Search Intent Mining by Word Vectors Clustering at NTCIR-IMine

Jose G. Moreno
LIMSI, CNRS,
Université Paris-Saclay,
F-91405 Orsay
moreno@limsi.fr

Gaël Dias
GREYC/CNRS
gael.dias@unicaen.fr

ABSTRACT

This paper presents a method for intent mining based on semantic vectors and search results clustering. Our algorithm represent words as documents and performs a state-of-the-art approach for query log driven clustering. Similarities between query logs and words are calculated by using semantic vectors. Based on a manual selection of vertical representatives, our method is able to correctly identify intents and to classify them into vertical classes. Results in the *Query Understanding* of the *iMine2@NTCIR* subtask show that our unique run submission outperforms other participants in terms of $D-nDCG@10$ and achieves the second position in the general team raking.

Team Name

HULTECH

Subtasks

Query Understanding Subtask (English)

Keywords

query log clustering, intent mining, semantic vectors

1. INTRODUCTION

Identifying correctly the intent of users when the use a search engine is a challenging task. Nowadays, users not only expect high quality results but also adequately classification in vertical interfaces. Vertical interfaces allow the visualization of results in different contexts. Consider when a user search using a keyword as "star wars". In this case the user could be searching by the movie or for a recent news about the movie. The intent "star wars the force awakens" is clearly related to the movie released on December 2015 and the user is maybe interested in *buying* something associated with the movie. For example, buying a streaming service. Another intent such as "star wars rogue one", the new sequel movie, is more with *news*[1]. These users' expectations are in constant developing by the major commercial search engines.

This paper present our participation in the *search intent mining* subtask of iMine2 at NTCIR. It is our third participation in the NTCIR tasks related to intent identification. In previous years [9][2], we have proposed two different solutions: a modified version of the classical k-means algorithm to identify intents through the clustering of search results [4] and a semantic vector-based strategy which make use of arithmetic operation to identify the user intents [5]. This year the organizer are included the vertical classification task, in order to reinforce the intent mining subtask. In short, the subtask consist in given a query provide a two levels set of intents that better match the user request with a vertical class for the lower level. First level is limited to four and second to 10 intents. However, less intents could be provided. Fully description of the task and examples could be find in the description paper of the task [10].

In order to continuing with our research in clustering for intent mining, this year we have explored the integration of semantic vectors into Dual C-means [8], a generalization of an algorithm [7] that has shown good performance in the previous editions of this task [4]. Semantic vectors is a recent developed technique that uses neural networks to efficiently represent words as vectors keeping interesting syntactic and semantic information. This technique has showed to achieve state-of-the-art results in many natural language processing tasks. In our previous participation we have used collocation measures that were calculated over a set of Web results. For this year, we have replaced which have deal with a drastically simplification of our algorithm by the integration of semantic vectors.

The remainder of this paper includes a description of our intent identification algorithm in Section 2. Vertical identification strategy is presented in Section 3. The experimental results and discussions are presented in Sections 4 and 5. Finally, conclusions are presented in Section 6.

2. INTENT IDENTIFICATION

Our method is based on a semantic representation of words within the Dual C-means (DCM) algorithm [8]. DCM is a iterative search results clustering (SRC) method that allow to cluster documents and simultaneously select the appropriate label from a set of candidates. In each step, DCM first calculate the similarity between labels –query logs– and documents to assign documents to cluster and later a label is selected from a list of candidates for each cluster. The algorithm iterates until convergence is achieved. Documents are not clustered by DCM, but words instead. Indeed, words are used as documents and query logs as centroid candidates. Similarities between the words and query logs are obtained by calculating the cosine similarity of the vector of each word and the a label candidate. These semantic vectors

[1]At the moment of writing this paper the movie was not released yet.

are a dense vector which encode the semantic information obtained from a corpus. Several strategies allow to obtain semantic vector. In this work, a recurrent neural network based algorithm is used for the calculation of this vectors[2]. Our used matrix based representation and our adapted version of DCM are described in the following sections.

2.1 Semantic vectors

Many recent works in natural language processing and information retrieval have been interested in this kind of techniques. A basic technique to calculate semantic vectors was based on singular value decomposition, but a new neural networks based strategy has attracted a lot of attention [3]. These vectors allow to encapsulate semantic and syntactic information in a 300[3] dimension vectors. The code to calculate these vectors is publicly accessible and it has been also implemented in many different libraries[4] and frameworks[5]. In this paper, we are not interested in the developing of adapted vectors, but in the use of them. For that reason, we have downloaded and used general proposes pre-calculated vectors calculated by [3].

