Hashtagger+: Efficient High-Coverage Social Tagging of Streaming News

Bichen Shi, Gevorg Poghosyan, Georgiana Ifrim, and Neil Hurley

Abstract—News and social media now play a synergistic role and neither domain can be grasped in isolation. On one hand, platforms such as Twitter have taken a central role in the dissemination and consumption of news. On the other hand, news editors rely on social media for following their audience's attention and for crowd-sourcing news stories. Twitter hashtags function as a key connection between Twitter crowds and the news media, by naturally naming and contextualizing stories, grouping the discussion of news and marking topic trends. In this work we propose Hashtagger+, an efficient learning-to-rank framework for merging news and social streams in real-time, by recommending Twitter hashtags to news articles. We provide an extensive study of different approaches for streaming hashtag recommendation, and show that pointwise learning-to-rank is more effective than multi-class classification as well as more complex learning-to-rank approaches. We improve the efficiency and coverage of a state-of-the-art hashtag recommendation model by proposing new techniques for data collection and feature computation. In our comprehensive evaluation on real-data we show that we drastically outperform the accuracy and efficiency of prior methods. Our prototype system delivers recommendations in under 1 minute, with a Precision@1 of 94% and article coverage of 80%. This is an order of magnitude faster than prior approaches, and brings improvements of 5% in precision and 20% in coverage. By effectively linking the news stream to the social stream via the recommended hashtags, we open the door to solving many challenging problems related to story detection and tracking. To showcase this potential, we present an application of our recommendations to automated news story tracking via social tags. Our recommendation framework is implemented in a real-time Web system available from **insight4news.ucd.ie**.

Index Terms—Learning-to-rank, Dynamic Topics, Social Tags, News, Real-time Hashtag Recommendation, Digital Journalism.

1 Introduction

Twitter is a fast growing social media platform that has taken a central role in the consumption, production and dissemination of news. In a recent study, nearly 9 in 10 Twitter users say they use Twitter for news, and the vast majority of those (74%) do so daily [1]. Most of the news stories spreading on Twitter have names, in the form of *hashtags* that contextualize the stories. These keyword-based tags, describing the content of a tweet, are a natural way to label tweets for news stories. For example, #Brexit, #Remain, #VoteLeave were heavily used for Twitter discussions on the EU membership referendum held in UK in June 2016. Hashtags tend to appear spontaneously around breaking news or developing news stories, and are a way for news readers to connect to a particular story and community, to get focused updates in real-time. News organizations use hashtags to target Twitter communities in order to promote original content and engage readers. Journalists sometimes introduce new hashtags, but the Twitter crowd is the one that most often creates and drives the usage of a few of many competing hashtags, thus echoing the current social discourse (e.g., #Brexit, and the opinion camps of #VoteLeave and #Remain for the EU referendum story).

A news story can have multiple hashtags, and is likely to have different hashtags at different stages of the story. For example, in the Umbrella Revolution story (a series of street protests in Hong Kong in 2014), Twitter played a huge role: thousands of people were protesting and reporting on ongoing events by tweeting with their phones. Three main hashtags are used during the event: #HongKong, #OccupyCentral and #UmbrellaRevolution. As shown in Figure 1, each hashtag dominates the discussion at different time points: #HongKong, the location of the events, is popular at the beginning of the story. #OccupyCentral becomes popular when sit-in protests begin to attract wide attention, particularly on Twitter. Finally, #UmbrellaRevolution dominates the topic as it refers to the protesters using umbrellas to protect themselves from teargas. The relationship between the news story and the hashtags is very dynamic, with new hashtags being created and adopted by Twitter users at a rapid pace. It may be seen from this example that for applications aiming to exploit hashtagging, it is critical to capture the dynamic co-evolution of news and hashtags, as the news story evolution influences the Twitter discussions, which in turn may affect the news. We note that the content of some articles may not be obviously related to a story, but a hashtag recommender can use the social discourse to create a bridge between news articles. Figure 2 showcases this idea of bridging news articles using social tags.

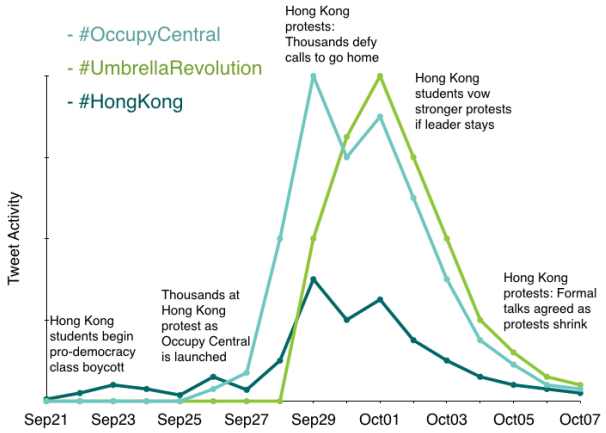


Fig. 1. Hashtags usage during the Umbrella Revolution event in 2014.

In this work we propose a real-time hashtag recommendation approach that is able to efficiently and effectively capture the dynamic evolution of news and hashtags. Most prior approaches for hashtag recommendation work on static datasets and do not account for the emergence and disappearance of hashtags. Many approaches use topic/class modeling [2], [3], [4], by considering hashtags as topics, and mapping news articles to topics using content similarity. As the relevant hashtags change quickly and the news and Twitter environments are highly dynamic, approaches that use multi-class classification need continuous retraining to adapt to new content. Additionally, to train models, these methods rely on tweets that contain both hashtags and

• *B. Shi, G. Poghosyan, G. Ifrim, and N. Hurley are with Insight Centre for Data Analytics, University College Dublin, Ireland*
{bichen.shi, gevorg.poghosyan, georgiana.ifrim, neil.hurley}@insight-centre.org

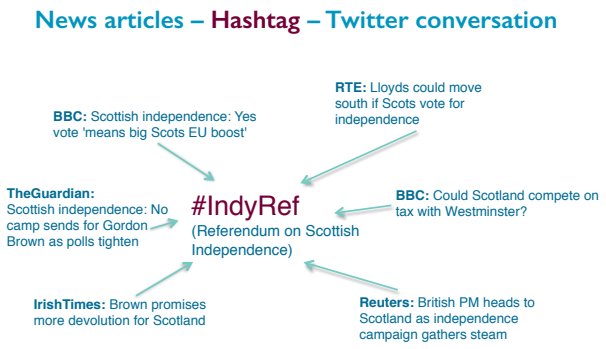


Fig. 2. Linking news articles via real-time recommended Twitter hashtags for the Scottish referendum story.

URLs. Such tweets are very few and tend to be noisy, which may explain the low accuracy of prior methods (e.g., 50% precision reported in recent work [4], [5]).

Our previous work [6] has introduced an accurate learning-to-rank (L2R) approach for streaming hashtag recommendation to news, but its efficiency and coverage is still not appropriate for practical use. The model in [6] has a time consuming data collection stage for each new article, and thus requires 12 hours to deliver 89% precision and 66% article coverage. In this paper we refine the problem statement to explicitly account for these requirements and propose new techniques to solve this problem.

Problem Statement. We aim to map a stream of news articles to a stream of Twitter hashtags, in **real-time**, with **high-precision** and **high-coverage.** Real-time refers to the efficiency of a solution: given a new article, how quickly can we recommend hashtags? For example, we prefer a solution that can deliver recommendations in under 5mins, while a solution that takes hours is not acceptable. High-precision refers to the quality of recommendations. For example, for a news headline such as, *Deadly car bomb targets Afghan bank*, we prefer focused recommendations, e.g., *#afghanistan #helmand,* that refer to specific entities involved in this story, instead of generic hashtags such as *#news*. High-coverage refers to how many articles get recommended hashtags within 5mins. For example, 8 out of 10 articles with at least one recommendation is an example of good coverage, while 1 out of 10 is not.

New Techniques. To address the above problem, we propose the following new techniques:

- An efficient L2R algorithm that works in real-time streaming environments. The choice of L2R modeling in Hashtagger+, in contrast to multi-class classification (MCC) modeling in prior work, enables us to address the following challenges:
  - **Many Classes:** In MCC modeling, training data is partitioned based on hashtags (a hashtag is interpreted as a class), and a model is trained for each single hashtag. As there are thousands of hashtags emerging every few minutes, we need to train thousands of models. Furthermore, by splitting the available training data, MCC needs to collect enough labeled data for each class. In our L2R model, we can use all the labeled data to train a single relevance model.
  - **Dynamic Classes:** Hashtags are very dynamic, thus if modeled as classes, previously trained models will not be useful for predicting on new data, since old hashtags are discarded and new hashtags emerge rapidly. This means an MCC model needs to be retrained often, while our L2R model does not need retraining.
  - **Concept Drift:** The usage and meaning of a hashtag may change over time, thus its content profile will be affected by concept drift. This means pre-trained MCC models will not work well if the meaning of the hashtag changes in new data. Our L2R model is robust to concept drift because hashtag relevance is stable over time.
- New approaches for efficient and effective data collection and feature engineering for the L2R model. We propose new query generation methods to summarize the key information in a given news article. Query generation from text documents is still an open problem [7] and has generated a lot of interest in the research community. In our setting, we require queries that can describe well what the article is about, can be generated quickly and can retrieve many relevant tweets. We design new algorithms to address the cold-start challenge for real-time hashtag recommendation. The key challenge is to re-use the data collected for older articles to efficiently bootstrap recommendations for new articles.

Besides presenting new algorithms, we also run a comprehensive empirical study. We compare 16 L2R algorithms to understand which ones are better suited for real-time hashtag recommendation and find that pointwise L2R with a RandomForest algorithm delivers good precision and efficiency. We also extend the analysis of our method and the comparison to the state-of-the-art. We show that our new model behaves very well for recommending hashtags to both popular and more niche articles, i.e., those articles that do not get a lot of attention/tweets on Twitter, a setting where prior models do not have enough training data to perform well.

Our new model delivers recommendations in **under 1 minute, with 94% precision and 80% coverage**, a 20% improvement in coverage and an order of magnitude improvement in efficiency, as compared to prior work [6].

Contributions. Our main contributions are as follows:

1) **Improved hashtag recommendation model:** We improve the state-of-the-art hashtag recommendation model in [6] by proposing new approaches for data collection and feature computation. We study four approaches for query extraction from news articles and evaluate the end-to-end effect on the recommendation precision, coverage and running time. We analyse three cold-start strategies to enhance the efficiency and coverage of recommendation for new articles, by bootstrapping the feature computation using data collected for older stories.

2) **Extensive empirical study:** We provide an extensive study of different approaches for streaming hashtag recommendation, and show that pointwise L2R is more effective than MCC and other more complex L2R approaches. We also show that relevance modeling allows us to deliver recommendations to popular as well as niche (less popular) stories, providing high coverage and precision even with small amounts of available data, a setting where most prior models do not perform well.

3) **Applications of social tags:** We deliver an effective approach for merging news and social streams in real-time, therefore opening the door for addressing problems that are difficult to solve in either the news or social domain (e.g., story linking, detection and tracking). We apply our hashtag recommendations to a real problem faced by newsrooms: automating story retrieval and tracking by using social tags.

