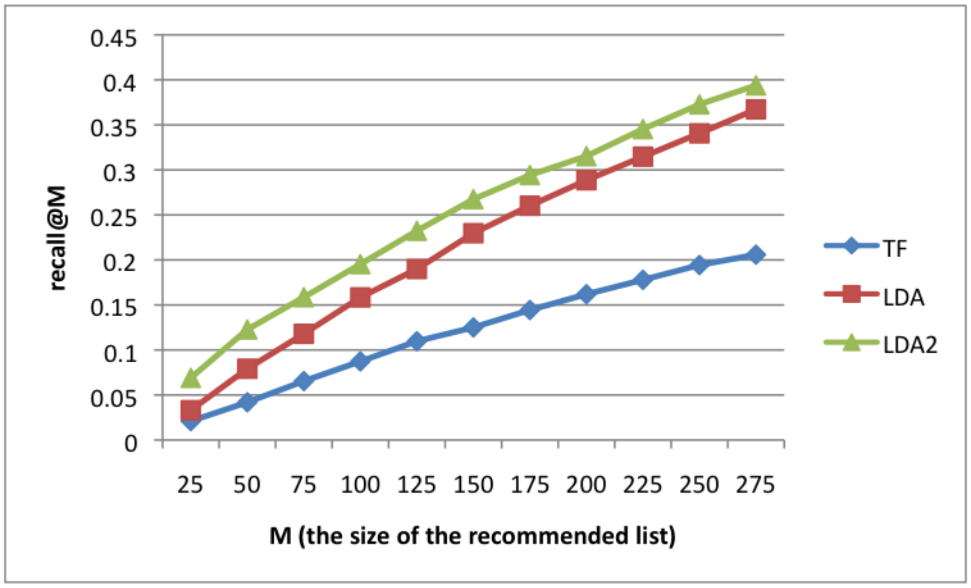


Figure 6: Comparison of recall@M values for various algorithms for different values of M

7.2 Percentile rank

Let's now measure the expected percentile rank. The percentile rank of an article is the percentage of articles that are the same or lower than it in its frequency distribution. Let $peRank_{ij}$ be the percentile rank of the article c_j in the personalized list of M articles presented to user u_i. To give some intuition, $peRank_{ij} = 0\%$ would mean that the story c_j is the most interesting article for user u_i and it is above all other articles in the top chart. Respectively, $peRank_{ij} = 100\%$ would mean that the article c_j is the least interesting one from user u_i's perspective and is placed at the end of the recommendation list. Then the expected percentile rank for all the articles shared by Twitter users analyzed is given by the following formula:

$$\overline{peRank} = \frac{\sum_{i,j} r_{ij} peRank_{ij}}{\sum_{i,j} r_{ij}}$$

The lower the $\overline{peRank}$, the better the recommender system, as low values of this quality measure would mean that the articles actually shared by users on Twitter were ranked high in the personalized recommendation lists. For the baseline algorithm which just makes random rating predictions and on average puts the article c_j half-way the recommendation list, expected value of the rating $E[r_{ij}] = 50\%$. Hence, any algorithm with $\overline{peRank} > 50\%$ performs worse than random guessing. In figure 7 we can see the measured values of $\overline{peRank}$ for algorithms TF, LDA, LDA2 (as defined in 6.1 and 6.2) for varying values of M (the number of recommended articles in the list): 25, 50, 75, ..., 275 on the subset of system with 20 users and 1,692 articles total. We can see that all 3 algorithms compared achieve far

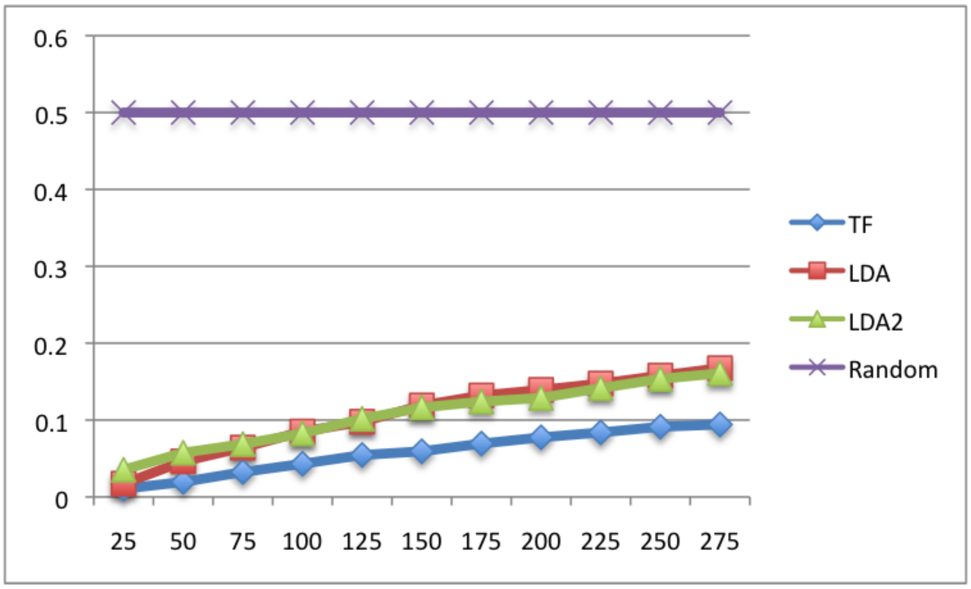


Figure 7: Comparison of expected percentile rank values for various algorithms on different values of M

better results than the random predictor. LDA and LDA2 ($\overline{peRank} = 16.7\%$ and 16.0% for $M = 275$ respectively) personalize the lists, but TF approach ($\overline{peRank} = 9.4\%$ for $M = 275$) obtains even better personalization.