2.2 Matrix based representation

Our modified version of DCM uses a matrix with the similarities between each label candidate and the words contained by all the candidates (full vocabulary). Let Q be the set of query logs candidates to the query q. M^q is nxm the matrix that contains the semantic similarity between each q_i intent in Q and the full vocabulary extracted from Q. Note that $n = |Q|$ and m correspond to all the words find in Q. The M^q_{ij} value correspond to the cosine similarity calculated between the semantic vector for the q_i intent and the w_j word.

2.3 Matrix based Dual C-means

As input DCM receives the M^q matrix, a desired number of clusters and a maximal number of iterations. Figure 1 shows the *python* code for DCM. First, an initialization step is performed to identify the top-k relevant label candidates (lines $2-3$). Then, the iterative process is started. Each word is assigned to the label with maximal similarity (line 5) and then each for cluster the maximal label is assigned (lines $6-11$). These previous steps are performed a predefined number of maximal iterations $iter_{max}$. As output, the DCM algorithm provides the labels that represent each cluster. To ensure that k cluster are provided some candidates are added in cases when a number of k labels are selected in the iterative process (lines $12-16$).

3. VERTICAL IDENTIFICATION

Similarly that in the intent identification section, semantic vectors are used to automatically assign the vertical class for each selected intent. Indeed, our method is based in a small set of words used a represent of the concept expressed by each vertical class. The size of each set variate between 2 and 4 manually selected words. Table 1 show the set of words for each vertical class. So, each intent string is mapped to the semantic space and its similarity to each set of words is calculated using the cosine similarity between the average intent vector and the average vector of the vertical class. The vertical class with highest value is assigned to the intent.

[2]Actually, the pre-calculated vectors distributed with *word2vec* where used.

[3]This value is a parameter.

[4]Gensim: https://radimrehurek.com/gensim/

[5]Tensorflow: http://www.tensorflow.org

```
1  def dcm(mq,k,iter_max):
2    init = numpy.sum(mq,1)
3    ind_i = init.argsort()[−k:][::−1]
4    for i in range(iter_max):
5      pi_d = mq[ind_i,:].argmax(axis=0)
6      ind_i = []
7      for id_pi_d in numpy.unique(pi_d):
8        partit = numpy.where(pi_d == id_pi_d)[0]
9        sum_partit = numpy.sum(mq[:,partit],1)
10       ind_i.append(sum_partit.argsort()[−1:][::−1][0])
11     ind_i = [ x for x in iter(set(ind_i)) ]
12     for e in init.argsort()[−k:][::−1]:
13       if len(ind_i)>=k:
14         break
15       elif not e in ind_i:
16         ind_i.append(e)
17   return ind_i
```

Figure 1: Python code of our matrix based Dual C-means algorithm.

Vertical class	Words
Web	web, internet and online
Image	image and picture
News	news and event
QA	qa, question, how and what
Encyclopedia	encyclopedia, wikipedia and facts
Shopping	shopping and mall

Table 1: Verticals and words used to represent the respective concept.

4. EXPERIMENTS AND RESULTS

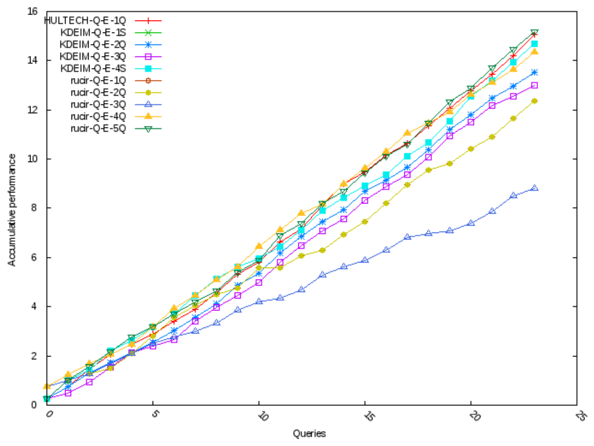
4.1 Experimental setup

A total of 100 strings were used as queries for the evaluation. In the final results, only 98 queries were considered[6]. For each of these queries, the system must provide a list of a maximum of four first level intents with a maximum of ten second level intents associated to each one in the first level. Each second level discovered intent was manually classified into relevant or not, as well as their relevance with its respective first level intent. For evaluation, we used independent metrics such as $I-rec$@10, $D-nDCG$@10 and $V-score$, and their combination into $D\#-nDCG$@10 and $QU-score$. Finally, the $QU-score$ value is used to compared the evaluated systems. A full description of the evaluation metrics and the used dataset can be found in [10]. A total of three teams participated in the task with 10 runs. Our team have submitted only one run.

