# Leveraging Noisy Lists for Social Feed Ranking

**Matthew Burgess, Alessandra Mazzia, Eytan Adar, Michael Cafarella**
University of Michigan, Ann Arbor
{mattburg, amazzia, eadar, michjc} @umich.edu

## Abstract

Active users of social networks are subjected to extreme information overload, as they tend to follow hundreds (or even thousands of other users). Aggregated social feeds on sites like Twitter are insufficient, showing superfluous content and not allowing users to separate their topics of interest or place a priority on the content being pushed to them by their "friends." The major social network platforms have begun to implement various features to help users organize their feeds, but these solutions require significant human effort to function properly. In practice, the burden is so high that most users do not adopt these features. We propose a system that seeks to help users find more relevant content on their feeds, but does not require explicit user input. Our system, BUTTERWORTH, automatically generates a set of "rankers" by identifying sub-communities of the user's social network and the common content they produce. These rankers are presented using human-readable keywords and allow users to rank their feed by specific topics. We achieve an average top-10 precision of 78%, as compared to a baseline of 45%, for automatically generated topics.

## Introduction

Over the past few years, online social networks have shifted focus from socialization platforms to information hubs. Users log onto social networks to obtain the latest updates from their personal friends as well as commentary, links, and news from colleagues and subject-matter experts. The conventional mechanism for displaying this information is the *social feed*, a long (usually) time-ordered list of updates. Active users of social networking sites like Twitter can follow the updates of hundreds or thousands of other users. This can produce feeds that contain hundreds of new items per hour, many of which are irrelevant to the user. As social networks snowball in popularity and friend networks grow in size, users become increasingly less able to find relevant content as both context and channels collapse.

Users' feeds often contain a heterogeneous mixture of information from many sources and on a variety of topics. This heterogeneity is caused by *context collapse*, the phenomenon in which many people, usually situated in different contexts, are suddenly grouped together. For instance, a user Alice may be primarily interested in data mining, but also interested in cooking and literature. She follows topic-matter experts on all these subjects, but because the primary consumption mechanism is a single social feed, all messages will become intermingled regardless of what they are about.

The converse of the context collapse problem is *channel collapse*. While context collapse describes the consumption of information—all the people a user follows push their updates into one place—channel collapse describes the production of content. A particular user, Jane, may be followed by others for many reasons: her family for pictures of the grandchildren, gamer buddies for the latest news on new releases, and work colleagues for Java coding tips. However, because Jane has only one output mechanism available to her, any content she creates—be it pictures of her children or Java tips—will be seen by all subscribers to her feed, whether they are interested in the topic or not. Alice, who follows Jane only because of their shared interest in games, will be forced to sift through irrelevant Java tips and family pictures. Conversely, Jane's father must get through coding and gaming messages before getting to the pictures.

In response to the problems of context and channel collapse, social networking sites have implemented features that allow users to group their friends into *lists*. Lists let users better organize their feeds by grouping friends who share a similar context; users can then access sub-feeds that only contain content from friends in that grouping. In many situations these lists are intended to aid in consumption, but recent trends in interfaces for these systems also allow lists to be used for targeting produced content. Allowing users to target their posts based on the topic of the content has the potential to alleviate channel collapse. While such features can help users better organize their feeds, they require significant human effort, as well as widespread adoption, in order to be effective.

In this paper we propose BUTTERWORTH, a system that provides a solution to both context collapse and channel collapse. We observed in pilot experiments that when users create lists, they are also implicitly indicating which content produced by members of that list they are interested in. This is most often the topics discussed in common by all members of that list. For example, when creating a *visualization* list, a user includes other users who post about visualization. Each user in the list may post about whatever other topic she

also finds interesting, but a common topic across all of their posts will likely be visualization. By finding this common core, it becomes easy to train a ranking algorithm to highlight posts related to visualization. Unfortunately, most users do not create lists.

To address this issue, we designed BUTTERWORTH to leverage manually created lists when available, but also to automatically identify people who should be placed together in a list. Unlike topic-modeling approaches, which are highly sensitive to input, computationally complex, require substantial tuning, and are difficult to present to users, BUTTERWORTH's novel approach leverages the inherent homophily of social networks. By mining a specially weighted version of a user's social network, BUTTERWORTH can scalably build lists that closely simulate manually created lists, down to picking human readable list labels. For each of these topics BUTTERWORTH generates a "ranker" that re-orders the user's feed by the relevance of the items to the selected topic. BUTTERWORTH leverages only a user's social network and content produced by their friends; it therefore requires no direct supervision from the user.

We have tested each component of BUTTERWORTH and have shown that it is able to achieve high levels of precision and recall, on both topic discovery and ranking. We have designed BUTTERWORTH to be broadly applicable to all mainstream social network platforms, but have chosen Twitter as our platform due to its popularity and public API.

## Related Work

There has been a substantial amount of recent research on managing social media data. However, this work has either emphasized text-only approaches (e.g., better ranking and topic modeling) or network-only strategies (e.g., community detection). BUTTERWORTH seeks to leverage both approaches to create a more robust technique that can be readily integrated into user interfaces in a manner consistent with users' expectations.

The bulk of work in social media ranking has focused on classifying posts in a user's feed (Dahimene, Mouza, and Scholl 2012; Das Sarma et al. 2010; Hong et al. 2012; Paek et al. 2010; Pal and Counts 2011; Sriram et al. 2010; Uysal and Croft 2011). Both Hong *et al.* (Hong et al. 2012) and Das Sarma *et al.* (Das Sarma et al. 2010) proposed ranking mechanisms based on collaborative filtering techniques. Hong *et al.* used a click-through rate based model while Das Sarma *et al.* compared various mechanisms based on different types of user supervision, including having the user provide pair-wise comparisons of feed items. Paek *et al.* and Dahimene *et al.* (Paek et al. 2010; Dahimene, Mouza, and Scholl 2012) built ranking and filtering methods that are personalized for each user and which explicitly model the user that is producing the tweet. Uysal *et al.* (Uysal and Croft 2011) describe methods to predict the likelihood of retweets (a user propagating a tweet to friends), and used the likelihood of retweet as the ranking score. Pal *et al.*'s algorithm for finding relevant Twitter users (Pal and Counts 2011) is an unusual point in the space as it attempts to identify topic experts using a classifier that leverages features such as follower counts and retweeted content. These approaches address both content and context collapse by learning user preferences and hiding irrelevant content. However, these models do not necessarily aid a user in organizing her information–an explicit goal of BUTTERWORTH. Additionally, while effective with enough training data, such systems often require a great deal of training data to be useful. One of the goals for BUTTERWORTH was not to require direct input from the user, but instead leverage the inherent homophily in a user's social network to provide high quality groups, and subsequently, rankings.

