Using paraphrases for improving first story detection in news and Twitter

Saša Petrović
School of Informatics
University of Edinburgh
sasa.petrovic@ed.ac.uk

Miles Osborne
School of Informatics
University of Edinburgh
miles@inf.ed.ac.uk

Victor Lavrenko
School of Informatics
University of Edinburgh
vlavrenk@inf.ed.ac.uk

Abstract

First story detection (FSD) involves identifying first stories about events from a continuous stream of documents. A major problem in this task is the high degree of lexical variation in documents which makes it very difficult to detect stories that talk about the same event but expressed using different words. We suggest using paraphrases to alleviate this problem, making this the first work to use paraphrases for FSD. We show a novel way of integrating paraphrases with locality sensitive hashing (LSH) in order to obtain an efficient FSD system that can scale to very large datasets. Our system achieves state-of-the-art results on the first story detection task, beating both the best supervised and unsupervised systems. To test our approach on large data, we construct a corpus of events for Twitter, consisting of 50 million documents, and show that paraphrasing is also beneficial in this domain.

1 Introduction

First story detection (FSD), sometimes also called new event detection (NED), is the task of detecting the first story about a new event from a stream of documents. It began as one of the tasks in Topic Detection and Tracking (TDT) (Allan, 2002) where the overall goal of the project was to improve technologies related to event-based information organization tasks. Of the five TDT tasks, first story detection is considered the most difficult one (Allan et al., 2000a). A good FSD system would be very useful for business or intelligence analysts where timely discovery of events is crucial. With the significant increase in the amount of information being produced and consumed every day, a crucial requirement for a modern FSD system to be useful is efficiency. This means that the system should be able to work in a streaming setting where documents are constantly coming in at a high rate, while still producing good results. While previous work has addressed the efficiency (Petrović et al., 2010) aspect, there has been little work on improving FSD performance in the past few years. A major obstacle is the high degree of lexical variation in documents that cover the same event. Here we address this problem, while keeping in mind the efficiency constraints.

The problem of lexical variation plagues many IR and NLP tasks, and one way it has been addressed in the past is through the use of paraphrases. Paraphrases are alternative ways of expressing the same meaning in the same language. For example, the phrase *he got married* can be paraphrased as *he tied the knot*. Paraphrases were already shown to help in a number of tasks: for machine translation to translate unknown phrases by translating their paraphrases (Callison-Burch et al., 2006), for query expansion in information retrieval (Spärck Jones and Tait, 1984; Jones et al., 2006), or for improving question answering (Riezler et al., 2007). A much more detailed discussion on the use of paraphrases and ways to extract them is given in (Madnani and Dorr, 2010). Here, we present the first work to use paraphrases for improving first story detection. Using paraphrases, we are able to detect that some documents previously thought to be about new events are actually paraphrases of the documents already

seen. Our approach is simple and we show a novel way of integrating paraphrases with locality sensitive hashing (LSH) (Indyk and Motwani, 1998). This way we obtain a very efficient FSD system with all the benefits of using paraphrases, while avoiding computationally expensive topic modeling approaches such as Ahmed et al. (2011).

First story detection was introduced as a task before the popularization of social media. Event detection in social media, especially Twitter is a very good fit: we cover a much larger set of events than would be possible by using newswire, and the stories are reported in real time, often much sooner than in news. Of course, social media carries additional problems not found in traditional media: we have to deal with huge amounts of data, the data is very noisy (both due to spam and due to spelling and grammar errors), and in the case of Twitter, documents are extremely short. There has been little effort in solving these problems for FSD. Arguably the main reason for this is the lack of a TDT-style corpus for Twitter that researchers could use to test their approaches. Here we build such a corpus and use it to measure the performance of TDT systems on Twitter.

Our main contributions are: i) we create a first corpus of events on Twitter, ii) we show how to use paraphrases in FSD, and how to combine it with LSH to handle high-volume streams, iii) our unsupervised system that uses paraphrases achieves the highest reported results on the TDT5 corpus, beating both the supervised and unsupervised state of the art, while still keeping a constant per-document time complexity, and iv) we show that paraphrases also help in Twitter, although less than in TDT.

