Overview of the TREC-2013 Microblog Track

Jimmy Lin[1] and Miles Efron[2]

[1] The iSchool, University of Maryland, College Park
[2] Graduate School of Library and Information Science, University of Illinois, Urbana-Champaign

jimmylin@umd.edu, mefron@illinois.edu

1. INTRODUCTION

This year represents the third iteration of the TREC Microblog track, which began in 2011. There was no substantive change in the task definition, which remains nominally real-time search, best summarized as "At time T, give me the most relevant tweets about topic X." However, we introduced a radically different evaluation methodology, dubbed "evaluation as a service", which attempted to address deficiencies in how the document collection was distributed in previous years. This is the first time such an approach has been deployed at TREC. Overall, we believe that the evaluation methodology was successful, drawing participation from twenty groups around the world.

2. EVALUATION AS A SERVICE

Understanding the rationale behind this year's new evaluation methodology requires comparisons to the approach in previous years. Here, we provide a quick recap, but refer the reader to previous track overview papers for more details [1, 2]. In TREC 2011 and 2012, the Microblog track used the Tweets2011 collection, specifically created for those evaluations. Since Twitter's terms of service prohibit redistribution of tweets, it was necessary to develop an mechanism for researchers to obtain the collection. The track organizers devised a system whereby NIST distributed the *ids* of the tweets, rather than the tweets themselves. Given these ids and a downloader developed by the track organizers, a participant could "recreate" the corpus. Since the downloading program accessed the twitter.com site, the tweets were delivered in accordance with Twitter's terms of service.

The "download-tweets-yourself" approach adequately addressed the no-redistribution issue: in 2011, there were 59 official participants, which meant that at least 59 teams were able to successfully acquire the collection. However, the approach has scalability limits: the speed of the downloading program, which had built-in rate limiting for "robotic politeness", set a practical upper bound on the size of the collection. The Tweets2011 collection originally contained only 16 million tweets, which is small by modern standards. For 2013, we hoped to increase the collection size by at least an order of magnitude, which required rethinking data gathering and data access procedures.

Our solution, implemented in TREC 2013, was the "evaluation as a service" model. In summary: we gathered a collection of tweets centrally, but instead of distributing the tweets themselves, we provided a service API through which participants could access the tweets to complete the task. Below, we describe this approach in more detail.

2.1 Collection Construction

To build the official collection, we developed a custom crawler using the twitter4j Java library[1] that gathers tweets from Twitter's streaming API.[2] We crawled all tweets from the public sample stream between February 1 and March 31, 2013, UTC (inclusive). This level of access is available to anyone with a Twitter account and does not require special authorization. The collection was gathered from two separate virtual machine instances on Amazon's EC2 service, one on the east coast of the US, and the other on the west coast of the US. The redundant setup guarded against network outages and other operational issues during the collection period. Fortunately, no downtime was experienced during the data collection period, so one of the copies was simply designated as the official collection.

Messages are delivered in JSON from Twitter's streaming API: these messages contain posted tweets as well as notifications of tweets that have been deleted. The crawler packages all messages in one-hour compressed chunks. Thus, the collection is comprised of 1416 gzipped files. In total, we gathered 259 million tweets, although at the time of the evaluation the collection was reduced to 243 million tweets after the removal of deleted tweets.

We made a decision early in the track planning phase that all software infrastructure associated with the evaluation would be open source and hosted on GitHub.[3] The code for the crawler was developed during January 2013, with input and discussion on a mailing list for developers. By mid-January, we had an operational crawler ready for testing by a few volunteers, and on January 23, 2013, the crawler was released to all participants.

The official collection period was publicized on the track mailing list well in advance of the actual start date, which gave participants the opportunity to run the crawler themselves to gather contemporaneous tweets. Although these crawls may not have the same content as the official collection, they are nevertheless useful for computing term statistics, background models, etc. Based on an informal survey conducted over the track mailing list in November 2013, at least half a dozen groups from around the world gathered their own local private collections.

