QU at TREC-2015: Building Real-Time Systems for Tweet Filtering and LiveQA

Reem Suwaileh, Maram Hasanain, Marwan Torki, Tamer Elsayed

Computer Science and Engineering Department
Qatar University
Doha, Qatar
{reem.suwaileh,maram.hasanain,mtorki,telsayed}@qu.edu.qa

ABSTRACT

This paper presents our participation in the microblog track and LiveQA tracks in TREC 2015. Each of the two tracks required building a "real-time" system, i.e., a live system that monitors a stream of data and responds to the user's need in real-time, which was a new and exciting experience to our team.

For the microblog track, we developed two online filtering systems for recommending "relevant" and "novel" set of tweets from Twitter stream given a set of users' interest profiles. The two systems simulate real scenarios: push notifications on a mobile phone and periodic email digest. One idea we tried evolves around applying static versus dynamic thresholds to control the filtered output. We also experimented with different profile expansion strategies that account for potential topic drifts. Our preliminary results (based solely on submitted TREC runs) show that the run of push notifications scenario that used static threshold with light profile expansion achieved better results than other runs with dynamic threshold and larger expansion. Similar phenomenon was observed in the email digest scenario, where the run that used a short representation of the interest profiles without any expansion achieved the best result.

For the LiveQA track, the system was required to answer a stream of more than 1000 real-time questions from Yahoo! Answers platform. We adopted a very simple approach for this year; we have simply searched an archived Yahoo! Answers QA dataset for similar questions to the asked ones and retrieved back their answers. We plan to use this system as a baseline for our future participation in the track.

1. INTRODUCTION

Twitter has rapidly developed over the past years to become a massive information sharing network. It gained a reputation for carrying the *heartbeat* of the world by allowing users to continuously post and read tweets about current events and news. With a huge flood of tweets shared daily, users are overwhelmed by the amount of data they need to follow in order to track a certain event or a topic. Automatically-satisfying the user interest in following a certain topic over a non-stop stream of tweets is challenging as it requires a system to *filter* both relevant and novel tweets from a huge stream in real-time. Moreover, the nature of tweets increases the challenge since they are extremely short with a maximum of 140 characters, conversational, and usually highly-temporal.

Proposing a filtering system in such case requires addressing the problems originating from these challenges. Some of the main problems to handle include the short length of tweets and the queries the user usually posts to describe her information need. One successful way of overcoming this problem is by expanding the context of the tweets and the queries. Along with this solution comes the possible case of topic drift [1] especially that the stream is rapidly developing and topic representation should keep-up with that. In this work, we propose an online tweet filtering system that tries to handle these challenges for the two scenarios in the TREC-2015 microblog track. The problem and our approach is further discussed in section 2.

As with Twitter, social question answering (sQA) systems are increasingly gaining importance and attracting millions of users to post and and answer questions. Yahoo! Answers[1] is by far one of the largest sQA platforms. Questions and answers on such platforms share some characteristics with tweets in terms of being conversational and often very socially-oriented. In the past few years, question answering (QA) as a problem has received significant attention; while there is a lack of suitable datasets to train automated QA systems, this problem becomes a real challenge especially when the focus goes beyond seeking short answers to factoid questions, to questions that require reasoning, explanations, etc. The datasets used in such previous systems are either small to train a good system or not extensive to involve multiple domains [4]. However, the existence of large social question answering websites, such as Yahoo! Answers specifically, makes the development of automated answering systems more interesting due to the scale of the available datasets. But the amount of questions posted on such platforms is not usually matched by the number of answers provided; many questions are left unanswered and many are repeated. Automatically answering questions in such cases is a useful feature users of these sQA can benefit from which intrigues a need for designing automatic sQA systems. This is in fact the problem we tackle as part of the LiveQA track in TREC 2015.

The rest of this paper is organized as follows: Section 2, describes our approach in the online filtering track for both tasks: tweets as push notifications on a mobile phone

TREC'15 Gaithersburg, Maryland USA

[1] https://answers.yahoo.com/

and a periodic email digest. The section also covers the configurations for different runs and the results. Section 3, presents our approach and evaluation results for the LiveQA track. We present some concluding remarks in Section 4.

