A Systematic Investigation of Blocking Strategies for Real-Time Classification of Social Media Content into Events

Timo Reuter
Semantic Computing
CITEC, Universität Bielefeld
Bielefeld, Germany
treuter@cit-ec.uni-bielefeld.de

Philipp Cimiano
Semantic Computing
CITEC, Universität Bielefeld
Bielefeld, Germany
cimiano@cit-ec.uni-bielefeld.de

Abstract

Events play a prominent role in our lives, such that many social media documents describe or are related to some event. Organizing social media documents with respect to events thus seems a promising approach to better manage and organize the ever-increasing amount of user-generated content in social media applications. It would support the navigation of data by events or allow one to get notified about new postings related to the events one is interested in, just to name two applications. A challenge is to automatize this process so that incoming documents can be assigned to their corresponding event without any user intervention. We present a system that is able to classify a stream of social media data into a growing and evolving set of events. In order to scale up to the data sizes and data rates in social media applications, the use of a candidate retrieval or blocking step is crucial to reduce the number of events that are considered as potential candidates to which the incoming data point could belong to. In this paper we present and experimentally compare different blocking strategies along their cost vs. effectiveness tradeoff. We show that using a blocking strategy that selects the 60 closest events with respect to upload time, we reach F-Measures of about 85.1% while being able to process the incoming documents within 32ms on average. We thus provide a principled approach supporting to scale up classification of social media documents into events and to process the incoming stream of documents in real time.

Introduction

Social Media Applications (SMA) are proliferating and they are characterized by an ever-increasing amount of content that represents a never-ending data stream growing at high rates. As of November 2011, around 140,000 messages per minute are for example posted via Twitter[1], a popular service for posting short text messages of at most 140 characters to following peers. Flickr[2], an image database where people can upload pictures to, counts an average of 2500 images uploaded per minute. In Facebook[3], a popular social network, around 500,000 messages are posted per minute.

As events play a prominent role in our lives, many of the postings and documents uploaded to social media sites are related to some event. Classifying social media documents by the events they represent or are related to thus represents a promising approach to better manage and organize the ever-increasing amount of user-generated content in social media applications.

Many social media sites allow users to tag content, thus supporting the better organization of the content and facilitating search. Indeed, tags can also be used to assign documents to their corresponding event. For this purpose, *last.fm*[4] hosts an event database with unique identifiers in the form of so-called *machine tags* that can be used to tag pictures – for example when uploading them to Flickr – thus assigning them to one or more uniquely identified events. While some users exploit such tags, the majority of pictures on Flickr are not assigned to events via such machine tags. A random sample of Flickr data in fact shows that only 0.0003% of the data are assigned to at least one event via a machine tag from last.fm, and only 0.0007% of the data has a machine tag at all. While these numbers might change in the future, we think that it is reasonable to assume that most of the data uploaded will not be tagged with any machine tags as this always represents additional effort for a user. In the case of using tags from last.fm, the user would have to search for the event in last.fm, copy the corresponding machine tag and assign this tag to their pictures when uploading them to some social media site (e.g. Flickr).

Given the above explanations, it seems clear that there is an impending need for automatic techniques that perform the assignment of a social media item (newly uploaded to some social media site) to its corresponding event (if it already exists) or create a new event to which future data items can be assigned to. In line with Becker et al. we refer to this problem as the *event identification problem* (Becker, Naaman, and Gravano 2010). A challenge in developing such techniques is to scale to the amounts of data available in social media applications and to support the classification of incoming documents into events in real time.

[1]http://www.twitter.com
[2]http://flickr.com
[3]http://www.facebook.com
[4]http://last.fm

A crucial step to scale-up and achieve real-time behavior is to filter down the set of events that are considered for every incoming document. As there might be billions of events in the database, it is unfeasible to scan all these events to compute the likelihood that the incoming document belongs to one of these. For this purpose, *candidate retrieval* or *blocking* strategies are typically considered (Baxter, Christen, and Churches 2003) which reduce the amount of pairs – in our case pairs of incoming document and event – to be considered. When using a blocker, a crucial issue that arises is to ensure that the blocker is neither too strict, thus filtering too much, nor too lenient, possibly passing on too much noise to the classifier. A certain blocking method might thus be efficient from a computational point of view but too inaccucrate, thus having an overall detrimental effect on the overall performance of the classifier. Given the importance of having a suitable blocker that is tailored to the domain in question, in this paper we investigate different candidate retrieval or blocking strategies in terms of their cost-effectiveness tradeoff with respect to the task of classifying a stream of social media documents into an evolving set of events. We show that using a blocking strategy that selects the 60 closest events with respect to upload time, we reach F-Measures of about 85.1% while being able to process the incoming documents within 32ms on average. We thus provide a principled approach supporting to scale up classification of social media documents into events and process the incoming stream of documents in real time.