2.2 API Specification

The idea behind the "evaluation as a service" model is to provide an API that participants can use to complete the

[1]http://twitter4j.org/en/index.html
[2]https://dev.twitter.com/docs/streaming-apis
[3]http://twittertools.cc/

```
struct TQuery {
  1: string group,
  2: string token,
  3: string text,
  4: i64 max_id,
  5: i32 num_results
}

struct TResult {
  1: i64 id,
  2: double rsv,
  3: string screen_name,
  4: i64 epoch,
  5: string text,
  6: i32 followers_count,
  7: i32 statuses_count,
  8: string lang,
  9: i64 in_reply_to_status_id,
 10: i64 in_reply_to_user_id,
 11: i64 retweeted_status_id,
 12: i64 retweeted_user_id,
 13: i32 retweeted_count
}
```

Figure 1: Thrift definition of a search query and a retrieval result.

```
service TrecSearch {
  list<TResult> search(1: TQuery query)
    throws (1: TrecSearchException error)
}

exception TrecSearchException {
  1: string message
}
```

Figure 2: Thrift definition of the search API. The service accepts a query and returns a list of results (as defined in Figure 1).

evaluation task without needing access to the raw collection. To this end, we provided a search API built using Thrift.[4]

Thrift is a software framework for developing scalable services. It was originally developed at Facebook, but is now an open-source Apache project. The framework has gained popularity over the last several years and is currently an integral part of production software stacks at many internet companies, including Facebook and Twitter. Thrift provides an Interface Definition Language (IDL) for describing services and data types. From these definitions, the Thrift compiler automatically generates RPC clients and servers as well as code for serializing, deserializing, and manipulating the defined data types. Thrift handles generation of boilerplate code for communications protocols, object transport, method invocation, and other functionalities necessary to build distributed services. The framework provides support for Java, C++, Python, Ruby, as well as many other languages, which allows the development of language-neutral services. For example, a Python Thrift client can communicate easily with a Thrift server written in Java because the communication protocols and data types are defined in a language-independent manner.

The Thrift definitions of the two main data types in the TREC Microblog search API are shown in Figure 1. The Interface Definition Language is similar to a C struct, and contains an enumeration of numbered fields, each with a type and a name. Most of the types are self-evident; `i32` represents a 32-bit integer (`int` in Java), while `i64` represents a 64-bit integer (`long` in Java). The `TQuery` object represents a query, which contains the query text, a `max_id` (i.e., requests the service to return only results smaller than the id), and the number of results requested. For simplicity, the service is stateless; access control is granted through a `group` and `token`, which must be passed in the query each time. The `TResult` object defines the retrieved result (more details later). The service definition is shown in Figure 2. The single method `search`, receives a `TQuery` object and returns a list of `TResult` objects.

The service for the evaluation was written in Java using the open-source Lucene search engine (version 4.3.1 at the time of the evaluation).[5] We provided a sample client in Java to illustrate the features of the API. In addition, we received the contribution of a Python client from the community, which was later integrated into the code base.

Search ranking was provided using Lucene's implementation of query-likelihood (`LMDirichletSimilarity`). Results were filtered such that tweets with ids greater than `max_id` (as specified in the `TQuery` object) were discarded. Each search result was populated with the fields described in Table 1 (corresponding to the Thrift definition in Figure 1). For each field, the table also provides its corresponding element in the original JSON messages from the Twitter streaming API, whether the element is optional (for example, only retweets have certain fields), the corresponding Java data type, and a short description.

The service endpoint was developed during Spring 2013 and was released to all registered TREC participants starting in June. We released two distinct services: one on the Tweets2011 collection created for the Microblog tracks in TREC 2011 and TREC 2012, and another on the new collection gathered in 2013. Both services behaved exactly the same, except on different document collections. Since evaluation data were available for the Tweets2011 collection, that service allowed participants to train their systems on old topics.

3. RESULT OVERVIEW

In this year's evaluation, we received 71 runs from twenty groups. Relevance judgments were created using the standard pooling methodology by NIST assessors. Runs were pooled to depth 90, according to the retrieval scores indicated in each run. Simple retweets were removed from the pools (as they were deemed to be non-relevant). The tweets were clustered so that textually similar tweets could be judged consistently. Relevance judgments were made on a 3-point scale ("not relevant", "relevant", "highly relevant"), but the following results consider both "relevant" and "highly relevant" tweets to be relevant.

To provide a reference baseline, we constructed a post hoc run that processed the raw API output in three minor ways:

- All retweets were discarded.
- Duplicate tweets were removed (a small number of tweets were delivered multiple times via the streaming API due to transient network glitches).