The rest of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present our

improved recommendation model. In Section 4 we present the evaluation setting and experimental results for our approach and a comparison to the state-of-the-art. In Section 5 we discuss a real-world application of our recommendations to the task of automated story tracking. We conclude in Section 6 and present directions for future work.

2 Related Work

Existing work on hashtag recommendation tackles the problem from either a class/topic modeling point of view, or from a learning-to-rank perspective. We discuss recent literature from both categories of approaches, as applied to tweets or news articles.

Hashtag Recommendation for Tweets. Prior work focusing on hashtag recommendation for tweets relies on MCC modeling on static datasets. The work of [9], [10] builds Naïve Bayes or SVM classifiers for hashtags, where (i) a hashtag is seen as a class and (ii) the tweets tagged with that hashtag are assumed to be labeled data for that class. Hashtag recommendation for tweets can be adapted to recommendation for news, by treating the news headline as a rich tweet. As we show in our experiments, MCC approaches are overwhelmed by the data scale, sparsity and noise characteristics of tweets.

Many other approaches employ topic modeling with PLSA [2], [3], DPMM [4] and LDA [11], [12], [13], [14], [15]. For example [12] fits an LDA model to a set of tweets in order to recommend hashtags. They combine the LDA model with a translation model, to address the vocabulary gap between tweets and hashtags. LDA-type approaches face drastic challenges regarding both scalability and accuracy of recommendation, since either hashtags that are too general are recommended, e.g., #news, #life, or ones that are not actively used by Twitter users. This happens because the focus is on recommending hashtags solely driven by the content of tweets [12]. These models are also not efficient as they need to be constantly retrained to adapt to newly emerging hashtags.

Some recent methods formulate hashtag recommendation based on multi-class modeling with deep neural nets. The work in [5] proposed an attention-based Convolutional Neural Network model for hashtag recommendation to tweets. This approach works on a static dataset and improves the state-of-the-art results, but the recommendation precision is still around 50%.

The work in [16] uses pairwise L2R for hashtag recommendation for tweets. This work is tailored for tweets with at least one URL and one hashtag in their body, a very small subset of the overall tweet pool discussing news. Training on this small and noisy tweet set can pose serious problems for the recommendation, resulting in low Precision (reported P@1 of 40% in [16]) and low coverage, i.e., few tweets receiving any recommendation at all (reported coverage of 50%). The data collection is seeded by an external set of 135 trending hashtags collected from hashtags.org each day. This means that many of the hashtags used as seed do not relate to news at all, but just happen to be trending on hashtags.org at the time of collection. Furthermore, there is no focus on news nor on efficient recommendation which is critical for our setting. In contrast to the approach in [16], we use the actual news articles to drive the selection of tweets and candidate hashtags. In our experiments we compare to the method of [16] and show that our model achieves much better coverage and Precision@1.

Hashtag Recommendation for News. There is little prior work focusing specifically on hashtag recommendation for news. The approach in [17] relies on a manual user query to retrieve related articles, which are then clustered to create a topic profile. A hashtag profile is also created from tweets collected from a set of manually selected accounts. Hashtags with a similar profile to a cluster, are recommended to that cluster. Since the experiments are done on a static collection, the user engagement with the hashtag is not considered.

In [6] we proposed a high-precision pointwise L2R framework for hashtag recommendation for news. In this paper, we improve the efficiency and coverage of that method, while preserving high-precision. We explore different methods for retrieving relevant tweets for news articles and evaluate the end-to-end effect on recommendation. There are several published methods for retrieving tweets for news articles. The method in [18] gathers news articles and tweets independently of each other, then uses co-occurence terms for the two datasets (i.e., articles, tweets) to connect the news and tweets. This approach to collect tweets is not appropriate, as it may result in little or no overlap with the news dataset. The method in [19] explores different ways of generating queries from news articles, but is limited to working only with tweets with URLs. We also generate queries from news articles, but do not restrict ourselves to tweets with URLs. We further advance the work in [6] with an extensive study of MCC and L2R approaches evaluated on the task of streaming hashtag recommendation for news.

The work in [20] focuses on temporal aspects of hashtag recommendation and proposes two content-based models implemented in a distributed manner. Again, the focus is on hashtag recommendation for tweets with URLs, this time aiming to extend the recommendation to the linked news documents. Similar to other topic modeling approaches, since the focus is on modeling content, rather than hashtag relevance, these methods need to be constantly retrained to keep up with the rapid shift in the news focus. As we show in our experiments, our L2R approach does not need to be retrained, and delivers higher precision due to the focus on modeling hashtag relevance. Related to our work are also recent approaches to real-time tag recommendation for streaming scientific documents and webpages [21], [22]. In that work, the set of tags is assumed to be static, and fairly small, which facilitates a lot of pre-processing steps. In our scenario, both articles and tags are continuously streaming into the system, and the set of hashtags is very large and dynamic, which makes the problem more challenging.

Learning to Rank (L2R). In classic Information Retrieval (IR) L2R approaches, a ranked list of documents is returned for a user query. Depending on the input representation and loss function, L2R algorithms can be categorised (see [8]) as pointwise [23], [24], pairwise [25], [26], [27] and listwise [28], [29], [30]. Although listwise and pairwise approaches are commonly used, we show here that they are not suitable for our setting because of efficiency constraints and the dynamic nature of our data. From an efficiency point-of-view, the computational complexity of listwise and pairwise approaches is usually high [30], [31], making them less suitable for a real-time setting. Pointwise approaches were shown to be efficient and effective for binary relevance labels [23], [32], [33], a finding reinforced in our experiments. We compare our method to existing MCC approaches (Naïve Bayes [9], SVM, Neural Networks) and L2R approaches (pointwise and pairwise L2R [6], [16]).

3 Recommendation Model

In this section we discuss the design of the proposed recommendation model. There are two key technical aspects of our approach: (1) the **choice of modeling** (e.g., learning-to-rank versus multi-class classification), (2) the **data collection** for feature computation (e.g., generating article queries and addressing cold-start). The work in [6] focused on modeling and showed that an L2R approach works better than a MCC approach. In this work, we improve our model to address a critical component for *real-time recommendation*, the data collection stage, which enables efficient, high quality, feature computation for the learning-to-rank model. We first review the basics of the L2R model, then present our new approaches for efficient data collection. Figure 3 gives a schematic view of the stages of the model.

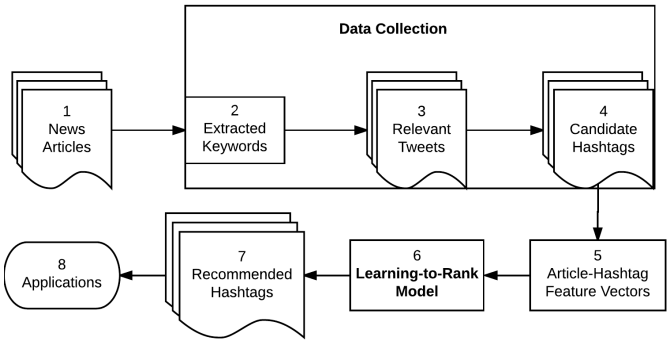


Fig. 3. Building blocks for the real-time recommender Hashtagger+.

In Algorithm 1 we provide the detailed steps of our recommendation algorithm. As input, we have a stream of articles A, a database D (storing past articles, tweets and recommendations, if any), access to the Twitter stream T, and a pre-trained L2R model m (described in Section 3.1). The output is a stream of recommended hashtags for each article in A. The algorithm proceeds as follows. For each article a, we start a new thread for processing that article (we process several articles in parallel). We first generate a query from the article (function $GenerateQuery(a)$). The query is a set of keywords that summarizes the article, and it affects the amount and quality of tweets collected for computing features. For example, if the query is too general or too narrow, we will collect very noisy/large or very little data from Twitter streams. Variable t tracks time, and evolves from an origin point, $t = 0$, when the article arrives on the stream, up to a maximum of 24h. We increase t in window increments of 5 minutes. A key term for feature computation is the set $T_a = T_{a,search} \cup T_{a,stream} = \cup_t T_{a,t}$. This is a set of tweets associated with article a, named the tweet-bag. In Section 3.1.1 we discuss the time complexity of feature computation as a function of data collected. Function $ColdStartSearch(Q_a, D)$ employs techniques for re-using past articles and recommendations, to quickly collect data for a new article. Depending on the technique used, this function will return a set of relevant tweets for a given article, or directly a recommended hashtag. In Section 3.2.2 we describe the cold-start approaches and discuss their time complexity. In each time window, we update the data collected for article a, using $TwitterStream(Q_a, T, t)$, select a set of candidate hashtags, and prepare a feature vector for each pair of article and candidate hashtag. Finally, the pre-trained model m can take in a feature vector and produce a recommended hashtag.

Algorithm 1: Hashtagger+ Recommender Model

Input: Stream of news articles A, database of past articles D, public Twitter stream T, pre-trained L2R model m.
Output: Stream of recommended hashtags R_A.
Method:

```
R_A = ∅
for a ∈ A do
    StartThread(a) //Start a new thread dedicated to processing article a.
    t = 0 //Time tracking variable.
    T_a = ∅ //Set of tweets for a.
    R_a = ∅ //Set of recommended hashtags for a.
    R_{a,t} = ∅ //Set of recommended hashtags for a, at time t.
    Q_a = GenerateQuery(a)
    T_{a_search}, H_{a_search} = ColdStartSearch(Q_a, D)
    T_{a,t} = T_{a_search} //Set of tweets retrieved from D for a.
    T_a = T_a ∪ T_{a,t}
    if H_{a_search} ≠ ∅ then
        R_{a,t} = H_{a_search}
        R_a = R_a ∪ R_{a,t}
        WaitUntil(t, 5) //Hashtag recommendation updated every 5mins.
    while t <= 60 * 24 do
        //Update recommendations for given article up to 24h.
        R_{a,t} = ∅
        T_{a_stream} = TwitterStream(Q_a, T, t) //Set of tweets streamed from Twitter for article a.
        T_{a,t} = T_{a,t} ∪ T_{a_stream}
        T_a = T_a ∪ T_{a,t} //Update tweet bag T_a of article a.
        H_{a,t} = CandidateHashtags(T_a)
        F = ∅ //Set of feature vectors for a and its candidate hashtags H_{a,t}.
        for h ∈ H_{a,t} do
            x_{a,h,t} = ComputeFeatures(a, h, t, T_a, D)
            F = F ∪ x_{a,h,t}
        for x_{a,h,t} ∈ F do
            x_{a,h,t} = NormalizeVector(x_{a,h,t}, F)
            //Normalize feature vector wrt article a. This step makes feature vectors comparable across articles.
            r_{a,t} = L2RModel(m, x_{a,h,t}) //If relevance score above threshold, recommend hashtag.
            R_{a,t} = R_{a,t} ∪ r_{a,t}
        R_a = R_a ∪ R_{a,t}
        StoreArticle(a, T_a, R_a, D)
        WaitUntil(t, 5)
    R_A = R_A ∪ R_a
    StopThread(a) //Stop tracking article a.
Return R_A
```