2 Paraphrasing and FSD

2.1 Current approaches to efficient FSD

State-of-the-art FSD systems (Allan et al., 2000b) use a fairly simple approach. Documents are represented as TF-IDF weighted vectors, their distance is measured in terms of the cosine distance, and they use a k-nearest neighbors clustering algorithm, with k usually set to 1. The novelty score for a document is the cosine distance to the nearest neighbor:

$$score(d) = 1 - \max_{d' \in D_t} \cos(d, d'). \quad (1)$$

D_t is the set of all documents up to time t when document d arrived.

Because the $\max$ in equation (1) takes $O(|D_t|)$ time to compute in the worst case, Petrović et al. (2010) introduced a way of using locality sensitive hashing (LSH) to make this time $O(1)$, while retaining the same accuracy level. In particular, instead of computing the $\max$ over the entire set D_t, like in (1), they compute it over a smaller set S of potential nearest neighbors. The set S is the set of documents that collide with the current document under a certain type of hash function:

$$S(\mathbf{x}) = \{\mathbf{y} : h_{ij}(\mathbf{y}) = h_{ij}(\mathbf{x}), \exists i \in [1..L], \forall j \in [1..k]\}, \quad (2)$$

where the hash functions h_{ij} are defined as:

$$h_{ij}(\mathbf{x}) = \text{sgn}(\mathbf{u}_{ij}^T \mathbf{x}), \quad (3)$$

with the random vectors $\mathbf{u}_{ij}$ being drawn independently for each i and j. The efficiency of this algorithm stems from the fact that it can be shown that the set S of potential nearest neighbors can be made constant in size, while still containing the nearest neighbor with high probability.

2.2 Paraphrases

There are several levels of paraphrasing – lexical paraphrases, where the relationship is restricted to individual lexical items, phrasal paraphrases, where longer phrases are considered, and sentential paraphrases, where entire sentences are in a paraphrastic relationship. Here we use the simplest form, lexical paraphrases, but our approach, described in section 2.3, is general and it would be trivial to use phrasal paraphrases in the same way – we leave this for future work.

We use three sources of paraphrases: Wordnet (Fellbaum, 1998), a carefully curated lexical database of English containing synonym sets, Microsoft Research paraphrase tables (Quirk et al., 2004), a set of paraphrase pairs automatically extracted from news texts, and syntactically-constrained paraphrases from Callison-Burch (2008) which are extracted from parallel text. We also considered using paraphrases from Cohn et al. (2008), but using them provided only minor improvement over the baseline model. This is likely due to the small size of that corpus (a total of 7

thousand pairs). We do not show results for this paraphrase corpus in our results section.

Wordnet paraphrases contained 150 thousand word pairs extracted from Wordnet's synsets, where all the pairs of words within one synset were considered to be paraphrases. MSR paraphrases were extracted from the phrase tables provided by MSR. Two words were considered paraphrases if they were aligned at least once in the most probable alignment, with the probability of both backward and forward alignment of at least 0.2. In our initial experiments we varied this threshold and found it has little effect on results. Using this method, we extracted 50 thousand paraphrase pairs. Finally, we use the method of Callison-Burch (2008) to extract syntactically constrained paraphrases from a parallel corpus. This method requires that phrases and their paraphrases be the same syntactic type, and has been shown to substantially improve the quality of extracted paraphrases (Callison-Burch, 2008). We extracted paraphrases for all the words that appeared in the MSR paraphrase corpus, and then kept all the pairs that had the paraphrase probability of at least 0.2. This way, we extracted 48 thousand pairs. All three resources we use are very different: they come from different domains (news text, legal text, general English), and they have very little overlap (less than 5% of pairs are shared by any two resources).