The article is structured as follows: in Section *System Description* we briefly describe our overall system. In Section *Candidate Retrieval*, we describe the candidate retrieval step in more detail, presenting different blocking strategies that we experimentally analyze with respect to their cost-effectiveness tradeoff. In Section *Experimental Setup and Results* we describe how we created our dataset consisting of 500,000 Flickr pictures together with an appropriate gold standard developed using last.fm machine tags. We also describe how the machine learning components were trained and provide experimental results. Before concluding, we discuss related work on event identification in social media as well as different blocking strategies that have been proposed in the literature.

System Description

Our system processes an incoming stream of documents as described in the following. For every incoming document $d \in D$:

1. A set E of k events that d is likely to belong to is retrieved from the event database (*Candidate Retrieval)*.
2. For each of these candidates $e \in E$, the probability that d belongs to e, $P(e|d)$, is computed, and all candidates are ranked according to this probability (*Scoring and Ranking*). Let e_{max} be the top scoring candidate.
3. Given this ranked list of candidates, the probability that d belongs to a new event, $P_{new}(d)$, or that it belongs to the first event in the list, $P_{belongs\_to\_top\_candidate}(d)$, is computed. We assume that these are the only two options, i.e. $P_{new}(d) + P_{belongs\_to\_top\_candidate}(d) = 1$.

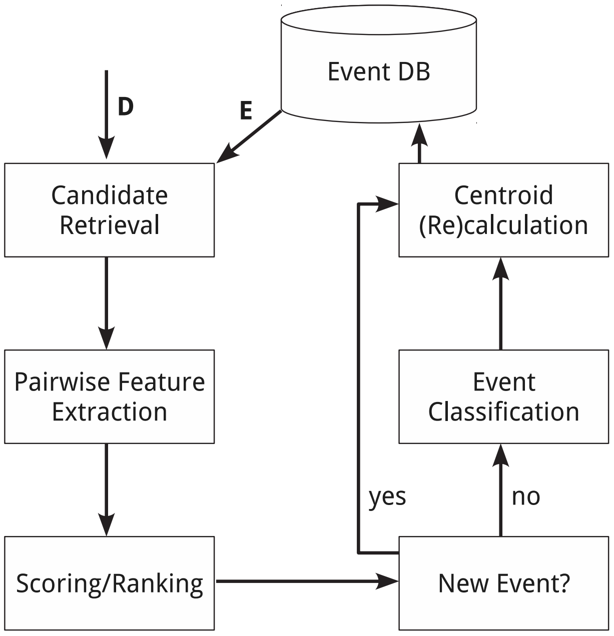

Figure 1: Overview of the event identification system

4. If $P_{new}(e) > \theta_n$, a new event e' is created and d is assigned to this newly created event; $\vec{e} := \vec{d}$ (*new event detection*).
5. Otherwise, d is assigned to e_{max}; the centroid $\vec{e}_{max}$ is recomputed.

The above procedure is illustrated by the pseudocode in Algorithm 1 and depicted graphically in Figure 1.

Algorithm 1 Stream-based classification into events

```
for all $d \in D$ do
    $Top_k(d)$ = retrieve a ranked list of promising event candidates to which $d$ could belong
    for all $e \in Top_k(d)$ do
        compute $P(e|d)$ – the probability that $d$ belongs to $e$
    end for
    $e_{max} = max_{e' \in Top_k(d)} P(e'|d)$
    compute $P_{new}(d)$ – the probability that $d$ belongs to a new event
    if $P_{new}(d) > \theta_n$ then
        create a new event $e'$
        $e'$ = {d}
        $\vec{e'} = \vec{d}$
    else
        $e_{max} = e_{max} \cup \{d\}$
        recompute $\vec{e}_{max}$
    end if
end for
```

We briefly describe each of the main components besides the candidate retrieval below. The candidate retrieval strategies are described in Section *Candidate Retrieval*.

Scoring and Ranking A pair consisting of a document and a candidate event is described in terms of a vector of nine features that describe the match between the document and the event. In particular, we use as features nine similarity measures which exploit the following information sources:

- **Temporal information:** refers to the moment the document was created (e.g. by taking a picture) or the time when the user uploaded the document. We define the temporal similarity between an event and a document as

$$sim_{time}(d,e) = 1 - \frac{\log(|time(d) - time(e)|)}{y}$$

where $time(d)$ and $time(e)$ are timestamps represented as integer values denoting the number of minutes elapsed since the Unix epoch, and y is the logarithmic value of minutes of one year. This yields two similarity measures: one calculated on the basis of capture time ($sim_{capture}(d,e)$) and one calculated on the basis of upload time ($sim_{upload}(d,e)$).

