Control and Cybernetics

vol. **36** (2007) No. 4

Extending k-means with the *description comes first* approach

by

Jerzy Stefanowski and Dawid Weiss

Institute of Computing Science, Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland
e-mail: {jerzy.stefanowski, dawid.weiss}@cs.put.poznan.pl

Abstract: This paper describes a technique for clustering large collections of short and medium length text documents such as press articles, news stories and the like. The technique called *description comes first* (DCF) consists of identification of related document clusters, selection of salient phrases relevant to these clusters and reallocation of documents matching the selected phrases to form final document groups. The advantages of this technique include more comprehensive cluster labels and clearer (more transparent) relationship between cluster labels and their content. We demonstrate the DCF by taking a standard k-means algorithm as a baseline and weaving DCF elements into it; the outcome is the *descriptive k-means* (DKM) algorithm. The paper goes through technical background explaining how to implement DKM efficiently and ends with the description of an experiment measuring clustering quality on a benchmark document collection *20-newsgroups*.

Short fragments of this paper appeared at the poster session of the RIAO 2007 conference, Pittsburgh, PA, USA (electronic proceedings only).

Keywords: document clustering, cluster labels, k-means algorithm, information retrieval

1. Introduction

Organizing unstructured collections of text documents into semantically related groups, from now on referred to as *text clustering* or *clustering*, provides unique ways of digesting large amounts of information. According to typical definitions of clustering, documents within clusters (groups) should be similar to each other, while being dissimilar to documents from other clusters. To evaluate the similarity of text documents, first it is necessary to transform them into a mathematical model where each document is described by certain features. The most popular representation of text data is the *vector space model* (VSM), Salton (1989). In the VSM, documents are represented as vectors of features in

a multidimensional space. Each dimension corresponds to a given *term* (word) and the dimension's value in a document vector expresses the weight of this term in the document. For a given set of documents one can build a *term-document matrix* and define various similarity measures between document vectors (rows of this matrix). A very common definition of document similarity is the angle between document vectors in term vector space (or a cosine of this angle which is easier to compute). Once similarity between documents is defined in this way, one can apply a number of strategies for grouping similar documents into clusters.

Historically, text clustering was applied in information retrieval to extend the set of documents matching a query beyond those containing the query's terms (by adding similar documents from the clusters which matched the query). In such applications clusters are not directly presented to the user — they are in the background of an information retrieval system. This situation has changed as new applications of text clustering emerged. For example, systems like Vivisimo (www.vivisimo.com) or Carrot^2 (www.carrot2.org) cluster results retrieved from Internet search engines and display cluster *labels* to present an overview of a larger set of search results. Note that accuracy and comprehensiveness of cluster labels determines the potential gain a user may receive from clustering functionality. Very similar conclusion could be given for systems clustering news stories (event detection). Their usefulness can be measured by the quality of the label assigned to each group of stories and the relationship between this label and the actual stories inside a cluster.

These new emerging applications require new solutions. In particular, if clustering is moved directly to the user interface level, it is necessary to assure that clusters contain human-readable descriptions — something to represent the information that makes documents inside a cluster similar to each other and that would convey this information back to the human user. Most text clustering algorithms have not been designed for such requirements. A typical solution for creating labels is to extract and present a list of the most salient terms inside a cluster, so called *keywords* (see Fig. 1). But stripped from the surrounding syntactical information, keywords leave a lot of room for guessing the true context they appear in, which is often confusing and requires considerable effort on behalf of the user. In general, returning from a mathematical model of cluster representation to comprehensive, explanatory labels will be difficult because text representation models, such as the VSM, do not preserve the inflection and syntax of the original text.

Some research has been done to replace word-based models with more complex features like entire phrases (we present related work in Section 6), however the semantics of cluster descriptions is still unsatisfactory. We think that creating comprehensive and accurate cluster labels is an interesting research problem and that it is distinctively different from regular document clustering. For this reason we formulated a problem of *descriptive clustering*, Weiss (2006):

> Descriptive clustering is a problem of discovering diverse groups of semantically related documents described with meaningful, comprehensible and compact text labels.

The three additional expected properties of cluster labels in descriptive clustering are (with respect to normal clustering):

- *comprehensibility* of labels, understood as grammatical correctness (word order, inflection, agreement between words if applicable);
- *conciseness* of labels — phrases selected for a cluster label should minimize its total length (without sacrificing its comprehensibility);
- *transparency* of the relationship between cluster label and cluster content, best explained by the ability to answer questions such as: "Why was this label selected for these documents?" and "Why is this document in a cluster labeled X?".