2.3 Efficient paraphrasing in FSD

In this section, we explain how to use paraphrases in a first story detection system. We account for paraphrases by changing how we compute the cosine in equation (1). Because the cosine measure depends on the underlying inner product, we change the way the inner product is computed. We model paraphrasing by using a binary word-to-word matrix of paraphrases $\mathbf{Q}$. An entry of 1 at row i and column j in the matrix indicates that words i and j are paraphrases of each other.[1] Note, however, that our approach is not limited to using single words – if the document representation includes n-grams with $n > 1$, the matrix $\mathbf{Q}$ can contain phrases, and thus we can capture non-compositional paraphrases like *he died* $\leftrightarrow$ *he kicked the bucket*. We use the matrix $\mathbf{Q}$ to define a new inner product space:[2]

$$\langle \mathbf{x}, \mathbf{y} \rangle_Q = \mathbf{y}^T \mathbf{Q} \mathbf{x}. \quad (4)$$

This way of using paraphrases basically achieves expansion of the terms in documents with their paraphrases. Thus, if two documents have no terms in common, but one has the term *explosion* and the other has the term *blast*, by knowing that the two terms are paraphrases, their similarity will be different from zero, which would have been the case if no paraphrasing was used. Alternatively, the new inner product in equation (4) can also be seen as introducing a linear kernel.

One problem with using $\mathbf{Q}$ as defined in (4) is that it is not very suitable for use in an online setting. In particular, if documents come in one at a time and we have to store each one, only for it to be retrieved at some later point, simply storing them and computing the inner product as in (4) would lead to frequent matrix-vector multiplications. Even though $\mathbf{Q}$ is sparse, these multiplications become expensive when done often, as is the case in first story detection. We thus have to store a modified document vector $\mathbf{x}$, call it $\mathbf{x}'$, such that when we compute $\langle \mathbf{x}', \mathbf{y}' \rangle$ we get $\langle \mathbf{x}, \mathbf{y} \rangle_Q$. Note that the inner product between $\mathbf{x}'$ and $\mathbf{y}'$ is computed in the original inner product space. It is clear that by using:

$$\mathbf{x}' = \mathbf{Q}^{\mathbf{1/2}} \mathbf{x} \quad (5)$$

we have achieved our goal: $\langle \mathbf{x}', \mathbf{y}' \rangle = \mathbf{y}'^T \mathbf{x}' = (\mathbf{Q}^{\mathbf{1/2}} \mathbf{y})^T (\mathbf{Q}^{\mathbf{1/2}} \mathbf{x}) = (\mathbf{y}^T \mathbf{Q}^{\mathbf{1/2}^T})(\mathbf{Q}^{\mathbf{1/2}} \mathbf{x}) = \mathbf{y}^T \mathbf{Q} \mathbf{x} = \langle \mathbf{x}, \mathbf{y} \rangle_Q$. Again, if we view equation (4) as defining a kernel, we can think of equation (5) as performing an explicit mapping into the feature space defined by the kernel. Because ours is a linear kernel, performing this mapping is fairly efficient.

Unfortunately, the square root of the paraphrasing matrix $\mathbf{Q}$ is in general dense (and can even contain complex entries), which would make our approach infeasible in practice because we would have to expand every document with all (or a very large number of) the words in the vocabulary. Thus, we have to

[1] This is of course a simplification – in general, one might like the entries in the matrix to be real numbers corresponding to the probability that the two words are paraphrases. We leave this for future work.

[2] Equation (4) does not define a proper inner product in the strict technical sense because the positive definiteness property does not hold. However, because vectors $\mathbf{x}$ and $\mathbf{y}$ in practice always have positive entries, equation (4) behaves like a proper inner product for all practical purposes.

approximate $\mathbf{Q^{1/2}}$ with a sparse matrix, preferably one that is as sparse as the original $\mathbf{Q}$ matrix. To this end, we introduce the following approximation:

$$\tilde{\mathbf{Q}}^{\mathbf{1/2}}_{ij} = \frac{\mathbf{Q}_{ij}}{\sqrt{\sum_k (\mathbf{Q}_{ik} + \mathbf{Q}_{kj})/2}} \tag{6}$$

To see how we arrive at this approximation, consider the paraphrase matrix $\mathbf{Q}$. If there was no polysemy in the language, $\mathbf{Q}$ would be a block matrix, where each non-zero submatrix would correspond to a single meaning. The square root of such a matrix would be given exactly by (6). While this approximation is somewhat simplistic, it has two major advantages over the exact $\mathbf{Q^{1/2}}$: i) it is very easy to compute and, with proper implementation, takes $O(n^2)$ time, as opposed to $O(n^3)$ for $\mathbf{Q^{1/2}}$, making it scalable to very large matrices, and ii) matrix $\tilde{\mathbf{Q}}^{\mathbf{1/2}}$ is guaranteed to be as sparse as $\mathbf{Q}$, whereas $\mathbf{Q^{1/2}}$ will in most cases become dense, which would make it unusable in real applications.