3.1 Problem Modeling via Learning-to-Rank

In this section we describe the L2R framework for our recommender model. In an IR setting we work with a collection of documents C. Given a query q, we retrieve a subset of documents C_q from the collection, rank the documents by a global ranking model $f(q, d)$, and return the top ranked documents. The $f(q, d)$ model is typically built automatically using supervised machine learning techniques trained with labelled ranking data [34]. In our recommendation setting, the query q is extracted from an individual article $a \in A$, where A is a stream of news. The document collection is a stream of hashtags H, extracted from a stream of tweets T. Since efficiency is critical in our setting, we take a pointwise L2R approach by transforming the ranking problem into a classification problem [23], [34]. First, a subset of candidate hashtags H_a is retrieved for article a using the function $CandidateHashtags(T_a)$ shown in Algorithm 1. This step prunes the huge stream of hashtags to a subset that is potentially relevant for article a. Then, for each article-hashtag pair (a, h), $h \in H_a$, we create a feature vector $\mathbf{x}$, with label $y \in \{0, 1\}$ ($y = 1$ if the hashtag is relevant for the article). Given n training examples $M = \{\mathbf{x}_i, y_i\}$, $i = 1, 2, ..., n$, we construct a global classifier $f(\mathbf{x})$ to predict y for any feature vector $\mathbf{x}$ of an arbitrary article-hashtag pair. To address the dynamic aspect of real-time hashtag recommendation[1], we extract time-aware features $\mathbf{x}_t$. We write $f(\mathbf{x}_t) = y_t$ to denote that the feature vector is dependent on time, while the classification function f is not. We employ two sliding time windows to transform the dynamic environment to a static one. The global time window γ pools together all articles published in the past 24h from the current time. This allows us to compute tf.idf type features for describing each candidate hashtag. The local time window λ, depends on a given article a, and restricts the computation of features to a local tweet-bag T_a. To select the time window parameters we are guided by the application domain and efficiency constraints. We restrict the local window/tweet-bag T_a to the most recent 4h of tweets collected for a. The global window is set to 24h, due to the news lifecycle where most news are typically ignored after 24h or get updated and become new articles [35].

1. Hashtags and articles come as streams and the relevance of hashtags to articles is time-dependent since the hashtag representation changes due to the arrival of new tweets.

Because we learn a single relevance model across all articles and hashtags, and the concept of hashtag relevance is stable over time, Hashtagger+ can cope well with a large number of dynamic hashtags and concept drift. In the next section we discuss the time complexity of feature computation for our L2R model.

3.1.1 Time-Aware Features

Given an article a and corresponding time-dependent candidate hashtags $h \in H_{a,t}$, we compute a feature vector $\mathbf{x}_{a,h,t}$ for each article-hashtag pair. We investigate features relevant for our real-time recommendation setting [6], [37], [38]. Because we aim to learn a single ranking function for all queries (i.e., articles) using the same set of features, all feature vectors have to be normalized at query-level for dealing with the issue of different number of candidate hashtags, and the variance between queries. This is achieved with function $NormalizeVector(x_{a,h,t}, F)$ in Algorithm 1. We select features that describe properties of the article and the hashtag, and features that reflect social network characteristics of users that engage with the hashtags. The efficiency of feature computation is directly affected by the size of data collected for a given news article. We now review the features used in our model and the time complexity of feature computation.

Local similarity $LS_{a,h,\lambda}$**:** Compares the article text to a local hashtag tweet bag via the cosine similarity as shown in Equation 1. Let $T_{a,h,\lambda}$ be the subset of tweets in T_a that mention h within time window λ. $||.||$ denotes the L2 norm. We form a bag of words representation of a and each hashtag h which we denote as the vectors $\mathbf{a}$ and $\mathbf{h}$. The bag-of-words representation of h is obtained from a bag of tweets T_h associated with h and is hence denoted as $h(T_h)$.

$$LS_{a,h,\lambda} = \frac{\mathbf{a} \cdot \mathbf{h}(T_{a,h,\lambda})}{\|\mathbf{a}\|\|\mathbf{h}(T_{a,h,\lambda})\|} \tag{1}$$

The local similarity is an important content feature that indicates how relevant a hashtag is to an article. The time complexity for computing this feature is $O(|a| \times |h|)$, where $|a|$ is the length of the article and $|h|$ is the number of terms in the bag-of-words representation of candidate hashtag h.

Local hashtag frequency $LF_{a,h,\lambda}$**:** Captures local popularity of usage for a given hashtag h present in the article tweet-bag $T_{a,\lambda}$.

$$LF_{a,h,\lambda} = \frac{|T_{a,h,\lambda}| - \min\{|T_{a,v,\lambda}|\}}{\max\{|T_{a,v,\lambda}|\} - \min\{|T_{a,v,\lambda}|\}} \tag{2}$$

where $v \in H_a$. The local frequency feature compares all hashtags from the same set H_a, and indicates whether a hashtag is dominating the topic. The time complexity of computing this features is $O(|T_{a,h,\lambda}|)$.

Global similarity $GS_{a,h,\gamma}$**:** Distinguishes between general and topic specific hashtags. The article bag-of-words representation is compared with the global hashtag representation extracted from T_h within global window γ:

$$GS_{a,h,\gamma} = \frac{\mathbf{a} \cdot \mathbf{h}(T_{h,\gamma})}{\|\mathbf{a}\|\|\mathbf{h}(T_{h,\gamma})\|} \tag{3}$$

The time complexity of this feature is $O(|a| \times |T_{h,\gamma}|)$. The set of tweets containing h in global window γ can grow very rapidly. In practice, we take a random sample of $T_{h,\gamma}$ of size 5K to estimate the term-frequency score. The global profile of a hashtag is then re-used for all article feature vectors.

Global hashtag frequency $GF_{h,\gamma}$**:** Captures global popularity of usage for a given hashtag. Let $|T_{h,\gamma}|$ denote the number of tweets in T_h within global time window γ. $GF_{h,\gamma}$ is computed as in Equation 2 and has similar time complexity $O(|T_{h,\gamma}|)$.

Trending hashtag TR_{a,h,t_n}**:** Captures a significant increase in local hashtag frequency and aims to identify *article-wise trending hashtags*. It uses the local frequency of a hashtag in two consecutive time windows. Given time window $W_n = t_n - t_{n-1}$, the number of tweets containing h in tweet stream T_a in time window W_n is $|T_{a,h,W_n}|$, then:

$$TR_{a,h,t_n} = \frac{|T_{a,h,W_n}| - |T_{a,h,W_{n-1}}|}{|T_{a,h,W_{n-1}}|} \tag{4}$$

Expected gain EG_{a,h,W_n}**:** Captures the potential of h in the next time window, and is expected to boost trending hashtags while punishing fading ones. Based on trending feature TR_{a,h,t_n}, we compute the expected number of tweets in T_a mentioning h for the next time window W_{n+1}, denoted by $E(|T_{a,h,W_{n+1}}|)$:

$$EG_{a,h,W_n} = E(|T_{a,h,W_{n+1}}|) = (1+TR_{a,h,t_n}) \cdot |T_{a,h,W_n}| \tag{5}$$

Hashtag in headline $HE_{a,h}$**:** Many hashtags are a variation of the name of the people/place/event being discussed. We define $HE_{a,h}$ as a binary feature equal to 1 if the hashtag is in the pseudo-article (headline, sub-headline, first sentence) after removing space between terms. The time complexity of computing the trending, expected gain and headline features is $O(1)$.

Unique user ratio $UR_{a,h,\lambda}$**:** The ratio of unique Twitter users using h in T_a within time window λ, to the number of tweets. Function User(T) returns the set of users in tweet stream T.

$$UR_{a,h,\lambda} = \frac{|\text{User}(T_{a,h,\lambda})|}{|T_{a,h,\lambda}|} \tag{6}$$

User credibility $UC_{a,h,\lambda}$**:** The quality of a hashtag depends on the users using it. A common Twitter user credibility indicator is the number of followers. Users with more followers are usually celebrities, domain experts and experienced users that work hard to attract followers. Therefore, we define user credibility as the maximum, the average and the median of the followers of users tagging h in article tweet bag T_a in λ.

$$MaxF_{a,h,\lambda} = \max(\text{Follower}(u)), u \in \text{User}(T_{a,h,\lambda}) \tag{7}$$

The time complexity for computing user-related features is $O(|T_{a,h,\lambda}|)$.

3.2 Efficient Data Collection

The data collection for computing features plays a key role in the overall efficiency and effectiveness of the recommender model. In this section we discuss the design choices and present a theoretical analysis of our proposed algorithms for data collection.

3.2.1 Query Generation

The article query has a direct effect on the quality and quantity of candidate hashtags that are passed to the learning stage. We investigate a commercial tool for generating queries from articles, as well as an approach used in related literature [16], [19]. We further investigate two efficient heuristics for extracting queries. The goal is to extract queries (article-keyphrases) to maximize the retrieved number of tweets (a form of tweet Recall), as well as the content similarity of the retrieved tweets to that of the article (a form of tweet Precision). We do not constrain the target data to tweets that contain URLs, as done in most prior work. It is interesting to study the effect of different ways to generate queries, on the amount and quality of data collected, but we also want to find out how different trade-offs in Precision and Recall influence the final recommendation efficiency and quality.

We study four approaches for generating keyword queries from news articles:

1) **POS + Tf.idf:** Part-of-speech-tagging of the pseudo article. This method selects the first 5 nouns/phrases by giving priority to noun-phrases, proper nouns, frequent nouns, all other nouns. It then pairs the single nouns to 2-grams and breaks long noun phrases into 2-grams. These pairs are then ranked by the average tf.idf score of individual terms, and the top 5 pairs are selected as the query. This method is implemented by the model presented in [6].
2) **POS + NER + Tf.idf:** The second method follows a similar process as above, but adds one step of Named Entity Recognition (NER) after POS-tagging. We use the Stanford NER Tagger [36] to detect entities. When ranking all pairs by the average tf.idf score, pairs containing entities get a 1.5 boost. The aim is to focus more on key entities, rather than generic words. We select the top 5 keyphrases as the final query.
3) **AlchemyAPI**[2]: This is a commercial tool developed by IBM for extracting the most important keyphrases from a given article. Alchemy's keyword extraction algorithm employs sophisticated statistical algorithms and natural language processing technology to analyze the article content and identify the best keywords. Given the article's full content, AlchemyAPI returns a list of ranked keyphrases. We take the top 5 keyphrases as the query.
4) **Article URL:** Twitter Streaming API supports using URLs as keyphrases. It retrieves tweets containing both the full and shortened version of the target URL. By using the article URL as a query, we retrieve only relevant tweets that contain the article URL (as in [16], [19]). We expect these tweets to be cleaner regarding content, but very few.

Table 1 presents the keyphrases selected by each approach for an example article. In Section 4.2 we present an empirical study to evaluate the impact of each query type on the amount/quality of data collected, as well as how this influences the recommendation effectiveness.

TABLE 1
Example article and ranked article-keyphrases using 4 approaches.

