Valtuustopilvi: Building Wordclouds for Search

Teemu Kanstrén
KaTe SC
Oulu, Finland

Abstract—**This paper describes a search engine built for city meeting minutes. A minimum viable product was built to participate in an open data innovation challenge, using a word-cloud visualization as a tool to help the user guide the search. After success in the challenge, the searchengine was refined with new algorithms to better surface topics and information for the visualization and search. This paper describes the search engine, the designed algorithms, their applications, and experiences in their use.**

Index Terms—**search, nlp**

I. Introduction

This paper describes features of the ValtuustoPilvi (CouncilCloud in Finnish) search-engine for exploratory search over meeting minutes. The name stems from original version supporting search over city council meeting minutes using word-clouds, while later being extended to other types of documents. Valtuustopilvi aims to support exploratory search, to help the user identify concepts discussed in the minutes, and guide the search forward interactively. Instead of focusing the search on single documents, the documents are considered in groups, where all documents related to a specific meeting form a single group. These groups, along with the documents and words they contains are used as a basis to build word-clouds aiming to highlight different topics and concepts discussed in the meetings, and to help the user drive towards interesting information. The user can interact with the word-clouds by clicking on words to further refine their search. The word-cloud is automatically updated with the new results to match the refined search.

The service was initially developed in a time period of about two weeks, to participate in an open data innovation challenge presented by the city of Oulu in Finland. The initial version can be seen as a minimum viable product, which was quickly built for the competition. It won first place in the competition, and after the city expressed interest in taking it into wider use, it was further developed to refine the algorithms, the architecture and features. These improvements were based on information learned about the domain and the received user feedback. The refined algorithms for building the word-clouds, the final architecture, and experiences with the service are described in the following sections.

The rest of the paper is structured as following. Section II goes through background concepts and related works. Section III describes algorithms used to build the word clouds. Section IV describes the overall search-engine architecture. Sections V and VI discuss overall experiences, and conclude the paper respectively.

II. Background and Related Work

The ”traditional” search engines such as Google are good at performing keyword-driven searches, where the user types in a keyword query and receives a list of documents ranked as relevant to that query. While the ValtuustoPilvi search engine allows the user to do this (using the underlying Elasticsearch engine), it also provides the interactive word-cloud visualization as a basis for exploring the search space and building the queries. This type of search is often called ”exploratory search”, where the information needs and search intents are evolving as the user learns more about the search results, aided by tools to explore the information [10].

Exploratory search has been a popular topic in the recent years, and various services for it have been developed. For example, queries based on user selecting from a hierarchical set of questions is presented in [11], where the questions are based on a pre-defined document tag-matrix. Similarly, [7] presents and approach where questions generated from the user given queries, and resulting data, are used to guide the search.

Similar to word-clouds, ontology-based term-clouds are used to help user navigate document-spaces in [5]. They focus on providing summaries of news articles and mapping them to the defined set of topics in an ontology. ValtuustoPilvi focuses on exploratory search over the documents and groups of documents (meetings), using topics models and word frequency metrics inferred from the data to guide the search as opposed to predefined ontologies.

A radar-like view to explore identified topics in documents is presented in [10], allowing the user to navigate a map of topics and associated documents. Similarly, topic models are used in [6] to cluster documents and associated nodes, and to provide visual means to navigate the topics and associated nodes. An example is presented of mapping topics to universities, where research on those topics is performed. Exploratory search for twitter is presented in [9], where tweets are clustered based on identified topics, and the user can perform searches and get results highlighting which topics are discussed in resulting tweets.

As seen in these works, the concept of *topics* is very important in this field. Different approaches to identify topics can be performed, such as using existing topic mappings [10], [6], or clustering the documents using various measures such as term frequencies and n-grams in the texts [9]. Finally, specialized topic modelling algorithms to cluster documents into different topics have been presented. ValtuustoPilvi uses topic models as a tool for building the top-level word-clouds for its interactive search-visualization. The topic modelling algorithm used is Latent Dirichlet Allocation (LDA) as presented in [12].

