Meliper: Making News Personal

Arden Dertat
Department of Computer Science
Brown University
Providence, RI 02912
arden@cs.brown.edu

Abstract

Getting fresh and relevant news about our interest topics is a fundamental matter in today's fast-paced information era. But there is a lack of personal solutions that treat each user individually. While following technology news via popular websites, we are presented vast number of articles about any technology related area. But if we are interested in just certain aspects of technology, we still have to browse through all the articles and try to find the ones that we are actually interested among many that are irrelevant. Meliper [2] gives users control about what news they want to receive, and the users can filter the information based on their interest. So users dont have to deal with irrelevant news. The system also learns from user behavior and adjusts retrieval and ranking accordingly.

1 Introduction

Meliper is a personal news aggregation service. The system is based on lists and keywords. Users create lists and associate keywords with those lists. Then for each list, only the news containing the keywords in that list is presented to the user. For example, say a user is interested in technology news about Microsoft, Google, Facebook; and economy news about unemployment, inflation rate, and tax cuts. Then the user can create a list called technology containing the keywords Microsoft, Google, Facebook. And a separate list named economy with keywords unemployment, inflation rate, and tax cuts. Then in the technology list, only the news about Microsoft, Google, and Facebook will presented to the user. And the economy list will contain news about its corresponding keywords. The user can create many other lists and associate keywords with them, and she will receive news about only those keywords, in their respective lists.

Section 2 explains how crawling is performed to fetch the news, Section 3 discusses parsing, and Section 4 indexing. Section 5 gives more detail about lists and keywords. Section 6 describes how news is retrieved and ranked. Finally Section 7 presents recommendation algorithms that make the system clever.

2 Crawling

Articles are extracted from RSS feeds of popular news websites based on their category. Feeds of a website contain the most recent articles published at that site, so in order to obtain a continuous stream of all articles, feeds should be crawled on regular intervals. In Meliper architecture, a uniform crawling rate of 1 hour is used for every website. RSS requests are made every hour and data about the latest 100 articles are crawled.

After crawling we have 100 article structures with various data fields. The following fields are particularly important:

- Summary: contains the summary of the article

- Title: title of the article
- Date: publish date of the article
- Author: author of the article
- Link: URL of the article
- Tags: tags of the article given by the author

2.1 2 options for parsing

There are two options to obtain the actual content of the article. Either to use the summary as content, or to get article link and parse the original article web page.

2.1.1 Summary as Content

The summary of the article in an RSS feed is the first few sentences of the content. The scope of the summary differs among different websites, for example TechCrunch includes almost all article content in the summary but New York Times only contains the first paragraph. We can assume that the summary of the article is a good approximation of the actual content. Because the first few sentences generally describe the main intent.

The advantage of this approach is speed and simplicity of implementation. We only make 1 HTTP request to get the content of 100 articles, which is the request for the RSS feed. And then we simply extract the summary from the feed.

2.1.2 Parsing article page

As another option, we can parse the original web page of the article to get the whole content. This way we can obtain the complete content but with the price of efficiency. Because for each article we have to perform an HTTP request to the original URL. Which is expensive because of network latency. In order to get the content of 100 articles we now have to make 101 HTTP requests, 1 to get RSS feed and 100 to get each individual article. Compared with only 1 request in the summary approach, this is drastically slower. But the benefit is that we now have access to the complete article content, instead of the first few sentences.

Another major drawback of this approach is that it requires different parsers for every news website. Because in an articles web page, there's many other text than article content. For example, there are user comments, advertisements, links to other articles, website's header and footer etc. Extracting only the content from the page is problematic because every website puts the content within different html elements. In some news sources the content appears in paragraphs between <p> and </p> tags, others use divisions such as the <div> tag. Even the websites using div tag differ about where they put the content, because there are many divivions at the same page with different names and a specific one contains the content. And the name of the specific div containing the article text is different between websites, there isnt a universally accepted convention. Some name it ”articleBody”, others use ”newsDetail” or ”text” etc. Without a standard representation, extracting the actual content of the article becomes troublesome and requires customized parsers for each news source.

Considering the inefficiency and complexity of this approach, the summary solution is used to obtain the article content.

3 Parsing

Our main purpose is to create an inverted index mapping each term (word) to a list of documents that term appears in (postings list). The inverted index is a dictionary type data structure where each vocabulary term is a key with its postings list being the value [3]. The goal of parsing is to extract a list of terms from a given article.