[4]http://thrift.apache.org/

[5]http://lucene.apache.org/

Table 1: Detailed Description of a Search Result.

Thrift field	JSON element	Optional?	Type	Description
`id`	`status.id`	no	long	unique tweet id assigned by Twitter
`rsv`		no	double	retrieval status value, i.e., document score
`screen_name`	`status.user.screen_name`	no	String	user who posted the tweet
`epoch`	`status.created_at` (derived)	no	long	UNIX epoch second when the tweet was posted
`text`	`status.text`	no	String	text of the tweet
`followers_count`	`status.user.followers_count`	no	int	the number of followers the user has
`statuses_count`	`status.user.statuses_count`	no	int	the number of tweets the user has posted
`lang`	`status.lang`	yes	String	the two-character language of the tweet as described by the Twitter language id system
`in_reply_to_status_id`	`status.in_reply_to_status_id`	yes	long	the id of the tweet that this tweet replies to
`in_reply_to_user_id`	`status.in_reply_to_user_id`	yes	long	the id of the user who posted the tweet that this tweet replies to
`retweeted_status_id`	`status.retweeted_status.id`	yes	long	the id of the tweet that this tweet is a retweet of
`retweeted_user_id`	`status.retweeted_status.user.id`	yes	long	the id of the user who posted the tweet that this tweet is a retweet of
`retweeted_count`	`status.retweet_count`	yes	int	number of times this tweet has been retweeted

Table 2: Summary of Effectiveness Metrics. *Median* and *Mean* are calculated from all submitted runs.

	MAP	P30	R-prec
Baseline	0.2524	0.4500	0.3008
Median	0.2217	0.4311	0.2796
Mean	0.2271	0.4111	0.2758

Table 3: Number and Fraction of Runs above the Baseline.

Year	Number	Fraction
2011	20 out of 184	0.109
2012	79 out of 122	0.648
2013	30 out of 71	0.423

- Score ties were broken by recency (i.e., more recent tweet were ranked higher).

The effectiveness of all submitted runs is shown in Table 4, with the baseline inserted for comparison purposes. The final column ("Type") denotes whether the run received human intervention (Manual) or was completely automatic (Auto). Summary statistics are shown in Table 2. We see that the median submitted run appears to be worse than the baseline.

How do these results compare to previous years? In Figures 3 and 4 we sort all submitted runs from TREC 2011–2013 by either MAP or P30. The red line corresponds to the median run and the blue line corresponds to the baseline. In TREC 2011 and TREC 2013, the median lies below the baseline, but in TREC 2012, the median system outperforms the baseline. The number and fraction of runs that beat the baseline are shown in Table 3.

An important question for the community to consider is the effect of the evaluation-as-a-service model with respect to experimental innovation. It may be possible that forcing teams to interact with the collection via an API leads to less diversity in techniques than we would see if participants had direct access to the collection. The diversity of techniques contributes to the diversity of the document pools, which affects the reusability of the evaluation resources. We are currently conducting analyses that hopefully will shed some light on this question.

4. ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation under IIS-1217279 and IIS-1218043. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the sponsor.

5. REFERENCES

[1] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the TREC-2011 Microblog Track. In *Proceedings of the Twentieth Text REtrieval Conference (TREC 2011)*, Gaithersburg, Maryland, 2011.

[2] I. Soboroff, I. Ounis, C. Macdonald, and J. Lin. Overview of the TREC-2012 Microblog Track. In *Proceedings of the Twenty-First Text REtrieval Conference (TREC 2012)*, Gaithersburg, Maryland, 2012.