8 Future work

There are many possibilities for the work to be conducted in the future. First, we plan to launch a full-fledged website where users sign in with their Twitter usernames and the system presents them a personalized rank list of links from their friends and from the pool of popular links in the system to allow serendipitous discovery. This website will be used to get user judgements via up-voting/down-voting functionality to improve the algorithms and evaluate the system based on explicit feedback. Second, we plan to incorporate social signals in the recommendation engine: the number of retweets, the proximity to the author of the link in the Twitter social graph, average time between consecutive tweets to give a smaller weight to links posted by users who share too much. Third, in this work we only analyzed the in-matrix prediction task leaving out-of-matrix prediction unexplored. Out-of-matrix prediction, however, deals with the very common scenario when a new article is published and no one in the system has seen (has rated) it. Evaluating different algorithms on the out-of-matrix prediction setup will show how different methods solve the first rater problem on the set of news articles. Fourth, we plan to thoroughly evaluate the performance of the proposed boosting-inspired hybrid method and, if necessary, replace the worst performing algorithm components with the better ones. Fifth, it will be interesting to see how well the collaborative topic regression (the method which combines matrix factorization based collaborative filtering with probabilistic topic modeling) by Wang and Blei [3] does on the pool of highly sparse user-article data from Twitter. Sixth, to deal with sparsity of the user-article matrix, we plan to explore the possibility of treating the near-duplicate articles as a single document. It is worth noting that we do not mean the near-duplicate in the simple word overlap sense. Rather, we mean the articles which share a large portion of the same topics, but might be using a quite different vocabulary. Seventh, to improve the trust in the article recommender system, we plan to explain recommendations by accompanying every article recommended with a short description why it is in the recommendation list. Latent factor models which can identify the features with the highest contribution to the recommendation can be helpful to accomplish this. Eighth, we ignore the time of the submission when ranking the links. However, to surface hot and new articles we should let old ones expire by decreasing the relevance score as time goes by, regardless of their relevance based on content-based and collaborative filtering methods. Ninth, we ignored the text of the tweets which accompanied the links in the recommendation process due to inherent noisiness of the data (abbreviations, misspellings, etc.). As the future direction we are planning to explore the possibility of incorporating this piece of information via sentiment analysis and applying topic modeling on Twitter-style short text documents [32, 33, 34, 35]. Also, richer user models can help in improving recommendations. One idea we have is to do topic modeling on the titles of 'Twitter lists' (ad-hoc lists curated by users, e.g. 'technology pundits', 'web innovators', 'hackers') a user belongs to in order to find her affiliation, interest and thematical category. Finally, we should not eliminate the opportunity to use links shared outside of Twitter: on Facebook, LinkedIn, Google Plus, Hacker News, reddit, etc..

9 Conclusions

In this work, we explored how link sharing activity on Twitter can be leveraged in order to make personalized content-aware URL recommendations. The main contribution of this work is three-fold. First, we demonstrated the potential of the real-time web, and micro-blogging services like Twitter more specifically, to serve as a useful source of links to articles for recommendation. Second, using recall and expected percentile rank measures we evaluated several algorithms on in-matrix prediction task on the unique dataset of 78,508 links shared on Twitter by 836 users. Unlike most other Twitter-related reseach papers, we take a content-aware approach by considering in the analysis the title and actual text of articles. Finally, we showed how existing ideas in recommendation algorithms can be extended and propose a novel hybrid method for personalized article recommendation using the intelligence of the social web.

References

[1] R. Bohn and J. Short. "How Much Information? 2009 Report on American Consumers" (Global Information Industry Center of University of California, San Diego, San Diego, CA, 2009). Retrieved from http://hmi.ucsd.edu/pdf/HMI_2009_ConsumerReport_Dec9_2009.pdf.

[2] A. Java, X. Song, T. Finin, and B. Tseng. Why We Twitter: Understanding Microblogging Usage and Communities. In Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis. ACM, 2007.

[3] C. Wang and D. Blei. M. 2011. Collaborative Topic Modeling For Recommending Scientific Articles. In Proceedings of 2011 ACM SIGKDD International Conference on Knowledge Discovery And Data Mining (KDD'11). 448–456.

[4] A. Das, M. Datar, A. Garg, and S. Rajaram. Google News Personalization: Scalable Online Collaborative Filtering. In Proceedings of the 16th International Conference on World Wide Web (WWW'07), 2007.

[5] J. Chen, R. Nairn, L. Nelson et al. Short and Tweet: Experiments on Recommending Content from Information Streams. In Proceedings of CHI'10, ACM Press (2010).