LDA takes as input a set of documents (a corpus), and builds a model representing these documents as a mixture of topics. Each topic is represented by a list of words and their occurrence count (or probability) in the topic. As documents are represented as mixtures of topics, the documents can also be viewed as probabilistic mixtures of topics in the overall topic model. Each word in each document is mapped by LDA to a topic, and a single document can have words mapped to any number of topics. This leads to mapping documents to different topics with different weights (e.g., count of how many words in the document map to each topic). ValtuustoPilvi uses such models and weights as one input for building the word clouds.

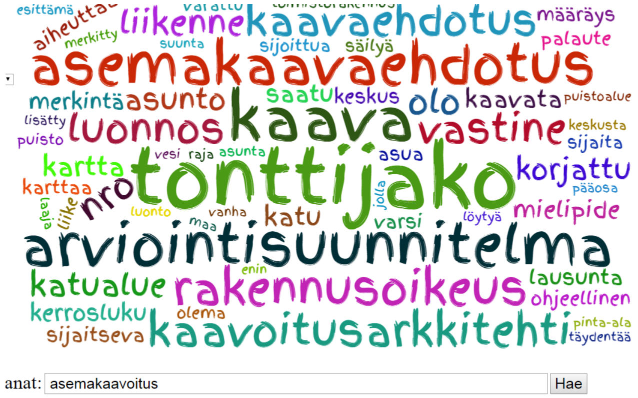
Another measure commonly used in document search is term frequencies (TF), inverse document frequencies (IDF), and their combinations (TF-IDF). TF describes the frequency of a term (word) in a document. IDF describes how common the term is across all documents. TF-IDF then describes how common a term is to a document vs how common is it across all the documents (the corpus). This allows identifying terms that are highly descriptive to a specific document (common in that document) but not found so often in many other documents (not so common in the overall corpus). In document search, this can be very helpful to, for example, rank documents most relevant to a search query. Document where the search terms have a high TF-IDF score can be considered more likely to be specifically relevant to the search terms, as they can be ranked as discussing those terms specifically vs other documents in the corpus. TF-IDF is also more generally used for various purpose in natural language processing (NLP), such as part of the clustering algorithm in [9].

ValtuustoPilvi uses term-frequencies as one building block for the deeper level word-clouds, but with a differing goal: finding words to describe a document set, which the user can use to focus their search further. The aim is not to find specific documents for given terms, but rather take a set of documents and summarize a set of potentially interesting search-terms for them. Terms are considered interesting if they are discussed in many meetings matching the search query, and are only present in a subset of the overall corpus (of all documents of type(s)).

III. Building Word-Clouds

In this section, the Valtuustopilvi word-cloud building algorithms are presented. This is based on two different approaches. The first world-cloud building approach is designed for top-level searches, where the goal is to model all the documents for a specific meeting type, or for several types. The set of topics in the full document set for all meetings is expected to be diverse, with many topics and large document sets. This set is always predictable as it covers the full set of documents for a single meeting type or a set of meeting types. The second word-cloud building approach is for deeper-level search results, where a search query limits the document set. This set could be any subset of the available documents, and is thus not predictable what documents need to be included in the word-cloud. In this case, the word-cloud needs to be tuned in real-time to match the dynamic document set.

Figure 1 shows an example screenshot of Valtuustopilvi in action, with the meeting type of council minutes chosen as search-target. The search-box at the bottom shows that the user has previously chosen the word ”asemakaavoitus” (”zoning” in Finnish) from the top-level word-cloud (based on topic-models), resulting in the shown word-cloud as given by the deeper-level word-cloud algorithm. This word-cloud presents words such as ”tonttijako” (plot division), ”kaava” (zone), ”kartta” (map), ”arviointisuunnitelma” (evaluation plan). The space below this cloud shows top-ranked documents matching the word ”asemakaavoitus” (in the search box) in the council minutes. This list is omitted here for space reasons. By choosing words from the cloud, or typing manually in the search-box, the user can further refine the query over the search results, and the word-cloud is updated in real-time at each step to match the new document set for query results.



anat: asemakaavoitus

Fig. 1: Example word-cloud

There are other search controls available also, including defining date ranges, and selecting meeting types as search targets. However, these are not different from typical search-engines and so are not described here.