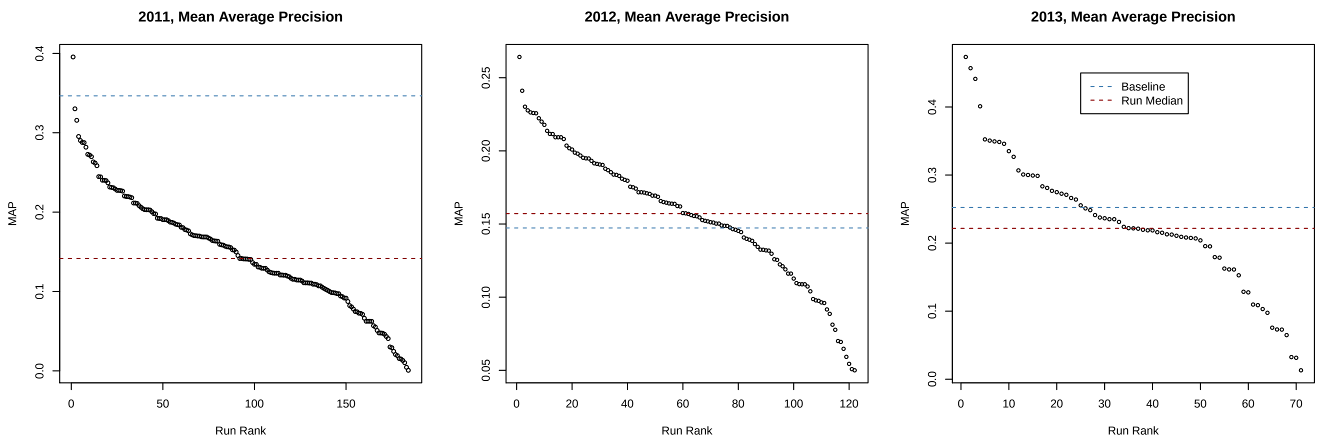

Figure 3: Runs from TREC 2011–2013 ranked by mean average precision.

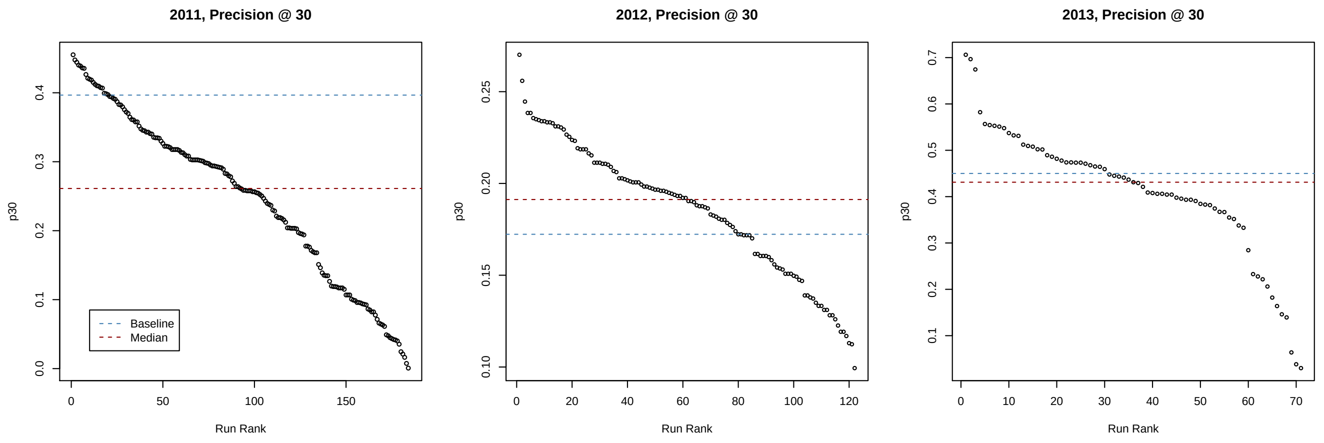

Figure 4: Runs from TREC 2011–2013 ranked by P30.

Table 4: Overview of Retrieval Effectiveness. All 71 runs submitted to the evaluation are ranked in decreasing order of mean average precision. The entry marked baseline was not part of the original evaluation, and is included only for reference purposes.