- **Geographical information:** locates the place where the document was created (i.e. the location of an event in terms of latitude and longitude). In our approach we use the haversine formula to determine the great-circle distance between two points. The geographic similarity is thus: $sim_{geo}(d,e) = 1 - H(L_1, L_2)$ where

$$H(L_1, L_2) = 2 \cdot \arctan^2(\sqrt{\phi}, \sqrt{1-\phi})$$
$$\phi = \sin^2(\frac{\Delta lat}{2}) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin^2(\frac{\Delta lon}{2})$$
$$\Delta lat = lat_2 - lat_1, \Delta lon = lon_2 - lon_1$$

- **Textual information:** contains tags, a title, a description, etc. We describe the textual content by way of TF.IDF vectors. To determine the similarity we rely on the cosine similarity ($sim^{cos}_{text}(e,d)$) as well as the BM25 formula (Robertson and Jones 1976) ($sim^{BM25}_{text}(d,e)$).

Overall, a vector describing the similarity between a pair of document and event looks as follows:

$$\vec{sim(d,e)} = \begin{pmatrix} sim_{capture}(d,e) \\ sim_{upload}(d,e) \\ sim_{geo}(d,e) \\ sim^{cos}_{tags}(d,e) \\ sim^{BM25}_{tags}(d,e) \\ sim^{cos}_{title}(d,e) \\ sim^{BM25}_{title}(d,e) \\ sim^{cos}_{description}(d,e) \\ sim^{BM25}_{description}(d,e) \end{pmatrix}$$

For each incoming document d, we compute the likelihood that it belongs to a given event e, $P(e|d)$, and order the events retrieved by the candidate retrieval step by decreasing probability. The likelihood that document d belongs to event e is calculated using a support vector machine. To do this we train a SVM that relies on an appropriate training dataset to discriminate between pairs of document and corresponding event (positive examples) and pairs of documents that do not belong to the given event (negative examples). We use 4,000 examples for each of these classes to train binary SVMs and compute the probability $P(e|d)$ as the probability that the pair (d,e) - described by the above mentioned vector of similarities - belongs to the positive class, i.e. $P(e|d) = P(positive|\vec{sim}(e,d))$. We use the C-SVMs implemented in libsvm, which uses Platt's algorithm (Platt and others 1999) to compute the above probability as follows:

$$P(positive|\langle\vec{e},\vec{d}\rangle) := \frac{1}{1 + exp(A\langle\vec{e},\vec{d}\rangle + B)}$$

where the parameters A and B are optimized by the SVM by minimizing the negative log likelihood of the training data. The accuracy of this classifier has been shown to be at around 99.4%.

New Event Detection Once the candidates are ranked by the likelihood that d belongs to them, an important question is whether the document d belongs to the top ranked event or rather to a new event to be created. This is what we refer to as *new event detection problem.* The top scoring candidate might actually represent an event that d is not related to, such that we need a decision function that decides whether to assign the document d to the top scoring candidate or to a newly created event.

For this purpose, we also employ a C-SVM trained on an appropriate dataset consisting of examples in which the document belongs to the top-scoring event (positive examples) and examples in which the document belongs to a new event (negative examples). As features for this task we use the following:

- **max**: $P(e_1|d)$, i.e. the probability that d belongs to the top-scoring event
- **min**: the probability $P(e_{10}|d)$ – the probability that d belongs to the 10-th ranked event
- **average**: $\frac{1}{10}\sum_{i=1}^{10} P(e_i|d)$ – the average probability of the top-10 most likely events
- **standard deviation**: the standard deviation of the probability of the top-10 most likely events
- **maximum capture time**: $sim_{capture}(d, e_1)$
- **maximum upload time**: $sim_{upload}(d, e_1)$

For each document d this yields a feature vector $\vec{new}(d)$ that is used to classify d into two classes: belongs to top scoring event (positive) or belongs to a new event (negative). The SVM classifier is trained using an equal number of positive and negative examples and as in the above case returns a probability that the document belongs to a new event: $P_{new}(d) = P(negative|\vec{new}(d))$.

We then use a threshold on this probability as a hyperparameter to be tuned which decides whether the event is assigned to a new event or assigned to the top ranked event. The optimal threshold is determined empirically using a gradient descent technique on a split of our training data. The accuracy of this classifier is around 85.9%.

Candidate Retrieval

In order to scale-up the system, a crucial issue is to reduce the number of events that the system considers as potential event candidates for the incoming document. The reason is that the predictions of the SVM need to be calculated for each candidate event: one to calculate the probability that the document belongs to this event – $P(e|d)$. The computation is thus linear in the number of events, which can be prohibitive if the number of events is large, e.g. in the region of millions or even billions. We thus require an event candidate retrieval strategy that, ideally, i) retrieves the correct event and minimizes the overall number of retrieved events and thus ii) filters out as many of the irrelevant events as possible in order to reduce the computational cost of post-processing the retrieved events.