[6] M. Kompan, M. Bieliková. Content-Based News Recommendation. In E-Commerce and Web Technologies, volume 61 of Lecture Notes in Business Information Processing, pages 61–72. Springer, 2010.

[7] Y. Lv, T. Moon, P. Kolari, Z. Zheng, X. Wang, Y. Chang (2011). Learning to Model Relatedness for News Recommendation. In Proceedings of WWW '11, ACM, pages 57-66.

[8] D. Wei, T. Zhou, G. Cimini, P. Wu, W. Liu, and Y.-C. Zhang. Effective Mechanism for Social Recommendation of News. Physica A: Statistical Mechanics and its Applications, 390, 11, 2117-2126, 2011.

[9] J.H. Liu, P. Dolan, E.R. Pedersen, Personalized News Recommendation Based on Click Behavior. In Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI), 2010, pages 31–40.

[10] I. Guy, I. Ronen, A. Raviv. Personalized Activity Streams: Sifting through the 'River of News'. In Proceedings of the ACM Recommendation Systems 2011 Conference, 2011.

[11] G. Shani, A. Gunawardana. Evaluating Recommendation Systems. In Recommender Systems handbook. Springer, Berlin, pages 257–298, 2011.

[12] G. Adomavicius and A. Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, IEEE Transactions on Knowledge and Data Engineering 17 (2005), pages 634–749.

[13] L. Li, L.Zheng, Li and T. Li. LOGO: A Long-Short User Interest Integration in Personalized News Recommendation. In Proceedings of the ACM Recommendation Systems 2011 Conference, 2011.

[14] E. Mannern, S. Coppens, T. De Pessemier, H. Dacquin, D. Van Deursen and R. Van de Walle. Automatic News Recommendations via Profiling. In Proceedings of the 3rd International Workshop on Automated Information Extraction in Media Production (AIEMPro'10), ACM Press, Oct. 2010, pages 45-50.

[15] F. Abel, Q. Gao, G.-J. Houben, and K. Tao. Analyzing User Modeling on Twitter for Personalized News Recommendations. In International Conference on User Modeling, Adaptation and Personalization (UMAP), Girona, Spain. Springer, 2011.

[16] R. J. Mooney and L. Roy. Content-based Book Recommending Using Learning for Text Categorization. In ACM SIGIR'99. Workshop on Recommender Systems: Algorithms and Evaluation, 1999.

[17] J. Hannon, M. Bennett, and B. Smyth. Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches. In Proceedings of the fourth ACM Conference on Recommender Systems, RecSys'10, pages 199–206, New York, NY, USA, 2010.

[18] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. M. Lukose, M. Scholz, and Q. Yang. One-class Collaborative Filtering. In IEEE International Conference on Data Mining (ICDM 2008), pages 502–511, 2008.

[19] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. IEEE Computer, 42(8): 30–37, 2009.

[20] L. Li, W. Chu, J. Langford, and R. E. Schapire. A Contextual-bandit Approach to Personalized News Article Recommendation. In 19th International World Wide Web Conference WWW, 2010.

[21] A. Z. Broder. On the Resemblance and Containment of Documents. In Proceddings of Compression and Complexity of SEQUENCES, pages 21–29. IEEE Computer Society, 1997.

[22] Y. Hu, Y. Koren, and C. Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In IEEE International Conference on Data Mining (ICDM 2008), pages 263-272, 2008.

[23] P. Melville, R. J. Mooney, and R. Nagarajan. Content-Boosted Collaborative Fitering. In Proceedings of the 2001 SIGIR Workshop on Recommender Systems, 2001.

[24] O. Phelan, K. McCarthy, M. Bennett, and B. Smyth. Terms of a Feather: Content-Based News Recommendation and Discovery Using Twitter. In Proceedings of the 33rd European Conference on Advances in Information Retrieval, ECIR'11, pages 448–459, Berlin, Heidelberg, 2011.

[25] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99), 1999.

[26] P. Cremonesi. Performance of Recommender Algorithms on Top-N Recommendation Task

[27] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. IEEE Computer, 42(8):30–37, 2009.

[28] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. Journal of Machine Learning Research, 3:993–1022, January 2003.

[29] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In TREC '94, pages 109–126, 1994.

[30] C. Zhai and J. D. Lafferty. A Study of smoothing methods for language models applied to ad hoc information retrieval. In SIGIR '01, pages 334–342, 2001.

[31] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In NIPS '07. 2007.

[32] L. Hong, B.D. Davison. Empirical study of topic modeling in Twitter. In proceedings of the SIGKDD Workshop on Social Media Analytics, 2010.

[33] D. Ramage, S. Dumais, D. Liebling. Characterizing Micorblogs with Topic Models. In proceedings of AAAI on Weblogs and Social Media, 2010.

[34] M. S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, and E. H. Chi. Eddi: Interactive Topic-based Browsing of Social Status Streams. In UIST '10, 2010.

[35] A. Ritter, C. Cherry, B. Dolan. Unsupervised Modeling of Twitter Conversations. NAACL 2010.