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,
Université Caen Normandie,
F-14032 Caen
gael.dias@unicaen.fr

ABSTRACT

This paper presents a method for intent mining based on semantic vectors and search results clustering. Our algorithm represents words as documents and performs a state-of-the-art approach for query log driven clustering. Similarities between query logs and words are calculated with semantic vectors. Based on 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 driven clustering, intent mining, semantic vectors

1. INTRODUCTION

Correctly identifying the intent of users when they use a search engine is a challenging task. Nowadays, users not only expect high quality results but also adequate classifications in vertical interfaces. Vertical interfaces allow the visualization of results in different contexts. Let's consider a user issuing the query "star wars". The user could be searching for different topics about star wars. One intent could be "star wars the force awakens", which is clearly related to the movie released on December 2015 and the user might be interested in *buying* some product associated with the movie e.g. a streaming service. Another intent might be "star wars rogue one", the new sequel movie. In this case, the user could be interested in *news*[1] about this topic. Understanding these users' expectations is a great issue for major commercial search engines.

This paper presents our participation in the *iMine2 Search Intent Mining* subtask at NTCIR. It is our third participation in the NTCIR tasks related to intent identification. In previous years [9, 2], we proposed two different solutions: (1) a modified version of the classical k-means algorithm to identify intents through the clustering of search results [4] and (2) a semantic vector-based strategy, which makes use of arithmetic operations to identify the user intents [5]. This year, the organizers included the vertical classification task in order to reinforce the intent mining subtask. In short, the subtask consists in providing a two-level set of intents that better match the users' search with a vertical class for the lower level. The topical level is limited to 10 second-level intents and 4 first-level intents, and the vertical level includes 6 specifications. However, less intents could be provided. Full description of the task can be found in [10].

In order to continue with our research on clustering for intent mining, we explore the integration of semantic vectors into Dual C-means [8], a generalization of an algorithm proposed in [7] that has shown good performance in the previous editions of this task [4]. Semantic vectors are built based on recurrent neural networks to efficiently represent words as continuous vectors keeping interesting syntactic and semantic information. This technique has proved to improve over most of state-of-the-art results in many natural language processing tasks. In our previous participation, we used collocation measures that were calculated over a set of Web results. This year, we propose a drastic 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. The 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 semantic representation of words within the Dual C-means (DCM) algorithm [8]. DCM is an iterative search results clustering (SRC) method that allows to cluster documents and simultaneously select the appropriate label from a set of candidates. At each step, DCM first calculates the similarity between labels (here, query logs) and documents to assign documents to clusters. Then, a label is selected from a list of candidates for each cluster. The algorithm iterates until convergence is achieved.

Here, instead of clustering web search results, we propose to cluster words contained in query logs. 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 respective semantic vectors. These semantic vectors are dense vectors, which encode the semantic information obtained from a corpus.

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

2.1 Semantic Vectors

Many recent works in Natural Language Processing (NLP) and Information Retrieval (IR) have been focusing on semantic vectors also known as word embeddings. A basic technique to calculate semantic vectors was based on singular value decomposition, but a new neural network-based strategy has attracted a lot of attention [3]. These vectors allow to encapsulate semantic and syntactic information in a 300[2] dimension vectors. The code to calculate these vectors is publicly accessible and it has also been implemented in many different libraries[3] and frameworks[4]. In this paper, we are not interested in developing adapted vectors, but in the use of them. For that reason, we used the pre-calculated vectors of [3].

2.2 Matrix-based Representation

Our modified version of DCM uses a matrix containing the similarities between each label candidate (any query log) and the words contained by all the candidates (full vocabulary, i.e. all words contained in query logs). 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 corresponds to all the words found in Q. The M^q_{ij} value corresponds to the cosine similarity calculated between the semantic vector of the q_i intent and the word w_j. Note that the semantic vector of a query log, which may contain different words is calculated as the average of the semantic vectors of its components.

2.3 Matrix-based Dual C-means

As input, DCM receives the M^q matrix, a k number of clusters and a maximum 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 maximum similarity (line 5) (i.e. assignment step). Then, for each cluster, the maximum label is assigned (lines $6-11$) (i.e. update step). These steps are performed a predefined number of maximum 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 to the topical intent identification, semantic vectors are used to automatically assign the vertical class to each selected intent. Our method is based on a small set of handcrafted words to represent the concept expressed by each vertical class. The size of each set variates between 2 and 4 words. Table 1 shows the set of words for each vertical class. So, each label cluster is mapped to the semantic space and the average vector is calcualted. The same is done for each vertical intent and the cosine similarity is used to compute similarity between both semantic vectors. The vertical class with highest cosine value is assigned to the intent.