In our experiments, we compare in particular the following blocking strategies:

1. **k-nearest by capture time**: By using this strategy, we retrieve those k events with the lowest temporal distance to the document. For this, we order all events e_i in the database by $\Delta(time(d), time(e_i))$ and then return the top k events in this ordered list.
2. **k-nearest by upload time**: We do the same as in *k-nearest by capture time* but use the upload timestamps.
3. **geo-blocker**: We define a window of $1.0° \times 1.0°$ and select the top-k events out of this window, i.e. we retrieve all events such that $latitude(d) - 1° < latitude(e) < latitude(d) + 1° \wedge longitude(d) - 1° < longitude(e) < longitude(d) + 1°$[5].
4. **Tag-TF.IDF**: We score each event by summing up the TF.IDF values of every tag which the document and the event share and return the k events having the highest scores.
5. **Title-TF.IDF**: The same as *Tag-TF.IDF* but using the tokens in the title.
6. **Description-TF.IDF**: The same as *Tag-TF.IDF* but using the tokens in the description.
7. **Uniform Combination**: We retrieve the same number of $\frac{k}{6}$ (with $k \ mod \ 6 = 0$) events for each of the blocking strategies $b_1 \cdots b_6$ described above.
8. **Optimal Combination:** We determine the optimal combination of the number of candidates to be retrieved by each blocker empirically. We thus retrieve a number $k(b_i)$ of events that is specific for each blocker. The optimal parameters are computed by exhaustive search on the training set with the goal of maximizing the effectiveness of the blocker (see below for the definition of effectiveness).

The effectiveness of a blocking strategy b_i at a number k of retrieved events is measured as:

$$\mathit{Effectiveness}(b_i, k) = \frac{|\{d \mid event(d) \in top_k(b_i)\}|}{|D|}$$

[5]This window roughly corresponds to an area of 50-60 square kilometers in Europe and the U.S. and to an ear of 110 square kilometers in the vicinity of the equator.

where $event(d)$ is the correct event for document d according to the gold standard and $top_k(b_i)$ is the set of top k events retrieved by blocker b_i.

Experimental Setup and Results

After having presented our system, we now describe how our dataset consisting of 500,000 pictures from Flickr has been created and our experimental results.

Experimental Settings

Dataset Creation Since 2007, *last.fm* has been providing a freely available event database in which every event contained has a unique event ID. A key function of Flickr is the possibility for the user to assign so-called *machine tags* to a picture. The main difference to normal *tags* is that machine tags follow a fixed schema. Such a machine tag might have the following form: `lastfm:event=#eventid`. This provides us with the following information about the picture: a) the picture belongs to an event contained in the last.fm database and b) the corresponding event on last.fm has the ID *#eventid*. Therefore, this allows us to assume that pictures marked with the same machine tag on Flickr belong to the same event. We thus constructed our gold standard by downloading pictures with last.fm machine tags from Flickr using their API and grouping them into events using the event IDs.

We considered pictures with a capture time between January 2006 and October 2011, yielding a dataset of 500,000 pictures assigned to 19,283 events. We divided this dataset into 5 splits consisting of 100,000 pictures each, in temporal order. Only 31.3% of the documents had a geo tag assigned, 90.2% had at least one tag, 97.9% had a title, and 45.9% had a description assigned. Machine tags were only used to create the gold standard and were no longer part of the dataset to conduct our experiments on. Splits 1-3 were used to train the machine learning components and to optimize parameters. We used split 4 to test the blockers, reporting average performance of our system with respect to precision, recall and F-Measure (defined below).

Baseline & Evaluation Measures We report our results in terms of Precision, Recall and F-Measure as defined by Becker et al. (Becker, Naaman, and Gravano 2010).

$$P_b = \sum_{d \in D} \frac{1}{|D|} \frac{|Cluster(d) \cap GoldStandard(d)|}{|Cluster(d)|}$$

$$R_b = \sum_{d \in D} \frac{1}{|D|} \frac{|Cluster(d) \cap GoldStandard(d)|}{|GoldStandard(d)|}$$

$$F_1 - Measure = 2 \cdot \frac{P_b \cdot R_b}{P_b + R_b}$$

Hardware setup All our experiments were conducted on an Intel Xeon E5620, 2.4 GHz system with 96 GiB of main memory. The system was equipped with a solid state disk.