2. REAL-TIME TWEET FILTERING

Tweet filtering is the reverse of ad-hoc tweet search, where a user provides a topic at a certain point in time, and the filtering system is expected to filter tweets that are relevant to the topic from a stream of tweets posted after the query time.

Tweet filtering as a TREC task was first introduced in the microblog track in TREC-2012 [3]. The task ran on a simulated stream of tweets from a collection of 16M tweets [3] that was already crawled. This year, the microblog track in TREC-2015 ran a related tweet filtering task, but, differently from the task in TREC-2012, this year's task was a real-time task carried on a real stream of tweets in a 10-day evaluation period. Moreover, the 2015 task has two modes: 1) a scenario that assumes the user is expecting few filtered tweets as push notifications on a mobile phone, and 2) a scenario where the user expects periodic email digest of a larger set of tweets summarizing the topic [2].

In both scenarios, the filtering system is expected to send a set of *novel* (i.e., non-redundant) and *relevant* (i.e., on-topic) tweets every day. The interest of a user in a topic can be represented in different ways. In this task, an *interest profile* is represented as title, description, and narrative of the topic. The title is short, having few keywords to represent the topic; the description is one sentence stating the information need of the user; and the narrative is a full paragraph describing that information need. Figure 1 shows a high-level overview of the filtering system.

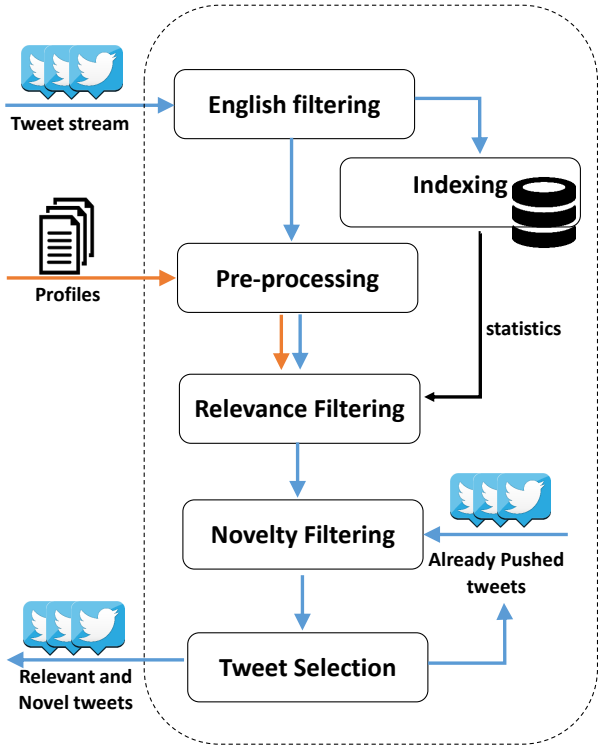

Figure 1: An Overview of microblog filtering system

2.1 Approach

In this section, we first present the main approach of our solution for both scenarios, and then discuss further details that are specific to each of them.

2.1.1 Cold Start and Preprocessing

Following the track guidelines, we only consider English tweets for filtering. We detect English tweets using an open-source language detection tool[2]. Additionally, we perform simple preprocessing on tweets and profiles including stemming, stopwords removal, and URL removal. We represent our tweets and profiles as vectors of terms weighted using BM25 model. The term weights are computed using a history of past tweets. To solve that "cold start" problem, the system is initialized with a 3-day stream of tweets preceding the beginning of the evaluation period. We index the tweets and update the index periodically with incoming tweets during the evaluation period.

2.1.2 Relevance and Novelty

For both scenarios, we used the following approach for filtering based on relevance and novelty. Given an interest profile for a topic, the filtering system processes the incoming stream one tweet at a time. For each tweet, a relevance score to the interest profile is computed using cosine similarity. A tweet with a similarity score above a relevance threshold τ_r is considered relevant to the topic. A tweet to be pushed should be both relevant and novel within and across all the evaluation days. To achieve this goal, we maintain a list of filtered tweets that were already sent to user. We measure novelty of an incoming relevant tweet by computing its similarity to each of the tweets in that list, using a modified version of Jaccard similarity. If the similarity of the tweet to any of the tweets in the list exceeds a novelty threshold τ_n, then the system elects not to send it to user, as it is considered redundant. Otherwise, the tweet is considered both relevant and novel, and thus the system tags it to be sent to the user; it is also added to the list of sent tweets.