To build a useful word-cloud for search, the goal was to find words that represent the query results, and summarize them in a meaningful way, highlighting different concepts and topics in the documents matching the query. This aims to highlight interesting search options in the word-cloud for the user, to drive deeper into the search results. This needs to be done in near real-time, to be able to adapt to any query result as the query and the result set changes, while maintaining a reasonable user-experience. To address these requirements, the queries to are divided to two types. First is top-level word-sets covering all documents of all types, or all documents of specific types. These sets can be seen as quite predictable but common starting points. Second is focused word-sets for specific query results, where the documents are those matching the query. These sets can be seen as varying and changing for every query.

A. Top-Level Words

Algorithms such as topic modelling can summarize document sets by identifying clusters of words that commonly represent some topic within the given document set. However, running these algorithms can take hours to days on a document set, depending on the number of documents, document sizes, number of topics, and other such parameters. To make use of topic models but side-step the long run-times, ValtuustoPilvi uses pre-trained topic models as a basis for the top-level word-clouds. These topic-models are re-generated at specified intervals by a process separate from the main search-engine, and uploaded to the search-engine after updates.

The first page users see when opening the search page shows the word cloud summarizing all documents for all the meetings. As the starting page, this is also the most common configuration to load. Other common pages to load are the top-level search pages for the different meeting types. These are the pages summarizing all documents for a given meeting type. Users searching for information about specific city organization will likely start many searches from this configuration.

As these search configurations are commonly used, target the largest document sets, and likely contain the most diverse topics, pre-trained topic models are used to provide the word-clouds for them. The goal is to provide the best topics, and the fastest response for the biggest and most common queries, while leaving more resources to answer more specific queries. A single topic model is built to cover the full document set over all meeting types, and specific topic models are built to cover all documents for each type. For a top-level query covering all documents for all meeting types, the topic model built just for that purpose is used. For a top-level query over all documents of a specific meeting type, the topic model for that meeting type is used. For a top-level query over multiple meeting types, the topic models for those meeting types are combined into one (taking all the top-level topics for each meeting type, and merging them to one set).

These topic models are built using LDA based algorithms, which produces a list of words associated with each topic, along with their count. The goal is to focus on finding terms describing broader concepts within the overall document set, where the document set consists of all documents for 1-N meeting types. The top words from all the topics for the document set are taken, merged, and a weighted random selection from those is presented to the user to describe the top-level word-cloud as a starting point for the search. This way, the word-cloud is expected to cover a broad range of high-level topics, across a broad range of documents (and meetings). By using highly ranked words across all the topics, the aim is to provide a starting point where each of the words should lead to explore a sizable subset of the overall document space, but more focused on a specific topic area.

Listing 1 describes in more detail the process of building the word cloud from these topic models in Python-like pseudocode.

Listing 1: Top-level word selection

```
weights = {}
MT = maximum how many meetings a word appears in
DT = the type of documents for query
DTS = the set of all documents of type DT
DTC = the size of DTS (count of documents in DTS)
DTSWC = count of unique words in DTS
TS = set of all topics generated

for each topic T in TS:
  selected = 0
  for each word W in top 10 words for T:
    MC = count of how many meetings W appears in
    if MC > MT: skip W
    if random(0,2) > 1: skip W
    WC = number times W is assigned to T
    weights[W] = weights[W] + WC
    selected += 1
    if selected > 2: move to next topic

TWC = total weight of top 10 words for all topics T in TS
TSAVG = average size of all topics
for each word W in weights:
  WDC = count of documents in DTS containing W
  weight = weights[W]
  weights[W] = 10 * WDC/DTC + weight/TSAVG

words = []
cloud_size = 100
for x = 0 to cloud_size:
  word = wrandom(weights)
  words.append(word)
```