[2]This value is a parameter.
[3]Gensim: https://radimrehurek.com/gensim/
[4]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, online
Image	image, picture
News	news, event
QA	qa, question, how, what
Encyclopedia	encyclopedia, wikipedia, facts
Shopping	shopping, mall

Table 1: Words to represent vertical intents.

4. EXPERIMENTS AND RESULTS

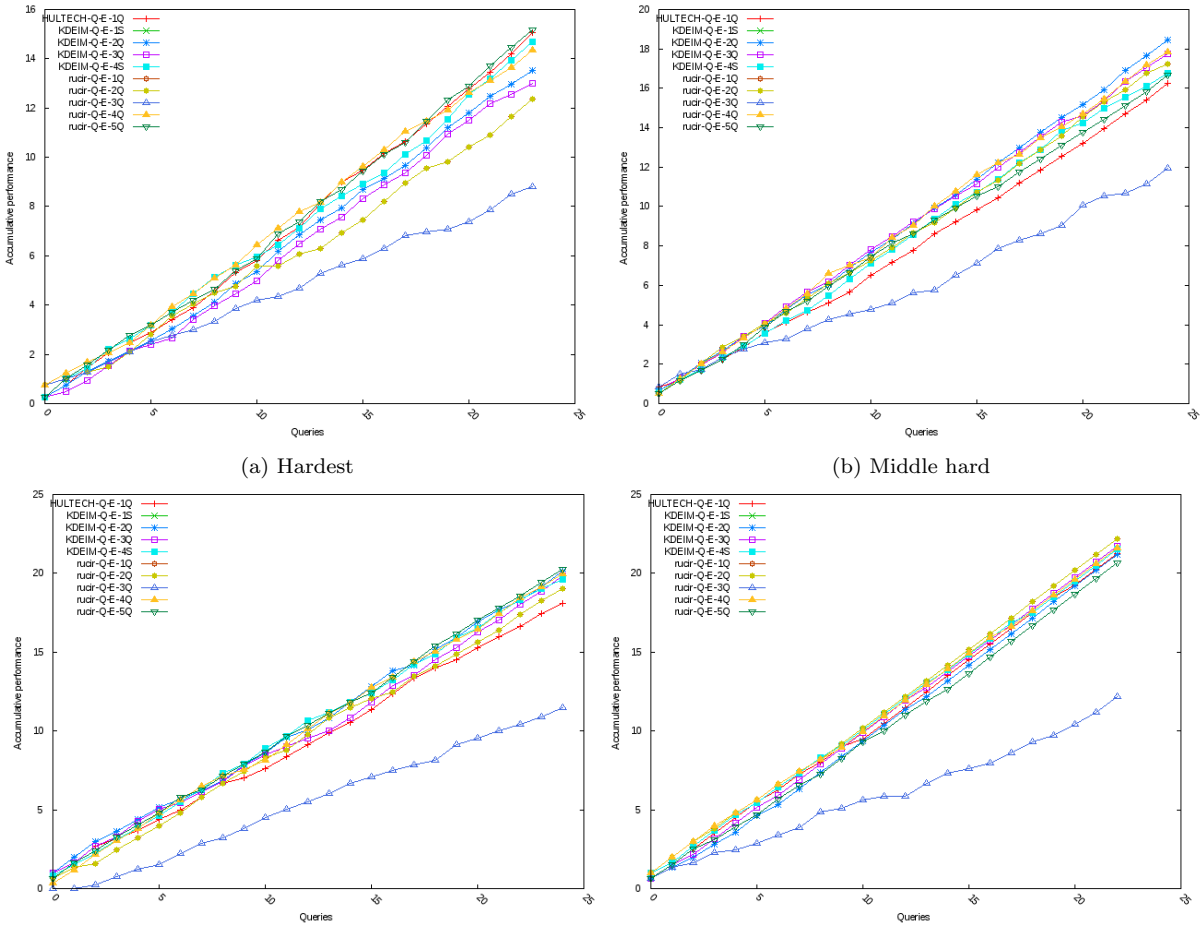
4.1 Experimental setup

A total of 100 query strings were used for evaluation. In the final results, only 98 queries were considered[5]. 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 each one associated to exactly one intent of the first level. All discovered second-level intents were manually classified into relevant or not, as well as their relevance with their respective first-level intent. For evaluation, independent metrics such as $I-rec$@10, $D-nDCG$@10 and $V-score$, and their combination into $D\#-nDCG$@10 and $QU-score$ were used. Finally, the $QU-score$ value is used to compare 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 submitted only one run.

4.2 Results

In order to access the results, we present an analysis of 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 the results into four different levels of difficulty (hardest, middle-hard, middle-easy and easiest). Results are showed in Figures 2, 3 and 4 where each point represents the accumulative performance starting from the easiest query to the hardest on average for each group. This accumulative result allows to draw some conclusions about the evolution

[5]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.

of each algorithm into each group of queries. Note that the order within the group to discriminate between each query 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 similarly 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 improve when the task becomes easier.