Run	MAP	P30	R-prec	Type
Direrank	0.4735	0.6967	0.5172	Manual
Avgrank	0.4570	0.7061	0.5033	Manual
FSsvm	0.4413	0.6744	0.4882	Manual
RvsDir	0.4010	0.5822	0.4931	Manual
PrisRun4	0.3524	0.5528	0.3861	Auto
PrisRun3	0.3506	0.5544	0.3850	Manual
QCRI4	0.3494	0.5372	0.3905	Auto
PKUICST3	0.3486	0.5567	0.3827	Auto
PrisRun2	0.3459	0.5511	0.3844	Auto
PKUICST1	0.3351	0.5478	0.3721	Auto
PKUICST2	0.3268	0.5311	0.3637	Auto
QCRI3	0.3068	0.4817	0.3472	Auto
ILPSub	0.3008	0.5322	0.3456	Auto
QCRI2	0.3001	0.4733	0.3441	Auto
QCRI1	0.2993	0.4678	0.3476	Auto
PrisRun1	0.2988	0.5078	0.3429	Auto
ILPSl2rE	0.2834	0.4894	0.3281	Auto
UDInfoMB	0.2811	0.5022	0.3285	Auto
PKUICST4	0.2768	0.5017	0.3260	Auto
QUTemporal	0.2748	0.4739	0.3245	Auto
NOVAsearch00	0.2726	0.4711	0.3171	Auto
QUQueryExp	0.2710	0.4433	0.3128	Auto
ICTNETBO1EXP	0.2663	0.4644	0.3117	Auto
kobeTSFRM	0.2640	0.4861	0.3146	Manual
QUBaseline	0.2555	0.4294	0.2936	Auto
Baseline	**0.2524**	**0.4500**	**0.3008**	**Auto**
ICTNETCOCCUR	0.2510	0.4594	0.2923	Auto
UDInfoMTB1	0.2484	0.4778	0.2873	Auto
GSAA	0.2412	0.4061	0.2996	Auto
stan2kl	0.2376	0.4411	0.2787	Auto
kobeTSFRMU	0.2365	0.4733	0.2867	Auto
GSAS	0.2351	0.4044	0.2920	Auto
GSAT	0.2351	0.4044	0.2920	Auto
QUDocExp	0.2311	0.4478	0.2809	Auto
NOVAsearch02	0.2239	0.4450	0.2738	Auto
CIRGIRDISCO4	0.2220	0.4078	0.2871	Auto
kobeU	0.2217	0.4211	0.2696	Auto
BAUENGPGRNK	0.2212	0.3933	0.2866	Auto
QEClustIDF	0.2196	0.5094	0.2640	Auto
QEDiscIDF25Good	0.2188	0.5122	0.2626	Auto
BAUENPRKST	0.2187	0.3906	0.2818	Auto
CIRGIRDISCO2	0.2160	0.3817	0.2796	Auto
CIRGIRDISCO3	0.2152	0.3828	0.2788	Auto
BAUENGFLT	0.2129	0.3956	0.2789	Auto
kobeRMU	0.2125	0.4311	0.2538	Auto
UDInfoMINT	0.2108	0.3978	0.2654	Auto
BAUENGSTAT	0.2092	0.3844	0.2750	Auto
NOVAsearch01	0.2082	0.4367	0.2530	Auto
ModelSEL922	0.2076	0.4739	0.2514	Auto
UDInfoMTB2	0.2069	0.4061	0.2614	Auto
DFRBase	0.2040	0.4650	0.2494	Auto
scunce4	0.1955	0.3933	0.2590	Auto
BNTSrK	0.1952	0.3672	0.2628	Auto
scunce1	0.1794	0.3744	0.2457	Auto
scunce2	0.1786	0.3667	0.2494	Auto
scunce3	0.1626	0.3517	0.2335	Auto
ICTNETBASE	0.1613	0.3378	0.2136	Auto
NOVAsearch03	0.1612	0.3550	0.2092	Auto
ILPSdf	0.1527	0.4089	0.2020	Auto
UCASgem	0.1285	0.2844	0.2034	Auto
UCASqe	0.1276	0.2217	0.1880	Auto
stan4col	0.1097	0.2278	0.1426	Auto
BJUTFreq	0.1088	0.2328	0.1610	Auto
BNTSrKSO	0.1031	0.2061	0.1379	Auto
WISSySeCo	0.0976	0.3328	0.1430	Auto
iritfdUrlRoc	0.0757	0.1461	0.1339	Auto
BJUTEntr	0.0731	0.1639	0.1174	Auto
BJUTCnor	0.0729	0.1822	0.1137	Auto
iritfdUrl	0.0648	0.1394	0.1146	Auto
stan1kl	0.0325	0.0383	0.0425	Auto
stan3kl	0.0315	0.0300	0.0401	Auto
ILPSl2rB	0.0131	0.0639	0.0370	Auto