A different approach to data management focuses on organizing and classifying rather than ranking. If content is clustered for a user based on topic, she may identify and select the information that is relevant to her. Topic modeling based methods (both on users and content) feature prominently in this space (Hong and Davison 2010; Lin, Snow, and Morgan 2011; Ramage, Dumais, and Liebling 2010; Xu et al. 2011). Inspired by these ideas, an earlier version of BUTTERWORTH attempted to build Latent Dirichlet Allocation (LDA) topic models. While useful for describing lists, we encountered many common problems: they were computationally costly, highly sensitive to noise, and required too much tuning to work effectively across a broad spectrum of users. Further, providing interpretable descriptions for topic models is a notoriously difficult problem, and even "optimal" models may not be consistent with reader preferences (Boyd-Graber et al. 2009). Prior work has demonstrated that unlike topic models, the names of lists in which a particular Twitter user appears are much better representations of the expertise of that user (Wagner et al. 2012). Because of these common problems, we chose to move away from topic modeling.

In addition to the topic-modeling work for analyzing the individuals a user may follow, there has been a small amount of work on explicitly managing lists on social networks. Kim *et al.* (Kim et al. 2010) conducted an analysis of lists on Twitter and concluded that topic-centric lists contain users who tend to publish content on similar topics, reinforcing the motivation for our distant supervision heuristics. Another analysis by Fang *et al.* (Fang, Fabrikant, and LeFevre 2012) studied sharing "circles" on Google+, and discovered that circle sharing enabled users with few contacts to grow their networks more quickly. In this arena, our work is most similar to that of Guc (Guc 2010), which describes a filtering mechanism for "list-feeds"; however, that work is framed as a standard supervised learning problem that also requires explicit training data from the user.

Finally, a number of research systems have focused on creating interfaces that enable users to more efficiently browse their feed (Bernstein et al. 2010; Hong et al. 2010; Tseng, Chen, and Chen 2012). For example, *Eddi* (Bernstein et al. 2010) displays a browsable tag cloud of all the topics in a user's feed, allowing the user to more easily find tweets related to her interests. *FeedWinnower* (Hong et al. 2010) is an interface that allows users to rank tweets by different tunable parameters such as time and topic. Tseng *et al.* proposed a (graph) visualization system called *SocFeedViewer* (Tseng, Chen, and Chen 2012), that allows users to analyze a topological view of their social

graph. SocFeedViewer implements community detection algorithms to group similar users; as with BUTTERWORTH's lists, these community feeds may contain irrelevant content, a problem that SocFeedViewer does not address. Many of these systems apply one or more of the ranking and classification techniques described above. This, unfortunately yields lower precision and recall. The higher quality grouping and ranking strategy of BUTTERWORTH can be readily integrated into various commercial and research interfaces and directly improve their function.

## Problem Overview

Since BUTTERWORTH is focused on an end-user burdened by irrelevant feed content, we present user scenarios that describe today's current situation as well as the feed interaction enabled by BUTTERWORTH. We then give an overview of BUTTERWORTH's major components.

### User Scenario

To see how users struggle with today's social feeds, recall our example user Alice. Alice has joined Twitter to find information about data mining, literature, and cooking. She follows many users who generate content on her main interests, but she also follows other users for various reasons (*e.g.*, they are co-workers, family members, *etc*.). While sometimes Alice enjoys browsing her feed to see the latest content from her friends, at other times she wants to see only content related to her main topics of interest. With the traditional feed mechanism, Alice suffers from *context collapse*; if she wants to see only content related to cooking, she must manually search through her feed to find cooking tweets. Since Alice is interested in cooking, she could devote some time to using Twitter's list feature to group together all of her friends who write about cooking. When Alice clicks on her cooking *list* she is shown a sub-feed consisting of only content from the users in the cooking list. Sadly, this does not help Alice very much, since she realizes that the friends she follows for cooking information also write about many other topics. Alice's lists allow her to break up her feed into contextualized groups, but this is not enough, as channel collapse fills even her list-specific feeds full of irrelevant content.

Alice's experience is very different when she uses BUTTERWORTH. Once Alice logs into Twitter, BUTTERWORTH automatically identifies her topics of interest and presents them to her. If Alice has already created a list, such as her cooking list, then BUTTERWORTH can incorporate it into the topics presented (though manual list creation is not required). Alice is then able to choose a topic that interests her; when she does, messages in her feed are ranked by their relevance to the topic selected. Alice's feed is no longer affected by context collapse, since BUTTERWORTH automatically generates lists that correspond to her interests. In addition, her feed is no longer affected by channel collapse, since BUTTERWORTH pushes the relevant content to the top of her feed.

### System Architecture

BUTTERWORTH comprises three main components, as seen in Figure 1. The **list generator** partitions friends into lists by analyzing their social network. These user lists are then fed into the **list labeler** that generates a concise label representing the list's (central) topic. The generated lists are then sent to the **topic ranker**, which trains ranking models for this core topic. The models can then be used to rank the user's feed by the selected topic.

**List Generator** The first step in the BUTTERWORTH pipeline is to automatically discover groups of users that discuss similar topics within a user's *ego network*, and to partition the members of this network into lists. The ego-network is a subset of the social network (in this case, a modified and weighted follower/followee graph). It is derived from the friends connected to a core "ego" individual and the links among these friends, excluding any edges to the ego node. The goal of this module is to generate lists (we call these *topical lists*) such that each list corresponds to one of the user's topics of interest. Even if a user's main objective on Twitter is to obtain information on topics of interest, they likely follow users for a variety of other reasons. For instance, a user may follow their co-workers or family members. The generated lists for a user may therefore also include these *contextual lists* that are less topically coherent (e.g., lists that contain users who all live in the same town or are all family members). After computing these topical and contextual lists, the **list generator** filters out all contextual lists (as described in the Graph Clustering section).