Once the RSS feed of a news website is crawled, the content of each article is obtained from the summary field as explained in Section 2.1. In order to make this content suitable for indexing, we parse it to get individual terms using the following set of operations in the following order:

1. Convert Unicode string to its ASCII counterpart. Since Meliper is a global news service providing news in multiple languages, non-English characters are converted to standard ASCII for better search results.
2. Clean HTML elements/tags such as <div> and   (represents space). Also remove non-alphanumeric characters such as punctuation, and lowercase all letters.
3. Split on whitespace to get individual words. These are the terms in our inverted index.
4. Filter out stop words, which are very common terms that dont give much information about the content, such as a, an, the etc. Very common verbs and nouns are also included in stop words. There are around 3,000 words in the stop words file and it currently covers the English language only.
5. Stem each word using Porters Stemmer [4] if the language of the article is English (language information obtained from the news source).

By performing these operations on the article content we get a list of terms representing the article. And they are added to the inverted index.

3.1 Named Entity Extraction

Users are generally interested in news about named entities and proper nouns, such as names of persons, organizations, locations etc. So they're more likely to add named entities to their keyword lists such as Federer or San Francisco, instead of regular verbs or nouns. Named entities are generally capitalized, and thats how we are going to extract them.

While parsing the article content to obtain a list of terms, we lowercase all the letters and remove non-alphanumeric characters. So we lose capitalization information that helps us to discover named entities. Punctuation is also needed to identify sentence boundaries. Therefore, a separate parsing routine is created to specifically address named entity extraction. The input is the same article content given to parsing, and the following operations are performed:

1. Convert Unicode string to ASCII, and clean HTML elements. Dont perform any other normalization such as lowercasing or filtering out punctuation.
2. Get all the sentences from normalized content and process each sentence individually. Sentence delimiters are dot, colon, question mark and exclamation point.
3. Extract named entities from sentences. A word is considered named entity if it contains a capital letter or a number. Continuous words that qualify to be named entities (n-grams) are combined together to form a single named entity. For example, New York City contains 3 capitalized words so it results in one named entity New York City. Also iPad 2 is formed of a capitalized word and a number so its combined as well.

As a result, we have a list of named entities for every article. Then we add these named entities to their inverted index, which is different from the word index described in Section 3. Further details about the inverted index is given in the next section.

4 Indexing

Every time we crawl the RSS feed of a news website, we get the most recently published 100 articles. Some of these articles are fresh stories that we havent seen before, but others are older articles that we have already indexed. So deduplication is performed to filter out the redundant articles, and only the new articles (if any) are added to the index after being parsed. Figure 1 demonstrates the system architecture and information flow.

For each crawled article, we check whether it has been indexed before. Some news websites assign unique IDs to their articles. If the article contains an ID, we check whether we already indexed an article with the given ID of that news website before. If the article was indexed we skip it and avoid parsing. Otherwise, we parse and index the article as discussed in Section 3. If the article doesnt contain an ID in the first place, then we use the title-date pair to check whether it exists in the index.

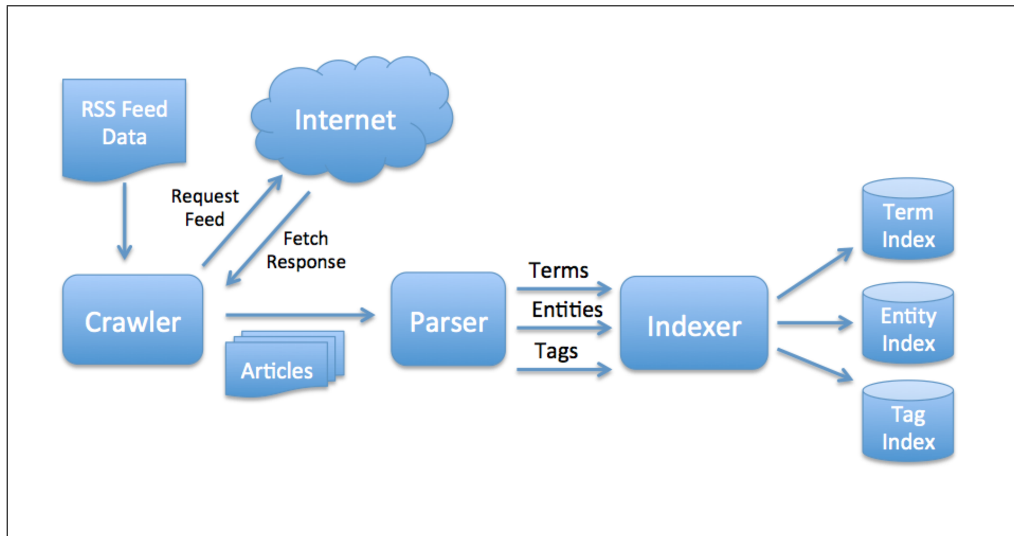

Figure 1: System Architecture

Inverted index is the main data structure in the system for news retrieval. It maps each term to the list of documents it appears in. Search for articles is performed using this index. There are thre types of indexes: term index, named entity index and tag index. Term and named entity indexes are constructed by parsing the articles as explained in Section 3 and 3.1 respectively. We refer the reader to [3] for a comprehensive discussion of inverted indexes.