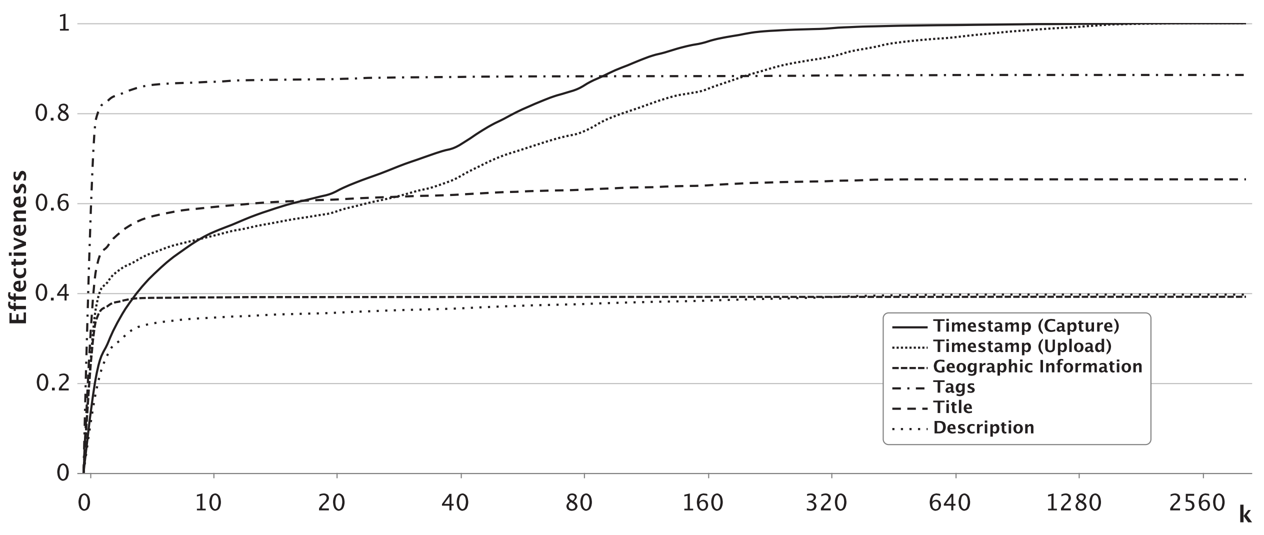

Figure 2: Effectiveness of different blockers

Results

Effectiveness of Blockers

First, we examine the effectiveness of the single-strategy blockers $b_1, .., b_6$. The effectiveness of all these blockers over the number k of events retrieved is depicted in Figure 2. We can observe that only the temporal blockers and the *Tag-TF.IDF blocker* reach higher effectiveness levels of more than 80%. Only the temporal blockers come close to an effectiveness of 100%, but at the cost of having to retrieve more than 300 and 1000 events, respectively. It is interesting to see that the *geo-blocker* reaches a plateau at an effectiveness of close to 40%, but reaches its maximal effectiveness very quickly for $k = 2$.

Table 1 shows for each blocker the number of events that need to be retrieved in order to reach an effectiveness of 30%, 60%, 80% and 90%, respectively. It is remarkable that only three blocking strategies reach an effectiveness of over 90%: the two temporal blockers and the optimal combination. An important observation here is that while the temporal blockers achieve this effectiveness for higher ks (over 100), the optimal combination blocker reaches an effectiveness of 90% after only 4 events retrieved! This is interesting as it suggests that this blocker has a much better cost-effectiveness ratio compared to the timestamp-based blockers. Figure 3 shows the effectiveness and the retrieval time for the optimal combination blocker over different values of k. The optimal combination blocker seems to bring together the advantages of the tag blocker – which returns reasonable results with only few candidates returned but is limited to about 88.5% effectiveness due to the lack of tag data – and the time blockers which need a lot of candidates but can reach up to 100% effectiveness. The combination reaches an effectiveness of 98% at $k = 25$. Further, the diagram also shows that the retrieval time remains fairly constant across different values of k.

It is important to note, however, that the above observations do not imply that the optimal combination blocker has the best cost vs. classification performance tradeoff with respect to our classification task. Thus, we turn to examine the cost-performance ratio of the different blocking strategies with respect to the overall classification performance of the system as measured by the F-Measure.

Table 1: Number of needed k to reach x% effectiveness

Blocking Strategy	**90%**	**80%**	**60%**	**30%**
Capture Time	109	60	17	3
Upload Time	254	112	26	1
Geo	-	-	-	1
Tags	-	2	1	1
Title	-	-	15	2
Description	-	-	-	4
Optimal Combination	4	2	1	1

Classification Performance

Figure 4 compares the four most effective blocking strategies (*Capture Time Nearest, Upload Time Nearest, Uniform Combination, Optimal Combination*) with respect to their F-Measure on the task of classifying documents into their corresponding event. We see that Upload Time Nearest is the best blocking strategy as it yields F-Measures greater than 80% for $k = 10$. The retrieval cost for this blocking strategy is also relatively low as there is only one feature involved and it can be implemented by a single query to the database.