Article Headline	Easyjet doubles number of female pilots
Subheadline	Easyjet says it has doubled the number of female pilots this year and is on the hunt for more.
First Sentence	The Amy Johnson initiative, named after the first female pilot to fly solo from the UK to Australia, caused a surge in applications.
POS + Tf.idf	(1) australia easyjet, (2) easyjet number, (3) easyjet uk, (4) australia number, (5) australia uk
POS + NER + Tf.idf	(1) amy johnson, (2) australia easyjet, (3) easyjet uk, (4) australia uk, (5) easyjet number
AlchemyAPI	(1) amy johnson initiative, (2) female pilots, (3) easyjet, (4) female pilot, (5) surge
URL	(1) bbc.com/news/business-38326523

3.2.2 Cold-Start Search Algorithms

In this section we describe methods to deal with **cold-start**: the challenge of quickly collecting data and providing recommendations for new articles. In our model, the extraction of dynamic features relies on a collection of relevant tweets for each article, collected via search over historical data or via streaming. When using only streaming, for each new article our system waits until sufficient tweets are gathered, before computing features and recommending hashtags to that article. Depending on the popularity of the news event on Twitter, the waiting time varies from a few minutes to a few hours. This is a problem, in particular in a journalistic setting where the editor cannot wait more than a few minutes for getting recommendations. To address this challenge we exploit the fact that many news are related to ongoing stories, locations or events, and can benefit from the data and recommendations collected for past articles.

2. http://www.alchemyapi.com/products/alchemylanguage/keyword-extraction

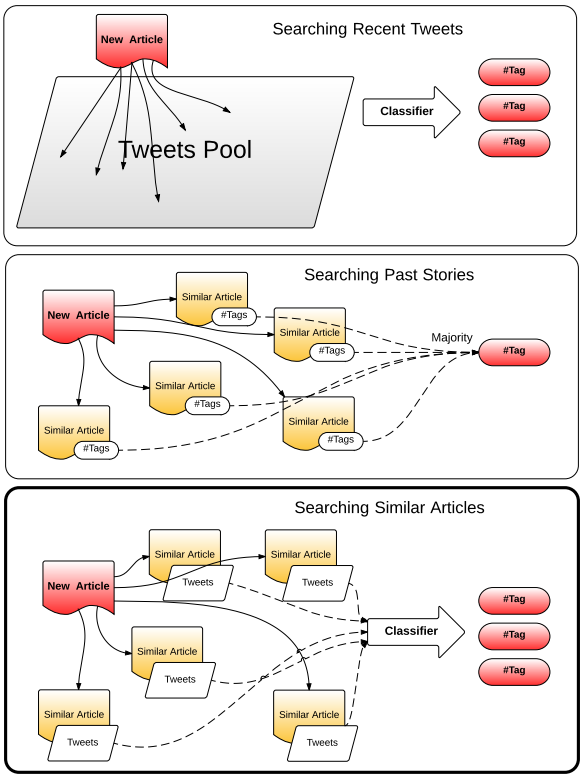


Fig. 4. Overview of proposed cold-start search methods.

There are two types of Twitter APIs for collecting tweets: Search and Streaming. Given current time t_n, the Twitter Search API retrieves tweets posted before t_n while the Streaming API retrieves tweets as they are posted on Twitter (after t_n). For a real-time system, we naturally prefer using existing tweets, rather than waiting for new tweets. However, the restricted rate limit of Twitter Search API (1 request per min) makes it infeasible to use. We therefore only use the Twitter Streaming API to retrieve tweets. For search, we study methods that exploit the historical tweets and articles collected by our system. We explore three ideas for using past data (visually sketched in Figure 4):

1) **Searching Recent Tweets:** Using the query extracted for a given article, this method searches the database for matching tweets and uses them for feature computation.
2) **Searching Past Stories:** This method first clusters old articles into stories, then assigns the new article to its most similar cluster and uses the majority hashtag for that cluster as a recommendation.
3) **Searching Similar Articles:** This approach searches the database of past articles for articles similar to the new article and uses their tweets for feature computation.

Searching Recent Tweets: This first approach uses the old tweets collected for previous news articles processed by our system. Given a time-window w (e.g., past 12h from current time point), this method searches in the old articles' tweet-bags for tweets that match any of the keyphrases of the new article. These old tweets are then added to the tweet-bag of the new article. After collecting a sufficient

number of relevant tweets (at least 100), we extract features from the tweet-bag and apply our L2R model. This method helps solve the cold-start problem by computing features and recommending hashtags as soon as a new article is received, without waiting for streamed tweets.

Algorithm 2 presents the steps of this method. We first find the subset of tweets T_w from the database D that falls within the time window w ($SubsetTweets(D, w)$). The complexity of this step is $O(|D_T|)$, where D_T is the number of tweets in the database. An efficiently indexed database can drastically reduce this step. Given a fixed length keyphrase query ($|Q_a| = 5$ keyphrases), the complexity of the keyphrase filtering step is $O(|T_w|)$. The overall complexity for Algorithm 2 is $O(|D_T| + |T_w|)$. Since T_w is in the scale of millions, Algorithm 2 can take a very long time in practice.

Searching Past Stories: The second method uses the past recommendation results. Given a collection of old articles, this method partitions them into clusters. One cluster is considered as one story. The average cosine similarity of an article to all other articles in the same cluster is a good indicator of how strongly the article is connected to the cluster. Each old article has few recommended hashtags and clusters are labelled by the most frequent hashtag in that cluster. A new article is then assigned to one of the clusters by content similarity, and the cluster hashtag is the hashtag recommended for the new article. We use k-means as the clustering algorithm. The runtime of our second approach is better than the first one, as it only considers a few thousand articles at one time. However, as news articles spread across many topics and stories, the number of clusters k can get large. Also, because the recommendation of this method relies on the quality of k-means clustering (an unsupervised method), its precision can be low.

In Algorithm 3, the function $SubsetArticles(D, w)$ has a time complexity of $O(|D_A|)$, where D_A is the set of articles in the database. The bottleneck of Algorithm 3 is the k-means clustering. We use the sklearn k-means implementation[3] which has an average time complexity of $O(A_w \times k \times d \times i)$, where A_w is the number of articles to be clustered, k is the number of clusters, i is the number of iterations and d is the number of terms in the articles. When the number of articles is large, the runtime of Algorithm 3 increases considerably.

Searching Similar Articles: The third method takes a collection of old articles, and uses a kNN classifier to find the k most similar articles to the new article. Then, using those articles' tweet-bags, it searches for tweets relevant to the new article by query matching, and feeds them to the feature computation module. By giving priority to the tweets of similar articles, this method dramatically prunes the amount of tweets to be filtered and it also increases the likelihood of finding relevant tweets. For this reason, the search time-window size can be extended to months, unlike the $12h$ restriction of the first method.

In Algorithm 4, the function $SubsetArticles(D, w)$ has a time complexity of $O(|D_A|)$, where D_A is the set of articles in the database. The complexity of $NearestNeighbour(A_w, a, 20)$ is $O(A_w \times |V|)$,where $|V|$ is the size of vocabulary for articles A_w. The query filtering has time complexity $O(|T_{pa}|)$, similar to Algorithm 2. Algorithm 4 is much more efficient than Algorithm 2 or 3, as kNN is much faster than k-means when given the same number of articles as input, and the size of T_{pa} is much smaller than T_w of Algorithm 2.

3. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Algorithm 2: ColdStart1: Searching Recent Tweets

Input: Query Q_a extracted from article a, database of past articles D.
Output: Relevant tweets T_{a_search}, relevant hashtags H_{a_search}.
Method:
$T_{a\_search} = \emptyset$
$w = 60 * 12$ //Timewindow for past tweets: 12h
$T_w = SubsetTweets(D, w)$ //Retrieve all tweets within timewindow w.
for $tweet \in T_w$ **do**
  for $phrase \in Q_a$ **do**
    if $phrase \in tweet$ **then**
      //If tweet contains the keyphrase $phrase$.
      $T_{a\_search} = T_{a\_search} \cup \{tweet\}$
      break
Return T_{a_search}, $\emptyset$

Algorithm 3: ColdStart2: Searching Past Stories

Input: Query Q_a extracted from article a, database of past articles D.
Output: Relevant tweets T_{a_search}, relevant hashtags H_{a_search}.
Method:
$H_{a\_search} = \emptyset$
$w = 60 * 24 * 30$ //Timewindow for past stories: 1 month
$A_w = SubsetArticles(D, w)$
$C = ClusterArticles(A_w, 0.1)$ //Cluster past articles into stories. The number of clusters is $0.1 \times |A_w|$.
$c = MostSimilarCluster(C, a)$ //Find the cluster of a
$H_c = \emptyset$
for $pa \in c$ **do**
  //For each past article in cluster
  $H_a = LastRecommededTag(pa, D)$
  $H_c = H_c \cup H_{pa}$
$H_{a\_search} = MostCommon(H_c)$ //Get the most common hashtag of the cluster
Return $\emptyset$, H_{a_search}

Algorithm 4: ColdStart3: Searching Similar Articles

Input: Query Q_a extracted from article a, database of past articles D.
Output: Relevant tweets T_{a_search}, relevant hashtags H_{a_search}.
Method:
$T_{a\_search} = \emptyset$
$w = 60 * 24 * 30$ //Timewindow for past articles: 1 month
$A_w = SubsetArticles(D, w)$
$N = NearestNeighbour(A_w, a, 20)$ //Find 20 nearest neighbours of article a.
$T_{a\_search} = \emptyset$
for $pa \in N$ **do**
  //For each past article in the neighbour set
  $T_{pa} = GetTweetBag(pa, D)$
  for $tweet \in T_{pa}$ **do**
    for $phrase \in Q_a$ **do**
      if $phrase \in tweet$ **then**
        //If tweet contains the keyphrase $phrase$
        $T_{a\_search} = T_{a\_search} \cup \{tweet\}$
        break
Return T_{a_search}, $\emptyset$

4 Evaluation

In this section we present extensive experiments analyzing the building blocks of Hashtagger+. We also compare to state-of-the-art methods and discuss the strengths and weaknesses of the methods compared.

4.1 Gathering Labeled Data

One cheap way to collect labeled data for training is to collect tweets that contain both article URLs and hashtags, and consider those hashtags as relevant labels. This is done

TABLE 2
Details of labeled article-hashtag pairs.

Total	Positive	Negative	Collection Period
1,238	348(28.1%)	890(71.9%)	04/12/2014-08/01/2015
Articles			Hashtags
217			725

in many related approaches, including [16], [20]. We found that such data is too little and too noisy. Most tweets with hashtags, although relevant to the article, do not contain the article URL. A quarter of the tweets with article URLs are tagged with general hashtags, #news #breaking, while many others are tagged with a mixture of relevant and irrelevant hashtags. For this reasons we decided to collect high quality labels by involving manual annotators and a Web application. The users can see the hashtags recommended by simple baselines for each article (e.g., most frequent hashtag, most similar hashtag content-wise) and provide feedback for each hashtag: if a hashtag is relevant or irrelevant for the article, at the time of labeling. We gathered around 1,200 labeled examples[4]; details are provided in Table 2. We use this data as ground truth for evaluating different approaches.