Initially, words that appear in more than X% of meetings for the given type are filtered out. This helps remove domain-specific stop-words, document templates, and similar terms. For example, the meeting documents for all organizations in the city of Oulu will likely contain the word ”Oulu”. Similarly, meeting minutes for the city board are likely to contain the term ”city board”. Showing these to someone searching the city board documents does not provide much value as they are quite obvious. Filtering helps focus the words shown to more interesting ones. The default value of X% here is 70%, which was defined through empirical experimentation. This value can be controlled in the search user interface by selecting a filtering level from *no filtering* (show all words), *remove most common* (default, 70%), *show rarer* (50%), *show rare* (25%), and *show very rare* (10%).

The value *MT* in Listing 1 is calculated from this filtering percentage. The value *WC* is simply the number of times a word is assigned to a topic by the LDA algorithm. Since the same word can be assigned to different topics in different locations in a document, and in different documents, the total weight is calculated as a sum of the ”weight” of that word in all the topics combined. For this purpose, only the top 10 words of each topic are taken, as they are considered to represent the main concepts of the topic, and wish to limit the processing overhead to reasonable amount.

For example, consider these two topics with their top five words shown here:

1) Topic14= yhtio[352] toiminta[267] liikelaitos[234] oulu[233] henkilosto[220]
2) Topic28= liikelaitos[49] johtokunta[46] energia[39] serviisi[33] vastine[32]

First, the words with meeting count over the filtering threshold are removed, removing ”oulu” from this set. Words with only a single appearance (in a single topic) are assigned their word count (in that topic) as their weight. Some examples of such single appearance words here are ”yhtio”=352, ”toiminta”=267, and ”johtokunta”=46. These numbers also represent the weight that would be assigned to these words in this case. Words in multiple topics are merged, so ”liikelaitos”=234+49=283. Average size of all topics (here just these 2) = (352+267+233+220+49+46+39+33+32)/2 = 636. Assume there are 1000 documents of this type, and the word ”liikelaitos” appears in 50 of them, the final weight for ”liikelaitos” becomes 50/1000*10+283/636 = 0.95.

To avoid simply building a word-cloud with words from a few largest topics, the number of words to pick from a single topic is limited to 2 in this example (in the listing).

Once the list of potential words is collected (variable *weights* in Listing 1), a weighted random selection is applied to this list (function *wrandom()* in Listing 1). This gives a more interesting result for the user, as the suggestions have variance, and yet they are weighted more towards the words that are ranked as potentially more interesting. Finally, before presenting the words to the user in the word-cloud, the weights of the words are normalized to be between 0.1-1.0, and these scaled values are used to scale the size of the words in the word-cloud visualization.

B. Deeper-Level Words

From the main top-level search page, the users progress to deeper pages by entering search terms manually or using the word-cloud. At this point, the words in the word-cloud are selected based on the specific documents matching the given search query. Listing 2 shows the algorithm for building the word-cloud from these documents. This is similar to the algorithm shown in Listing 1 for topic-model based word selection. The main difference is in how the weight of a word is calculated. Here, the second part of the formula is based on the number of words in the query results document set, as opposed to the count in topics.

Listing 2: Deeper-level word selection

```
weights = {}
MT = maximum how many meetings a word appears in
DS = query result document set
DT = the type of documents for query
DTS = the set of all documents of type DT
DTC = the size of DTS (number of documents in DT)
DSWC = count of unique words in DS
DTSWC = count of unique words in DTS

for each word W in DS:
  MC = count of how many meetings W appears in
  if MC > MT: skip W
  WC = count of W in DS
  WDC = count of documents in DTS containing W
  weights[W] = weights[W] + WC

for each word W in weights:
  WDC = count of documents in DTS containing W
  weight = weights[W]
  weights[W] = 10 ∗ WDC/DTC + weight/DSWC

words = []
cloud_size = 100
for x = 0 to cloud_size:
    word = wrandom(weights)
    words.append(word)
```

Again, once the weights are calculated, a similar weighted random choice of the words is taken. At any time, the user can also manually type new words into the search, or modify the existing search built using the word-cloud.