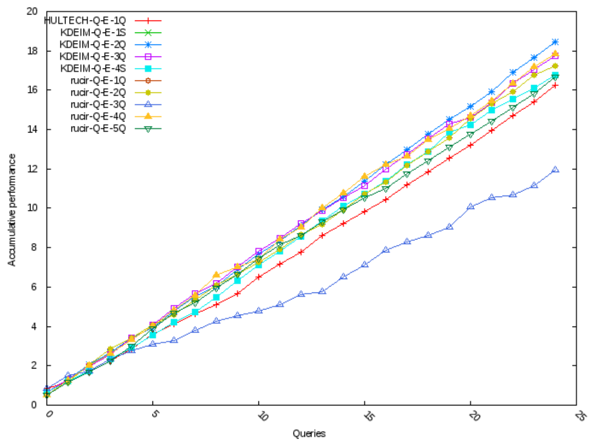
4.2 Results

In order to access the results, we present an analysis on the independent metrics. For each metric, we classified the results into four equally distributed groups sorted by the respective results of all participants. This allows us to split

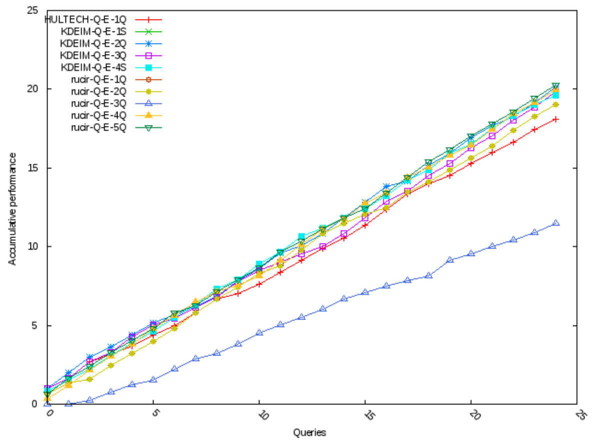
[6]For major details check [10].



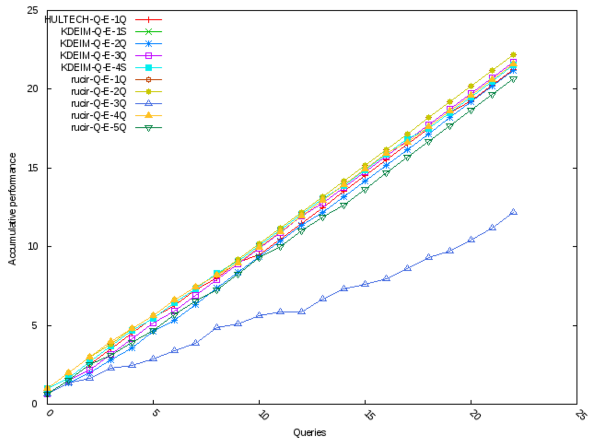
(a) Hardest



(b) Middle hard



(c) Middle easy



(d) Easiest

Figure 2: $I-rec@10$ results for the submitted runs. From left to right and top to bottom, the hardest to the easiest queries.

the results into four different levels of difficulty (hardest, middle-hard, middle-easy and easiest). Results are showed in Figures 2, 3 and 4 were each point represent the accumulative performance starting from the easiest query to the hardest in average for each group. This accumulative result allows to draw some conclusions the evolution of each algorithm into each group of queries. The order within the group and to discriminate each query into a group is determined by the average between all the runs.

In the case of $I-rec@10$, our algorithm performs well for the *hardest* group and all the algorithms perform very similar for the *easiest* group excluding the *rucir-Q-E-3Q* which obtains significantly lower results in the three groups. Indeed, the performance of this run does not improves when the task become easier.