4.2 Query Generation

Experiment Setup. We collect 150 news articles and extract keyphrases using the four approaches presented in Section 3.2.1. We track these article-keyphrases for 12h, via the Twitter Streaming API, and for each article we gather four tweet-bags corresponding to the four methods. As the purpose of this step is to gather high quality candidate hashtags, we only consider tweets with at least one hashtag in the following evaluation. To estimate the precision of each approach, we compute the cosine similarity between the article and its tweet-bag tf.idf profile, then average over the 150 articles. This gives us an indication of the focus of the tweet-bag. To estimate the recall, we average the sizes of the tweet-bags (number of tweets per article) over all articles. Also, we average the number of unique hashtags appearing in the tweet-bags, and the average frequency of the hashtags.

Experiment Result. Table 4 shows the results of the four methods evaluated on the collected tweet-bags. The URL approach has the highest cosine similarity (0.265), but very few tweets (4.2), and only 1.5 candidate hashtags per article. The best approach is the commercial tool AlchemyAPI, which retrieves the most tweets (5K) and candidate hashtags (976), and the retrieved tweets are also very similar to the article content (0.246). The second best approach is POS+NER+Tf.idf. When compared to POS+Tf.idf, the tweets retrieved are fewer (3K vs 3.7K) but of higher quality (cosine similarity: 0.242 vs 0.221).

End-to-End Evaluation. To examine the impact of keyphrase extraction on the final hashtag recommendation quality, we conduct an end-to-end experiment where we use the tweets collected by the four approaches with our L2R framework (no search, only streaming) to recommend hashtags to the 150 target articles. We evaluate the recommendation result using *Precision@1 (P@1)*, article *coverage* (percentage of articles that get at least one hashtag recommended), and total *running time*. Table 3 shows the end-to-end evaluation results.

URL has the worst P@1 and article coverage, due to the insufficient amount of tweets. POS+NER+Tf.idf has the highest P@1, revealing that NER helps retrieve a more focused tweet-bag. Although AlchemyAPI has the highest tweet content similarity to the article, the P@1 of AlchemyAPI is not as high as expected. This is mainly because AlchemyAPI often extracts unigram keywords from the article, leading to noisier tweets. For instance, the keyword "game" is extracted from an article about a football game. When collecting tweets with "game", tweets about video games are also retrieved, and result in wrongly recommended hashtags: #game, #gamedev. This problem is avoided in POS+Tf.idf and POS+NER+Tf.idf which only use selected 2-grams. AlchemyAPI has 71.3% article coverage as compared to 63.3% of POS+NER+Tf.idf. On the other hand, AlchemyAPI retrieves more tweets and hashtags and therefore runs slower, as the speed of hashtag recommendation depends on the number of tweets in the tweet-bag. POS+NER+Tf.idf takes less than half the amount of time needed by AlchemyAPI (200s vs 588s). Overall, POS+NER+Tf.idf has the highest P@1, and shortest running time. We show later that its low article coverage can be addressed by the cold-start search step.

Key Take-away: Although AlchemyAPI, a commercial keyphrase extraction tool, results in more tweets and candidate hashtags retrieved, it does not result in the best end-to-end recommendation quality, due to the noise introduced by single keywords or long keyphrases. Additionally, the commercial API is restricted to a limited number of free calls each day. Our POS+NER+Tf.idf approach for generating queries results in better end-to-end recommendation quality and efficiency as compared to [6].

4.3 Learning-to-Rank Algorithm Selection

In this section we investigate many different L2R algorithms to study which are the most appropriate for our problem.

Experiment Setup. There are three main types of L2R approaches [8]: pointwise, pairwise and listwise. The RankLib[5] library implements 8 L2R algorithms (2 pointwise, 2 pairwise and 4 listwise methods), and is commonly used in L2R research. We also use the scikit-learn (sklearn)[6] library for 7 additional pointwise classifiers and Cornell's RankSVM[7] (pairwise) implementation. RankLib expects the training data in the format of RankSVM, where labeled data is organized by query lists, and within each list, labels are ranked from 0 to r, where 0 indicates the item is irrelevant to the query and a bigger r means the item is more relevant. We use 1.2K labeled pairs as ground-truth data, and transform them into the RankLib required format. We only have relevant and irrelevant labels, so we group the label pairs by articles, then rank the hashtags as relevant (1) and irrelevant (0). The transformed labeled data consists of 217 ranked lists (one list per article), with an average of 5.7 hashtags per list. We compare the performance of L2R algorithms by using 10-fold-cross-validation. For evaluation we use P@1 and NDCG@3, as in our data each article typically has less than 3 relevant hashtags.

Experiment Result. Table 5 shows the results for 16 L2R ranking methods. When compared to pairwise and listwise ranking algorithms, pointwise methods have higher P@1 and NDCG@3 (around 0.8), and shorter running time (around 2s). Among the 9 pointwise methods, RandomForest(sklearn) has the highest P@1 (0.8526) and NDCG@3 (0.8488) and acceptable training time (2.75s). Multilayer Preceptron, AdaBoost and Linear Regression have the second best P@1, NDCG@3 and running time. We tested two implementations of Random Forest (ranklib vs sklearn) and found that the sklearn implementation has better ranking performance, which is similar to prior findings[8].

For the pairwise methods, RankBoost does the best in ranking, but has the longest running time compared to RankNet, and RankSVM (15.67s, 7.45s, 2.05s). Among the four listwise methods, AdaRank is the fastest (2.53s) while LambdaMART is the slowest (54.48s). CoordinateAscent is

4. Labeled data and results available from https://gitlab.com/claireshi/hashtagger-plus-data

5. https://sourceforge.net/p/lemur/wiki/RankLib/
6. http://scikit-learn.org/stable/index.html
7. www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
8. sourceforge.net/p/lemur/discussion/ranklib/thread/1d30431d/

TABLE 3
P@1, article coverage, and running time of end-to-end hashtag recommendation using tweets collected using four query generation methods.

	L2R (POS + Tf.idf)	L2R (POS + NER + Tf.idf)	L2R (AlchemyAPI)	L2R (URL)
P@1	0.930	**0.947**	0.901	0.410
Coverage	67.3%	63.3%	**71.3%**	22.1%
Time	301s	**200s**	588s	48s

TABLE 4
Average cosine similarity, number of tweets, number of candidate hashtags and hashtag frequency using tweets collected using four query generation methods.

	POS + Tf.idf	POS + NER + Tf.idf	AlchemyAPI	URL
Cosine	0.221	0.242	0.246	**0.265**
Tweets	3696.2	2982.9	**5083.8**	4.2
Hashtags	529	442	**976**	1.5
Tag Freq	5.26	5.73	**5.81**	1.49

TABLE 5
Comparing the P@1, NDCG@3 and running time of 16 ranking methods using Ranklib, sklearn and Cornell's RankSVM.

L2R	Algorithm	P@1	NDCG@3	Time(s)
Pointwise	**RandomForest(sklearn)**	**0.852**	0.848	2.75
	MultilayerPerceptron(sklearn)	0.835	0.803	6.14
	SVM(poly)(sklearn)	0.823	0.827	0.78
	GradientBoosting(sklearn)	0.810	0.817	1.71
	LinearRegression(sklearn)	0.803	0.824	0.16
	AdaBoost(sklearn)	0.801	0.840	1.51
	RandomForest(ranklib)	0.792	0.778	2.01
	MART(ranklib)	0.783	0.768	49.87
	GaussianNaiveBayes(sklearn)	0.764	0.757	0.05
Pairwise	**RankBoost(ranklib)**	**0.774**	0.773	15.67
	RankSVM(cornell)	0.728	0.734	2.05
	RankNet(ranklib)	0.654	0.718	7.45
Listwise	**CoordinateAscent(ranklib)**	**0.778**	0.765	28.11
	LambdaMART(ranklib)	0.769	0.766	54.48
	ListNet(ranklib)	0.751	0.756	14.56
	AdaRank(ranklib)	0.737	0.749	2.53

the best listwise L2R method, with P@1 of 0.77. Across the three types of algorithms, the pointwise RandomForest (sklearn) has the highest P@1 and NDCG@3, with good running time. In the followup experiments, we use pointwise RandomForest for the relevance classifier.

Key Take-away. The common understanding in the IR community is that listwise approaches typically outperform pairwise and pointwise approaches. We have found that this is not the case in our setting. We work with binary relevance labels and high class imbalance (the number of relevant hashtags for each article is low, in the range of 1-3 hashtags). In this setting, we find that pointwise approaches have clear advantages regarding both ranking quality (higher precision) and efficiency (no need to pair hundreds of hashtags). Our finding is similar to work in [33] on question answering tasks with binary relevance judgements (answers are either relevant or irrelevant), where there are very few relevant answers (one correct answer for a given question, out of hundreds of candidate answers). The authors in [33] found that for this type of ranking problem, pointwise methods outperform pairwise and listwise ones. We have similar findings from our extensive experiments with 16 L2R approaches.

4.4 Cold-Start Algorithms Evaluation

To evaluate and compare our cold-start approaches we gathered 150 test articles published between 8am-12pm Nov 09, 2016. We first study the influence of parameter settings for each method, and then compare them using their best set of parameters.

4.4.1 *Searching Recent Tweets*

The first approach reuses the tweets collected for older articles and has only one parameter: the size of the time window for collecting tweets.

Parameter Tuning: Time-window Size. The length of the search time-window decides the amount of historical tweets that will be used for filtering and affects the time this method needs for making a recommendation. For each test article, we retrieve tweets using different time-window sizes: 1h, 4h, 12h, 24h, 36h. There are about 500K historical tweets with hashtags collected between 8pm Nov 07 to 12pm Nov 09, 2016 (around 36h before the articles publishing time). Then, we apply our hashtag recommendation approach over the collected tweets and measure the recommendation quality when using different time-window sizes, by P@1, article coverage, total running time (search time plus recommendation time), and the average number of tweets retrieved for the 150 test articles.

TABLE 6
Time-window Size: P@1, Article Coverage, and total running time of hashtag recommendation using recent tweets.

	1h	**4h**	**12h**	**24h**	**36h**
Tweets	49.8	258.1	1179.1	2247.4	3221.1
P@1	0.820	0.801	0.824	0.883	0.850
Coverage	26.0%	41.3%	48.0%	57.3%	60.0%
Time	71s	132s	261s	456s	639s

Results. As shown in Table 6, longer searching time-windows allow retrieving more relevant tweets and thus lead to better recommendation coverage. Unfortunately, the running time also increases with the window size (51s for 1h vs 339s for 36h). The time-window size has small impact on P@1, which is around 0.85. This shows the robustness of our hashtag recommendation approach to the number of tweets. Considering the article coverage and running time trade-offs, we choose 12h as the search time-window in the follow-up experiments.

4.4.2 *Searching Past Stories*

The second method uses k-means clustering and past recommendations. The performance of this method depends on two factors: the number of clusters and the size of the time-window for historical articles used for clustering. We test the number of clusters and conduct an end-to-end study to measure the final hashtag recommendation quality.