It is easy to see that the above goals are quite difficult to achieve. We try to fulfill them at least in part and propose a general algorithmic scheme called *description comes first* (DCF) (see Section 2). The DCF is a very general approach which can be used to construct concrete algorithms depending on the building blocks used in each step. In our previous work we demonstrated DCF in a new algorithm called Lingo (Osiński, Stefanowski and Weiss, 2004; Osiński and Weiss, 2005). Lingo was designed to cluster (and label) search results from Internet search engines and demonstrated the ability to create diverse, meaningful cluster labels. However, Lingo's weak point was its limited scalability, which we would like to address in this work.

The main goal of this paper is to show how the popular and efficient k-means algorithm can be modified to benefit from the DCF approach. We called the modified algorithm *descriptive k-means* (DKM) to highlight the emphasis placed on proper cluster descriptions.

A secondary goal of this paper is to experimentally study how our DCF modifications affect clustering quality compared to the baseline version of k-means. We suspected that shifting emphasis to labels may lead to decreased document-to-cluster assignment quality, but we needed evidence if this was really the case.

Finally, our intention was to combine theoretical aspects with pragmatic approach. We wanted to see if an efficient implementation of the proposed algorithm is possible and which data structures could be used to improve its actual runtime performance.

The rest of the paper is organized as follows. In the next two sections we describe the fundamental ideas behind the DCF and show how k-means algorithm can be modified to benefit from it. In Section 4 we delve into certain implementation details that make the algorithm implementation efficient for large document collections. Section 5 shows the results of an experiment comparing two versions of the descriptive k-means algorithm against the baseline k-means. The paper ends with discussion of certain related works and final conclusions.

2. An overview of *description comes first*

In a typical text clustering algorithm, cluster labeling procedure naturally follows cluster discovery (see Fig. 1). As we mentioned already, this leads to problems because text representation based on vector space model makes sensible cluster label extraction very difficult. One solution to this problem could be to use more intuitive features instead of isolated terms, for example entire phrases appearing in the text — this was the idea employed in the STC algorithm (Zamir and Etzioni, 1999). But phrase-based clustering introduces its own difficulties: meaningless, but frequent phrases are hard to identify (i.e., "home page") and unnecessarily proliferate in the set of features. Even the common understanding of a phrase as an ordered sequence of words is not natural in many languages, in which words have more positional freedom inside sentences.

In the description comes first approach we use a trick. To avoid the difficult phase of constructing labels for existing clusters, we change the troublesome conventional order of a typical clustering algorithm. First, we separate two threads — *candidate cluster label discovery* and *document clustering* (see Fig. 1):

- *Candidate label discovery* phase is responsible for providing comprehensive labels (usually phrases but also salient keywords) that can be used to label future clusters. These candidate labels can be acquired in many ways, but in this paper we show two methods that extract them directly from input documents (Section 3.2).
- *Cluster discovery* provides the model and information about similar document groups present in the input data (we call this a model of *dominant topics*). Note that we can and actually prefer to use a clustering algorithm based on words as individual features here — such algorithms tend to work better, especially for languages with less strict word order in a sentence. What is important is that these clusters never surface to the user interface level, merely providing information about document groups in the input.

To better explain how DCF works let us demonstrate the entire procedure using a simple example. Let us assume only two dimensions of feature space. At first we collect cluster candidate labels and documents and represent them in the same vector space model (Figs. 2(a) and 2(b)). Angle vectors are also shown for clarity. Document clustering phase returns groups of similar documents, with a centroid vector (average of all document vectors inside the cluster) for each cluster (Fig. 2(c)).

After candidate labels and document clusters are known, the *matching step* takes place. The assumption here is that the set of candidate labels contains a number of good-quality, but redundant entries (not corresponding to any document clusters). On the other hand, the model of clusters contains useful information about document groups, but lacks the expressive power of syntactically correct phrases. We combine these two information sources by filtering the set of candidate labels and leaving only those labels that are most simi-