Some news sources associate tags with their articles, and make them accessible through their RSS feeds. Tags basically summarize the article with various individual keywords. This is a valuable information since tags are selected by the author of the website, and considered to be relevant. We index tags corresponding to every article in a separate index and make use of this information at recommendation systems.

5 Lists and Keywords

Lists and keywords are the main building blocks of Meliper. Users create lists and associate keywords with them. Lists represent interest areas of the user and the keywords concretize those interests. Users create lists and associate keywords with them as discussed in Section 1. This section elaborates the concept of list and keywords.

5.1 List and Keyword Details

Users aggregate keywords about a particular interest topic into a list. They can have as many lists as they want, with as many keywords as they wish. List and keyword names are strings. The list name doesn't influence news retrieval, articles are selected based on keywords. A list just aggregates the articles retrieved using its keywords (news retrieval explained in Section 6).

There are only two constraints about list and keyword names. The list name should be unique for a user, so a user cant have two lists with the same name. And a keyword should be unique in a list, but the same keyword can appear in different lists

Users have the option of making their lists can public or private. Public lists of a user are visible to everyone in the system, and can also be ”followed” by other users (explained in Section 5.2). Private lists are only visible to their creators. A user can switch the privacy setting of her list anytime.

Figure 2 shows the user interface for creating list, adding/removing keywords, and changing privacy settings. The system is tried to be kept intuitive and easy to use. Figure 3 demonstrates an example

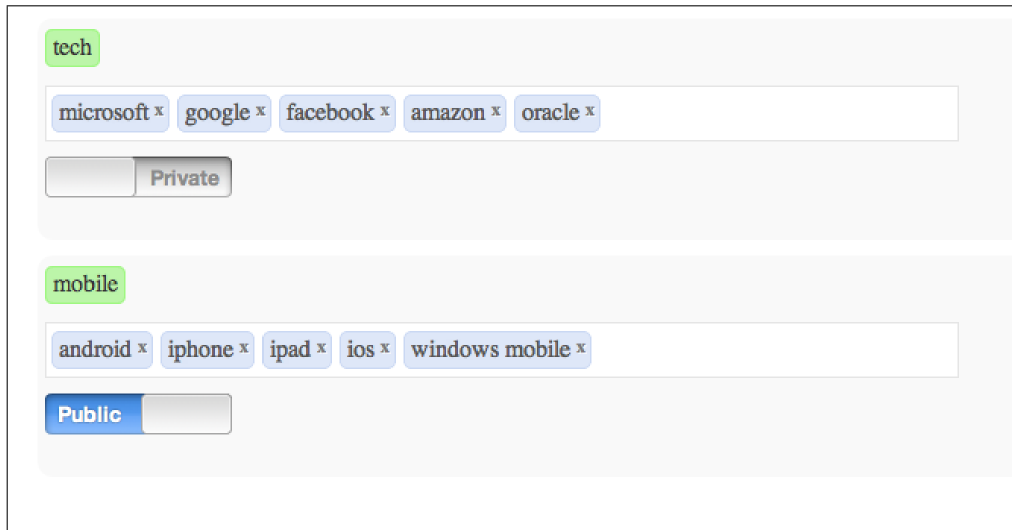

Figure 2: User interface for editing lists and keywords.