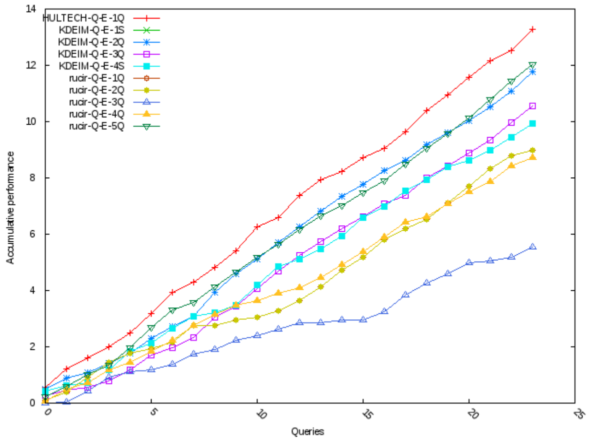
Regarding the $D-nDCG@10$ metric, the results are quite different. For the hardest group, we can distinguish five groups of curves with similar results with our run in the top, which is clearly superior to the runs of all the other participants and again the *rucir-Q-E-3Q* run in the bottom. It is clear that runs from the same participant tend to behaves similarly. For the *middle hard* many of the runs manage to get the top performance, but *KDEIM-Q-E-4S*, *rucir-Q-E-2Q* and *rucir-Q-E-4Q* does not manage it yet. For the graphs *middle easy* and *easiest* all the runs get similar results, excluding *rucir-Q-E-3Q*.

Finally, for results for the $V-score$ metric are clearly different to the previous ones. Note that in this case, the runs from *rucir* outperforms the other participants. Only in the middle easy and easiest our run manage to slightly approach the *rucir* lower performance. *KDEIM* submissions are clearly the less performer for this metric. In the easiest case they are far from the other runs.

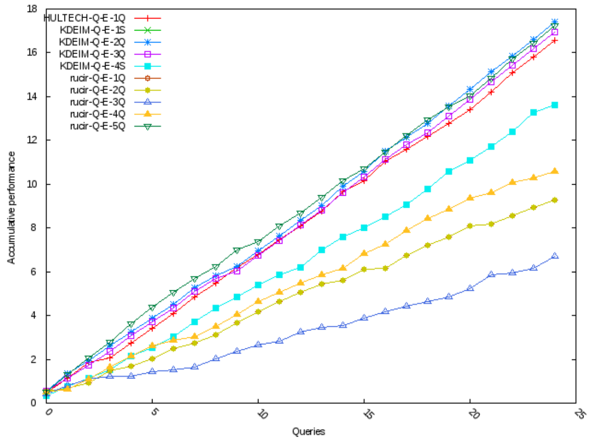
The average results by team are shown in Table 2. We were the only team with one submission which explains the no standard deviation in our results. It is interesting to remark that the *rucir* team manage to get the best and the worst performance for the final metric, the $QS-score$. This could be explained by a problem with their *rucir-Q-E-3Q* run. In general, our team performs well obtaining the second position by teams and the fifth by runs. Indeed, we have undesired performances for the $V-score$ which can be explained by the few number of words used to represent each vertical class (see Table 1). Note that this is consistent with the results of Figure 4. *Hardest* vertical classes are not adequately classified but the *easiest* are. An intuitive solution is the expansion of the sets, however selecting larger set could deal with a semantic shift.

5. DISCUSSION

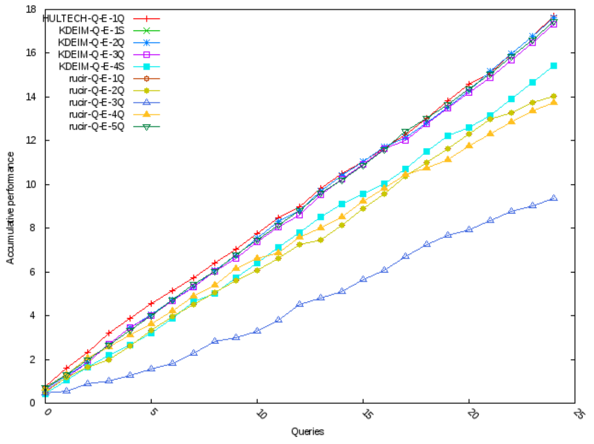
The results obtained by our algorithm are quite promis-