2.1.3 Topic Updates

Since the topics are followed over a period of time, we periodically update the topic representation to reflect the topic development over time. To achieve that, the system performs profile expansion using pseudo relevance feedback; expansion terms are extracted from pseudo relevant tweets filtered before the expansion time.

2.1.4 Push notifications on a mobile phone scenario

This scenario simulates a situation where updates about a topic are sent as mobile notifications to the user. The task requires pushing a maximum of 10 tweets only per day not to overload the user with notifications. The main challenge in pushing such small number of tweets is considering how fresh the tweets are in addition to how relevant and novel they are. The freshness of tweets can be measured in this context considering the elapsed time between the tweet's

[2]https://code.google.com/p/language-detection/

creation time (the time when tweet appears in the stream) and the time it was pushed to the user.

We represent the interest profile (i.e., topic) as a vector of terms. The initial interest profile $\vec{q}$ is represented as follows:

$$\vec{q} = \vec{title} + \alpha(\vec{desc} + \vec{narr}) \quad (1)$$

Where $\vec{title}$, $\vec{desc}$ and $\vec{narr}$ are the vector representations of the title, description, and narrative of the profile respectively. The title is given higher weight since it contains the most concise and relevant description of the topic while the description and narrative contribution to the topic is controlled by the parameter α (< 1). Since narrative and description are much longer than the title and might share terms, we further control their contribution to the topic vector by selecting top k weighted terms of the resulting vector $(\vec{desc} + \vec{narr})$ to be added to the final topic vector.

Following all interest profiles in parallel, the system monitors the Twitter stream and filters tweets based on their similarity and novelty given each of the profiles. Over time, the system maintains a list of relevant and novel tweets for each profile, only pushing a tweet from that list periodically with a time period of length δ or when the size of the list exceeds a limit l. We select a tweet to be pushed from that list by selecting the tweet with the maximum score after scoring all tweets in the list by the following scoring function:

$$S(t) = S_r(t) * \frac{100 - (CurTime - time(t))}{100} \quad (2)$$

Where $S_r(t)$ is the relevance score of tweet t computed using cosine similarity between a profile and the tweet, $CurTime$ is the current system time, and $time(t)$ is the tweet creation time. The tweet score $S(t)$ considers how relevant it is to the profile and how fresh as well.

When computing relevancy of a tweet to a profile, we consider a tweet to be relevant if its relevance score exceeds a threshold τ_r. As the system filters more tweets over time which provides an opportunity for learning, and since the topic itself can evolve over time, we experiment with using a dynamically-updated relevance threshold τ_r. We explore using both static and dynamic relevance threshold τ_r settings. In the static threshold mode, the relevance threshold τ_r is set before the system starts and it does not change over the evaluation period. The approach we follow in the dynamic threshold mode is to start the system with an initial threshold for each interest profile p_i and the system updates per-profile relevance threshold τ_r periodically. To update the threshold, we increase or decrease it by a simple ratio based on the number of already filtered relevant tweets. If the profile p_i gets no relevant tweets, then the relevance threshold τ_r is decreased by 0.025, ensuring the final threshold does not go below 0.5. In the opposite case, if a profile gets more than one relevant tweet the threshold is updated as follows:

$$\tau'_{r_i} = \tau_{r_i} + min(\frac{R_{p_i}}{100}, 0.15) \quad (3)$$

Where τ_{r_i} is the current threshold of profile p_i, τ'_{r_i} is the updated threshold of that profile and R_{p_i} is the number of relevant tweets filtered for profile p_i within a time period t_T. The threshold upper bound is 0.95. Furthermore, in order to get the maximum possible throughput out of the filtering system, profile expansion is performed as follows:

$$\vec{q}' = \vec{q} + \beta(\vec{e}) \quad (4)$$

Where $\vec{q}$ is the initial profile represented in 1, $\vec{q}'$ is the expanded profile and $\vec{e}$ is the expansion terms extracted from the pseudo relevant tweets for each profile independently.