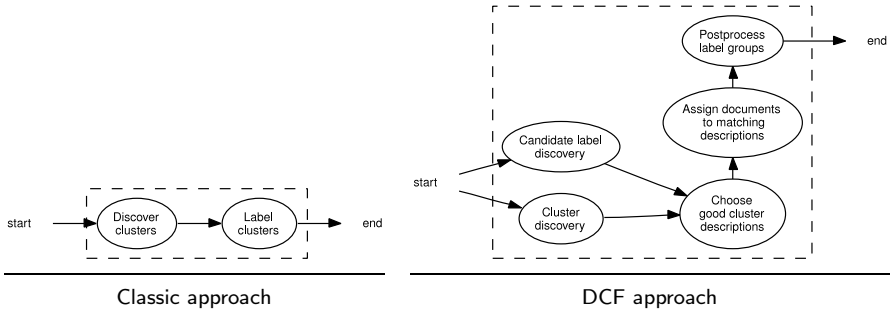
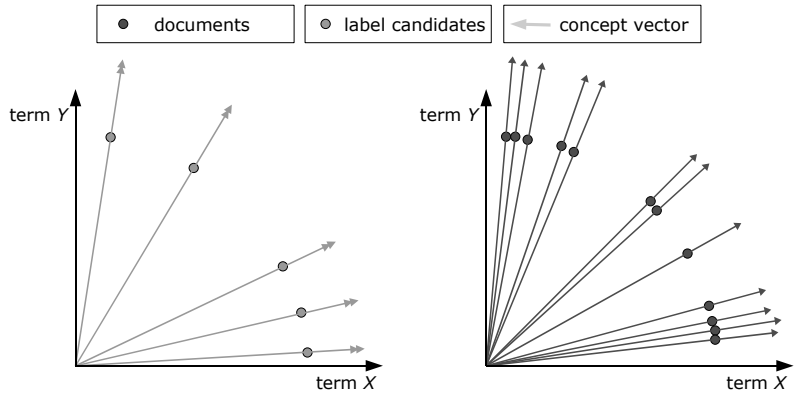


Figure 1. Steps of a clustering procedure in a typical clustering algorithm and in the DCF approach. Note the two steps of candidate label discovery and cluster discovery in the DCF.

lar to any centroid vector of the discovered document clusters (Figs. 2(d) and 2(e)). The result of the matching step is a set of comprehensive phrases that should constitute good representation of actual clusters discovered in the collection of documents. We call these phrases *pattern phrases* because they can be perceived as "seeds" of the final set of document groups to be created in the last step. The model of clusters (dominant topics), computed using traditional clustering algorithm, is no longer required at this point and can be discarded.

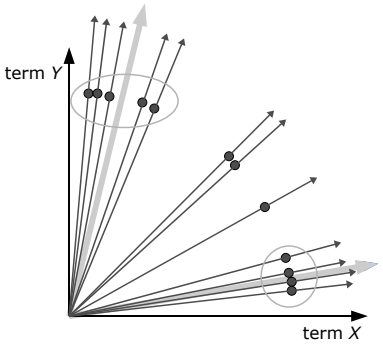
In the final step the documents are once again assigned to pattern phrases to assure every document has a clear relationship to the cluster — the label containment. For each pattern phrase we look for documents that contain the phrase in an exact, or slightly distorted shape by allowing matches with minor rewordings and foreign words injected inside (see Fig. 2(f)). The output is a set of clusters, each consisting of a pattern phrase (label) and the documents allocated to it.

Note a few interesting aspects of the procedure presented above. First, there is strong interdependence between clustering, documents and label discovery: cluster centroids formed around weak keywords are unlikely to find matching candidate labels and will be discarded early on. At the same time, even if a pattern phrase has been mistakenly selected, it must still collect a certain required number of documents to actually form a final cluster. We thus increase the likelihood that each cluster is truly sensible and tightly related to its label. Second, the procedure naturally caters for two natural elements expected from a text clustering algorithm — overlapping clusters and documents not belonging to any cluster. Clusters may overlap because documents are re-assigned to pattern phrases independently (a single document may match more than one pattern phrase and hence belong to more than one cluster). At the same time, a document may not find any matching pattern phrase and thus not belong to any cluster at all. The third aspect worth mentioning is that there is a great deal of freedom in putting together different basic blocks of the entire DCF procedure.

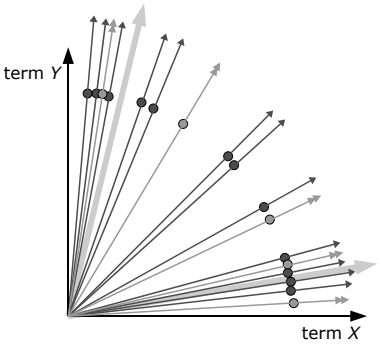


(a) Candidate labels and their "angle vector" in the model.

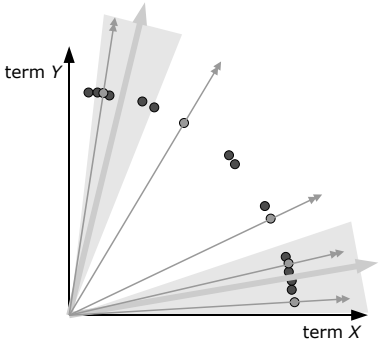
(b) Documents and their "angle vector" in the model.