For an example we turn back to Alice. The list generator takes as input Alice's ego network and partitions all of her friends into lists. Some of these lists will correspond to her interests like cooking and data mining, and others will contain her college friends, co-workers, *etc*. The **list generator** then removes the contextual lists, so that only the topical lists remain.

While generating high-quality lists is an important problem, it is not BUTTERWORTH's end goal; we formulate these lists in order to produce high quality topic rankers. While low-quality lists can have a bad effect on downstream ranking performance, we show experimentally that perfect lists are not required for high-quality ranking. The ideal output of the **list generator** is a set of lists that comes close to the quality of lists that users generate themselves, which is how we evaluate this component in the experiments.

**List Labeler** BUTTERWORTH's second component is the **list labeler**. It generates relevant and human-understandable labels for each of the lists generated in the previous step. These labels are also attached to the generated rankers and are intended for display to the user in the system interface, enabling her to select which topic ranking she wants applied. For example, the list labeler should name Alice's cooking list as "cooking" or something similar. While it is, of course, possible to formulate lists without generating accompanying labels, the resulting interface would be substantially harder to use—Alice would need to resort to clicking on all the BUTTERWORTH-given topics and reading sets of tweets to discover the topic behind each one.

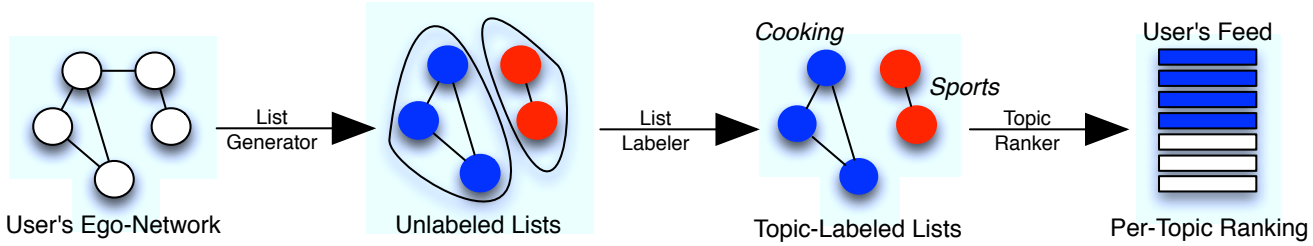


Figure 1: BUTTERWORTH has three components. The **list generator** groups a user's social contacts into topically coherent lists. The **list labeler** synthesizes labels for those lists. The **topic ranker** learns a ranking model for each topic by using each list to heuristically label training data.

We propose two different labeling algorithms, using network as well as textual features (including list labels generated by others, content, and user biographies). An ideal output for the list labeler is a topic string that is semantically close to the label a human would give to the same list. We evaluate the list labeler's output by comparing it to some simple synthetic baselines, as well as asking human judges to rate its relevance.

**Topic Ranker** The final component of BUTTERWORTH is generating a ranking model for each identified topic. The **topic ranker** takes as input the past tweets from the set of users in each list that was labeled in the previous step. It proceeds in two steps. First, it generates a label ("relevant" or "irrelevant") for a subset of the user's tweets. Second, it uses this synthetically generated labeled data to train a naïve Bayes model. Once a ranking model is trained for each topic, the model can be applied to rank a user's feed by the corresponding topic. For example, after Alice selects the cooking topic, her *entire* feed would be re-ordered such that the cooking tweets are pushed to the top of the list. Note that we purposely rank the entire feed to deal with both false-positive and false-negative assignment of users in the list generation.

In order to train each ranker, we propose various heuristics to automatically generate "relevant" and "irrelevant" labels for training examples—a form of distant or self-supervised learning. The main intuition behind these heuristics is that since the users contained in each list discuss similar topic(s), the most frequently discussed topics are most likely to be relevant. An ideal output of the ranker would order a user's feed such that all of the content relevant to the list appears at the top of their feed. We use standard information retrieval techniques to evaluate the ranking quality.

## Algorithms

Here we describe how we implement each of the components above: the list generator, the labeler, and topic ranker.

### List Generation

The goal of list generation is to take a user's ego-network and partition the friends into topically coherent lists. Fortunately, partitioning the nodes of a graph into subsets is one of the topics addressed by graph clustering research. In particular, the algorithm of Pons and Latapy (Pons and Latapy 2005) takes as input an undirected weighted graph, then produces a disjoint set of node sets. An edge weight in the input network generically describes the strength of the link between two nodes. In principle, we can take the (unweighted, directed) ego-network to create a (weighted, undirected) input to the clustering algorithm, and then call each of the algorithm's clusters a *list*.

Of course, the quality of clustering depends largely on the network we formulate as input. By choosing edge weights differently, we can obtain different kinds of clusterings. We have found that using weights that take into account shared topics leads to groupings that emphasize these topics while de-emphasizing *contextual* relationships (family, childhood friends, *etc*.) that are, for our purposes, "spurious."

Each ego-network consists of a set of nodes, $N$, and a set of directed edges, $E$, where each edge indicates the presence of a follower-followee relationship. For simplicity, we first convert each directed edge in $E$ to an undirected edge. Then, we use a combination of network features and textual features to produce $E'$, the set of weighted edges.

For each pair of nodes $(u, v)$, where $u \in N$ and $v \in N$, we produce a weighted, undirected edge $e_{u,v}$ using a modification of the user similarity measure presented in (Adamic and Adar 2003):

$$sim(u,v) = \sum_{x \in sharedneighbors} \frac{1}{log[degree(x)]} + \sum_{t \in sharedterm} tfidf(u_t) * tfidf(v_t) + E(u,v), \quad (1)$$

where $E(u,v) = 1$ when $e_{u,v} \in E$, and 0 otherwise. If $sim(u,v) > 0$, then $e_{u,v}$ is added to $E'$ with weight $sim(u,v)$. To produce the "shared term" weights, we first produce a TF-IDF-weighted term vector for each node $n \in N$, limiting the per-user vocabulary to the 10 top-scoring words. In examining the networks produced by our edge weighting algorithms, we realized that many networks appeared extremely dense. To correct for this, we discard all edges with weight less than a threshold parameter $\alpha$.