2.4 Locality-sensitive hashing with paraphrasing

Here we explain how to integrate paraphrasing with efficient FSD, using LSH described in section 2.1. As we mentioned before, a single hash function h_{ij} in the original LSH scheme hashes the vector $\mathbf{x}$ to:

$$h(\mathbf{x}) = \text{sgn}(\mathbf{u}^T\mathbf{x}), \tag{7}$$

where $\mathbf{u}$ is a (dense) random vector. If we want to use paraphrases with LSH, we simply change the hash function to

$$h_1(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\tilde{\mathbf{Q}}^{\mathbf{1/2}}\mathbf{x})). \tag{8}$$

It is not difficult to show that by doing this, the LSH bounds for probability of collision hold in the new inner product space defined by the matrix $\mathbf{Q}$. We omit this proof due to space constraints.

Space efficient LSH. While LSH can significantly reduce the running time, it is fairly expensive memory-wise. This memory overhead is due to the random vectors $\mathbf{u}$ being very large. To solve this problem, (Van Durme and Lall, 2010) used a *hashing trick* for space-efficient storing of these vectors. They showed that it is possible to project the vectors onto a much smaller random subspace, while still retaining good properties of LSH. They proposed the following hash function for a vector $\mathbf{x}$:

$$h_2(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\mathbf{A}\mathbf{x})), \tag{9}$$

where $\mathbf{A}$ is a random binary matrix with exactly one non-zero element in each column. This approach guarantees a constant space use which is bounded by the number of rows in the $\mathbf{A}$ matrix. Here we show that our paraphrasing approach can be easily used together with this space-saving approach by defining the following hash function for $\mathbf{x}$:

$$h_3(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\mathbf{A}\tilde{\mathbf{Q}}^{\mathbf{1/2}}\mathbf{x})). \tag{10}$$

This way we get the benefits of the hashing trick (the constant space use), while also being able to use paraphrases. The hash function in (10) is the actual hash function we use in our system. Together with the heuristics from Petrović et al. (2010), it guarantees that our FSD system will use constant space and will take constant time to process each document.

3 Twitter Event Corpus

3.1 Event detection on Twitter

As we mentioned before, research on event detection in social media is hampered by the lack of a corpus that could be used to measure performance. The need for a standard corpus is evident from the related work on event detection in Twitter. For example, (Petrović et al., 2010) address the scaling problem in social media and present a system that runs in constant time per document, but the evaluation of their system on Twitter data was limited to very high-volume events. The only attempt in creating a corpus of events for Twitter that we are aware of was presented in Becker et al. (2011). Unfortunately, that corpus is not suitable for FSD evaluation for two main reasons: i) the events were picked from the highest-volume events identified by the system (similar to what was done in Petrović et al. (2010)), introducing not only a bias towards high-volume events, but also a bias toward the kinds of events that their system can detect, and ii) the authors only considered tweets by users who set their location to New York, which introduces a strong bias towards the type of events that can appear in the corpus. While these problems were not relevant to the

work of (Becker et al., 2011) because the corpus was only used to compare different cluster representation techniques, they would certainly pose a serious problem if we wanted to use the corpus to compare FSD systems. In this paper we present a new corpus of tweets with labeled events by taking a very similar approach to that taken by NIST when creating the TDT corpora.

3.2 Annotating the Tweets

In this section we describe the annotation process for our event corpus. Note that due to Twitter's terms of service, we distribute the corpus as a set of tweet IDs and the corresponding annotations – users will have to crawl the tweets themselves, but this can be easily done using any one of the freely available crawlers for Twitter. This is the same method that the TREC microblog track[3] used to distribute their data. All our Twitter data was collected from the streaming API[4] and consists of tweets from beginning of July 2011 until mid-September 2011. After removing non-English tweets, our corpus consists of 50 million tweets.