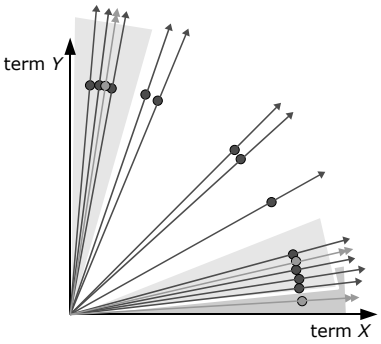
(c) Clusters (and centroid vectors) discovered in the documents.



(d) Merging: cluster label candidates and topics are represented in the same space.



(e) Candidate labels similar to cluster centroids (within cones) become pattern phrases.



(f) Documents matching pattern phrases are selected to form final three groups (note the overlap).

Figure 2. The DCF approach applied to documents and labels in a two-dimensional term vector space.

The descriptive k-means algorithm presented in the next section shows one such possibility.

3. Modifying k-means for the DCF: *descriptive k-means* algorithm

3.1. Rationale for k-means and the variant selected for the baseline

The *descriptive k-means* (DKm) algorithm is, depending on how one looks at it, either a modification of the widely known k-means algorithm, adjusted to the DCF scheme or a concrete instance of the DCF scheme, where k-means is used to discover dominant topics. We considered a few different "baseline" text clustering algorithms and finally selected k-means for the following reasons.

- k-means is widely recognized and used, so minimal effort is required to understand which DCF modifications we introduced.
- Baseline k-means has a few limitations when applied to text clustering — crisp document-to-cluster partitioning, spherical clusters, non-overlapping clusters and, perhaps most of all, the requirement to specify the number of clusters in advance. There are many variants of k-means that attempt to overcome the problems mentioned above, but our intention was to demonstrate that even this inconvenient algorithm can be adopted to the DCF approach and yield good results.
- At the time of choosing the baseline algorithm we thought it would be a good idea to somehow utilize our good experiences with the singular value decomposition (SVD) used in the Lingo algorithm. SVD is applied to the term-document matrix. Although this is difficult to objectively prove, the matrices resulting from SVD decomposition reveal the latent co-occurrence relationships between terms and correspond to different "topics" present in the documents. Interestingly, it has been shown that SVD decomposition can be approximated with cluster centroids discovered by k-means (Dhillon and Modha, 2001). Our hope was to create a very efficient algorithm based on k-means and at the same time employ the interesting properties of SVD.
- Finally, k-means (or one of its many variants) is often used for comparing clustering quality between algorithms. Any definition of "quality" in text clustering is usually controversial and prone to subjective bias because no single ideal result exists (Manning and Schütze, 1999). Our motivation was to evaluate DKM and k-means in *the same experiment conditions* to see if DCF improves (or at least retains) the quality of the output clusters measured using typical document allocation indicators, at the same time hoping that the DCF scheme should yield clearer cluster labels and more transparent relationship between documents and the label of a cluster. Since these aspects are quite difficult to formally define, quality

measurement using methods other than user satisfaction surveys is nearly impossible. In fact, we finally decided that comprehensibility of cluster labels will not be taken into account at this phase of our evaluation and we would focus just on document-to-cluster assignment quality, leaving some systematic judgment concerning cluster labels for later.

After we decided to pick k-means to be the baseline clustering algorithm, there was still a great deal of freedom in choosing the configuration of parameters and components of this algorithm. We eventually decided to use the following variant.

- The initial maximum number of clusters k must be given in advance (we do not try to estimate it, although it is possible, see Manning, Raghavan and Schütze, 2008). As we discuss it later on, the final number of clusters resulting from the DKM may be smaller than the one provided for underlying k-means (depending on how many matching label candidates we can find for the discovered k cluster centroids). The fact that k must be given in advance is of course a weak point and should be addressed in the future.
- The initial state of cluster centroid vectors is computed by taking a random subset of input documents and finding k most diverse elements in this subset. This method of bootstrapping k-means is reportedly stable and leads to good cluster diversity (Dhillon, Fan and Guan, 2001).
- The convergence criterion is an alternative of two conditions: either there are no more reassignments of documents to clusters or the global gradient criterion (sum of distances to cluster centroids) no longer significantly changes.

Once we had a concrete baseline algorithm, we could proceed to modifying it toward the DCF.

3.2. Overview of the descriptive k-means algorithm

Our adaptation of plain k-means to DCF focused on adding a candidate label discovery, using k-means for discovering cluster centroid vectors and combining these two elements to select pattern phrases. The algorithm pseudocode would be a bit lengthy[1] because of so many data structures and algorithms involved, so instead we present a state chart of major steps (see Fig. 3).