2.1.5 Periodic email digest scenario:

Email digest system is a periodic filtering system that populates a daily list of top $n = 100$ relevant and non-redundant tweets from Twitter stream to each interest profile as an email digest. Initial profiles are represented in the vector space by the weighted profiles' title terms:

$$\vec{q} = \vec{title} \quad (5)$$

By the end of each day in the evaluation period, we search the aforementioned index; that is built before the evaluation period started and is updated periodically during the 10 evaluation days using all the initial profiles as queries. By doing so, we retrieve the most relevant $2n$ tweets to each profile by using the LM Dirichlet similarity model. Both indexing and searching are done by Lucene 4.5.0[3]. After that, the novelty filtering is applied on each $2n$ result list to compile an email digest that should contain only the $n = 100$ novel tweets per profile for the current day. The novelty of a tweets is determined by computing it is similarity to all tweets in all previous email digests of the past days within the evaluation period. We compute that similarity using a modified version of Jaccard similarity. If the similarity exceeded novelty threshold τ_n, the tweet is treated as redundant and then discarded, otherwise the tweets is added to the email digest of a certain interest profile that will be sent to the user.

Our focus in this scenario is to experiment different sources of a periodic profile expansion; starting with a baseline run that does not execute any expansion, then compare it to two different configuration settings. In the first expansion configuration, both description and narrative fields of the interest profile are used as a source to extract expansion terms since they contain the most relevant terms to the topic. The expansion in this situation is performed as follows:

$$\vec{q}' = \vec{q} + \alpha(\vec{desc} + \vec{narr}) \quad (6)$$

Where $\vec{q}'$ is the expanded profile and $\vec{q}$ is the initial profile represented in 5. The $\vec{desc}$ and $\vec{narr}$ are the terms' vectors of both description and narrative, respectively. α is a parameter used to give more weight for the initial profile terms, the profile's title, in the expanded profile representation. A main constraint here is that there should not be any overlap between the extracted expansion terms and the initial profiles terms.

In the second type of expansion, the expansion terms are combination of terms extracted from the interest profile fields; the same as 6, in addition to the top m weighted terms extracted from the pseudo relevant tweets returned by searching the index. The expansion equation used is the following:

$$\vec{q}' = \vec{q} + \alpha(\vec{desc} + \vec{narr}) + \beta\vec{e} \quad (7)$$

[3]https://lucene.apache.org/

Where $\vec{e}$ is the vector of the extracted expansion terms from pseudo top results. All selected expansion terms should not overlap with profile title terms.

2.2 Experimental Evaluation

2.2.1 Evaluation measures

The push notification scenario is evaluated using two main measures: the primary measure which is expected latency-discounted gain (ELG) and the normalized cumulative gain (nCG) [2]. Measures can be computed as follows:

$$ELG = \frac{1}{R_t} \sum Gain(t_i) \tag{8}$$

$$nCG = \frac{1}{Z} \sum Gain(t_i) \tag{9}$$

where $Gain(t_i)$ is 0 if the tweet is not interesting, a spam or junk, $Gain(t_i)$ is 0.5 for a tweet that is somewhat interesting and a gain of 1 is given to very interesting tweets. The gain of a tweet is reduced using the following time latency penalty:

$$latency = max(0, \frac{100 - delay}{100}) \tag{10}$$

where delay is the difference in minutes between tweet creation time and tweet push time.

As for the tweet e-mail digest scenario, the e-mail digest is considered as a ranked-list of tweets. The evaluation measure used in this task is the normalized discounted cumulative gain (NDCG) computed at length k of this list.

For all evaluation measures in all scenarios, the score of a topic is the average measure values over all days and a score of a run is the average of that over all topics.

2.2.2 Submitted Runs

This section describes our submitted runs of both scenarios. Table 2.2.2 shows the different configuration settings is each run.

Push notifications on a mobile phone runs:

- **QUBaseline:** is a baseline that uses a static relevance threshold τ_r when comparing a tweet to a profile. The system in this run uses title, narrative and description to represent the topic. The parameter $\alpha = 0.2$ controls the weight given to terms from description and narrative fields in profile representation, and we only use 8 terms from these two fields. The profile is periodically expanded using a maximum of 4 terms extracted from pseudo relevant tweets, while controlling weights of those terms by a parameter $\beta = 0.2$
- **QUDyn:** Similar to QUBaseline, but the relevance threshold is updated dynamically for each profile independently.
- **QUDynExp:** This run uses dynamically-set threshold as in QUDynm, but the number of expansion terms from pseudo relevant tweets is set to 12 and we select 10 terms from narrative and description fields to include in profile representation. We also set $\beta = \alpha = 0.3$

Table 2 shows the results for the push notification task. The score of a run is computed as an average score of all profiles over all days. It can be clearly seen that the baseline run outperforms the other runs.

