Udel @ NTCIR-11 MobileClick Track

Ashraf Bah and Ben Carterette
Computer and Information Sciences
University of Delaware
Newark, DE 19716
{ashraf, carteret} @udel.edu

ABSTRACT

This paper describes our participation in the MobileClick track of NTCIR-11. We present our methods and results for both iUnit retrieval and summarization. Our ranking methods for the retrieval task are essentially two-step methods in which we first create a model pseudo-nugget, and then promote iUnits that are the most similar to that model pseudo-nugget. Our summarization methods consist in simply concatenating the iUnits (parts of sentences) that we already ranked in our retrieval sub-task.

1. INTRODUCTION

Given a query, participants are asked to return a structured textual output. The expected output is a two-layered text, where the first layer contains the most important information and the outline of relevant information, while the second layer, which consists of several parts of text, contains detailed information that can be accessed by clicking on an associated part of the text at the first layer.

2. RETRIEVAL RUNS

Participants in this task must provide a ranking of information units (iUnits). iUnits are pieces of texts extracted from documents. Evaluation is done considering those iUnits, rather than full documents.

The first four rankings use the iUnits provided by the organizers of the track.

All ranking use either cosine similarity or Jaccard similarity.

Cosine Similarity

Cosine similarity measures the cosine of the angle between two document vectors. Documents are represented as vectors where every term in the vocabulary is an independent dimension in the vector space. The inner-product between those 2 vectors is computed and used as their similarity value [4].

$$CosSim = \frac{A \cdot B}{\|A\|\|B\|}$$

where A and B are document vectors, and ||A|| and ||B|| are the magnitudes of A and B respectively.

Jaccard Similarity

Jaccard similarity coefficient is obtained by dividing the size of intersection of terms between two documents by the size of the union of terms between those two documents [1].

$$JacSim = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are the set of terms in two different documents and |A| is the size of the set A.

2.1 RET-udel-E-MAND-1

In this experiment, we use a two-step method. First, we generate and select dynamic "model" pseudo-nuggets. Secondly, using a baseline ranking, we re-rank iUnits by promoting the ones that are most similar to the model pseudo-nugget. Specifically, we compute the cosine similarity between each iUnit and the dynamic 'model' pseudo-nugget. At each iteration, the new model pseudo-nugget is constructed by adding a new iUnit to the concatenation of iUnits that have been ranked so far. The new iUnit that is added is the iUnit least similar to the current model pseudo-nugget. The idea behind selecting the iUnit that is the least similar to the model pseudo-nugget is to provide a diversified ranking that could be the basis for a well-diversified summary. Examples of "dynamic" model pseudo-nuggets are shown in Table1.

2.2 RET-udel-E-MAND-2

This run is very similar to the previous run. The only difference is in how we build the pseudo-documents. For each query, instead of using a dynamic model-nugget, we construct a static nugget that is simply the concatenation of all iUnits. Again, the ranking proceeds by promoting the iUnit that is the most similar (using cosine similarity) to the model pseudo-nugget. An example of "static" pseudo-nuggets is shown in Table 2.

2.3 RET-udel-E-MAND-3

Here again, just like in the second run, we use a static model pseudo-nugget. The only difference between this run and the second run is that instead of cosine similarity, we use Jaccard similarity. The idea is to investigate which one of cosine similarity and Jaccard similarity is well-suited for this task.

2.4 RET-udel-E-MAND-4

In this run, we use a static model pseudo-nugget as well. The model pseudo-nugget in this case, however, is the concatenation of the *full-documents* from which the top-10 relevant iUnits were extracted. The measure used for promoting iUnits is the cosine similarity.

2.5 RET-udel-E-MAND-5

The only difference between this run and the second run is that we use our own iUnits, instead of the ones provided by the organizers. These iUnits are constructed by breaking the documents from which the original top-10 iUnits were extracted by the organizers. Each new iUnit has a maximum of 70 characters long. Like in the second run, we use a two-step method. For each query, we construct a static nugget that is simply the concatenation of all iUnits. Then we proceed with the ranking by promoting the iUnit that is the most similar (using cosine similarity) to the model pseudo-nugget.

3. Summarization Runs

For each one of the runs described here, the text that describes the links is obtained by extracting the most significant words from the second-layer summaries.