As new documents are added for processing, candidate cluster labels are extracted and unique entries are added to an inverted index (Baeza-Yates and Ribeiro-Neto, 1999). A similar inverted index is created for document content. These data structures are crucial for keeping the whole procedure efficient and are used in searching for pattern phrases and allocation of documents in subsequent stages of the algorithm.[2]

[1] DKM's pseudocode is available in Weiss (2006).

[2] We use the Lucene library extensively for building document and candidate label indexes

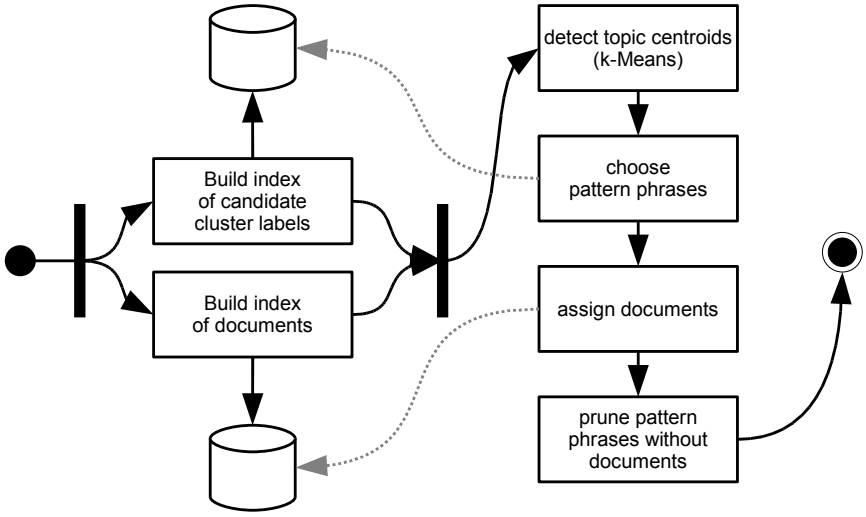


Figure 3. State chart of major steps of the descriptive k-means algorithm.

Two different methods are used to extract candidate cluster labels: frequent phrase extraction and noun chunking. The first method (described in Zamir and Etzioni, 1999) looks for frequently occurring ordered sequences of terms. Additional constraints are added to prevent proliferation of common collocations: a frequent phrase must not cross sentence boundary and it must not start or end with a very common word. Extraction of frequent phrases is efficient and can be implemented using generalized suffix trees or suffix arrays (Larsson, 1999). While appealingly fast, frequent phrases are based solely on word statistics and co-occurrence. An alternative method is applying *shallow text chunking* with the aim of detecting and extracting noun phrases. Since nouns are usually used to refer to objects and events, noun phrases should provide sensible and unambiguous cluster labels. We used a pre-trained statistical chunker for this task to avoid deep linguistic processing and keep the process reasonably efficient (Abney, 1991; Zhang, Damerau and Johnson, 2002).

Candidate label extraction is a time-consuming process compared to other phases of the algorithm, especially when shallow NLP methods need to be applied. The observation that helps here is that the extraction of candidate labels from a document requires computations that are local to it and independent of other input. Therefore, the effort required to extract phrases can be easily divided into independent units and distributed. We successfully deployed this idea and performed massive candidate label extraction using map-reduce computing paradigm (Dean and Ghemawat, 2004), where the input for a single map operation was a document and the result a set of phrases. Reducers simply

(http://lucene.apache.org), appendix A defines some terminology that we will use in further part of this paper.

emitted unique phrases with a minimal threshold on the number of occurrences in all documents.

Once all documents have been added, the algorithm proceeds to building a model of topics using k-means clustering (the variant described earlier). We only care about the final centroid vectors of clusters and not about each document assignment, so we can subsample the entire input collection of documents and approximate cluster centroids fairly closely (Goswami, Jin and Agrawal, 2004). With proper use of on-disk data structures created at indexing time, detection of cluster centroids is in the order of seconds for a few thousand documents on commodity hardware — we discuss these implementation details in Section 4.

In the merging step, we search for candidate cluster labels "closest" to the discovered cluster centroids. We already discussed it when describing DCF, but just to stress it again: an intuitive explanation of the rationale of this phase is that we want to "describe" a cluster (group of similar documents) using phrases we know are sensible for a human and at the same time match closely the cluster centroid. To handle this goal efficiently, for each cluster centroid we build a Boolean query (see Appendix A for definitions of query types) with terms taken from the cluster centroid term vector. The scoring boosts for each term correspond to weights in the centroid vector. We then execute such a query against the index of label candidates. To give the user the ability to express his or her preference of the final length of cluster labels, we multiply the matching score of each candidate label by an adjustment function taking into account the number of words in the label phrase — this helps promote phrases of the desired length. The exact shape of this penalty function is discussed in Section 4. The outcome is a score-ordered list of best matching labels. From this list we select a few top scoring entries and add them to the set of pattern phrases.