In our annotation process, we have adopted the approach used by the National Institute of Standards and Technology (NIST) in labeling the data for TDT competitions. First, we defined a set of events that we want to find in the data, thus avoiding the bias of using events that are the output of any particular system. We choose the events from the set of important events for our time period according to Wikipedia.[5] Additionally, we used common knowledge of important events at that time to define more events. In total, we define 27 events, with an average of 112 on-topic tweets. This is comparable to the first TDT corpus which contained 25 events and average of 45 on-topic documents. However, in terms of the total number of documents, our corpus is three orders of magnitude larger than the first TDT corpus, and two orders of magnitude larger than the biggest TDT corpus (TDT5). Our corpus contains very different events, such as the death of Amy Winehouse, downgrading of US credit rating, increasing of US debt ceiling, earthquake in Virginia, London riots, terrorist attacks in Norway, Google announcing plans to buy Motorola Mobility, etc. The event with the most on-topic tweets had over 1,000 tweets (death of Amy Winehouse), and the smallest event had only 2 on-topic tweets (arrest of Goran Hadzic).

We faced the same problems as NIST when labeling the events – there were far too many stories to actually read each one and decide which (if any) events it corresponds to. In order to narrow down the set of candidates for each event, we use the same procedure as used by NIST. The annotator would first read a description of the event, and from that description compile a set of keywords to retrieve possibly relevant tweets. He would then read through this set, labeling each tweet as on- or off-topic, and also adding new keywords for retrieving a new batch of tweets. After labeling all the tweets in one batch, the newly added keywords were used to retrieve the next batch, and this procedure was repeated until no new keywords were added. Unlike in TDT, however, when retrieving tweets matching a keyword, we do not search through the whole corpus, as this would return far too many candidates than is feasible to label. Instead, we limit the search to a time window of one day around the time the event happened.

Finally, the annotator guidelines contained some Twitter-specific instructions. Links in tweets were not taken into account (the annotator would not click on links in the tweets), but retweets were (if the retweet was cut off because of the 140 character limit, the annotator would label the original tweet). Furthermore, hashtags were taken into account, so tweets like *#Amywinehouseisdead* were labeled as normal sentences. Also, to be labeled on-topic, the tweet would have to explicitly mention the event and the annotator should be able to infer what happened from the tweet alone, without any outside knowledge. This means that tweets like *Just heard about Lokomotiv, this is a terrible summer for hockey!* are off topic, even though they refer to the plane crash in which the Lokomotiv hockey team died.

In total, our corpus contains over 50 million tweets, of which 3035 tweets were labeled as being on-topic for one of the 27 events. While searching for first tweets (i.e., tweets that first mention an event), *fake* first tweets were sometimes discovered. For example, in the case of the death of Richard Bowes (victim of London riots), a Telegraph journalist posted a tweet informing of the man's death

[3] http://trec.nist.gov/data/tweets/

[4] https://stream.twitter.com/

[5] http://en.wikipedia.org/wiki/2011

more than 12 hours before he actually died. This tweet was later retracted by the journalist for being incorrect, but the man then died a few hours later. Cases like this were labeled off-topic.

4 Experiments

4.1 Evaluation

In the official TDT evaluation, each FSD system is required to assign a score between 0 and 1 to every document upon its arrival. Lower scores correspond to old stories, and vice versa. Evaluation is then carried out by first sorting all stories according to their scores and then performing a threshold sweep. For each value of the threshold, stories with a score above the threshold are considered new, and all others are considered old. Therefore, for each threshold value, one can compute the probability of a *false alarm*, i.e., probability of declaring a story new when it is actually not, and the *miss probability*, i.e., probability of declaring a new story old (missing a new story). Using the false alarm and the miss rate, the cost C_{det} is defined as follows:

$$C_{det} = C_{miss} * P_{miss} * P_{target} + C_{FA} * P_{FA} * P_{non-target},$$

where C_{miss} and C_{FA} are costs of miss and false alarm (0.02 and 0.98, respectively), P_{miss} and P_{FA} are the miss and false alarm rate, and P_{target} and $P_{non-target}$ are the prior target and non-target probabilities. Different FSD systems are compared on the minimal cost C_{min}, which is the minimal value of C_{det} over all threshold values. This means that in FSD evaluation, a *lower* value of C_{min} indicates a better system.