Figures 5 and 6 show the precision and recall values for these four strategies over k. In general, the precision decreases and the recall increases with growing k, which is as expected. We can observe that the Capture Time Nearest strategy has the highest precision compared to all other blockers, thus being indeed the preferable strategy if precision is important. The highest recall, however, is achieved

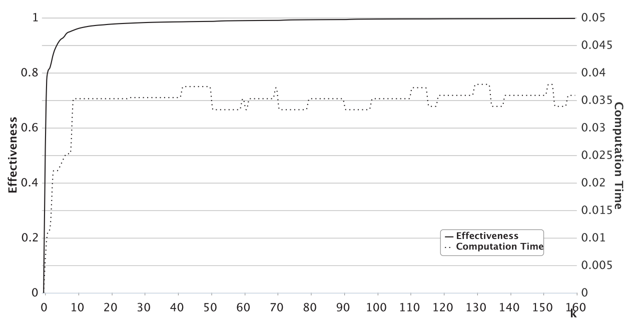

Figure 3: Effectiveness and retrieval time for optimal combinations of single ks

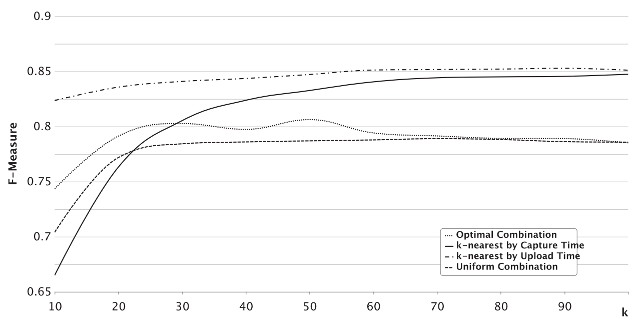

Figure 4: Comparison of the performance (F-Measure) for different blocking strategies over k

by the Upload Time Nearest, which is thus to be preferred if recall is important, i.e. if all documents which actually belong to one event should be grouped together by the classifier.

In Table 2 we compare the F-Measure, Precision and Recall values for our different blocking strategies at $k = 60$ with a configuration of our system that uses no blocking strategy.

We clearly see that the Upload Time Nearest outperforms all other blocking strategies for $k = 60$ and with respect to F-Measure. We also see that, in spite of considering only 60 events, all the blocking strategies outperform a configuration

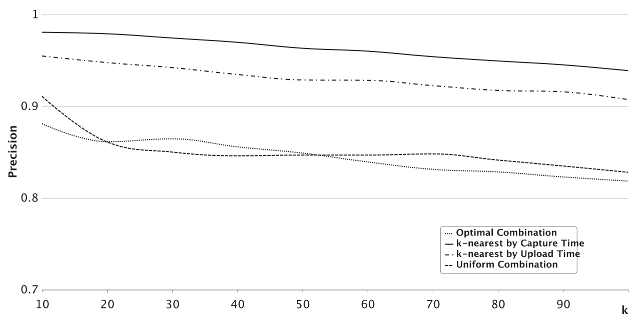

Figure 5: Comparison of the performance (Precision) for different blocking strategies over k

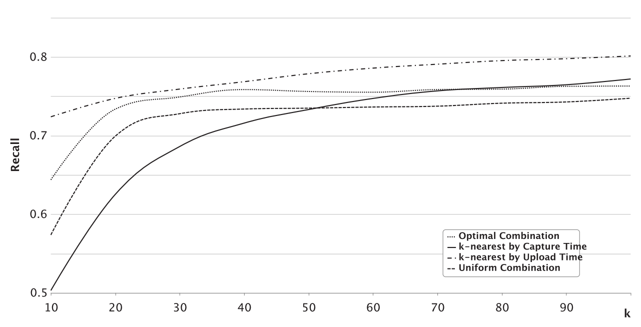

Figure 6: Comparison of the performance (Recall) for different blocking strategies over k

Table 2: Best blocking strategies with $k = 60$ compared to no blocker

Blocking Strategy	F-Measure	P	R
Upload Time Nearest	0.851	0.928	0.786
Capture Time Nearest	0.840	0.960	0.747
Uniform Combination	0.786	0.828	0.745
Optimal Combination	0.794	0.839	0.755
No Blocking	0.589	0.464	0.818

of our system that does not use any blocker.

Overall, our results license the following conclusions:

- The blockers that have the highest cost-effectiveness ratio (e.g. Optimal Combination) are not the best ones with respect to the overall classification performance, being outperformed by simpler blocking strategies that consider only the timestamp (Capture Time Nearest and Upload Time Nearest) and can thus be computed efficiently by a single query, not requiring to combine results from multiple queries.
- Our system reaches already its top performance of 85.1% measure using the *k-nearest by capture time* blocker at only 60 events retrieved.
- Our blocking strategies clearly outperform an approach not using any blocking strategy.