**Graph Clustering** After we have produced a weighted, undirected graph, we perform graph clustering to obtain a set of lists. Currently, we apply a common approach based on

Table 1: The top three ranked tweets for automatically generated lists labeled *environment*, *fashion*, and *sports* (list labels are also automatically generated).

| | |
|---|---|
| **Environment** | Join us in helping get the 2012 Olympics off plastic bags!... |
| | Heading to Patagonia Santa Monica store to bring 5 Gyres display on plastic pollution.. |
| | Starbucks Trash: Behind the Scenes :: My Plastic-free Life — Less Plastic... |
| **Fashion** | Anchors away in this Vintage Yellow Sailor dress! Newly listed... |
| | STUNNING Vintage Paisley Teal Spring Dress!... |
| | Vintage Silk Heart Blouse with a Tie Neck listed... |
| **Sports** | I posted 12 photos on Facebook in the album "Warrior Around the NHL... |
| | LeBron James and Michael Jordan (92): are thus the only players to win NBA title... |
| | Denver Nuggets Sign Quincy Miller... |

random walks (Pons and Latapy 2005), as implemented in the iGraph R package. The clustering algorithm uses properties of random graph walks to produce a disjoint set of communities, and we consider each community produced as a *list*.

After identifying the lists, we must decide whether each list and its members are topically cohesive. To do this, we compute the *entropy* (Shannon 2001) of the list, in which the list's probability distribution is defined as a bag-of-words model over all tweets in the list. Our intuition behind using entropy is that if a list's tweets share common topics, then the distribution over words will be skewed to a small subset of the whole vocabulary and thus have low entropy. We classify all lists with entropy lower than a empirically learned $\epsilon$ value as *topical*.

## Topic Labeling

When generating a label for a topic list, we use one of two algorithms: the BESTOVERLAP method, or the USERINFOBIGRAM method. This specific choice was the result of experimenting with a variety of other techniques described in the Experiments section.

In the BESTOVERLAP method, we exploit Twitter's large user base to implicitly "crowdsource" topic list labeling. For each topical list, we create a set of candidate names by looking at all previously-created Twitter lists that contain members of our newly generated list. To label our new list, we simply pick the name of the previously created list that maximally overlaps with the individuals in our list. For example, if our list contained users A, B, and C and we find two other previously created lists—"foodies," which contains A and B and "cooking," which contains A, B, C, and D—we label our new list "cooking." If multiple Twitter lists overlap equally, up to three names are concatenated to form the list label.

For the USERINFOBIGRAM method, we examine the optional text that Twitter users can enter to describe themselves (e.g. "I love NYC, tech & funk" or "CS PhD student at the University of Rochester... My research involves real-time crowdsourcing, human computation, and AI"). For each newly created list we generate a "corpus" of all user information fields of list members. The list is then labeled with the most frequent bigram in this synthetic corpus.

We found through experimentation that the BESTOVERLAP method works best when our generated lists have more members (increasing the chance of overlap and the size of overlap with previously created lists). After considering various limits, we use a minimum threshold of 10 list members to decide when to switch from BESTOVERLAP to USERINFOBIGRAM.

## Topic Ranking

The last step in the BUTTERWORTH pipeline is to build ranking models for each topic. The input for each ranking model is the past tweets of each member of the corresponding list, comprising a set of unlabeled tweets $T$. Of course, these tweets are not labeled as "relevant" or "irrelevant." However, we can exploit the fact that all of the tweets come from users grouped together in a single list. We propose various heuristics for automatically labeling a subset of the examples in $T$. Our learning scenario falls into the category of distant supervision, in which a heuristic labeling function $H$ is applied over all of the examples in $T$ to produce a labeled set $T_{\mathcal{L}} \subseteq T$. We only use $H$ to try to infer positive training examples—obtaining high-quality negative examples is much easier. For each list, we randomly sample $|T_{\mathcal{L}}|$ from the corpus of tweets not produced by users in the list. After obtaining a labeled training set, we then extract the bag-of-words features from each training example and train a naïve Bayes model for each list. The final feed ranking is produced by sorting the tweets by their relevance probability, as scored by the trained model for each list. Below we propose three different variants of the labeling function $H$.

**Naïve Method** In the naïve method we use all of the tweets produced by the list members as our positive training set, and sample from out-of-list tweets for our negative training set. More formally, we let $H$ be a constant function, labeling all of the examples in $T$ as positive, to create $T_{\mathcal{L}} = T$. While this method is very simple, it achieves surprisingly good results, as our experiments show.

**Top-K Hashtags and Unigrams** By labeling all of the examples in $T$ as positive, the naïve method creates a noisy training dataset. Instead, we would like a heuristic that tries to label as positive only the examples that are surely positive. Here we propose heuristics that try to leverage the observation that the more prevalent content in $T$ likely corresponds to the more relevant content to the end user. Instead of labeling all of the examples in $T$ as positive, we label a smaller

subset that contains examples that are the most likely to belong to the positive class. We use a modified TF-IDF score to find the unigrams and hashtags[1] that are most likely to be contained in the positive class. We score each hashtag or unigram occurring in $T$ using the following TF-IDF scoring function for a given list $l$:

$$TF * IDF(w) = F_l(x) \times \log\left(\frac{N}{IDF(x)}\right), \qquad (2)$$

where $F_l(x)$ is the number of occurrences of the hashtag or unigram $w$ in the list $l$, $N$ is the number of tweets in the corpus and $IDF(w)$ is the number of tweets that contain $w$. We define two heuristic functions, one for unigrams and one for hashtags. $H$ labels all of the examples in $T$ that contain at least one of the top-$k$ unigrams or hashtags, where $k$ is a chosen parameter.

## Experiments

Below we describe a set of four experiments to test various aspects of BUTTERWORTH. All of our experiments use data collected from Twitter which we describe below.

### Dataset Description

In order to evaluate BUTTERWORTH, we need a ground-truth dataset containing real users who have manually created their own lists. If our algorithmic list-generation component can accurately recreate these lists, and if we can use the resulting lists to formulate accurate topic labels and rankings, then we can be confident that BUTTERWORTH will be effective in a general deployment.