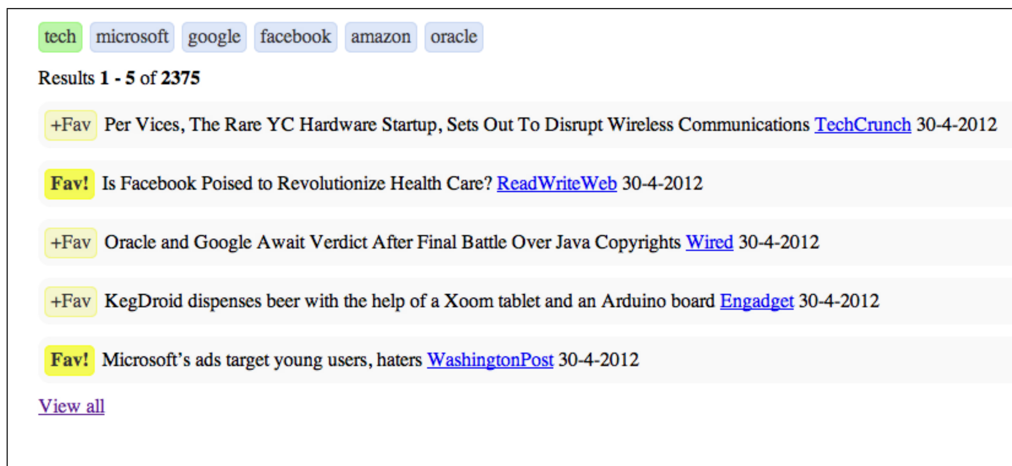

Figure 3: An example user list.

list with its associated keywords and retrieved news results. Bold yellow "Fav" to the left of an article means that it's bookmarked by the user (explained in Section 6).

5.2 Following Lists

Each list is associated with its creator. But a useful feature in the system is that users can follow other users' public lists. For example, say we have a friend that has a deep knowledge in technology, and he constructed a detailed public list with many keywords spanning various areas. We trust our friend's technology knowledge and we're also interested in that area. So instead of spending time creating our own list from scratch, we can just follow our friend's list and start receiving the same news as him. Other users can also follow his list and his list's "popularity" will increase, encouraging him to create high qualiy lists. The system is similar to Twitter's follow interface. It is non-symmetric and there's no concept of "being friends". A user can follow any public list created by other users.

When the owner of the list updates it by adding or removing keywords, all the followers of that list automatically see the changes. And only the owner of a list can modify the keywords of the list.

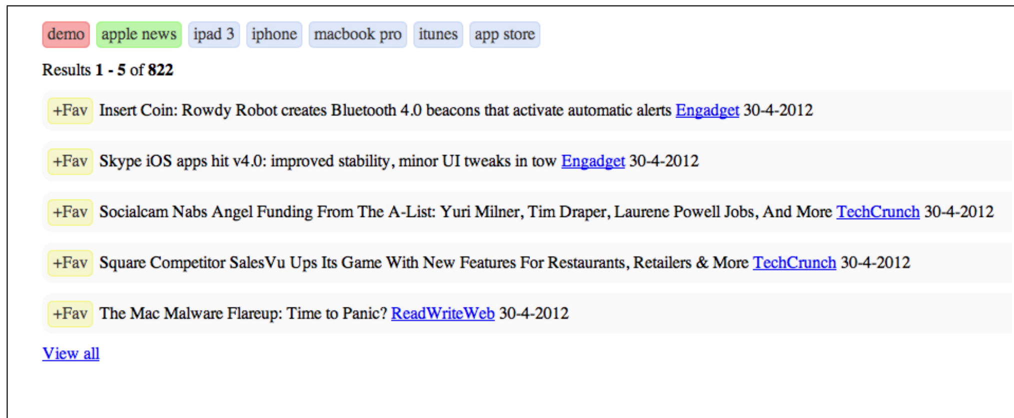

Figure 4: An example followed list.

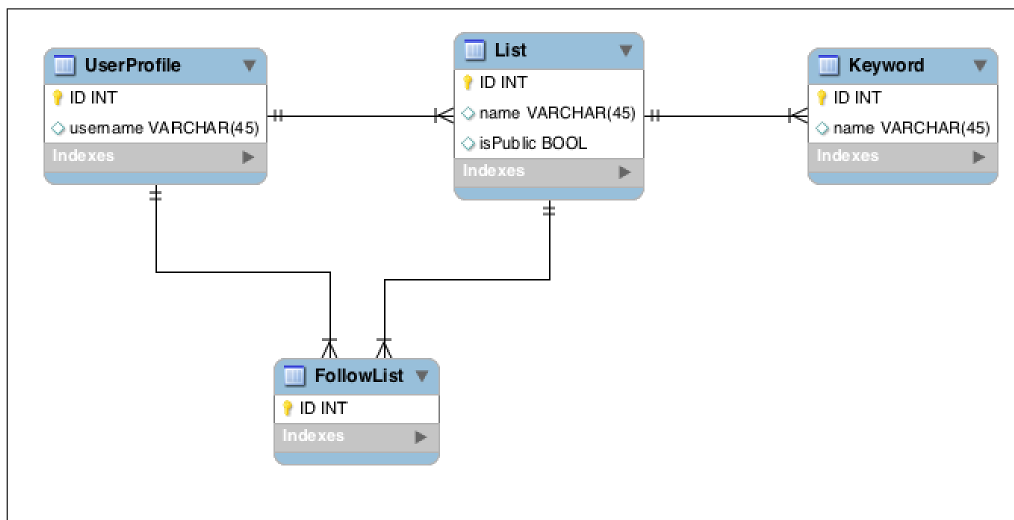

Figure 5: Database Architecture

Followers just have read permission. Additionally, if the owner changes the list's privacy setting from public to private, or if he deletes the it, then followers of the list won't be able to access it.