The above results are very interesting and have significant impact on the design of any system attempting to classify social media documents into an evolving set of events.

Scalability and Processing Time

While the time needed to retrieve the candidates is fairly constant across different values of k (see Computation Time in Figure 3), the overall processing time required clearly increases with k. The graph in Figure 7 depicts the overall processing time of our system using the optimal combination blocker over the number k of event candidates considered. We see that the more candidates are retrieved, the higher the computation time per document. In fact, the increase in computational cost seems to be linear in k, so that minimizing k is the key issue towards scaling up the system. Note that once k is fixed, the computation time is constant

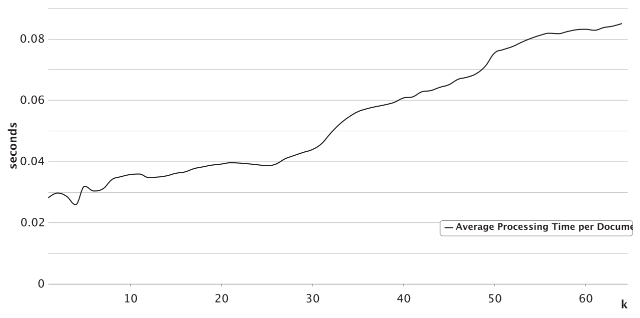

Figure 7: Average processing time for one document using different k (based on Optimal Combination blocker)

regardless of the size of the event database, an important requirement for scalability.

In Table 3 we compare the average processing times per document for the different blocking strategies at the smallest k value that achieves a performance of at least 80% F-Measure. We thus compare the efficiency of the different blocking strategies by maintaining the performance at a comparable level.

Due to the fact that the upload time blocker only needs to retrieve few candidates and therefore less comparisons are needed, the overall computation time for this blocker is the lowest. The reason why the upload time blocker has a higher performance compared to the capture time blocker is the way the events are stored in the database. They are stored in the database ordered by upload time. This saves extra computation time as the data does not need to be resorted. In general, the average processing times per document are in the range of 32-63 ms. Given the high efficiency of the upload time blocker and the fact that it achieved the highest F-Measures overall, it is clearly the overall best blocking strategy for our task.

Related Work

Blocking strategies have been mainly considered in duplicate detection or record linkage tasks where the task is to find equivalent elements. As the search space consists of all pairs over the original set, it is crucial for efficiency reasons to reduce the number of pairs considered. This is typically accomplished by applying an appropriate blocking strategy (Fellegi and Sunter 1969; Hernández and Stolfo 1998; Jaro 1989; Jin, Li, and Mehrotra 2003; Chaudhuri et al. 2003).

Some blockers rely on pre-defined and simple similarity measures such as in our approach, e.g. relying on the Jaccard similarity (McCallum, Nigam, and Ungar 2000) or on some sort of edit distance (Jin, Li, and Mehrotra 2003; Chaudhuri et al. 2003). More sophisticated blockers include canopy blockers (McCallum, Nigam, and Ungar 2000; Rendle and Schmidt-Thieme 2006; 2008) and adaptive blockers (Bilenko, Kamath, and Mooney 2006; Michelson and Knoblock 2006). Canopy blockers form blocks of pairs by randomly selecting one pair from a candidate set and then adding all pairs within a given distance threshold. The pairs not contained in the canopy cluster are then removed from the candidate list (Cohen and Richman 2002; McCallum, Nigam, and Ungar 2000). Hernández and Stolfo (1998) form blocks based on lexicographic sorting of pairs, an approach similar to the one presented by Winkler (). Adaptive blockers learn the blocking function automatically and have been shown to outperform non-adaptive blockers on tasks where address data has to be filtered (Bilenko, Kamath, and Mooney 2006).

In many cases, it is desirable to optimize blockers to maximize their performance on a given task and domain. This is what we have done for the task of classifying social media documents into events. We have in particular shown that a suitable blocking strategy can not only increase the efficiency but most importantly also improve the overall classification performance.

Baxter, Christen, and Churches (2003) have provided an overview and comparison of different blocking methods in the context of the record linkage task. They compare in particular bigram indexing and canopy clustering with sorted neighborhood blocking and standard blocking as proposed by Jaro (1989) and Kelley (1984).

The problem of event identification in social media was introduced by Becker, Naaman, and Gravano (2010). In their paper they presented an incremental clustering algorithm which classified documents into a growing set of events. As the datasets considered by Becker et al. are smaller than ours, no blocking strategy was applied in their work.