Unfortunately, it is not straightforward to find a random sample of users that has created high quality lists. We obtained such a sample from Twitter as follows:

1. We chose by hand a set of 10 high-quality lists drawn from www.listorius.com. They cover a range of topics, including computer science, cooking, and others. Each list has between 300 and 500 users. Together, these users formed a large seed set of users who are likely to be members of many lists.
2. We then obtained all lists that contained *any* of the seed users. This formed a large set of lists likely to be of high quality.
3. We found the creator of each list, yielding a set of nearly 400,000 users. We have now found a subset of Twitter users who have created high-quality lists — this is exactly the subset from which we want to draw our test sample.
4. Last, we randomly sampled 100 users from the follower set, and removed users who were using private accounts or who were non-English speakers. The resulting 80 users formed the *test-user* set. These users had created anywhere from 1 to 20 lists (mean of 7.79 and median of 7 lists). Figure 2 shows the final distribution of lists created by these test users.

[1]Hashtags are user-specified single-word descriptions that signal the topic of a tweet and always begin with a "#", i.e., #Food or #Climate

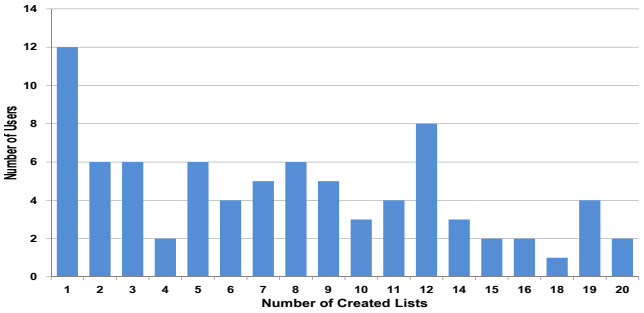


Figure 2: Distribution of the number of lists created by users in the test set (20 is the maximum allowed by Twitter)

Each of the obtained lists, which we call *organic lists*, has a user-given name (i.e., label) and a set of list members. Finally, we created an ego-network for each test user that comprised all of the test user's followers/followees, all members of a test user's organic lists, and any friend relationships among these users. The average ego-network size is 1383.72 and the median is 805. For our experiments we downloaded up to 1,000 tweets from each user in the resulting network.

### List Generation

To evaluate our list-generation algorithms, we experimented with several alternative edge-weighting schemes. For each ego user in our dataset, we produced one weighted, undirected ego-network graph for each weighting scheme. Next, we ran the clustering algorithm on each graph, producing a set of disjoint lists. Because all users in our dataset had at least one organic list, we evaluated the sets of lists produced by the community detection algorithm against the user's manually created organic lists. We evaluated the following weighting schemes: network only, in which the edge-weighting function $sim(u, v)$ considers only the neighbors as features, and three additional schemes in which the edge-weighting function $sim(u, v)$ considers neighbors and a TFIDF-weighted term vector as features, limiting the per-user vocabulary to $k$ words, where $k \in \{10, 100, 1000\}$.

We also tested the edge-weight parameter $\alpha$ with values that were functions of the average edge weight and standard deviation for each network: $\alpha \in \{1$, average edge weight - standard deviation, average edge weight, average edge weight + standard deviation$\}$. Figure 3 shows the results of this experiment. We see that taking $k = 0$ and $\alpha = ($ average edge weight$-$standard deviation) results in the highest value of F-measure, 0.83. Thus, we chose this algorithm as our edge-weighting scheme.

Our goal in this list-generation experiment was to produce lists that mimicked each user's manually created organic lists. However, community finding was performed on each user's entire ego-network, which included friends that were not placed on any organic lists. Therefore, in evaluating the machine produced lists against the manually created organic lists, we created a confusion matrix for each user as follows: for each pair of friends that was placed in any organic list $(f1, f2)$, the pair is a true positive if the

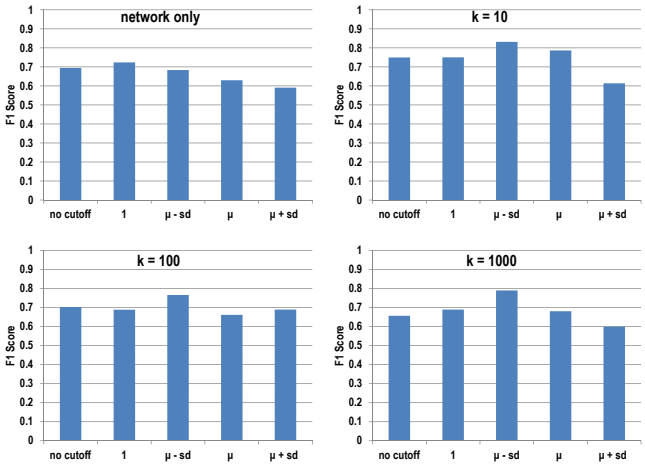


Figure 3: F1 scores for the list-generation module, varying $k$ and $\alpha$.

friends were both placed in the same organic list, and were both placed in the same machine produced list; the pair is a false positive if they were both placed in the same machine-produced list, but were not placed in the same organic list, etc. We cannot include users who were not placed on any organic lists in our evaluation metrics because we do not know why they were not included in the list — they may belong on the list, but the user lazily left them off, or they truly do not belong. However, we will show in the next few sections that even with imperfect list generation, our results are still quite good.

To evaluate our topicality threshold, we performed a 5-fold cross-validation on a set of 100 manually classified lists. Two human evaluators constructed the training and testing set by manually classifying 100 randomly chosen organic lists as either topical or non-topical. We then calculated the entropy of the lists, and learned an $\epsilon$ cutoff value by choosing the cutoff to maximize the F-measure. Our final $\epsilon$ value was then chosen as the average of the 5 runs, yielding $\epsilon = 1.374$ and F-measure $= 0.92$.

## Topic Labeling

We evaluated eight potential labeling algorithms before choosing the BESTOVERLAP and USERINFOBIGRAM methods, described in the Algorithms section. We performed two experiments to evaluate these algorithms. First, we chose a random sample of 100 organic lists from our dataset. For each list, we produced 8 labels—one from each labeling algorithm. To evaluate how similar each label was to the organic list's original label, we computed the pointwise mutual information (PMI) score of each label, relative to the original label. To compare all labeling techniques, we then calculated the root mean squared error (RMSE) of each algorithm's PMI scores.