Significant key-phrases were extracted using a model that exploits linguistic and statistical meth-ods. The method uses statistical lexical analysis to determine the most significant single-word terms, and extracts those terms as well as their immediate context to form complex terms – using noun-chunks where applicable or n-grams. Then it proceeds by clustering similar complex terms – using Monge-Elkan distance as the string-similarity measure – and selecting a representative for each cluster to be a candidate key-phrase. For selecting a representative, a similarity maximization algorithm is used that prefers the key-phrase that resembles the remaining key-phrases most closely. Finally, all the candidates are analyzed in order to determine confidence scores for each in the context of the document in question. The confidence scores are obtained by combining the significance of cue tokens in the representing candidate, the scope, as determined by the distribution of the candidate cluster over the document, and number of words contained in the candidate. Xtrak4Me [3] is an open-source library that performs this.

3.1 SUM-udel-E-MAND-1

This summarization run is based on the ranking created in the RET-udel-E-MAND-1 run of the retrieval task. Instead of providing a ranking, this time we concatenate the re-ranked iUnits to form several second-layer summaries. Each second layer summary is made of about 280 characters. First layer summaries are obtained by simply using the first iUnit as first-layer summary. The text that describes the links is obtained by extracting the most significant words from the second-layer summaries.

3.2 SUM-udel-E-MAND-4

This second summarization run is based on the ranking created in the RET-udel-E-MAND-4 run of the retrieval task. Again, instead of providing a ranking, we concatenate the re-ranked iUnits to form several second-layer summaries. Each second layer summary is made of about 280 characters. First layer summaries are obtained by simply using the first iUnit as first-layer summary. The text that describes the links is obtained by extracting the most significant words from the second-layer summaries.

3.3 SUM-udel-E-MAND-5

This summarization run is based on the ranking created in the RET-udel-E-MAND-5 run of the retrieval task. Like in the previous summarization runs, instead of providing a ranking, we concatenate the re-ranked iUnits to form several second-layer summaries. Again, each second layer summary is made of about 280 characters, and first layer summaries are obtained by simply using the first iUnit as first-layer summary. The text that describes the links is obtained by extracting the most significant words from the second-layer summaries.

Table 1. Example pseudo-nuggets for RET-udel-E-MAND-1 (from iterations 1 through 4) for query 1.

Current Pseudo-nugget (starting with iteration1 and ending with iteration 4)	**Selected Pseudo-nugget (i.e. the one least similar to current pseudo-nugget)**	**New Pseudo-nugget**
-	Java documentation is extensive	Java documentation is extensive.
Java documentation is extensive.	Gensim, Pattern, and other Python modules are good at text processing	Java documentation is extensive. Gensim, Pattern, and other Python modules are good at text processing.
Java documentation is extensive. Gensim, Pattern, and other Python modules are good at text processing.	static typing	Java documentation is extensive. Gensim, Pattern, and other Python modules are good at text processing. static typing.
Java documentation is extensive. Gensim, Pattern, and other Python modules are good at text processing. static typing.	semantic similarity	Java documentation is extensive. Gensim, Pattern, and other Python modules are good at text processing. static typing. semantic similarity.

Table 2. Pseudo-nuggets for RET-udel-E-MAND-2, RET-udel-E-MAND-3 and RET-udel-E-MAND-5 for query 1

Static (unchanging) model pseudo-nugget used for query
Java documentation is extensive Python is more expressive Java is more verbose Java is easy Java is faster Python has clear, concise syntax Python is easier to learn Python is difficult for beginners Java has weird syntax Use the best tool for the job natural language processing Python can be written more quickly Python can be maintained more easily Natural Language Toolkit (NLTK) provides a lot of tools for natural language processing Python is a more natural language Python has extensive libraries Java Python Text processing Python reduces development time, increases productivity code Kivy concurrency array scripting language non-scripting object oriented language platform regular expressions immutable sequences freely available high-level language string portability character classes Python is a dynamically-typed language, allowing higher productivity static typing Programming languages are tools text similarity semantic similarity Gensim, Pattern, and other Python modules are good at text processing

4. Results

Table 3. nDCG@10 and nDCG@400 results for iUnit Retrieval Subtask

Runs	nDCG@10	nDCG@400
RET-udel-E-1	0.4485	0.4354
RET-udel-E-2	0.5720	0.6915
RET-udel-E-3	0.5365	0.6787
RET-udel-E-4	0.5570	0.6906
RET-udel-E-5	0.1538	0.2185

Table 3 shows the nDCG results for all our runs. As can be seen in Table 3 – as well as Table 4 – there is a very big difference between the first four runs and RET-udel-E-5. This is simply because the first four runs use the iUnits provided by the organizers (gold iUnits). Gold iUnits were generated manually by assessors, hence those iUnits turned out to be of a very good quality. The first four runs can thus be considered manual runs (as opposed to automatic runs that did not involve any human input), and are not directly comparable to RET-udel-E-5 and the other runs submitted to the track. It is important therefore to note that, although our manual runs greatly outperform all other runs submitted to the track, the latter are on par with our automatic run RET-udel-E-5.