The algorithm ends with allocation of documents to each pattern phrase. We use the inverted index again (this time of documents, not candidate labels) and build phrase queries, which retrieve documents matching the pattern phrase. Exact matches receive a higher score than those where the phrase terms appear in different order or are mixed with non-query terms. This way documents where the pattern phrase is slightly distorted may still end up in one cluster, but documents just happening to contain the phrase terms at random positions should not be taken into account. To give an example: for a pattern phrase "Hillary Clinton" we would permit documents containing: "Hillary Rodham Clinton" and "Clinton, Hillary" but not documents where these two words appear too far apart.

If not enough documents can be collected for a given pattern phrase, it is discarded. Note that the relationship between a cluster label and its documents is very explicit — the label must be present in each document. We believe this *transparency* of relationship between a cluster label and its content is one of the most desirable features of document browsing interfaces, much like the default conjunction (AND) between query terms has become the default behavior of popular search engines.

3.3. The k (number of clusters) problem

We decided that the number of clusters to be discovered inside the k-means part of the DKM will be given a priori. Our decision was motivated by a few factors. First, for our planned experiments with cross-algorithm comparisons we needed the same number of clusters from all algorithms under consideration — a different number of clusters complicates measuring quality greatly (although is possible). Second, estimation of k is possible (Manning, Raghavan and Schütze, 1999), but counterintuitive in text clustering; many authors decide to put this problem away as too problematic to solve together with other issues (see Bekkerman et al., 2007, for an example). Third, descriptive k-means and DCF in general exhibits an interesting "pruning" property that decreases the number of clusters automatically if they have no coverage in the data set. Let us investigate this phenomenon in more detail. The following scenarios are possible:

- No good matches can be found in the set of cluster label candidates for a given cluster centroid. This may be the case, for example, when cluster centroids contain noisy groups of words not occurring anywhere in the text as a phrase. In such situation, the cluster centroid is simply discarded.
- No documents or very few documents are assigned to a pattern phrase. This happens when the pattern phrase was similar to the dominant topic model, but is not relevant to any documents. Consider this example: a cluster centroid vector contains terms *lemony* and *snicket*. A candidate cluster label *Lemony Snicket*[3] will be added to the set of pattern phrases, but since no documents contain this phrase, it is pruned and a potential final cluster is discarded.

As a result, the number of final clusters in DKM may be smaller than the k given to the algorithm. Nonetheless, we are aware this is a weakness of the algorithm and that k could be at least roughly estimated from the source data using one of the known methods (Manning, Raghavan and Schütze, 1999).

3.4. Discussion of computational complexity

Estimation of computational complexity of the entire descriptive k-means is difficult. A great deal depends on the method used to extract candidate cluster labels, for example; shallow linguistic preprocessing algorithms like noun chunking heuristics rarely specify computational complexity. An alternative label candidate extraction method — frequent phrase extraction — can be implemented more efficiently (an implementation based on Ukkonen's, 1995, algorithm is of the order of $O(n)$, where n is the number of input symbols), but have complex memory access characteristics that often turn out inefficient in practice. Our

[3]Lemony Snicket is a pseudonym of Daniel Handler, an American novelist and the author of a series of darkly comic children books known as *A Series of Unfortunate Events*.

experiments show that candidate label extraction phase easily becomes the most expensive phase of the entire algorithm.

Candidate label discovery put aside, the overall algorithm complexity is bound by k-means. Theoretical approach to estimating k-means complexity is given in Arthur and Vassilvitskii (2006). Encountering the pessimistic theoretical bound in practice is very doubtful, however, especially when the termination criterion is a combination of the convergence function and the number of iterations performed by the algorithm. The use of sampling reduces the problem size even further. We found the performance of the entire DKM to be very satisfactory in practice.

4. Implementation aspects

4.1. Term vectors, their properties and efficient implementation of k-means

Let us recall that in the implementation of DKM we used the Lucene library. The indexes created at the time documents are added to the system (documents index and candidate labels index) contain VSM representation of each entry (a sparse vector of terms with non-zero weight). For the needs of k-means clustering we trimmed the size of document vectors even more in the following way:

- a random sample of term vectors for documents in the index is retrieved;
- for each selected document, its features (terms) are weighted and then sorted in descending order of their weights;[4]
- the number of features finally used in k-means to represent a single document is fixed (vectors are truncated and length-normalized). We experimented with representation lengths equal to 30, 50, 70 and 100 features. According to results shown in Schütze and Silverstein (1997), the optimal value of the term vector length should be somewhere within this range and our results support these guidelines.