The algorithms we evaluated were: (1) the BESTOVERLAP method; (2) the LISTUNIGRAM method, in which topic lists are labeled with the most common unigram from the names of all previously created Twitter lists; (3) the TWEETUNIGRAM method, in which topic lists are labeled with the most common unigram occurring in the past 1,000 tweets from all list members; (4) the TWEETBIGRAM method, in which topic lists are labeled with the most common bigram occurring in the past 1,000 tweets from all list members; (5) the TWEETWIKI method, in which we compute the top-5 unigrams occurring in the past 1,000 tweets from all list members, search Wikipedia with these unigrams, and label the topic list with the most common unigram occurring in the parent categories for the top 10 search results; (6) the USERINFOUNIGRAM method, in which topic lists are labeled with the most common unigram occurring in the user info fields from all list members; (7) the USERINFOBIGRAM method; and (8) the USERINFOWIKI method, which is identical to the tweet wiki method, but chooses the top-5 unigrams occurring in the user info fields from all list members in the first step. Results from this experiment are presented in Table 2.

Table 2: RMSE and average relevance values for labeling algorithms.

| Algorithm | RMSE | Avg relevance |
|---|---|---|
| BESTOVERLAP | 6.79 | 1.72 |
| LISTUNIGRAM | 7.38 | – |
| TWEETUNIGRAM | 7.54 | – |
| TWEETBIGRAM | 6.64 | 0.91 |
| TWEETWIKI | 7.87 | – |
| USERINFOUNIGRAM | 7.35 | – |
| USERINFOBIGRAM | 6.56 | 1.14 |
| USERINFOWIKI | 7.52 | – |

Because the BESTOVERLAP, TWEETBIGRAM, and USERINFOBIGRAM methods all had similarly low RMSE scores, we conducted a second evaluation, just using these three algorithms. Our goal in this second evaluation was to evaluate the meaningfulness and relevance of a label. Two human evaluators manually classified each label for the 100 random organic lists as very relevant (a score of 2), relevant (a score of 1) or irrelevant (a score of 0) when compared with the organic list's original label. These results are also presented in Table 2. The BESTOVERLAP method outperforms the TWEETBIGRAM and USERINFOBIGRAM methods significantly ($p < .01$). Although the USERINFOBIGRAM method appears to outperform the TWEETBIGRAM method, the difference is not significant.

## Topic Ranking

In order to evaluate the ranking, we needed to obtain ground truth about the relevance of tweets to the topic name of a list. We randomly sampled 100 organic lists from the test user set for which we wanted to obtain labels. We used Amazon's Mechanical Turk to obtain binary labels of relevant or irrelevant to a prescribed list name. For each of the 100 organic lists we sampled 100 tweets, and asked three Mechanical Turk workers ("Turkers") to grade whether a tweet was relevant to the list name or not. Since the list names were generated for personal use by Twitter users, some were less informative (*e.g.*, "mac," "vegas tweeple"), and as a result the Turkers were not able to give high-quality

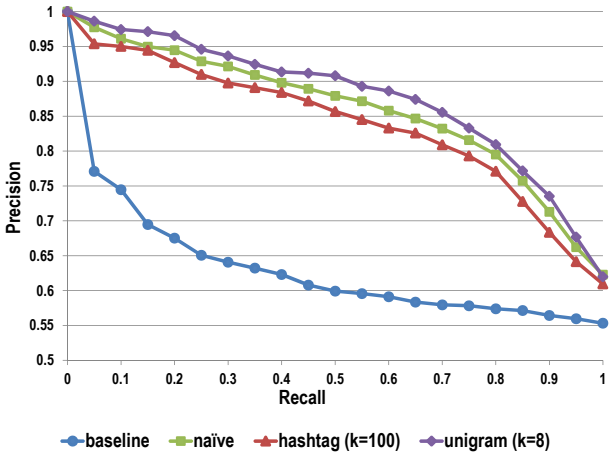


Figure 4: Precision and recall of the hashtag, unigram, naïve ranking methods against a baseline.

ratings. Since we only gave the Turkers two options, for the incoherent list names the vast majority of the answers selected were "irrelevant," as the Turkers did not understand the list name. We used for ranking only those lists for which Turkers unanimously agreed that at least 15% of tweets were relevant. After filtering out lists with incoherent names, we obtained a set of 55 lists that we used for testing. From each list, we used only the examples to which the Turkers responded unanimously, yielding a total of 3,215 labeled examples. The mean percentage of relevant tweets over the 55 lists was 52%. We evaluated the performance of the ranking models using standard information retrieval metrics.

Our first set of experiments test the effect of the $k$ parameter for the hashtag and unigram methods. We then compare the hashtag, unigram, and naïve methods against a baseline. We show the robustness of each heuristic under various amounts of noise in order to determine the sensitivity of the ranking module to the output of the list-generation module.

**Tuning the $k$ parameter** Figures 5 show the Mean Average Precision (MAP) scores for various values of the $k$ parameter. As the plots show, the MAP scores do not vary much between the different $k$ values. As long as we choose a $k$ value that is not very small, the performance of the ranker is consistently good. When $k$ is too small, the naïve Bayes model is not supplied enough training data and performance suffers. We choose $k = 8$ (unigram) and $k = 100$ (hashtag) for the following experiments, as those values performed slightly better than others.

**Comparing Proposed Methods to Baseline** By default Twitter ranks a user's feed in reverse chronological order, so we compared our method against this baseline. Ranking the tweets by time is essentially the same as randomly ordering with respect to their topical relevance. Figure 4 shows the precision-recall curves for the naïve, hashtag, and unigram methods as compared to the baseline. As shown in the figure, our methods significantly outperform the baseline, which quickly converges to a precision of around 50%, which is the average percentage of relevant topics over all lists. All three of our proposed methods have a very similar performance, in which the unigram method slightly outperforms the other two methods. The naïve Bayes classifier is robust to noise, so it is not surprising that the naïve method performs well.