Figure 4 shows an example followed list, with its keywords and retrieved articles. It's very similar Figure 3, but now the owner of the followed list is emphasized in red.

5.3 Implementation

Figure 5 shows the Entity-Relationship Diagram of database tables corresponding to lists, keywords, follow lists and user profile. Some fields are ommitted for brevity.

Each user has a unique user profile, containing detailed information about the user such as username, email, languages etc. List table contains the field isPublic which can take a boolean value of True or False, indicating the list is public or private respectively. Foreign keys establish one-to-many relationships between tables (denoted by arrows). As we can see from the figure user profile is a foreign key in list, meaning each list belongs to a single user but a user can have many lists. Similarly, list is a foreign key in keyword, stating each keyword belongs to a list and a list can have multiple keywords associated with it.

Follow lists are also similar, containing foreign keys for user profile and list. User profile field is bounded to the current user, and the list field is associated with the list the current user is following

(which belongs to a different user). Note that we don't have to keep a reference to the followed user, who is the owner of the followed list, because the list already keeps that information. Foreign keys are configured as on-delete cascade, which means when a list is deleted, all its associated keywords and followers are also deleted. The database is Mysql.

6 News Retrieval

News results in each list are aggregated and presented to the user together. The user can first browse news about technology, then politics, and sports. This way the users are able filter and localize the information as they wish. So news stories are not presented as a blended mixture of various topics, the system clusters them based on user's own choice of lists and keywords. This section elaborates news retrieval and ranking process, which is the most fundamental part of the system.

6.1 Ranking News

The articles within a users keyword list are ranked based on that users activity on the site. Personalized ranking is performed by analyzing user logs, and creating a model to represent the user and the news articles. This model is then used to retrieve ranked news results for each list of the user.

6.1.1 User Logs

Two types of actions of a user are logged: article and bookmarks (also called favorites). While a user is browsing news at Meliper, she clicks on articles that she finds interesting. These clicks provide valuable information about a users interest, since they're a sign of relevance. Therefore, clicks on articles are logged as user-article pairs. Every time a user clicks on an article, the articles ID is added to her click log. Multiple clicks to the same article constitute as one entry.

Apart from clicks, a user can also bookmark (favorite) an article, similar to bookmarking a web page in a browser. This could mean that the user has read (clicked) and appreciated the article, so she wants to bookmark it. Or maybe she thinks that its interesting but doesnt have the time to read it now, and she saves it for later. Either way, this is a strong signal of relevance, possibly even more than clicks. The number of articles a user bookmarks is considerably less then the number she clicks. And we generally see that bookmarked articles are also clicked, supporting our first intuition about bookmarking implying user appreciation. Therefore, in the system bookmarks are considered more important than clicks (we will quantify it later). Similar to clicks, bookmarked articles are added to the users favorite logs.

6.1.2 Article Representation

Content of parsed articles are added two separate inverted indexes, term index and named entity index (tag index doesn't include article content). Therefore an article is indexed by both the terms and named entities it contains. During ranking we only focus on named entity index, ignoring the regular term index. Because named entities provide more specific information about an article. We try to capture what exactly the article is talking about from the named entities it contains. For example the existence of the proper nouns Microsoft, Bing, and Search API gives a clear context about the article. Verbs or regular nouns provide broader information, not as specific and focused as named entities. Therefore only named entities will be used to represent an article in ranking. From now on, to conform standard information retrieval notations, we will refer named entities in an article as terms, and articles will be called documents. But again note that terms we refer to are actually the named entities in the article, not the word tokens.