4.2 TDT results

For the TDT experiments, we use the English portion of TDT-5 dataset, consisting of 126 topics in 278,108 documents. Similar to (Petrović et al., 2010), we compare our approach to a state-of-the-art FSD system, namely the UMass system (Allan et al., 2000b). This system always scored high in the TDT competitions and is known to perform at least as well as other systems that also took part in the competition (Fiscus, 2001). Our system is based on the streaming FSD system of (Petrović et al., 2010) which has a constant per-document time complexity. We use stemming (Porter, 1980) and, the same as (Petrović et al., 2010), we use 13 bits per key and 70 hash tables for LSH. Additionally, we use the hashing trick described in section 2.4 with a pool of size 2^{18}. Paraphrasing is implemented in this system as described in section 2.4.

While the UMass system was among the best systems that took part in the TDT competitions, there has been research in event detection since the competitions stopped. Recent work on event detection includes a hybrid clustering and topic model with rich features such as entities, time, and topics (Ahmed et al., 2011). We do not compare our system to Ahmed et al. (2011) because in terms of the numerical $C_{\min}$ score, their approach does not outperform the UMass system. This is not surprising as the primary goal in Ahmed et al. (2011) was not to improve FSD performance, but rather to create storylines and support structured browsing.

We compare our approach to the best reported result in the literature on the TDT5 data. To the best of our knowledge, the highest reported results in FSD come from a supervised system described in Kumaran and Allan (2005). This system uses an SVM classifier with the features being FSD scores from unsupervised systems (the authors used scores computed in the same way as is done in the UMass system) computed using i) full text, ii) only named entities in the document, and iii) only topic terms. The classifier was trained on TDT3 and TDT4 corpora and tested on TDT5.

Table 1 shows the results for TDT5 data. UMass 1000 is the run that was submitted as the official run in the TDT competition.[6] We can see that using paraphrases improves the results over the unsupervised state of the art, regardless of which source of paraphrasing is used. However, it is clear that not all types of paraphrases are equally helpful. In particular, the automatically extracted paraphrases from Callison-Burch (2008) seem to be the most helpful, and by using them our unsupervised system is able to beat even the best known supervised FSD system. This is a very promising result because it indicates that we can use automatically extracted paraphrases and do not have to rely on hand-crafted resources like Wordnet as our source of paraphrases.

[6] Our experiments, and experiments in Allan et al. (2000b) showed that keeping full documents does not improve results, while increasing running time.

System	C_{min}
UMass 100	0.721
UMass 1000	0.706
Best supervised system	0.661
Wordnet	0.657
MSR Paraphrases	0.642
Syntactic paraphrases	**0.575**

Table 1: TDT FSD results for different systems, lower is better. The number next to UMass system indicates the number of features kept for each document (selected according to their TFIDF). All paraphrasing systems work with full documents. Results for the best supervised system were taken from Kumaran and Allan (2005).

The difference between our system and the UMass system is significant at $p = 0.05$ using a paired t-test over the individual topic costs. We were not able to test significance against the supervised state-of-the-art because we did not have access to this system. In terms of efficiency, our approach is still $O(1)$, like the approach in Petrović et al. (2010), but in practice it is somewhat slower because hashing the expanded documents takes more time. We measured the running time of our system, and it is 3.5 times slower than the basic approach of Petrović et al. (2010), but also 3.5 times faster than the UMass system, while outperforming both of these systems.