Table 3: Precision at k.

| **precision@k** | **unigram ($k = 8$)** | **baseline** |
|---|---|---|
| 1 | 0.77 | 0.52 |
| 5 | 0.76 | 0.46 |
| 7 | 0.76 | 0.44 |
| 10 | 0.78 | 0.45 |

**Robustness to Noise** The previous experiments tested the proposed methods on user-generated lists, which we assume are high quality. Since part of BUTTERWORTH's purpose is to automatically generate lists, we also wanted to see how sensitive each of the methods were to varying levels of noise. We simulated mistakes in list generation by replacing a varying percentage of tweets in each test list with random tweets from outside the list. For example, a noise level of 30% for a given list would correspond to 30% of the tweets in the training set being replaced with random tweets sampled from users outside the list. Figure 6 shows the precision-recall plots for all three methods with various levels of noise. We can see that the naïve method is the most robust to increasing levels of noise—it is only truly affected at levels of noise above 70%. The other methods are more affected by noise, likely because they quickly begin to only include hashtags/unigrams that correspond to irrelevant topics.

## End-to-end Experiment

The previous experiments evaluated each component of BUTTERWORTH in isolation, but we have yet to test the effectiveness of BUTTERWORTH as a whole. In order to test BUTTERWORTH as a complete system, we need to experiment on an end-to-end workflow that tests the quality of the rankers given generated lists and their labels from the list generator and list labeler, respectively. We propose an experiment with the following workflow:

1. Randomly select 100 generated topical lists (from our set of test users) that contain at least 5 members
2. Label each of the 100 generated topical lists using the BESTOVERLAP list-labeling algorithm
3. Use the unigram (at $k$=8) topic ranker to rank a random sampling of 100 tweets from each of the generated lists and output the top 10 ranked results. For a baseline, we also output 10 randomly ordered tweets.

To evaluate the results of this experiment, we employed three human evaluators to score the top 10 results from both the baseline and the topic ranker. We present the results from this experiment in Table 3. As we can see, the topic ranker greatly outperforms the baseline, with 78% precision@k, compared with 45% precision@k achieved by the baseline. Table 1 shows a few examples of BUTTERWORTH's output.

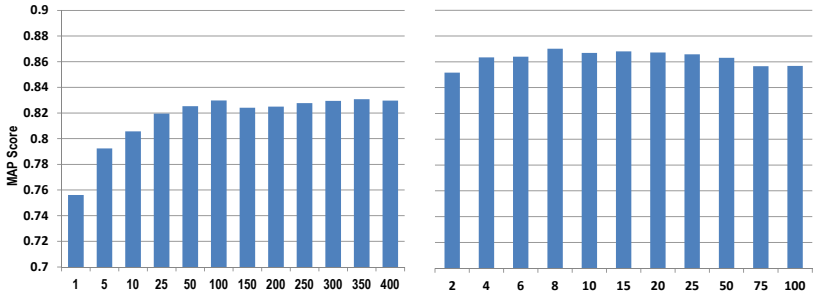


Figure 5: Varying the k parameter for the hashtag (left) and unigram (right) methods.

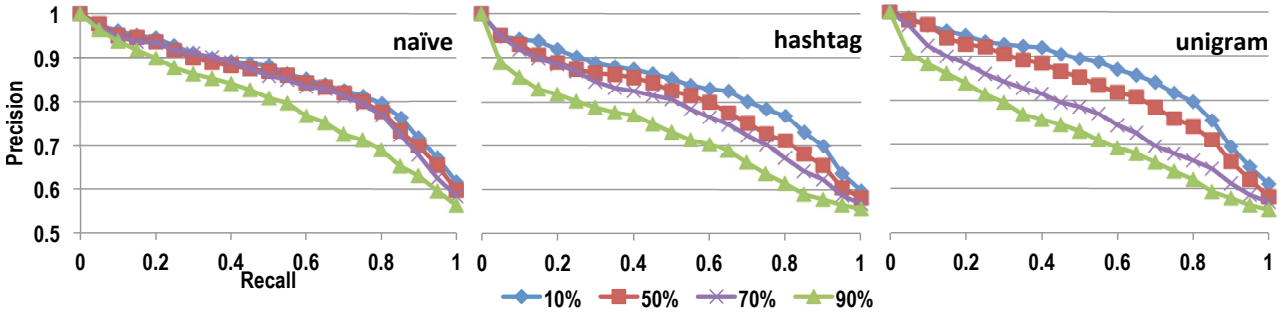


Figure 6: A comparison of the effects of varying the percentage of noise on precision and recall for the naïve, hashtag, and unigram methods.

We can compare the average precision achieved by the baseline, 45%, with the average rate of relevant tweets found in organic lists—52%. These very similar rates indicate that the topical lists generated by the list- generation module mimic the quality and topical coherence of human-constructed lists. Because there are many components in BUTTERWORTH's pipeline, there is a high potential for error propagation and compounding. However, we can see that BUTTERWORTH performs quite well, maintaining high-quality ranking despite potential noise in list generation.

## Discussion

While BUTTERWORTH performs well for the general scenarios we experimented with, there are some user cases which require adaptation of the strategies we employ. Currently, BUTTERWORTH places each friend in exactly one cluster (i.e., a hard-clustering). If a friend tweets about multiple topics of interest to the user, then the clustering procedure will identify only one of these interests. We propose two possible solutions to this problem. First, we could simply modify our clustering algorithm to allow friends to be part of many clusters (e.g., through soft- or hierarchical-clustering). Figure 6 demonstrates that BUTTERWORTH is capable of ranking well under noisy list generation. Thus, we believe the potentially noisier clusters produced by a fuzzy clustering algorithm would not drastically affect ranking quality.

Instead of modifying our clustering algorithm, we could alternatively use the ranker on *all* tweets rather than those produced by a specific list. A solution at the interface level would enable users to choose the set of friends to which the ranker would be applied. For instance, if BUTTERWORTH detected that the user is interested in cooking, the modified interface would enable the user to apply the cooking-ranker to just a specific friend, just the generated list of cooking friends, or to their entire set of friends. To test the feasibility of this modified interface we re-ran the end-to-end experiment (Table 3) using a user's entire feed as input to each ranker. We found that BUTTERWORTH performs slightly better, giving 82% precision@10. This slight improvement may be due to the fact that BUTTERWORTH can find the "best" tweets on a given topic regardless of who produced them. In some cases, these may be as good or better than those produced by list members.