Documents (articles) are represented using tf-idf in vector-space model [5]. Term frequency of a term t in document d is the number of occurrences:

$$tf_{t,d} = n_{t,d} \tag{1}$$

where $n_{t,d}$ is the number of times term t appears in document d. T_d is the set of all terms in d, so the document frequency of a term in corpus is:

$$df_t = 1 + |d \in D : t \in T_d| \tag{2}$$

where D is the set of all documents (indexed news articles). Actual document frequency is incremented by 1 to avoid divide-by-zero error in inverse document frequency, which is calculated as follows:

$$idf_t = \frac{|D|}{df_t} = \frac{N}{df_t} \tag{3}$$

where N is the number of documents in the corpus. And the final tf-idf value of a term in a document is the product of term and inverse document frequencies:

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \tag{4}$$

Therefore, each document is represented by a vector of tf-idf values of terms it contains. Document vectors are normalized to have unit length.

6.1.3 User Representation

The users are also represented as normalized tf-idf vectors in the same vector-space as documents. We define C_u and B_u as the set documents (articles) the user clicked and bookmarked respectively. Note that a document can appear in both sets. Then we define the "user document" D_u as the collection of all documents the user clicked or bookmarked:

$$D_u = C_u \cup B_u \tag{5}$$

The user vector is then represented by the terms (named entities) in D_u. Term frequency of a term t in clicked and bookmarked documents is:

$$c_{t,u} = |d \in C_u : t \in T_d| \tag{6}$$

$$b_{t,u} = |d \in B_u : t \in T_d| \tag{7}$$

Using these frequencies, we define the term frequency of a term in user document as follows:

$$tf_{t,u} = \alpha \times c_{t,u} + \beta \times b_{t,u} \tag{8}$$

where the parameters α and β are the weights reflecting the importance of clicked and bookmarked documents respectively. Occurrence counts of terms are boosted by these weights. As mentioned before, terms that appear in bookmarked documents are considered more relevant than clicked ones, so bookmark weight β is higher than click weight α. Current values are $\alpha = 1$ and $\beta = 1.5$. An article in D_u can be in one of 3 states for the user: only clicked, only bookmarked, or both clicked and bookmarked. Term frequencies are boosted by 1, 1.5, and 2.5 in each case respectively. Which follows our intuition that if an article is both clicked and bookmarked, it's strongly relevant to user's interests. Finally, the tf-idf weight of a term for a user is defined as:

$$tf\text{-}idf_{t,u} = tf_{t,u} \times idf_t \tag{9}$$

So a user is represented similar to a document, with a normalized tf-idf vector in the same vector space. This enables us to use cosine similarity in ranking articles for a specific user (described in the next section).

Category	Website	No Articles
Technology	TechCrunch	10,000
	ReadWriteWeb	10,000
Economy	New York Times	10,000
	Washington Post	10,000
	Wall Street Journal	10,000
	CNN	10,000
Politics	New York Times	10,000
	Washington Post	10,000
	Wall Street Journal	10,000
	CNN	10,000

Table 1: News sources for categories.

6.1.4 Ranking

Ranking of news articles in a user's list is performed based on his interests. Personalized ranking is applied and as a result the ranking of news results are decided. Therefore the ordering of articles may not be the same for different users, even if they have the exact same keywords in their lists. Because user behavior such as clicking or bookmarking an article is logged, and these logs are used to create a specific vector representation for each user. Documents are similarly represented as vectors in the same vector space. Thus given a user vector $\vec{u}$ and a document vector $\vec{d}$, the similarity between these two vectors are calculated via cosine similarity. Since $\vec{u}$ and $\vec{d}$ are normalized tf-idf vectors (section 6.1.2 and 6.1.3), cosine similarity reduces to simple dot product of vectors:

$$sim(u, d) = \vec{u} \cdot \vec{d} \tag{10}$$

The cosine similarity value of two vectors ranges between 0 and 1, with 1 being perfectly similar and 0 implying strong dissimilarity. Meliper is at its core, search and ranking on news corpus. Given a list and its keywords, the system returns a ranked array of corresponding articles as follows:

1. For each keyword in the list, get the articles that keyword appears in from the inverted index. Filter out the articles that are more than a week old to present only fresh news to the user.
2. Now we have an array of articles for each keyword. Merge them together via conjunction. The result is a single article array where each article contains at least one of the keywords.
3. Get the vector representation of the articles in the array as described in Section 6.1.2.
4. Form the user vector as explained in Section 6.1.3.
5. Calculate the cosine similarity between every article and the current user vector. Sort the articles by decreasing similarity and return the result.

We perform the above operations for each list of the user one by one. And we get a ranked collection of articles for every list. These articles are then presented to the user within their corresponding list as shown in Figure 3.

7 Recommendation Systems

To make the system more clever and user-friendly, several recommendation systems are implemented using Machine Learning techniques. Such as keyword, user, and list recommendations, as well as mining Twitter data. In this section, we explain each recommendation system in detail.