How does quality of paraphrases affect results? We have shown that using automatically obtained paraphrases to expand documents is beneficial in first story detection. Because there are different ways of extracting paraphrases, some of which are targeted more towards recall, and some towards precision, we want to know which techniques would be more suitable to extract paraphrases for use in FSD. Here, precision is the ratio between extracted word pairs that are actual paraphrases and all the word pairs extracted, and recall is the ratio between extracted word pairs that are actual paraphrases, and all the possible paraphrase pairs that could have been extracted. In this experiment we focus on the syntactic paraphrases which yielded the best results. To lower recall, we randomly remove paraphrase pairs from the corpus, and to lower precision, we add random paraphrase pairs to our table. All the results are shown in Table 2. Numbers next to precision and recall indicate the proportion of added random pairs and the proportion of removed pairs, respectively (e.g., recall 0.4 means that 40% of pairs were removed from the original resource). We can see that the results are much more stable with respect to recall – there is an initial drop in performance when we remove the first 10% of paraphrases, but after that removing more paraphrases does not affect performance very much. On the other hand, changing the precision has a bigger impact on the results. For example, we can see that our system using a paraphrase corpus with 30% of pairs added at random performs even better than the system that uses the original corpus. On the other hand, adding 20% of random pairs performs substantially worse than the original corpus. These results show that it is more important for the paraphrases to have good precision than to have good recall.

Paraphrasing resource	$C_{\min}$
Precision 0.1	0.603
Precision 0.2	0.672
Precision 0.3	0.565
Precision 0.4	0.603
Precision 0.5	0.626
Recall 0.9	0.609
Recall 0.8	0.606
Recall 0.7	0.632
Recall 0.6	0.610
Recall 0.5	0.626

Table 2: Effect of paraphrase precision and recall on FSD performance. Numbers next to recall and precision indicate the sampling rate and the proportion of added random pairs, respectively.

4.3 Twitter results

Because the Twitter event corpus that we use consists of over 50 million documents, we cannot use the UMass system here due to its linear per-document time complexity. Instead, our baseline system here is the FSD system of (Petrović et al., 2010), without any paraphrasing. This system uses the same approach as the UMass system, and (Petrović et al., 2010) showed that it achieves very similar results. This means that our baseline, al-

though coming from a different system, is still state-of-the-art. We make some Twitter-specific modification to the baseline system that slightly improve the results. Specifically, the baseline uses no stemming, ignores links, @-mentions, and treats hashtags as normal words (i.e., removes the leading '#' character). While removing links and @-mentions was also done in (Petrović et al., 2010), our preliminary experiments showed that keeping hashtags, only without the hash sign improves the results. Additionally, we limit the number of documents in a bucket to at most 30% of the expected number of collisions for a single day (we assume one million documents per day).

Results for the different systems are shown in Table 3. First, we can see that not using stemming is much better than using it, which is the opposite from what is the case in TDT. Second, we can see that the improvements from using paraphrases that we had in TDT data are different here. Syntactic paraphrases and the MSR paraphrases do not help, whereas the paraphrases extracted from Wordnet did improve the results, although the gains are not as large as in TDT. A paired t-test revealed that none of the differences between the baseline system and the systems that use paraphrases were significant at $p = 0.05$.

To gain more insight into why the results are different here, we look at the proportion of words in the documents that are being paraphrased, i.e., the *coverage* of the paraphrasing resource. We can see from Table 4 that the situation in TDT and Twitter is very different. Coverage of MSR and syntactic paraphrases was lower in Twitter than in TDT, whereas Wordnet coverage was better on Twitter. While it seems that the benefits of using paraphrases in Twitter are not as clear as in news, our efficient approach enables us to answer questions like these, which could not be answered otherwise.

To illustrate how paraphrases help detect old tweets, consider the tweet *According to Russian aviation officials, two passengers survived the crash, but are in critical condition*. Before paraphrasing, the closest tweet returned by our system was *Shazaad Hussein has died in Birmingham after being run over, two others are in critical condition*, which is not very related. After applying paraphrasing, in particular knowing that *officials* is a paraphrase of *authorities*, the closest tweet returned was *Some Russian authorities are reporting one survivor, others are saying there are three. There were 37 total on board*, which is on the same event. There are also cases where paraphrases hurt. For example, before paraphrasing the tweet *Top News #debt #deal #ceiling #party* had the nearest neighbor *New debt ceiling deal explained*, whereas after paraphrasing, because the word *roof* is a paraphrase of *ceiling*, the nearest neighbor was *The roof the roof the roof is on fire!*. Cases like this could be fixed by looking at the context of the word, but we leave this for future work.