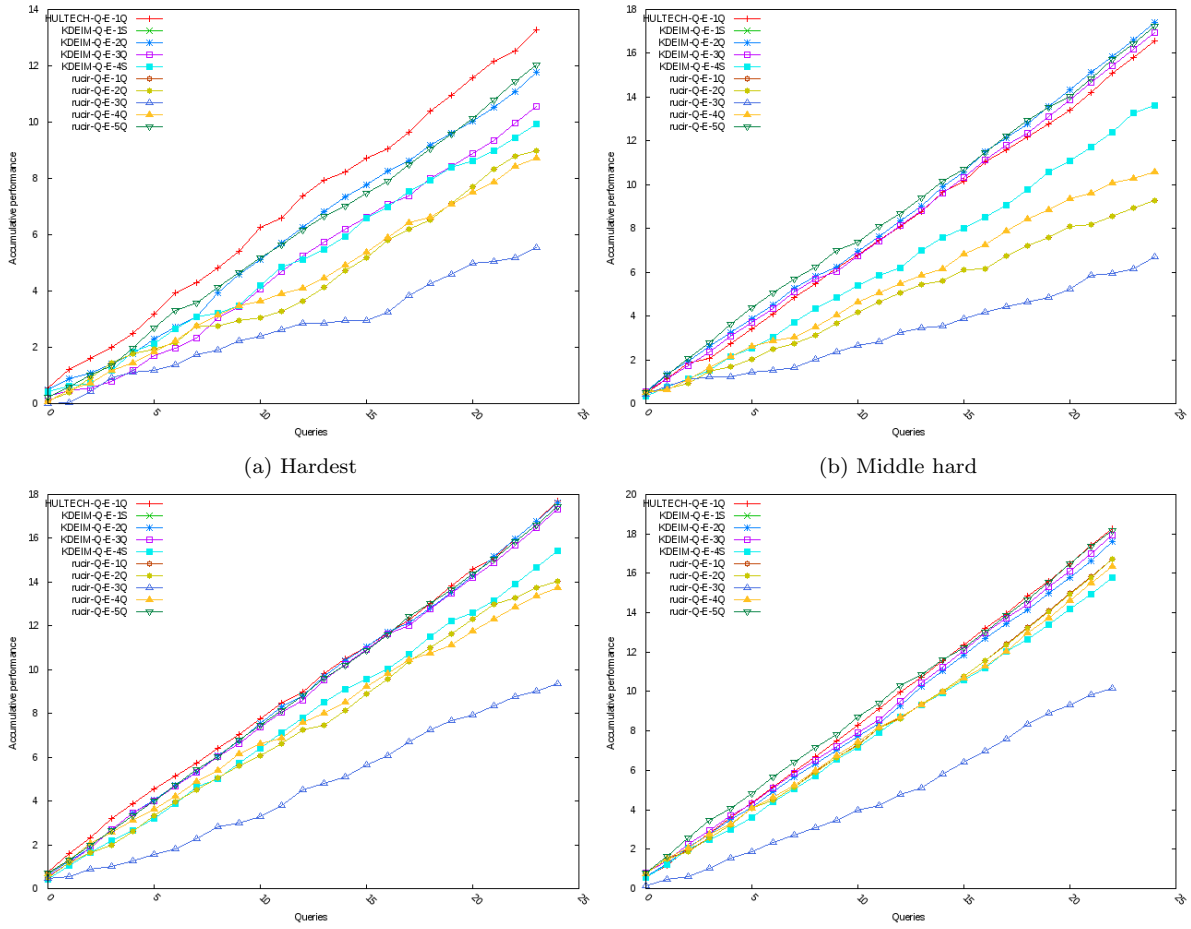
Regarding the $D-nDCG$@10 metric, the results are quite different. For the hardest group, we can distinguish five groups of curves with similar results with our run in the top, which is clearly superior to the runs of all the other participants and again the *rucir-Q-E-3Q* run in the bottom. It is clear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

Finally, results for the $V-score$ metric are clearly different to the previous ones. Note that in this case, the runs from *rucir* outperform all other participants. Only in the *middle easy* and *easiest*, our run manages to slightly approach the *rucir* lowest performance. *KDEIM* submissions are clearly the worst performing system for this metric. In the easiest case, results 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 "0 standard deviation" of our results. It is interesting to remark that the *rucir* team manage to gets the best and the worst performance for the final metric, the $QU-score$. 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) as well as the simplistic strategy defined for this task. 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

Results obtained by our algorithm are promising. 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 overall. 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



(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

not increase the number of lines of code. All this is possible due to power of the used semantic vectors as well as the simplicity of the DCM algorithm. These vectors were not adapted to this specific task and are general vectors calculated over a huge collection of web documents. These vectors are publicly available, which allows the replication of our experiments for 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 considering extra candidates 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.