The models described below are trained on a corpus of 100,000 news articles parsed from 3 categories: technology, business, and politics. The websites used for each category is in Table 1.

7.1 Keyword Recommendation

We want the system to recommend new keywords to the users, based on the current keywords they have in their lists. This way, Meliper becomes more user-friendly, because now the users don't have to manually input all the keywords that belong to an interest area. The system automatically understands users intent and suggests new similar keywords. The user can either follow the suggestions or ignore them. If they choose to aggree with suggested similar keywords, then they're added to the corresponding lists. As an example, if a user has the keyword android in a list, then it would be logical to recommend him similar keywords like iphone, ipad, ios to add that list.

7.1.1 Co-occurrence Count

The first method to find similar words of a given word is using co-occurrence counts. For each word, its co-occurrence counts with other words are computed. Then the most similar words to a given word is the ones with largest co-occurrence counts. Co-occurrence counts of words pairs are computed as follows. For each news article in the corpus, the following steps are performed:

1. Get the named entities from the article (Section 3.1). The article is represented by an array of named entities.
2. Get all pairs of named entities from the array. For an article with n named entities, this will result in $\binom{n}{2}$ pairs.
3. Increment the co-occurrence count of each pair.

After this procedure, we obtain a data structure (cooc-count) that contains co-occurrence counts of every named entity pair. Cooc-count is a nested hashtable of hastables, with keys being the named entities. First level keys are all the named entities in the corpus. Second level keys indexed by a first level key (named entity) are all the named entities that co-occur with that first level key. The value of combined first and second level key pair is the co-occurrence count of that named entity pair. Therefore given a word and cooc-count data structure, we obtain the most similar k words as follows:

1. Get all the words that co-occurred with the given word from cooc-count. These are all candidate similar words.
2. To get top k most similar words, sort the candidate words by decreasing co-occurrence counts, and return top k (or possibly less) most co-occurring words.

Now that we have a procedure to find most similar words to a given word, the main goal of recommending k new keywords to a user can be done using these procedures. Keywords are recommended to the user on a per-list basis. So for each list of the user, the following steps are performed:

1. Get the keywords in the given list of the user.
2. Get the most similar k keywords for each keyword in the list, together with their co-occurrence counts. Therefore, for each keyword in the list we have an array of similar keywords with counts.
3. Merge all similar keyword arrays. If a similar keyword has more than one co-occurrence count because it is associated with more than one keyword, get the largest count value.
4. Sort the similar keywords by decreasing co-occurrence counts, and return the top k most similar keywords.

We run the above algorithm on every list of the user, and we get an array of k new similar keywords for each list. These new keywords are then suggested to the user, and the is given the choice of following the recommendation or not. He can either add all recommended keywords, or he can selectively add some of them he finds relevant. The system tries to optimize precision rather than recall, so the k value is chosen to be a relatively small number such as 5. Table 2 shows some words together with their corresponding similar words.

apple	iphone, ipad, android, mac, google, steve jobs, ios
obama	white house, congress, caucus, democrats, republicans
romney	republican, gop, massachusetts, santorum, gingrich
stocks	wall street, europe, greece, bonds, federal reserve
fed	federal reserve, ben bernanke, stocks, banks, economy
unemployment	labor department, benefits, americans, obama, job

Table 2: Example of some words and similar words.

7.1.2 Pointwise Mutual Information

An alternative to Co-occurrence Counts appraoch is Pointwise Mutual Information (PMI). PMI between two words is calculated using co-occurrence counts as follows [1]

$$PMI(u,w) = log\frac{\frac{c_{uw}}{N}}{\frac{\sum_{i=1}^{n} c_{iw}}{N} \times \frac{\sum_{j=1}^{n} c_{uj}}{N}} \tag{11}$$

Where n is the number of unique words (named entities) in the corpus and N is the total count of words. PMI can take positive or negative values, where large positive and negative values asserting strong similarity and dissimilarity respectively. But if it's zero, then two words are independent.

New keywords are recommended using PMI very similar to co-occurrence counts. Instead of counts we now use PMI as similarity metric, so there's only one extra step performed.

7.2 List Recommendation

In addition to recommending new keywords to a user's lists, we also recommend user new lists to follow, based on his current lists and keywords. Recommended lists are public lists of other users, and they are more detailed than the current list of the user. It's the user's choice to follow the recommended lists.