Parameter Tuning: Number of Clusters. Given the total number of articles N, we vary the number of clusters c in the range $sqrt(N)$ and $0.1N$ to $0.9N$. The value $sqrt(N)$ is a common way to set the number of clusters, while the other values use a proportion of the article collection size N. As in the previous experiment, we use the 150 test articles. We fix the time-window size to 1 month, resulting in about 13.3K historical articles published at most 1 month before Nov 09, 2016. We cluster the articles into c clusters. For each test article, we allocate it to the closest cluster and recommend the most frequent hashtag of that cluster.

Results. As shown in Table 7, the number of clusters has a complex impact on the hashtag recommendation quality. In general, having more clusters leads to better article coverage, but lower P@1. If the number of clusters is small (e.g., $c = sqrt(N)$), both P@1 and article coverage are low, because articles of different news stories are grouped together, and test articles are less likely to find relevant clusters. Considering the overall performance, we set $c = 0.1N$ in the follow-up experiments.

Parameter Tuning: Time-window Size. Here we test what is a suitable time scope for the articles used for clustering, given that we fix the number of clusters to $c = 0.1N$. Longer time scope covers more long-term news stories, but also retrieves more articles. We test the following time span:

7 days, 14 days, 1 month, 2 months, 3 months, and use the same experiment setup as before, just that we include more articles for clustering.

Results. Table 8 shows that using articles from the past 1 month delivers the best P@1 and good coverage and running time. We fix $c = 0.1N$ and the time-window to 1 month in the follow-up experiments.

4.4.3 *Searching Similar Articles*

This method uses a kNN classifier to find related news articles and uses their tweet-bags for feature computation for the L2R module.

Parameter Tuning: Number of Neighbors. We first evaluate the number of nearest neighbors k retrieved for test articles. We vary $k = 5, 10, 15, 20$, and fix the time-window size of historical articles to 1 month. As in the previous experiment, we use 150 test articles. We find the k most similar articles to each test article, and from these articles' tweet-bags select tweets based on the article query. We then feed these tweets to the L2R module to get recommendations.

Results. Table 9 shows that increasing k leads to finding more relevant tweets, and to an increase in the article coverage and running time. The change of k has no obvious impact on P@1 showing the robustness of our hashtag recommendation approach. We chose $k = 20$ in the follow-up experiments.

Parameter Tuning: Time-window Size. Similar to the second method, the time scope of historical articles refers to how far back we search for nearest neighbors. Longer time scope covers more long-term news stories, but concept drift can have a negative impact on the recommendation quality. We evaluate the following time span: 7 days, 14 days, 1 month, 2 months, 3 months, and use $k = 20$ for kNN.

Results. Table 10 shows that the increase of time scope leads to finding less relevant tweets, showing recent articles are more relevant for the test articles. However, when the time scope is set to 1 month, the article coverage reaches its maximum (67.3%), indicating it is a good trade off between short-term vs long-term news stories.

4.4.4 *Comparing the Tuned Cold-start Methods*

We compare the performance of the three cold-start methods using their best tuned parameters, and as a baseline, we include a no cold-start approach.

Experiment Setup. We use the same 150 test articles and compare the impact of using different cold-start search strategies, and their combination with streaming. For each article published at t_a, besides searching, we also collect tweets until $t_a + 24h$, using the article query and the Twitter Streaming API.

1) **No Coldstart.** Baseline approach that has no cold-start solution, only streaming. This is similar to the recommendation approach presented in [6].

TABLE 7
Number of Clusters: P@1, Article Coverage, and total running time of hashtag recommendation using past articles. Time-window size set to 1 month.

	sqrt(N)	0.1N	0.3N	0.5N	0.7N	0.9N
Clusters	115	1330	3991	6652	9313	11974
P@1	0. 531	0.775	0.636	0.601	0.559	0.664
Coverage	48.6%	65.3%	95.3%	95.3%	95.3%	85.3%
Time	41s	101s	121s	187s	237s	265s

TABLE 8
Time-window Size: P@1, Article Coverage, and total running time of hashtag recommendation using past articles. The number of clusters set to $c = 0.1N$.

	7 days	14 days	1 month	2 months	3 months
Clusters	299	570	1330	2937	3917
P@1	0. 621	0.702	0.775	0.760	0.760
Coverage	51.3%	56.0%	65.3%	67.3%	66.7%
Time	3s	15s	101s	320s	653s

TABLE 9
Number of Neighbors: P@1, Article Coverage, and total running time of hashtag recommendation using past articles and tweets. Time-window size set to 1 month.

	5	10	15	20
Tweets	465.5	695.9	860.1	997.6
P@1	0.940	0.933	0.949	0.940
Coverage	57.3%	61.3%	66.0%	67.3%
Time	115s	139s	176s	199s

TABLE 10
Time-window Size: P@1, Article Coverage, and total running time of hashtag recommendation using past articles and tweets. The number of neighbors is set to $k = 20$.

	7 days	14 days	1 month	2 month	3 month
Tweets	1,344	1,071	997	832	835
P@1	0.897	0.930	0.941	0.900	0.948
Coverage	58.6%	58.6%	67.3%	64.0%	64.6%
Time	195s	196s	199s	191s	195s

2) **Searching Tweets.** Given an article published at t_a, we retrieve tweets posted in time window $[t_a - 12h, t_a]$, and filter them by the article query. After the first recommendation round, streaming is used to enhance the tweet-bags of articles, followed by L2R to update the recommendations.
3) **Clustering Articles.** We cluster 1 month worth of historical articles (13.3K) with the number of clusters set to $c = 0.1N = 1,330$. This approach uses the past recommendations to recommend hashtags to new articles. For a new article to be assigned to a cluster, we set a minimum similarity threshold (thus clustering will not automatically lead to 100% coverage). After the first recommendation round (at 1min), in the subsequent rounds (e.g., after 5mins of streaming) we keep the clustering recommendations if the L2R module has not provided a new recommendation, but update the recommendation if any was provided by the L2R module. This ensures a smooth transition from the cluster-based recommendations, to the L2R-based ones.
4) **Searching Articles (KNN).** The number of neighbors for kNN is set to $k = 20$, the time-window is set to 1 month, and the most recent 1K tweets from the articles' tweet-bags are filtered by the query of the new article. After the first recommendation round, the streaming will update the tweet-bags and the L2R module will update the recommendations.

The recommendation output of each of the four approaches is manually labeled as relevant/irrelevant.

Results An ideal cold-start solution should maintain high-precision, increase the article coverage rate and reduce the time needed to provide recommendations to new articles. Therefore, we evaluate the final recommendation via P@1 and article coverage at different time cut-offs, to measure how quickly each method can return high-quality recommendations. The P@1 and coverage results are shown in Figure 5 and Figure 6. Within 2s after the article publishing time, all three cold-start methods return recommended hashtags for at least 50% of test articles (i.e., have at least 50% coverage). In comparison, No Coldstart only provides recommendations to less than 1% of articles. The methods that use L2R for recommendation have high precision (above 0.8) from the first round of recommendation; in comparison the precision of clustering is low, around 0.7. The No Coldstart baseline, within 1min, only produces very few recommendations (only 2 articles out of 150 get any recommendations), but with high-precision (100%). Of the three cold-start methods, Clustering Articles provides good coverage (starting at 70%), but low precision. Searching Tweets has relatively low coverage (starting at 50%) and good precision (P@1 of 0.8).

The KNN Articles method achieves good coverage from the start (67% coverage after 1min), with P@1 around 0.9.

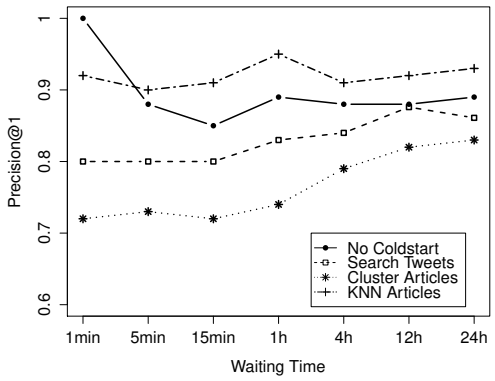


Fig. 5. Comparing the P@1 of three cold-start methods and a no cold-start baseline.

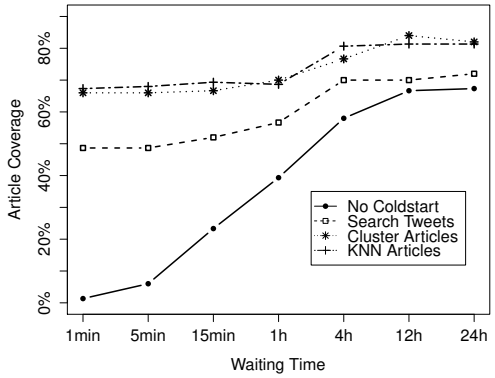


Fig. 6. Comparing the article coverage of three cold-start methods and a no cold-start baseline.

All search methods benefit from follow-up streaming which improves the article tweet-bags, leading to better features for the L2R module. After 24h streaming, the No Cold-start method reaches 67% coverage and P@1 of 0.87. The best cold-start method, KNN Articles, combined with 24h streaming, achieves a much better coverage of 80% and P@1 above 0.9, therefore dramatically improving the article coverage of [6].

Key Take-away. Combining search and streaming for collecting features for the L2R module leads to efficient, high-precision, high-coverage, hashtag recommendation. Among the cold-start approaches, the KNN Articles method achieves the best trade-off regarding speed, precision and coverage. One interesting lesson learned from comparing different cold-start approaches is that, for new articles to get good recommendations, they don't need to be part of an existing story. It is enough if the new article covers similar locations, events or entities, for the KNN Articles to retrieve relevant tweets, and for the L2R module to provide relevant recommendations.

4.5 Comparison to the State-of-the-Art

In this section we compare our model, **Hashtagger+**, to state-of-the-art hashtag recommendation techniques. We group the compared methods into MCC and L2R types. We compare the following methods (all of them use tweets collected using our POS+NER+Tf.idf query generation method):

MCC Methods: These are methods that use the hashtagged tweets as training data. Each hashtag is a class, and tweets with those hashtag are used as training data for that class. These methods need to wait to accumulate training data and, to cope with concept drift, need to be retrained often. To adapt them to a real-time setting, we retrain them using a sliding window. Two parameters affect these methods: the time-window for training data (e.g., 4h of recent tweets as training) and time before retraining the classifier (e.g., retrain every 5mins, using the 4h past tweets as training). For each model, these two parameters are determined by the time needed to train the model. For example, if we increase the training data from 2h to 4h of tweets, this leads to an increase in training time, therefore it constrains how often the model can be re-trained.