Table 2: Results for runs of the tweet push notification scenario

Run	ELG	nCG
QUBaseline	0.2750	0.2347
QUDyn	0.1850	0.1762
QUDynExp	0.1848	0.1763

Periodic email digest runs:

- **QUBaselineB:** is the baseline run where the initial profile is issued as a query against the index of tweets posted within the same day and no profile expansion is performed.
- **QUExpB:** in this run, a light query expansion is performed. The expansion terms are extracted only from the description and narrative fields of the profile.
- **QUFullExpB:** this run performs a daily full expansion, and by full expansion we mean the expansion terms used are extracted from the top weighted terms from description and narrative beside the top terms obtained from the pseudo relevant tweets returned by searching the index. The number of top terms taken from the both sources is controlled by parameters to balance their influence. The first and second parts of expansion equation are controlled by α and β parameters that should be in the range between 0 and 1. Where $\alpha = 0.3$ and $\beta = 0.2$. One main constrain in this run is that the number of terms in the final profile after expansion should not exceed 20 terms.

In all expansion types, the extracted expansion terms should not overlap with the profile's title terms. Table 3 shows the results of tweet e-mail digest scenario. The results of all runs in the table are somewhat similar and the baseline run is slightly better than other runs.

Table 3: Results for runs of the tweet e-mail digest scenario

Run	NDCG
QUBaselineB	0.1288
QUExpB	0.1180
QUFullExpB	0.1196

3. LIVEQA

The LiveQA task is introduced this year for the first time in TREC. In this track, systems are supposed to return answers to real-time questions originating from real users via a live question stream. The language for the track is English for both questions and answers.

Several restriction are applied to the track to make it more challenging. First restriction is the maximum allowed response time, which is set to be one minute only. Second is the maximum answer length, which is set to just 1000 characters. Third is the limit on returned answers by the system, which is set to be the top retrieved ranked answer by the system.

Table 1: Summary of submitted runs for the two scenarios

Scenario	Run	Dynamic τ_r?	Base τ_r	Expansion?	# Pseudo tweets	# Expansion terms
Mobile Push Notification	QUBaseline	×	0.6	✓	20	4
	QUDyn	✓	0.8	✓	20	4
	QUDynExp	✓	0.8	✓	20	12
Periodic Email Digest	QUBaselineB	×	-	×	-	-
	QUExpB	×	-	✓	5	3
	QUFullExpB	×	-	✓	10	10

The approach we followed for this task was to implement a simple system, which can be considered as a baseline for our future work on that problem/track for next year(s). The intuition behind that approach is also simple; since the expected questions are as of same type asked on Yahoo! Answers, we chose to use only Yahoo! Answers as the source of retrieving answers.

3.1 System Pipeline

We describe the pipeline for question answering as follows:

1. **Build/get access to an archive of questions and corresponding answers.** In this step, we use the question answers dataset provided by [5].
2. **Index the archived the question answer dataset.** In this step we use Lucene 4.7.0[4] with stop words removal. The objective here is to search the questions answers dataset for similar questions in real time.
3. **Retrieve potential answers.** For a newly-asked question, we search the pre-built index to find similar question(s). Answers for similar questions are retrieved and considered as potential answers for the asked question.
4. **Limiting the answer size.** In some cases, the size of the potential answer exceeds the 1000-character limit set by TREC LiveQA task. Therefore, we limit the number of characters to 1000 character at maximum. We truncate those answers by returning the first k sentences whose total size is less than 1000 character.
5. **Rank the potential answers.**

   We used a heuristic based on some of the features defined in [5]:

   - **Overall Match(AM):** Number of non-stop question terms matched in the complete answer.
   - **Answer Span(AS):** The largest distance (in words) between two non-stop question words in the answer.
   - **Same Word Sequence(WS):** The number of non-stop question words that are recognized in the same order in the answer.

   We score every possible answer a_i with respect to a question q as follows:

$$score(q, a_i) = AM(q, a_i) + WS(q, a_i) - AS(q, a_i) \quad (11)$$

   Finally, we return only the answer a_i with the highest score as our retrieved answer.