On another note, given that runs RET-udel-E-5 and RET-udel-E-2 use the same ranking method, it is clear that optimizing the methods used for the extraction of iUnits in order to obtain good quality iUnits before ranking them, is a crucial task. In fact in this case, using gold iUnits significantly increases the nDCG@10 value of RET-udel-E-2 by 272%.

As far as the manual runs are concerned, results vary greatly among them – albeit not as much as the variation between each manual run and our automatic run – depending on what ranking method is used. The first observation stems from the fact that RET-udel-E-2 performs much better than RET-udel-E-1. This suggests that using a static pseudo-nugget – which is simply the concatenation of all iUnits – is much more helpful than using a dynamic pseudo-nugget. This could be due to the fact that, in RET-udel-E-1, the idea of choosing a new iUnit that is the least similar to the current pseudo-nugget (for the purpose of obtaining a new diversified pseudo-nugget) does not work as intended. In fact, in some cases, the iUnit the least similar to the current pseudo-nugget could be an irrelevant or spammy iUnit. Such iUnit could lead to obtaining a poorer quality new pseudo-nugget. Further investigation will be needed to determine with certitude what causes RET-udel-E-1 to perform much worse than RET-udel-E-2.

It is interesting to note that using cosine similarity in our experiments leads to better results than using Jaccard similarity, as can be witnessed through the fact that RET-udel-E-2 performs better than RET-udel-E-3. Indeed, the only difference between the two runs is that RET-udel-E-2 uses cosine similarity while RET-udel-E-3 uses Jaccard similarity.

Also of note is the fact that the difference between RET-udel-E-2 and RET-udel-E-4 is not big (0.6915 vs 0.6909 for nDCG@400; 0.5720 vs 0.5570 for nDCG@10; 0.5503 vs 0.5440 for Q@400; and 0.6153 vs 0.6195 for Q@10). The difference between the two is simple. For RET-udel-E-4, a model pseudo-nugget of a query is the concatenation of the *full-documents* from which the top-10 relevant iUnits were extracted; whereas for RET-udel-E-2, a model pseudo-nugget of a query is simply the concatenation of all gold iUnits for that query. The fact that the difference between the results of these two runs is small suggests that there is not much difference between using iUnits to create static model pseudo-nuggets and using full-documents to create static model pseudo-nuggets.

Table 4. Q@10 and Q@400 results for iUnit Retrieval Subtask

Runs	Q@10	Q@400
RET-udel-E-1	0.5684	0.3028
RET-udel-E-2	0.6153	0.5503
RET-udel-E-3	0.6253	0.5493
RET-udel-E-4	0.6195	0.5450
RET-udel-E-5	0.0893	0.0324

nDCG measures (shown in Table 3) and Q-measures (shown in Table 4) have very similar trends, even though the former is a rank-based graded-relevance metric and the latter is a recall-based graded relevance metric. An analysis of the Q-measures leads to the same conclusions we have made using nDCG measures. One difference is that, using Q@10, RET-udel-E-3 is our best run. Though, RET-udel-E-2 remains the best run, if we use Q@400 instead. Another difference is that RET-udel-E-2's nDCG value increases as we increase the cutoff value from 10 to 400, and its Q value decreases as we increase the cutoff value from 10 to 400.

REFERENCES

[1] Grefenstette, E., Pulman, S.: Analysing Document Similarity Measures. Dissertation. (2010)

[2] Kato, M. P., Ekstrand-Abueg, M., Pavlu, V., Sakai, T., Yamamoto, T., & Iwata, M.: Overview of the NTCIR-11 MobileClick Task. (2014)

[3] Schutz, A. T.: Keyphrase extraction from single documents in the open domain exploiting linguistic and statistical methods. Dissertation, National University of Ireland. (2008).

[4] Singhal, A.: Modern information retrieval: A brief overview. IEEE Data Eng. Bull., 24(4) (2001) 35-43.