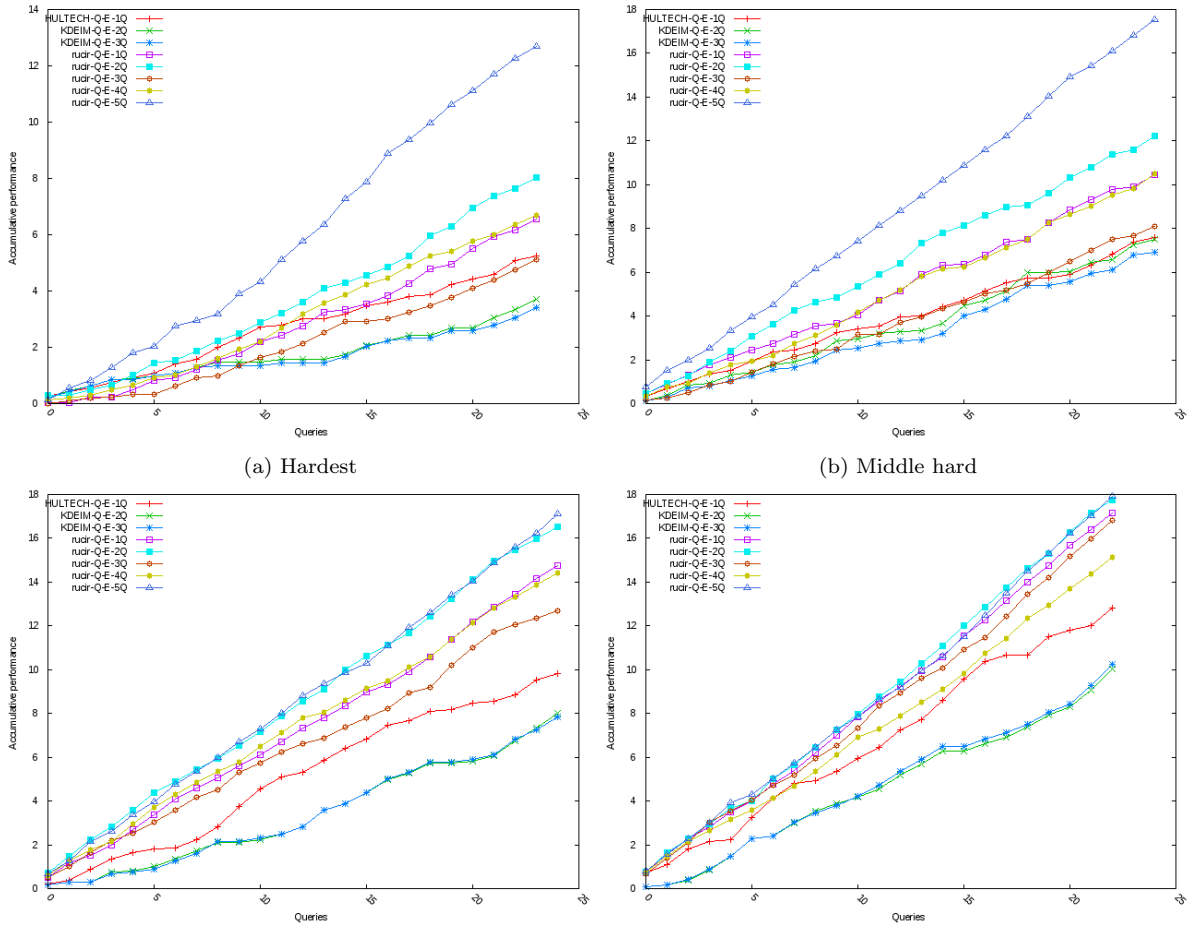
6. CONCLUSION

This paper presents the *HULTECH* participation in the *Query Understanding* subtask of the *NTCIR-12 IMine-2 Search Intent Mining* task. Our submission achieves the best performance in terms of $D-nDCG@10$ and the second position when evaluated with the overall metric $QU-score$ against all teams. Our algorithm is based on a SRC algorithm combined with semantic representations of words also known as word embeddings.

The main drawback of our system is the inadequate performance for the vertical class classification problem. Future work will focus on the improvement of this situation. Additionally, we plan to include a word sense induction system based on Web graph analysis [6] and its new developements when combined 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 induction based on content and interlink connections. In *21st International Conference on Applications of*



(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.

Natural Language to Information Systems (NLDB 2016), 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 *Proceedings of the 27th Conference on Neural Information Processing Systems (NIPS 2013)*, 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 2014)*, pages 465–466, 2014.

[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)*, pages 153–158, 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 2014)*, pages 777–786, 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.