Earlier work has also addressed the task of deciding whether a document represents an event or not. Rattenbury and Naaman (2009) learned a classifier which is capable to learn to distinguish Flickr documents which represent an event from those that do not. Firan et al. (2010) use Naive Bayes classifiers to to classify social media documents into events. They induce a function $class : D \rightarrow E$ from training data, thus reducing the problem of event identification to a standard classification task. This problem formulation suffers from the fact that a new classifier needs to be trained for each different event. In contrast, we induce a classifier: $D \times E \rightarrow [0..1]$ which does not have to be retrained for new events.

Conclusions

In this paper we have considered the problem of classifying a stream of social media documents into an evolving set of events. In particular, we have been concerned with the question of how to scale up the classification so that it can be performed in real-time. In order to scale up to the data sizes and data rates in social media applications, the use of a candidate retrieval or blocking step is crucial to reduce the number of events that are considered as potential candidates to which the incoming data point could belong to. In this paper we have experimentally compared different blocking strategies along their effectiveness/computational cost trade-off and shown that using a blocking strategy that selects the 60 closest events with respect to upload time delivers the best results, achieving an F-Measure of 85.1%. From a more general perspective, our results indeed show that a suitable

Table 3: Reaching 80% F-Measure (*Reaching 78.8% as this is the maximum here)

Blocking Strategy	**Precision**	**Recall**	**Processing Time per Document**	**ΔProcessing Time per Document**
Capture Time	**0.974**	0.686	0.054s	+0.022s (+68.7%)
Upload Time	0.955	0.724	0.032s	0.000s (0.0%)
Uniform Combination	0.841*	0.741*	0.063s	+0.031s (+96.9%)
Optimal Combination	0.860	0.748	0.039s	+0.006s (+18.8%)

blocking strategy does not only have the potential to speed up the classification but also to filter out false positives that could potentially confound the classifier, thus increasing the overall classification performance.

Acknowledgements

The research is funded by the Deutsche Forschungsgemeinschaft (DFG), Excellence Cluster 277 “Cognitive Interaction Technology”.

References

Baxter, R.; Christen, P.; and Churches, T. 2003. A Comparison of Fast Blocking Methods for Record Linkage. In *Proceedings of the 2003 ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 25–27.

Becker, H.; Naaman, M.; and Gravano, L. 2010. Learning similarity metrics for event identification in social media. In *Proceedings of the third ACM International Conference on Web search and Data Mining*, 291–300.

Bilenko, M.; Kamath, B.; and Mooney, R. J. 2006. Adaptive Blocking: Learning to Scale Up Record Linkage. In *Proc. of the 6th IEEE International Conf. on Data Mining*, 87–96.

Chaudhuri, S.; Ganjam, K.; Ganti, V.; and Motwani, R. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 313–324. ACM.

Cohen, W., and Richman, J. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 475–480. ACM.

Fellegi, I., and Sunter, A. 1969. A theory for record linkage. *Journal of the American Statistical Association* 1183–1210.

Firan, C. S.; Georgescu, M.; Nejdl, W.; and Paiu, R. 2010. Bringing order to your photos: event-driven classification of flickr images based on social knowledge. In *19th Int'l. Conf. on Information and Knowledge Management*, 189–198.

Hernández, M., and Stolfo, S. 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery* 2(1):9–37.

Jaro, M. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association* 414–420.

Jin, L.; Li, C.; and Mehrotra, S. 2003. Efficient record linkage in large data sets. In *Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on*, 137–146. IEEE.

Kelley, R. 1984. *Blocking considerations for record linkage under conditions of uncertainty*. SRD research report. Bureau of the Census.

McCallum, A. K.; Nigam, K.; and Ungar, L. 2000. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proc. of the 6th International Conf. on Knowledge Discovery and Data Mining*, 169–178.

Michelson, M., and Knoblock, C. 2006. Learning Blocking Schemes for Record Linkage. In *Proc. of the 21st National Conf. on Artificial Intelligence - Vol. 1*, 440–445.

Platt, J., et al. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers* 10(3):61–74.

Rattenbury, T., and Naaman, M. 2009. Methods for extracting place semantics from Flickr tags. *ACM Transactions on the Web* 3(1):1.

Rendle, S., and Schmidt-Thieme, L. 2006. Object identification with constraints. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, 1026–1031. IEEE.

Rendle, S., and Schmidt-Thieme, L. 2008. Scaling Record Linkage to Non-uniform Distributed Class Sizes. *Advances in Knowledge Discovery and Data Mining* 308–319.

Robertson, S., and Jones, K. 1976. Relevance weighting of search terms. *Journal of the American Society for Information science* 27(3):129–146.

Winkler, W. E. 2005. Approximate string comparator search strategies for very large administrative lists. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, D.C.