Compressing the input is one factor that speeds up the main k-means loop, but the key to efficiency is in engineering vector multiplication. If we use cosine distance to calculate similarity between documents then for normalized vectors d_i and d_j the cosine similarity simplifies to:

$$\mathrm{sim}(d_i, d_j) = \cos(\alpha) = \frac{d_i \cdot d_j}{|d_i||d_j|} = d_i \cdot d_j. \tag{1}$$

By representing cluster centroids as dense vectors and documents as (short) sparse vectors, calculation of similarity in (1) is implemented in one loop iter-

[4] We use standard IR term weighting schemes — *tfidf*, mutual information, modified mutual information and modified *tfidf*. The choice of a weighting formula was marginally important; with the exception of pointwise mutual information all other strategies behaved similarly so we omit their details here. A complete description of the term weighting functions used is available in Weiss (2006).

ating over the components of the sparse vector only (this is the actual reason for limiting the number of features). The total cost of calculating similarity between two documents is therefore $\Theta(m)$ multiplications, where m is the number of components of the sparse vector.

Note that sampling and truncation of document features limit memory requirements as well, so the whole k-means procedure can be squeezed into memory even for large document collections. These optimizations make the k-means loop efficient enough to neglect other possible improvements to the baseline algorithm, such as utilizing geometric properties of the clustered space as discussed in Pelleg and Moore (1999).

4.2. Choosing pattern phrases

Selecting pattern phrases is about searching among candidate cluster labels and selecting those similar (or "close to" in vector space model terminology) the discovered dominant topics.

For each cluster centroid we assemble a list of its top weighted terms and their scores. We then construct a weighted query and execute it against the index of cluster candidate labels, retrieving phrases that best match the "profile" of weights of the cluster centroid terms. The queries are constructed programmatically, but could be expressed in a human-readable form as a list of alternatives with numeric boosts (denoting preference) associated with each term. For instance:

$$\textbf{java}_{(0.52)} \text{ OR } \textbf{coffee}_{(0.24)} \text{ OR } \textbf{island}_{(0.12)} \text{ OR } \textbf{language}_{(0.09)} \text{ OR } \ldots$$

Note that at this stage we are not concerned with the order of words or their proximity — cluster label candidates that match the query, but have no coverage in the set of input documents will be pruned later in the document allocation phase anyway.

A query like the one shown in the example above fetches an ordered list of candidate cluster labels for a given dominant topic. Additionally, each label has a score, which is calculated as a relevance to this query. Let us denote this score of a given candidate label p as $\mathsf{query\_score}(p)$. We can influence the process of cluster label selection by allowing the user to express his or her preference of the expected cluster description length by adjusting the score of each label p and penalizing it for being longer or shorter then the desired length of m terms.

The penalty function is a simple bell-like curve:

$$\mathsf{length\_penalty}(p) = \exp \frac{-\big(\mathsf{length}(p) - m\big)^2}{2d^2}, \tag{2}$$

where $\mathsf{length}(p)$ is the number of terms in p and d controls the penalty strictness. A penalized score of a candidate label then becomes:

$$\mathsf{score}(p) = \mathsf{query\_score}(p) \times \mathsf{length\_penalty}(p). \tag{3}$$

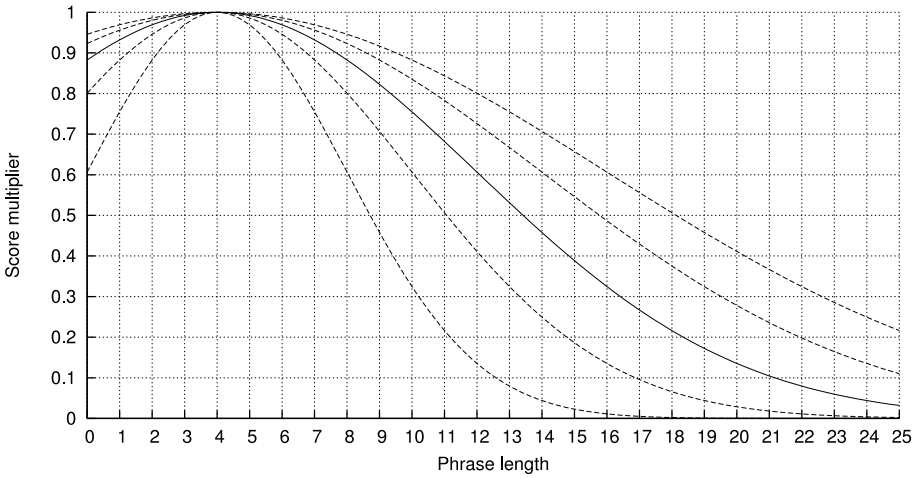


Figure 4. Shape of the phrase length penalization function 2 for desired phrase length $m = 4$ and strictness d replotted for 4 (innermost curve), 6, 8 (solid line), 10 and 12 (outermost curve).