The words provided have to be valid search terms, and the goal of the deeper-level word-cloud building algorithm is to find searchable words from the document set matching the resulting document set for the current query. For a word to be searchable, it has to be present in at least some of the resulting documents. The more broadly it is present, the broader a concept it is considered to be in matching the given query. Thus words gets ranked higher by the document count (but filtered by maximum meeting count to avoid overly common ones such as city names). The filtering threshold against all possible documents (not just the search results) removes the template words and similar stop-words. Algorithms such as TF-IDF can be seen as inverse to this ranking as it would rank terms higher if they are found in fewer documents. However, a mixture of results could be of interest, where different terms with high TF-IDF counts within the document set could weighted higher, in addition to the current algorithm.

IV. Service Architecture

The architecture of the search-service as a whole is illustrated in Figure 2. The functionality is split into several small services. The front-end interface to the user is provided by the *web-server*, responsible for providing the user the web-based interface to build queries and receive the results. This connects to the *search engine* component, which implements the algorithms described in Section III.

The actual documents are stored in *ElasticSearch*. A customized version of the Finnish language plugin for ElasticSearch provides lemmatized versions of the documents and queries, to match different morpohological forms of the Finnish words together. The customization enables to identify domain- and location-specific words that the general purpose Finnish lemmatizer (Voikko) does not recognize, and adding domain-specific lemmatization rules.

The *document downloader* component crawls the city document storage websites once per night to identify and download new documents. Identified new documents are downloaded, lemmatized, and indexed into ElasticSearch. The *topic modeller* is responsible for building the high-level topic-models

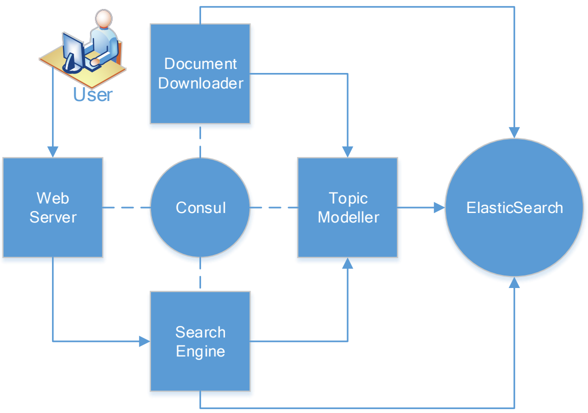


Fig. 2: Service Architecture

for the search-engine. The downloader component notifies the topic modeller component about new documents, allowing triggering of topic model rebuild. The search engine can query the topic modeller for the latest versions of the topic models. The high-level topics and concepts of the overall meetings do not change very often, allowing also the topic modeller to be run less often to conserve resources. The search engine updates its topic models to the latest version once per day.

Consul is a service discovery component the different service use to find other services they depend on. Each service registers to consul with a given identifier, and queries consul for known id's of other components. Communication between the different components is through Google Protocol Buffers Remote Procedure Call interface.

The system has been deployed on different cloud configurations, both on distributed and single (large) node configurations. All the components, excluding the topic modeller, have been run successfully on a single cloud node with 2 CPU cores and 4GB of memory, for document set sizes of about 40000 documents and 1GB of raw extracted text. This includes the web server, elasticsearch, consul, downloader, and the search engine all in the same, single, virtual machine of this size. Since this is in practice a very low resourced cloud server, this should indicate potential scalability of the approach to bigger document sets.

For the components in Figure 2, ElasticSearch and Consul are freely available open-source components, while the rest were developed for ValtuustoPilvi. The topic modeller component was developed and tested to work with rest of the components fully automatically. However, since it requires more resources, and does not need to run constantly, it was eventually run in a development environment, and the updated topic models uploaded to the search-engine component from there. Different tools to build the LDA topic models were also experimented with, including a self-written one, and the Python Gensim package.