A second potential issue with BUTTERWORTH is linked to our user interaction model. Our system explicitly ranks a user's feed by topics of their interest and therefore loses temporal ordering. We believe this issue can also be fixed with a modified interface design that surfaces BUTTERWORTH's ranking in a different way. Instead of completely reordering a users feed on a page, we propose an interface that uses the generated rankers to color the tweets in a user's temporally-oriented feed. Thus, the salience of a tweet in the feed would vary based on the score that the ranker produces for each tweet. This interface would allow a user to view both topically and temporally relevant content.

## Conclusion

We have a few ideas about how to improve BUTTERWORTH's future performance. First, users who have overlapping interests may also build overlapping lists, write similar text, retweet related content, and link to similar URLs;

the list-generation step should thus incorporate information sources beyond network structure. Second, each social networking site has some particular features — say, privacy settings unavailable in other systems—that might shed additional light on how to construct better lists, which we should exploit. In addition, now that we have constructed a robust back-end architecture, we have begun to consider the design of the UI for BUTTERWORTH. The mechanism by which the user accesses lists (manually created and automated, topical, and contextual), provides feedback, controls lists and ranking behavior can, and should, be taken into account to create a usable user experience. The interactions of the user with the system can be utilized to further inform the intelligent components of BUTTERWORTH.

In this work we have described BUTTERWORTH, a system for automatically performing topic-sensitive grouping and ranking of the messages in a user's social feed. Unlike existing approaches for message ranking, BUTTERWORTH requires no explicit guidance from the user and works across a spectrum of users and scales. By breaking apart the simple "follow" relationship into multiple topical lists, BUTTERWORTH addresses the context collapse problem. By leveraging each of these lists to generate rankers that can be applied to the user's feed, the system alleviates channel collapse. This is achieved through use of a novel architecture that leverages a user's social network. By weighting this network to encourage topical sub-community extraction, BUTTERWORTH is able to identify groups of users that share a common interest. The lists generated by BUTTERWORTH are accurately labeled, and the rankers trained from these lists improve the relevance of a feed from 45% to 78%.

## Acknowledgements

The authors would like to thank Avishay Livne for help in collecting data for this project, Qiaozhu Mei for helpful discussions, Andrea Pellegrini for help with formatting figures and the anonymous reviewers for their helpful comments. This work was supported by National Science Foundation grant IGERT-0903629.

## References

Adamic, L. A., and Adar, E. 2003. Friends and neighbors on the web. *Social Networks* 25(3):211 – 230.

Bernstein, M. S.; Suh, B.; Hong, L.; Chen, J.; Kairam, S.; and Chi, E. H. 2010. Eddi: interactive topic-based browsing of social status streams. In *UIST'10*, 303–312.

Boyd-Graber, J.; Chang, J.; Gerrish, S.; Wang, C.; and Blei, D. 2009. Reading tea leaves: How humans interpret topic models. In *NIPS'09*.

Dahimene, R.; Mouza, C.; and Scholl, M. 2012. Efficient filtering in micro-blogging systems: We won't get flooded again. In Ailamaki, A., and Bowers, S., eds., *Scientific and Statistical Database Management*, volume 7338 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 168–176.

Das Sarma, A.; Das Sarma, A.; Gollapudi, S.; and Panigrahy, R. 2010. Ranking mechanisms in Twitter-like forums. In *WSDM'10*, 21–30.

Fang, L.; Fabrikant, A.; and LeFevre, K. 2012. Look who I found: Understanding the effects of sharing curated friend groups. In *WebSci'12*, 137–146.

Guc, B. 2010. Information Filtering on Micro-blogging Services. In *Master's Thesis*. Swiss Federal Institute of Technology Zürich.

Hong, L., and Davison, B. 2010. Empirical study of topic modeling in Twitter. In *Proceedings of the First Workshop on Social Media Analytics*, 80–88. ACM.

Hong, L.; Convertino, G.; Suh, B.; Chi, E. H.; and Kairam, S. 2010. FeedWinnower: layering structures over collections of information streams. In *CHI '10*, 947–950.

Hong, L.; Bekkerman, R.; Adler, J.; and Davison, B. 2012. Learning to rank social update streams. In *SIGIR'12*.

Kim, D.; Jo, Y.; Moon, I.-C.; and Oh, A. 2010. Analysis of Twitter lists as a potential source for discovering latent characteristics of users. In *Workshop on Microblogging at the ACM Conference on Human Factors in Computer Systems. (CHI 2010)*.

Lin, J.; Snow, R.; and Morgan, W. 2011. Smoothing techniques for adaptive online language models: topic tracking in tweet streams. In *KDD'11*, 422–429. ACM.

Paek, T.; Gamon, M.; Counts, S.; Chickering, D. M.; and Dhesi, A. 2010. Predicting the importance of newsfeed posts and social network friends. In *AAAI'10*, 1419–1424.

Pal, A., and Counts, S. 2011. Identifying topical authorities in microblogs. In *WSDM'11*, 45–54.

Pons, P., and Latapy, M. 2005. Computing communities in large networks using random walks. In *Computer and Information Sciences - ISCIS 2005*, volume 3733 of *Lecture Notes in Computer Science*.

Ramage, D.; Dumais, S.; and Liebling, D. 2010. Characterizing microblogs with topic models. In *WDSM'10*.

Shannon, C. E. 2001. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.* 5(1):3–55.

Sriram, B.; Fuhry, D.; Demir, E.; Ferhatosmanoglu, H.; and Demirbas, M. 2010. Short text classification in Twitter to improve information filtering. In *SIGIR '10*, 841–842.

Tseng, C.-Y.; Chen, Y.-J.; and Chen, M.-S. 2012. Socfeedviewer: A novel visualization technique for social news feeds summarization on social network services. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, 616 –617.

Uysal, I., and Croft, W. B. 2011. User oriented tweet ranking: a filtering approach to microblogs. In *CIKM '11*, 2261–2264.

Wagner, C.; Liao, V.; Pirolli, P.; Nelson, L.; and Strohmaier, M. 2012. It's not in their tweets: Modeling topical expertise of Twitter users. SocialCom '12.

Xu, Z.; Lu, R.; Xiang, L.; and Yang, Q. 2011. Discovering user interest on Twitter with a modified author-topic model. In *WI-IAT'11*, volume 1, 422 –429.