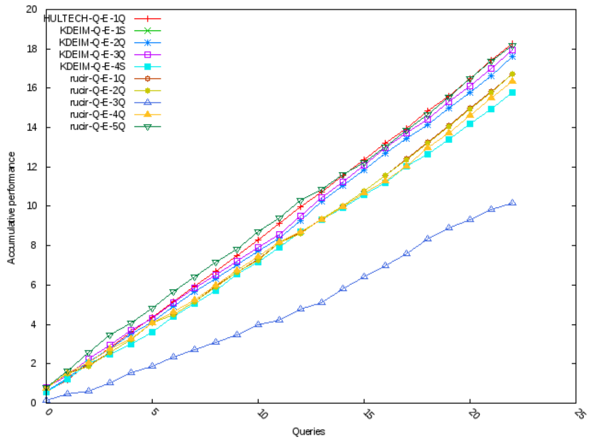
(a) Hardest



(b) Middle hard



(c) Middle easy



(d) Easiest

Figure 3: $D-nDCG@10$ results for the submitted runs. From left to right and top to bottom, the hardest to the easiest queries.

Table 2: Average and standard deviation results for each team. Best average performance marked in bold.

	$I-rec@10$	$D-nDCG@10$	$D\#-nDCG@10$	$V-score$	$QU-score$
HULTECH	0.728±0.000	**0.679**±0.000	**0.703**±0.000	0.366±0.000	0.534±0.000
KDEIM	**0.751**±0.005	0.635±0.048	0.693±0.025	0.297±0.006	0.500±0.008
rucir	0.686±0.128	0.504±0.121	0.595±0.119	**0.532**±0.090	**0.564**±0.098

ing. Note that the code of the entire core of our algorithm is presented in Figure 1. It takes only 17 lines and it is able to achieve a good performance in overall. Indeed, the code for selecting the vertical class is not shown but it takes just few lines and including more words to improve the performance will not increase the number of lines of code. It is possible also because the power of the used semantic vectors. These vectors were not adapted to this specific task and are general vectors calculated over a huge collection of web documents. This vector are publicly available which allows the replication of our experiments in further research. The full python code will be also available as extra content of this paper.

Another situation that could help to improve the results is in consideration of extra candidates in as intents. It will not deal with a significant increase of the code (or none). However, to achieve top-1 performance some parameters must be adjusted and extra pre-processing or heuristics must be considered.

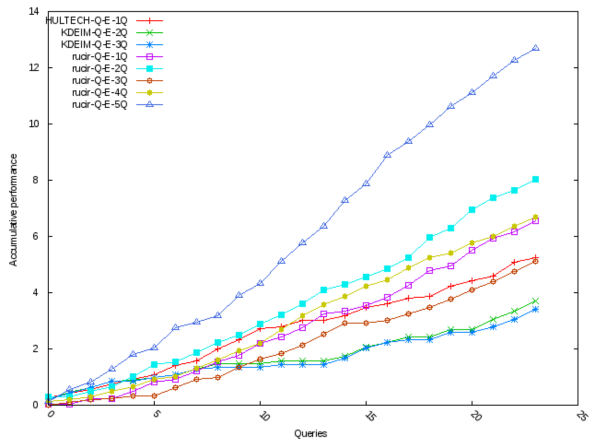
6. CONCLUSION

This paper present the *HULTECH* participation in the *iMine2 Query Understanding* subtask. Our submission achieves the best performance in terms of $D-nDCG@10$ and the second position when evaluated with the overall metric, $QU-score$ by teams. Our algorithm is based in a search results clustering algorithm combined with a new semantic representation for words.

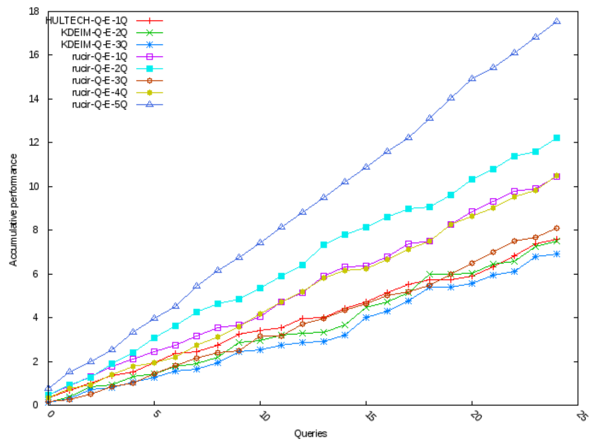
The main drawback of our system is the inadequately performance for the vertical class classification problem. Future work will focused on the improvement of this situation. Additionally, we plan to include a word sense disambiguation system based on the web graph [6] and their combination with text content [1].