For each list of the user, we recommend him $n = 3$ new lists to follow. Recommended lists are the ones that are most similar. The following operations are performed for every list of the user:

1. Get the keywords of the current list.
2. Get $k = 5$ most similar keywords to each keyword in the list (Sections 7.1.1 and 7.1.2), and merge them to form a vector of keywords.
3. Set the value of keywords in the list vector as their corresponding idf value. Normalize the list vector to have unit length.
4. Find all public lists of other users which contain at least one keyword from the list vector. These are the candidate similar lists.
5. Construct vectors of candidate lists as in step 3 to form candidate list vectors.
6. Find cosine similarity between the list vector and every candidate vector. Sort candidate vectors by decreasing cosine similarity values and return the top 3 most similar vectors.

The procedure is similar to ranking news articles in Section 6.1.4. Lists are represented as unit length vectors in the same vector space, and cosine similarity value is used as degree of similarity. These operations are performed on every list of the user, and for each list 3 similar public lists of other users are recommended. The parameters n and k can be modified. Increasing n results in more recommendations to the user, and k value effects how specific or general the recommendations are, where increasing k produces more general lists.

7.3 User Recommendation

The system can also recommend other a user other users who are similar. This way user a user can discover users with similar interests, and potentially follow their lists. This feature improves the social factor in the system and can lead to networks of users.

As we have seen in Section 6.1.3, the users are represented as tf-idf vectors in vector-space model. This leads to natural use of cosine similarity to calculate similarity between users. Therefore, similarity between two users u and v is defined as:

$$sim(u, v) = tf\text{-}idf_u \cdot tf\text{-}idf_v \tag{12}$$

This representation of the user is based on the articles he clicked and bookmarked. If he neither clicked nor bookmarked any articles, then this approach won't work.

We can represent the user in an alternative way based on all his keywords. Instead of using the named entities in the articles he clicked or bookmarked as features, his keywords become the features in the vector-space. We get all keywords from every list he creates or follows, and create a tf-idf vector from it. The process is very similar to Section 6.1.2. Now documents are lists (instead of articles) and terms keywords in the lists (rather than named entities in articles). Let L be all the lists in the system, L_u be the lists that the user u owns or follows. Term frequency of a term t for user u is defined as:

$$tf_{t,u} = |l \in L_u : t \in l| \tag{13}$$

Which is the number of lists the term appears in. The document frequency of a term t is defined as the number of lists it appears in + 1 (to avoid divide-by-zero error):

$$df_t = 1 + |l \in L : t \in l| \tag{14}$$

Then inverse document frequency of a term becomes:

$$idf_t = \frac{|L|}{df_t} \tag{15}$$

Finally, the weight of a term t in tf-idf representation of the user u is calculated as:

$$tf\text{-}idf_{t,u} = tf_{t,u} \times idf_t \tag{16}$$

The user is then represented as tf-idf values of the keywords in all lists he owns or follows, in the vector-space of keywords in the system. User vector is normalized to enable to use of cosine similarity. As we can notice, the formulas are almost the same as Equations 1-4. Because only the source of the information has changed, the methodology is the same.

After denoting each user as normalized keyword vectors, the similarity of two users is the same as Equation 12. In both representations, we calculate the similarity of current user with every other user. And the most similar 5 users are recommended.

8 Future Work

Meliper project is open to further improvements in many areas. More machine learning techniques can be added to make the system more intelligent. One particular example that we're working on right now is that instead of making users create lists by adding keywords manually, we try to mine user's interests from his Twitter account, if he grants us permission. The system then crawles and parses user's tweets, and tries to automatically extract keywords. Then similar keywords are clustered together to form lists, and these lists are then recommended to the user. If the user likes the list he imports it and becomes the owner of the list. Several clustering techniques are being experimented, and we get promising results. This feature is expected to be completed very soon.

Another improvement is increasing the number of news sources, and covering various categories from multiple perspectives. Adding more websites to the system also diversifies the content, and increases user engagement. This is relatively simple to accomplish since crawling and parsing routines are generic and website-independent. As long as the website is has an RSS feeds, it can easily

be indexed. The feature is delayed on purpose to keep the system simple and easy to test. Also multiple language support would engage more users spanning different countries.

Websites that have a social side improve as the user base increases. Especially recommendation systems work better with more data. Meliper will advance further as users register and actively provide data.

Acknowledgments

We would like to thank Melina Sabunci for valuable suggestions and useful feedbacks. Also ”Meli” part of Meliper is inspired from her name.

References

[1] Yejin Choi, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, Mauricio Mediano, and Bo Pang. Using landing pages for sponsored search ad selection. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 251–260, New York, NY, USA, 2010. ACM.

[2] Arden Dertat. Meliper, 2012. `http://meliper.com`.

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[4] Martin F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[5] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.