V. Discussion

The service was initially developed as a quick prototype to try the concept of word-cloud assisted search in the city of Oulu open data innovation challenge. This initial version (minimum viable product in modern terms) would allow searching one type of document set, the minutes of the city council. This was refined by adding lemmatization, customizing the lemmatization to the domain, and creating the customized word-highlighting algorithms for the word-clouds, as presented in this paper.

Other features were also prototyped, but due to resource limitations did not make it into the production version. This includes using N-grams (bi-grams, or pairs of words) in addition to single words for the word-clouds. This worked well to identify useful information such as names, as Finnish names are typically consisting of two parts (first name and last name). The experimentation revealed it would be useful to try extension to 2- and 3-grams to identify names (word-pairs), phone numbers (typically in this data represented as three sequences of digits), and domain specific compound terms. However, tracking even bi-grams (word-pairs) quickly becomes very resource intensive as there are so many possible pairs to track. Means to filter this set down would be needed to make n-gram inclusion viable. Processing documents in batches was considered, and tracking and filtering most and least common ones, but due to resource limitations, this was not studied further.

In general, with larger datasets, even without n-grams, the presented algorithms can end up taking large amount of memory to run when the vocabulary is very large. ValtuustoPilvi uses the lemmatizer to provide the baseforms of the words, and use those baseforms for the counts in the algorithms. If the lemmatizer did not recognize the word, custom spelling lists were checked, and if still not recognized the word was logged in a list of unrecognized words (for later manual checking) and used as such. This can lead to large vocabularies in documents with a large amount of special words, typos, and similar issues leading to unrecognized words. Limiting the vocabulary to only the words recognized by the lemmatizer was considered, and only tracking the unrecognized words during indexing. This would limit the size of the vocabulary, while providing a way to check problem cases and add them as custom spellings later. However, the set of words did not grow too big with the document sets used, and giving the user an ability to look for ”oddball” words by setting the common meeting filtering threshold very low (parameter MT in the listings) was found useful.

As seen in the listings, 10 was used as the multiplier for the document weight vs word weight in the word-weighting algorithms. This was chosen by empirical experiments with the data. For different types of document sets, and with different word vs document count distributions, different values could be useful. For example, for a large set of small documents (e.g., tweets), the relation between document vs word count would likely be quite different.

As interesting consideration would also be replacing the count of documents in the word-weighting formulas by the count of meetings. This would be interesting since topics are typically discussed in meetings and not just single documents. The topic can span multiple documents in a single meeting, or just few documents but over many meetings. Discussion in several meetings could be seen as putting more weight on the importance of the words in the results set. Currently (in the algorithms presented), the meeting count is simply applied as the first step filter.

A feedback form was provided as part of the service. This allowed users to post feedback and rate the usefulness of the service, or the service experience. Generally, the feedback was positive even though little actual textual feedback was received. Discussions with the city representatives were also used to collect feedback, and their experiences in using the service, and the feedback they had received when discussing the use of the service with their colleagues. This was generally seen as very positive and the use of the service making the overall search more efficient and interesting.

Regarding additional NLP techniques, part-of-sentence (POS) tagging would be a useful addition to be able to identify different uses of words. For example, ”nainen” is a Finnish baseform word for ”woman”, but also lemmatizes to ”naida”, which is a word describing a sexual act. Showing word-clouds highlighting names of sexual acts as results for searching for ”woman” is not necessarily a good experience. Nor is showing results for the search of a word describing a sexual act (”naida”, which could then be selected in the word-cloud) and highlighting all words describing ”woman” in the results (because they would match the same lemmatization). Such grave mis-taggings were the real issues regarding the interest in POS tagging (e.g., subject vs. verb). Unfortunately a high accuracy POS tagger for the Finnish language was not available. Ultimately, a self-made Finnish POS tagger was built and trained, which reached reasonable accuracy on general texts but still had issues. These issues include many dialect forms of words found in Finnish documents, as well as area local terms and names. Finnish has a large number of different forms of words (morphologies), which may further complicate this. Since the self-made POS tagger could not accurately enough solve these issues, it was discarded for this use case, as the benefit was not worth the added complexity. Rather, words and lemmatizations that showed up with high frequency but were obviously erronous were collected, including ones that would potentially cause issues for the users (e.g., ”nainen” vs ”naida”). These were then added to a list of custom words to ignore or to lemmatize only in a specific way.