7. REFERENCES

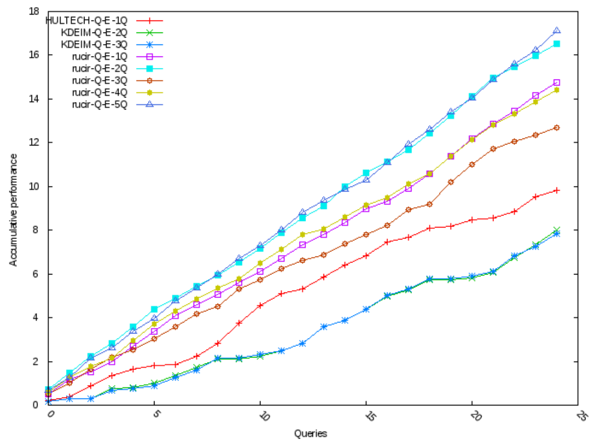
[1] S. Acharya, A. Ekbal, S. Saha, P. Santhanam, J. G. Moreno, and G. Dias. Multi-objective word sense



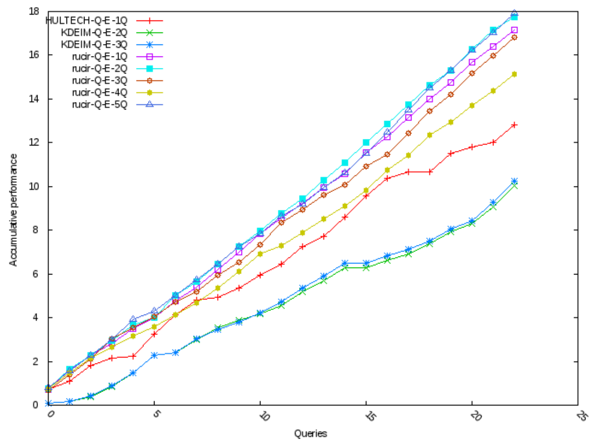
(a) Hardest



(b) Middle hard



(c) Middle easy



(d) Easiest

Figure 4: $V - score$ results for the submitted runs. From left to right and top to bottom, the hardest to the easiest queries.

induction based on content and interlink connections. In *21st International Conference on Applications of Natural Language to Information Systems*, 2016.

[2] Y. Liu, R. Song, M. Zhang, Z. Dou, T. Yamamoto, M. P. Kato, H. Ohshima, and K. Zhou. Overview of the ntcir-11 imine task. In *IMine Subtask of the NII Testbeds and Community for Information Access Research Workshop (NTCIR-11 2014)*, 2014.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[4] J. G. Moreno and G. Dias. Hultech at the ntcir-10 intent-2 task: Discovering user intents through search results clustering. In *IMine Subtask of the NII Testbeds and Community for Information Access Research Workshop (NTCIR-10 2013)*, 2013.

[5] J. G. Moreno and G. Dias. Hultech at the ntcir-11 imine task: Mining intents with continuous vector space models. In *IMine Subtask of the NII Testbeds and Community for Information Access Research Workshop (NTCIR-11 2014)*, 2014.

[6] J. G. Moreno and G. Dias. Pagerank-based word sense induction within web search results clustering. In *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '14, pages 465–466, Piscataway, NJ, USA, 2014. IEEE Press.

[7] J. G. Moreno, G. Dias, and G. Cleuziou. Post-retrieval clustering using third-order similarity measures. In *51st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2013.

[8] J. G. Moreno, G. Dias, and G. Cleuziou. Query log driven web search results clustering. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 777–786. ACM, 2014.

[9] T. Sakai, Z. Dou, T. Yamamoto, Y. Liu, M. Zhang, and R. Song. Overview of the ntcir-10 intent-2 task. In *Intent-2 Subtask of the NII Testbeds and Community for Information Access Research Workshop (NTCIR-10 2013)*, 2013.

[10] T. Yamamoto, Y. Liu, M. Zhang, Z. Dou, K. Zhou, I. Markov, M. P. Kato, H. Ohshima, and S. Fujita. Overview of the ntcir-12 imine-2 task. In *IMine Subtask of the NII Testbeds and Community for Information Access Research Workshop (NTCIR-12 2016)*, 2016.