1) **NaiveBayes:** [9] We select NaiveBayes due to its speed and evaluate it with different training size (e.g., 2h, 4h, 6h of recent tweets as training data) and retraining time (e.g., every 5mins, 10mins, etc.). We select the parameters that result in the best P@1 for this classifier: past 4h tweets for training and retraining every 5mins.
2) **Liblinear SVM:** The LibLinear classifier [39] supports millions of instances and features. and is popular due to its efficiency and high accuracy. Liblinear takes longer to run than NaiveBayes, but is more accurate. The parameters resulting in the best P@1 for Liblinear are: 2h training tweets and retraining every 30mins.
3) **MultilayerPerceptron:** To also evaluate a non-linear classifier, we use the neural network model implemented in sklearn. This approach is the slowest among the MCC methods compared. It needs 1h training time for 1h of tweets. Increasing the training data (e.g., to 4h of tweets) requires a much longer time to train (more than 4h) making frequent retraining infeasible. After trying different parameters, we set 1h for training data and retraining every 1h.

L2R Methods: These are methods that are trained only once (on a few manually labeled examples, as in [6] or on hashtagged tweets, as in [16]), and can be re-used for recommendation without the need to re-train.

1) **PairwiseL2R:** The method presented in [16] uses pairwise L2R and is trained on hashtagged tweets. We could not obtain the original code from the authors, but we implemented their method following [16]. We train a pairwise RankSVM on 4h of tweets that contain hashtags and URLs, as required by this method (there are 3K unique URLs, 23K training examples). As in [16], for each given test article, we use the tf.idf score to find the most similar 50 tweets and 50 articles from the 4h of tweets posted before the test article, and apply the trained model on the resulting candidate hashtags. The training size is constrained by the time needed to find similar tweets and articles (4h of tweets require 5min for getting recommendations). We compute 4 binary features for each candidate hashtag: (1) hashtag in the article headline, (2) URL is a popular domain (e.g., bbc/rte), (3) hashtag matches the domain of the article URL, (4) hashtag is popular (i.e., most frequent 100 hashtags).
2) **Hashtagger:** This is a pointwise L2R method presented in [6] which is trained on manually labeled article-hashtag pairs, and does not use cold-start algorithms for data collection. This method needs to wait to gather enough tweets for providing recommendations, and the waiting time varies between 15mins and 12h.
3) **Hashtagger+**: The method proposed in this paper uses pointwise L2R trained on manually labeled article-hashtag pairs and cold-start algorithms to deliver recommendations fast and with high coverage. This method delivers recommendations within seconds.

As evaluation metrics we use the article coverage and P@1 (by recommending the maximum score prediction of each

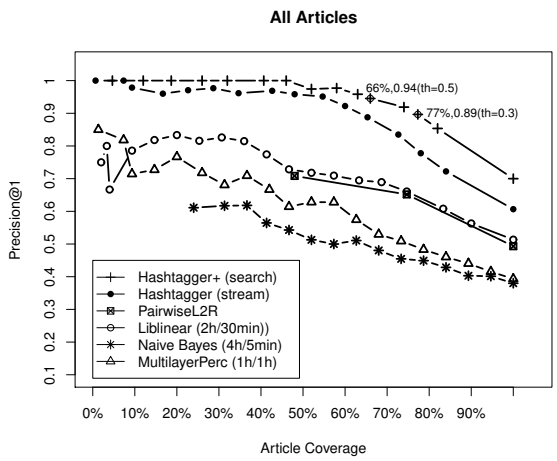


Fig. 7. P@1 and article coverage of all methods compared.

method). Both of these metrics are a function of the recommendation score. By setting a threshold on this score, we can study the sensitivity of each method. Setting a higher threshold for selecting a recommendation, results in better P@1, but less coverage, since recommendations that do not pass the threshold are not selected. The scores of all methods are mapped to the [0,1] interval using the logit function.

Experiment Setup. We randomly pick a starting time point t_0 (Nov 18th, 8am, 2016, UTC), then run the experiment for 6h, involving 150 articles and 400K tweets that have at least one hashtag. Each pseudo article (headline, sub-headline and first sentence) is considered as a rich tweet, and each method recommends one hashtag to each article.

Experiment Result. A group of annotators evaluated the resulting 900 test article-hashtag pairs as relevant/irrelevant and we averaged their results. For each method, we change the threshold to each unique predicted value in increasing order, and record the article coverage rate and the P@1 at that threshold, as shown in Figure 7. We observe the following behavior of the compared methods.

MCC Methods: The NaiveBayes, Liblinear, MultilayerPerc have a lower P@1 than the L2R methods, across all coverage thresholds. We believe the reasons for this behavior are as follows:

- Having one class per hashtag means that we have to train thousands of classifiers and thus we need to learn a much more complex decision boundary.
- MCC modeling leads to issues regarding training data size and quality. Regarding training size, MCC methods need to wait to collect enough training data to build robust classifiers. Furthermore, hashtagged tweets are a very noisy source of training data, since Twitter users do not always reliably tag tweets, e.g., they mix irrelevant, general and specific hashtags.
- Hashtags are very dynamic, many of them disappear and new ones emerge in a matter of minutes. MCC methods need to retrain often to deal with such dynamic classes, and are therefore less robust to concept drift.

L2R Methods: The pairwise L2R method proposed in [16] has a lower P@1 as compared to the other L2R methods, Hashtagger and Hashtagger+. We believe there are three main reasons for this. First, as for the MCC methods, the training data size and quality is problematic for this approach. PairwiseL2R of [16] requires tweets with hashtags and URLs for training. As above, hashtagged tweets are not a good source of labeled data, since the labels can be very noisy. Additionally, tweets containing URLs are very few, as compared to all the tweets discussing a story. Second, the feature engineering of this approach is not appropriate, as it uses cheap to compute, but weak features to describe hashtag relevance. Finally, we have shown in our experiments that pairwise L2R with RankSVM, which is used in [16], is not as accurate as RandomForest with pointwise L2R, since the goal is to learn a good threshold on hashtag relevance, rather than learning pairwise preferences for irrelevant hashtags. The second L2R baseline, our prior work Hashtagger [6], delivers high P@1, since it is trained with high quality labeled data, uses complex feature engineering and pointwise L2R. It nevertheless suffers from cold-start issues, as it has to wait to collect enough data for computing features for new articles. Better techniques for data collection in Hashtagger+, lead to higher P@1 across all coverage points. In [6] we report a P@1 of 0.89 (for $th = 0.5$) and coverage of 66%. In comparison, Hashtagger+ delivers P@1 of 0.94 (for $th = 0.5$) and coverage of 66%. If we set the threshold for Hashtagger+ to get the same P@1 of 0.89 as in [6], we achieve a coverage of 77%, a 10% improvement in coverage. Regarding efficiency, Hashtagger+ delivers recommendations in under 1min, while Hashtagger needs to wait an average of 80mins to deliver the first recommendations.

4.5.1 Niche versus Popular Articles

We refine the analysis of all compared methods to observe their capacity to provide good recommendations for articles that are popular versus articles that are more niche. We define a popular article to be one for which we can retrieve at least 300 matching tweets, resulting in 50 niche and 100 popular articles. We want to test the hypothesis that MCC methods trained with hashtagged tweets are not able to deal well with niche articles, as they do not have enough content to train from. We expect L2R methods to be robust even with less data available for computing features. Figure 8 shows this comparison for popular and niche articles. We note that for popular articles, MCC methods behave well, although their quality is well below that of L2R methods. For niche articles the advantage of L2R methods is clear. Although niche articles only have at most 300 tweets discussing the article, our method can also deliver good recommendations for these cases. For a coverage of 70% of niche articles, previous methods have P@1 of 0.5, while Hashtagger+ has P@1 of 0.8.

Key Take-away: Hashtagger+ can deliver good recommendations for popular as well as niche news stories. Our model advances the state-of-the-art by efficiently delivering high-precision, high-coverage recommendations (under 1min, P@1=0.89, coverage=77%).

5 Applications

In this section we study an application of our hashtag recommender to story tracking.

5.1 Story Tracking

News editors curate and continuously update collections of news articles to give readers an overview and updates on particular issues, e.g., referendums, elections, budgets. Preparing these story-pages relies on manually tagging documents with pre-agreed keyphrases. For example, *"brexit"* was used for the BBC story-page on the "UK leaves the EU"[9]. Our recommender method enables us to automatically link articles and Twitter hashtags and to index articles using *keywords and hashtags*. This means that we can formulate queries that mix keywords and hashtags, such as *EU crisis, #brexit*. Using Hashtagger+ and the Twitter crowd, we annotate stories with high quality social tags, in real-time, potentially capturing novel emerging concepts (e.g., #brexit, #remain, #leave). We coin this **social indexing**, and investigate its impact on a story tracking application.

9. http://www.bbc.com/news/politics/uk_leaves_the_eu

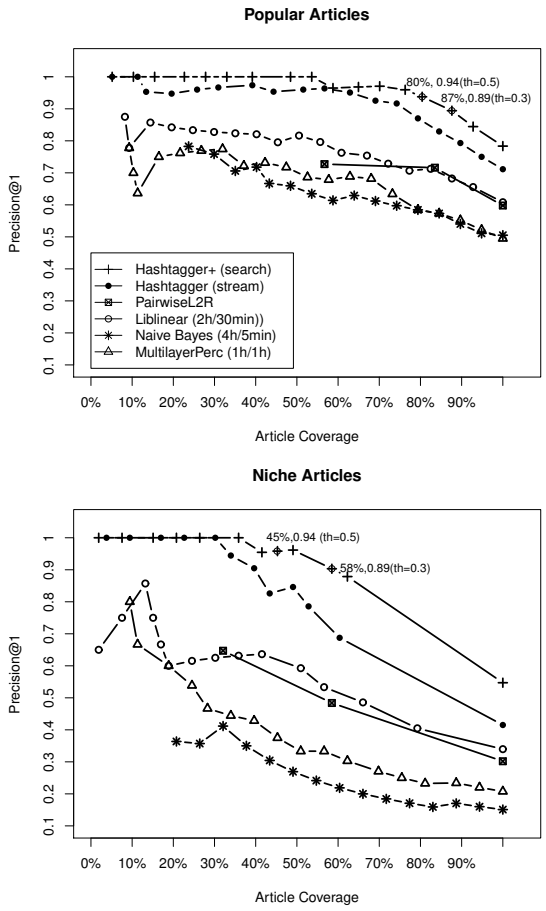


Fig. 8. P@1 and article coverage for popular versus niche articles.

TABLE 11
The number of articles collected from 16th June 2016 to 1st July 2016.

BBC EU Referendum Story Page					
130					
BBC	Reuters	Irish Times	Independent	RTE	The Journal
2,141	1,409	2,362	3,100	601	760

5.1.1 Retrieving Key Articles for the EU Referendum Story

As a target story, we select the EU membership referendum held in UK in June 2016, also known as Brexit. BBC created a dedicated story-page[10], listing key articles for the story and updating it hourly. Even with the help of key-phrase indexing, the story-page still relies on experienced journalists to select the important articles, from a daily pool of a few thousand story-related articles. We aim to automate story-page construction by using hashtags as a tool for retrieving story-related articles.