System	C_{min}
Baseline system (stemming)	0.756
Baseline system (no stemming)	0.694
Wordnet	**0.679**
MSR Paraphrases	0.739
Syntactic paraphrases	0.729

Table 3: Twitter FSD results for different systems, lower is better. The baseline system is that of (Petrović et al., 2010).

Paraphrases	Coverage TDT (%)	Coverage Twitter (%)
Wordnet	52.5	56.1
MSR	33.5	31.0
Syntactic	35.6	31.7

Table 4: Coverage of different resources.

5 Conclusion

We present a way of incorporating paraphrase information in a streaming first story detection system. To the best of our knowledge, this is the first work to use paraphrases in first story detection, and also the first work to combine paraphrases with locality-sensitive hashing to achieve fast retrieval of documents that are written with different words, but talk about the same thing. We compare different sources of paraphrases and show that our unsupervised FSD system that uses syntactically constrained paraphrases achieves state-of-the-art results, beating both the best supervised and unsupervised systems. To test our approach on very large data, we construct a corpus of events for Twitter. Our approach scales well on this data both in terms of time and memory, and we show that paraphrases again help, but

this time the paraphrase sources yield different improvements from TDT data. We find that this difference can be explained by the different coverage of the paraphrasing resources.

Acknowledgments

The authors would like to thank Mirella Lapata for her help with paraphrasing resources. We also acknowledge financial support from EPSRC grant EP/J020664/1.

References

Amr Ahmed, Qirong Ho, Jacob Eisenstein, Eric Xing, Alex Smola, and Choon Hui Teo. 2011. Unified analysis of streaming news. In *Proceedings of WWW*, pages 267–276. ACM.

James Allan, Victor Lavrenko, and Hubert Jin. 2000a. First story detection in tdt is hard. In *Proceedings of the CIKM*, pages 374–381. ACM.

James Allan, Victor Lavrenko, Daniella Malin, and Russell Swan. 2000b. Detections, bounds, and timelines: Umass and tdt-3. In *Proceedings of Topic Detection and Tracking Workshop*, pages 167–174.

James Allan. 2002. *Topic detection and tracking: event-based information organization*. Kluwer Academic Publishers.

Hila Becker, Mor Naaman, and Luis Gravano. 2011. Selecting quality twitter content for events. In *Proceedings of ICWSM*.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. 2006. Improved statistical machine translation using paraphrases. In *Proceedings of NAACL*, pages 17–24. Association for Computational Linguistics.

Chris Callison-Burch. 2008. Syntactic constraints on paraphrases extracted from parallel corpora. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 196–205. Association for Computational Linguistics.

Trevor Cohn, Chris Callison-Burch, and Mirella Lapata. 2008. Constructing corpora for the development and evaluation of paraphrase systems. *Computational Linguistics*, 34(4):597–614.

Christiane Fellbaum. 1998. *WordNet: An electronic lexical database*. The MIT press.

Jonathan Fiscus. 2001. Overview of results (nist). In *Proceedings of the TDT 2001 Workshop*.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA. ACM.

Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM.

Giridhar Kumaran and James Allan. 2005. Using names and topics for new event detection. In *Proceedings of EMNLP*, pages 121–128. Association for Computational Linguistics.

Nitin Madnani and Bonnie Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Saša Petrović, Miles Osborne, and Victor Lavrenko. 2010. Streaming first story detection with application to twitter. In *Proceedings of the 11th annual conference of the North American Chapter of the ACL*, pages 181–189.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Chris Quirk, Chris Brockett, and William Dolan. 2004. Monolingual machine translation for paraphrase generation. In *In proceedings of EMNLP*, pages 142–149.

Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *Proceedings of ACL*, volume 45, page 464.

Karen Spärck Jones and John Tait. 1984. Automatic search term variant generation. *Journal of Documentation*, 40(1):50–66.

Benjamin Van Durme and Ashwin Lall. 2010. Online generation of locality sensitive hash signatures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.