VI. Conclusion

This paper described the Valtuustopilvi search engine and the word-cloud building algorithms to support the user in exploratory search over document sets. In this case the search targets were city meeting minutes for the different organizations in different cities. In different experiments, the search-engine was successfully applied over the documents of several biggest Finnish cities, both separately (per city) and as a joint search over the documents of all the integrated cities. These all shared the same content-management system, allowing easy integration. With relatively little tuning, the same could have also been applied with other structures as well.

From the business perspective, the service was run for the period of about 1.5 years for the city of Oulu. After this, the city ran out of EU funding for their Open Data initiative that was used to fund this, and in a typically bureaucratic fashion the service was appreciated but not required by regulation or legistlation, or motivated by EU funding requiring further proof of concept or reporting of successful business collaboration. With the remaining bits of EU funding, several cities also tried to build a unified open data access layer for their meeting minutes, and a simple search interface, which they would consider to be enough to satisfy the need to give citizens some type of access, even if not as advanced or supporting exploratory features as ValtuustoPilvi.

Maybe the most interesting (for the author), and generally useful, aspect that was learned was how much more is needed to make a successful commercial product, beyond having nice technical features and happy users. Overall, we believe the algorithms, the concept, the architecture and experiences are useful inputs for many other similar applications, which is perhaps the best technical contribution of this paper.

References

[1] F. F. Heath III, R. Hull, E. Khabiri, ”Alexandria: Extensible Framework for Rapid Exploration of Social Media”, IEEE International Conference on Big Data, 2015.

[2] Q. Manon, ”Inbenta Semantic Search Engine: a search engine inspired by the Meaning-Text Theory”, 6th International Conference on Web Intelligence, Mining, and Semantics, 2016.

[3] Y. Jing, D. Liu, D. Kislyuk, A. Zhai, J. Xu, J. Donahue, ”Visual Search at Pinterest”, 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015.

[4] M. Kanakaraj, S. Kamath S., ”NLP Based Intelligent News Search Engine using Information Extraction from e-Newspapers”, IEEE International Conference on Computation Intelligence and Computing Research, 2014.

[5] W. Wong, W. Liu, M. Bennamoun, ”An Ontology-Based Interface for Improving Information Exploration”, 1st International Workshop on Intelligent Visual Interfaces for Text Analysis (IVITA), 2010.

[6] B. Gretarsson, et al., ”TopicNets: Visual Analysis of Large Text Corpora with Topic Modeling”, ACM Transactions on Intelligent Systems and Technology, no. 2, vol. 3, 2012.

[7] A. Kotov, C. Zhai, ”Towards Natural Question-Guided Search”, 19th International Conference on Word Wide Web (WWW), 2010.

[8] C. Fautsch, J. Savoy, ”Adapting the TF IDF Vector-Space Model to Domain Specific Information Retrieval”, ACM Symposium on Applied Computing, 2010.

[9] R. Gopi, O. Hoeber, ”TwIST: A Mobile Approach for Searching and Exploring within Twitter”, ACM Conference on Human Information Interaction and Retrieval (CHIIR), 2016.

[10] T. Ruotsalo, G. Jacucci, P. Myllamki, S. Kaski, ”Interactive Intent Modeling: Information Discovery Beyond Search”, Communications of the ACM, 2014.

[11] Y. Yang, J. Tang, ”Beyond Query: Interactive User Intention Understanding”, IEEE Internation Conference on Data Mining, 2015.

[12] D.M. Blei, A.Y. Ng, M.I. Jordan, ”Latent Dirichlet Allocation”, Journal of Machine Learning Research, vol. 3, 2003.