Experiment Setup. We use the BBC's EU referendum story-page as ground-truth. All articles that appear anytime on this page are considered as key articles for the story. We run a crawler and collect 130 ground-truth articles from 16th June 2016 to 1st July 2016 (one week ahead and one week after the referendum day). During the same time period, we collect a large set of articles from the RSS feeds of 6 news organizations, around 650 articles per day, 10.3K articles in total. To make sure all ground-truth articles are in our article set, the ground-truth articles that are not in the BBC RSS feeds, are also crawled and added to the set. Stats of the article collection are shown in Table 11. Once retrieved from RSS feeds or crawler, articles are tracked for 12h, during which the Hashtagger+ recommendations are updated every 15 min.

10. http://www.bbc.com/news/politics/eu_referendum

TABLE 12
Usage of five EU referendum hashtags for the tracked period.The total number of articles from six news sources and from BBC with this hashtag recommended by Hashtagger+ over 12h.

	Tweets	Articles	BBC Articles
#Brexit	790,872	1,035	234
#EURef	176,523	249	96
#EU	93,414	903	231
#VoteLeave	92,687	38	15
#Remain	83,667	67	21

TABLE 13
Comparison of keyword query baselines and hashtag indexing, via the retrieved number of BBC articles and the ground-truth articles.

		BBCArticles	KeyArticles	Precision	Recall	F1
Keyword Query	EURef+Full	266	91	0.34	0.70	0.46
	Brexit+Full	330	68	0.20	0.52	0.29
	EURef+Brexit+Full	**479**	**118**	**0.24**	**0.90**	**0.38**
	EURef+Title	198	81	0.41	0.62	0.50
	Brexit+Title	223	28	0.13	0.21	0.16
	EURef+Brexit+Title	**382**	**100**	**0.26**	**0.77**	**0.39**
Hashtag Query	#Brexit	234	68	0.29	0.52	0.37
	#EURef	96	52	0.54	0.4	0.46
	#EU	231	72	0.31	0.55	0.40
	#VoteLeave	15	13	0.86	0.10	0.18
	#Remain	21	11	0.52	0.08	0.14
	All Five Hashtags	**351**	**121**	**0.34**	**0.93**	**0.50**

We use both Hashtagger+ and the system described in [40], which expands a plain keyword query to a query with keywords and hashtags, to retrieve all relevant hashtags for a story. For the query "EU referendum", this approach returns related hashtags *#Brexit*, *#EURef*, *#EU*, *#VoteLeave*, and *#Remain*. These five hashtags are the most used in the discussions relevant to the EU referendum on Twitter. Each of them has generated a huge amount of traffic, as shown in Table 12. Both #Brexit and #EURef are specifically used for this story, but because #Brexit is more self-explanatory, it is popular in both mainstream and social media. #VoteLeave and #Remain are the two hashtags used by the opposite camps on social media, and are less used in mainstream media. For the baselines, we use plain keyword queries "EU Referendum" (EURef) and "Brexit" for retrieving articles using either the full article body (Full) or only the headline and subheadline (Title). The article title is a short text that describes the news article, however, the lack of certain keywords may hurt the retrieval recall. The article body contains rich information and should improve the recall, but may result in lower precision compared to a retrieval on title alone.

Experiment Result. To measure the performance of using hashtags for indexing and retrieving key articles for the story, we use standard IR metrics: Precision, Recall and F1. Since in this experiment we are focusing on retrieving most of the story key articles, the approaches with high Recall are preferred. Table 13 presents the full results of the baselines and the hashtag indexing approaches, by comparing their retrieved articles to the ground-truth. The top 3 approaches with highest Recall are AllFiveHashtags, EURef+Brexit+Full, and EURef+Brexit+Title. Among them, our proposed hashtag indexing has the highest recall (121 out of 130), followed by the query on full article content (118 out of 130), and the query on article headlines (100 out of 130). Although the hashtag indexing and full article keyword query approaches have similar Recall, the hashtags allow focusing on a much smaller pool of articles (351 vs 479), without compromising the Recall. Among the five hashtag indexes, #Brexit and #EU are the two hashtags that retrieve most of the key story articles (68 vs 72), and the intersection between their retrieved key articles is also large, 47 articles, meaning they are commonly used together and are interchangeable during that period of time. On the other hand, #VoteLeave and #Remain are more distinct from each other and from the other 3 hashtags. We find that the two groups of hashtags indexed two types of articles:

information-type and opinion-type articles. Thus hashtags can capture different aspects of the news story and are a good way to organize the news articles.

6 Conclusion

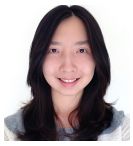
In this paper we have presented **Hashtagger+**, an approach for efficient, high-coverage *real-time hashtag recommendation* for streaming news. Our work has advanced the state-of-the-art by proposing an L2R model together with a set of efficient algorithms for data collection and feature computation. We have presented a detailed breakdown and analysis of our model, and provided an extensive empirical study of each building block. We showed that pointwise L2R approaches vastly outperform content-based and pairwise/listwise L2R approaches for real-time hashtag recommendation. Finally, we showed that L2R approaches behave better for recommending hashtags to niche news articles, a setting where most other approaches do not perform well due to lack of data for robust feature computation. We have showcased the value of our recommendations in an application to automated story tracking. By efficiently recommending hashtags to news article with high-precision (P@1 above 90%) and high-coverage (80% of articles get recommended hashtags), we believe we can address very interesting problems in text mining, ranging from news story detection and tracking, to entity linking. We intend to explore these research problems in our future work.

Acknowledgments

This work was funded by Science Foundation Ireland (SFI) under grant number 12/RC/2289. We would like to thank the reviewers for their helpful feedback.

References

[1] T. Rosenstiel. Twitter and the news: How people use the social network to learn about the world, American Press Institute, 2015.
[2] Z. Ma, A. Sun, Q. Yuan, and G. Cong, "Tagging your tweets: A probabilistic modeling of hashtag annotation in twitter," in *CIKM*, 2014.
[3] T.-A. Hoang-Vu, A. Bessa, L. Barbosa, and J. Freire, "Bridging vocabularies to link tweets and news.", in *WebDB*, 2014.
[4] Y. Gong, Q. Zhang, and X. Huang, "Hashtag recommendation using dirichlet process mixture models incorporating types of hashtags."
[5] Y. Gong and Q. Zhang, "Hashtag recommendation using attention-based convolutional neural network," in *IJCAI*, 2016.
[6] B. Shi, G. Ifrim, and N. Hurley, "Learning-to-rank for real-time high-precision hashtag recommendation for streaming news," in *WWW*, 2016.
[7] K. S. Hasan and V. Ng, "Automatic keyphrase extraction: A survey of the state of the art," in *ACL*, 2014.
[8] T.-Y. Liu, "Learning to rank for information retrieval," in *Foundations and Trends in Information Retrieval*, 2009.
[9] R. Dovgopol and M. Nohelty, "Twitter hash tag recommendation," *arXiv preprint arXiv:1502.00094*, 2015.
[10] A. Mazzia and J. Juett, "Suggesting hashtags on twitter," *EECS 545 Project*, 2011.
[11] F. Godin, V. Slavkovikj, W. De Neve, B. Schrauwen, and R. Van de Walle, "Using topic models for twitter hashtag recommendation," in *WWW*, 2013.
[12] Z. Ding, X. Qiu, Q. Zhang, and X. Huang, "Learning topical translation model for microblog hashtag suggestion," in *IJCAI*, 2013.
[13] Z. Huang, Q. Ding, and X. Zhang, "Automatic hashtag recommendation for microblogs using topic-specific translation model," in *COLING*, 2012.
[14] J. She and L. Chen, "Tomoha: Topic model-based hashtag recommendation on twitter," in *WWW*, 2014.
[15] Q. Zhang, Y. Gong, X. Sun, and X. Huang, "Time-aware personalized hashtag recommendation on social media." in *COLING*, 2014.
[16] S. Sedhai and A. Sun, "Hashtag recommendation for hyperlinked tweets," in *SIGIR*, 2014.
[17] F. Xiao, T. Noro, and T. Tokuda, "News-topic oriented hashtag recommendation in twitter based on characteristic co-occurrence word detection," in *Web Engineering*, 2012.
[18] O. Phelan, K. McCarthy, and B. Smyth, "Using twitter to recommend real-time topical news," in *RecSys*, 2009.
[19] M. Tsagkias, M. de Rijke, and W. Weerkamp, "Linking online news and social media," in *WSDM*, 2011.
[20] T. Gruetze, G. Yao, and R. Krestel, "Learning temporal tagging behaviour," in *WWW*, 2015.
[21] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles, "Real-time automatic tag recommendation," in *SIGIR*, 2008.
[22] X. Si and M. Sun, "Tag-lda for scalable real-time tag recommendation," *JCIS*, 2009.
[23] P. Li, Q. Wu, and C. J. Burges, "Mcrank: Learning to rank using multiple classification and gradient boosting," in *NIPS*, 2007.
[24] R. Nallapati, "Discriminative models for information retrieval," in *SIGIR*, 2004.
[25] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005.
[26] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "Frank: a ranking method with fidelity loss," in *SIGIR*, 2007.
[27] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *JMLR*, 2003.
[28] J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," in *SIGIR*, 2007.
[29] C. Quoc and V. Le, "Learning to rank with nonsmooth cost functions," *NIPS*, 2007.
[30] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML*, 2007.
[31] J. Fürnkranz and E. Hüllermeier, "Pairwise preference learning and ranking," in *ECML*, 2003.
[32] D. Sculley, "Combined regression and ranking," in *SIGKDD*, 2010.
[33] A. Agarwal, H. Raghavan, K. Subbian, P. Melville, R. D. Lawrence, D. C. Gondek, and J. Fan, "Learning to rank for robust question answering," in *CIKM*, 2012.
[34] L. Hang, "A short introduction to learning to rank," *IEICE TRANSACTIONS on Information and Systems*, 2011.
[35] C. Castillo, M. El-Haddad, J. Pfeffer, and M. Stempeck, "Characterizing the life cycle of online news stories using social media reactions," in *CSCW*, 2014.
[36] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL*, 2005.
[37] J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec, "Can cascades be predicted?" in *WWW*, 2014.
[38] N. Naveed, T. Gottron, J. Kunegis, and A. C. Alhadi, "Bad news travel fast: A content-based analysis of interestingness on twitter," in *International Web Science Conference*, 2011.
[39] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, "LIBLINEAR: A library for large linear classification," *JMLR*, 2008.
[40] G. Poghosyan, M. A. Qureshi, and G. Ifrim, "Topy: Real-time story tracking via social tags," in *ECML/PKDD*, 2016.



Bichen Shi is a PhD student at the Insight Centre for Data Analytics, School of Computer Science, University College Dublin, Ireland. Her research focuses on scalable machine learning, social data analysis and information retrieval.

Gevorg Poghosyan is a PhD student at the Insight Centre for Data Analytics, School of Computer Science, University College Dublin, Ireland. His research focuses on topic detection and tracking.

Georgiana Ifrim is an assistant professor and funded investigator at the Insight Centre for Data Analytics, School of Computer Science, University College Dublin, Ireland. Her research focuses on scalable machine learning and data mining.

Neil Hurley is an associate professor and principle investigator at the Insight Centre for Data Analytics, School of Computer Science, University College Dublin, Ireland. His research focuses on machine learning, social network analysis and recommender systems.