[4]https://lucene.apache.org/

3.2 Evaluation

3.2.1 Evaluation Measures

The evaluation for TREC LiveQA track is based on 1087 questions. These 1087 questions were judged and scored using 4-level scale:

- 4: Excellent – a significant amount of useful information, fully answers the question.
- 3: Good – partially answers the question
- 2: Fair – marginally useful information
- 1: Bad – contains no useful information for the question
- -2: the answer is unreadable (only 15 answers from all runs were judged as unreadable)

Based on the four levels labeling, the evaluation measures adopted by TREC are:

- avg-score(0-3) – average "quality" score over all queries
- succ@i+ – number of questions with i+ score (i = 1..4) divided by number of all questions
- prec@i+ – number of questions with i+ score (i = 2..4) divided by number of answered only questions

3.2.2 Results

We present our results as reported by TREC in table 4. The first column names the evaluation measure, the second column shows our reported results, and third column shows the average results over all submitted runs of the different participating teams in the track.

Table 4: QU-Results for LiveQA task compared to the average of the submitted runs.

Evaluation Measure	QU	Track Average
avg score (0-3)	0.256	0.465
succ@1+	0.995	0.925
succ@2+	0.163	0.262
succ@3+	0.070	0.146
succ@4+	0.023	0.060
prec@2+	0.164	0.284
prec@3+	0.070	0.159
prec@4+	0.023	0.065

We show in table 5 the breakdown of the labels given to our system. We compare our breakdown to the average reports over the participating systems in the track. These percentages are calculated using the reported **succ@i+** measures.

Table 5: Labels of QU run for LiveQA task compared to the average of the submitted runs.

Label	QU	Track Average
Excellent(4)	2.3%	6%
Good(3)	4.7%	8.6%
Fair(2)	9.3%	11.6%
Bad/Unrecognized(1& -2)	83.7%	73.8%

3.2.3 Discussion

The behavior of our system was fairly expected, as we did not incorporate more sophisticated steps in the body of our pipeline. Several enhancements can be applied in many parts of the system. For example, the system we presented uses only Yahoo! Answers; this is a limiting choice since there are more sources like Google web search, Wikipedia, and Quora that can be leveraged as well. Also, the ranking function is just a heuristic. We can use a more principled way to rank the answers using a learning to rank approach. Adding enhancements to the basic system we presented here might improve the performance in the future runs of the LiveQA track.

4. CONCLUSION

In our participation at TREC15, we developed filtering systems for Microblog track and question-answering system for LiveQA Track. The main focus of our filtering systems was to experiment with different relevance threshold modes: static and dynamic thresholds. In addition, we performed expansion of interest profiles to enrich the profile representation while giving the profile's title the largest influence to avoid drifting from the original topic. The expansion was applied in both push notification and email digest scenarios. The results show that the runs with static thresholds and light or no expansion in both scenarios outperform the other runs by all evaluation measures. As future work, we plan to investigate the reasons behind the relatively poor performance of dynamic threshold as well as profile expansion strategies.

For Live QA track, we implemented a very simple system for the question-answering task; For a given posted question, we retrieved answers to questions that are similar to the asked question but were posted in the past; we then rank those answers using a heuristic approach and return one within the limit of 1000 characters imposed by the track. The system can be considered as a baseline for our future work with many possible directions for improvements that include enriching the sources of the answers and incorporating a more principled way for ranking answers.

5. ACKNOWLEDGMENTS

This work was made possible by NPRP grant# NPRP 6-1377-1-257 and NPRP grant# NPRP 7-1313-1-245 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

6. REFERENCES

[1] M. Albakour, C. Macdonald, I. Ounis, et al. On sparsity and drift for effective real-time filtering in microblogs. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 419–428. ACM, 2013.

[2] J. Lin. Trec 2015 track guidelines. `https://github.com/lintool/twitter-tools/wiki/TREC-2015-Track-Guidelines`.

[3] I. Soboroff, I. Ounis, J. Lin, and I. Soboroff. Overview of the trec-2012 microblog track. In *Proceedings of TREC*, volume 2012, 2012.

[4] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers on large online qa collections. In *ACL*, pages 719–727, 2008.

[5] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to Rank Answers on Large Online QA Collections. In *ACL*, pages 719–727, 2008.