We used fixed values of $m = 4$ and $d = 8$, these proved to work well in practice; Fig. 4 illustrates the shape of this particular penalty function. After applying the penalty function, the set of candidate labels is sorted again and the highest scoring elements become pattern phrases.

4.3. Allocation of documents

Document allocation is more tricky since we want to have a clear relationship between a pattern phrase and the documents allocated to it. A heuristic that can be used for this task could be as follows:

- allocate all documents containing an exact copy of the pattern phrase,
- allocate all documents containing a possibly distorted version of the pattern phrase (reordered words, a few other words injected between the phrase terms); the user should be able to control the allowed level of distortion,
- allocate all documents containing the phrase and any synonymous phrases that could be related to it, but offer the user a possibility of expanding the cluster label to explain which phrases contributed to the allocated documents.

We focused on the first two points from the list above. For each pattern phrase a query is constructed and executed against the index of documents. This time, unlike previously, the query is not a simple Boolean alternative of weighted terms, but a phrase query with some allowance for distortions (slop factor, explained in Appendix A). For a given pattern phrase this query returns a list con-

taining relevance-ordered documents; documents with exact matches are scored higher than those with "sloppier" version of the pattern phrase terms.

The decision how much distortion (and hence potential confusion for the user) is allowed between cluster labels and documents is controlled using the slop factor. This threshold should be most likely left to the user. In our experiments we set it automatically as a linear function of the pattern phrase's length (allowed distortion is proportional to the number of terms in the phrase).

5. Evaluation of clustering quality

5.1. Goals and assumptions

The goals of descriptive clustering slightly differ from those defined for pure data clustering — the point is to find coherent, *well-described* groups of documents. Yet, in our experimental evaluation we tried to investigate how descriptive k-means differs in the clustering quality compared to its baseline k-means algorithm. Measuring quality (understood as document allocation) is fairly well defined and portable across various IR clustering techniques, whereas evaluation of the quality of labels is still a very challenging topic with very few references in literature. We tend to believe that improved cluster labels should be an outcome of the candidate cluster label extraction phase — if candidate labels extracted there are good, then final cluster labels should be comprehensive as well. Since we used two phrase extraction methods already rooted in literature — frequent phrases and noun chunks — our hope was that cluster labels would be comprehensive to humans. Obviously, whether this claim is true should be validated, but due to its inherent difficulty we shifted it to the future work on the subject. Instead, we focused on answering the following questions concerning document allocation quality in DCF and DKM:

- Two elements may degrade clustering quality in DCF: approximation of dominant topics using pattern phrases and document assignment to pattern phrases (instead of cluster centroids) — does DCF improve, degrade or retain clustering quality of the baseline k-means?
- Given the two phrase-extraction methods (frequent phrases, noun chunks), is the quality of clustering between them different and how does it change?
- Does document subsampling used to speed up k-means change or affect clustering quality?

5.2. Data set and evaluation methodology

To validate our results against previous research we decided to use a widely known *20-newsgroups*[5] data set consisting of approximately 20 000 mailing list messages partitioned into 20 different groups. Each group corresponds to a different topic, although some of them are more related than others. Fig. 5 shows the headers of newsgroups in the 20-newsgroups data set and their relation to

[5]See http://peple.csail.mit.edu/jrennie/20Newsgroups/

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Figure 5. Groups of messages in the 20-newsgroups data set, related groups are in the same box.

each other. We chose a subset of the original data set with removed redundancies and empty documents called a "bydate split" and consisting of 18 941 documents. We assumed that each original newsgroup in the data set should be reconstructed as a cluster in the output clustering.

Three combinations of algorithms underwent the evaluation: DKM using English noun phrases as candidate labels, DKM with frequent phrases, and pure k-means. Descriptive k-means appears in two variants because we wanted to see if the candidate label selection strategy affected clustering quality. The implementation of k-means was identical as the one used internally in DKM to detect clusters of dominant topics.

Standard IR evaluation measures assume complete partitioning of the input data set into k disjoint subsets and DKM produces partial partitioning into an unknown number of clusters. We forced the k-clusters output from DKM by modifying it slightly for the needs of the experiment — pattern phrases selected by each centroid formed one merged cluster with a union of documents assigned to each pattern phrase. Any remaining unassigned documents have been excluded from the result. It is worth mentioning that this change actually penalizes the DKM algorithm because by artificially merging documents from different pattern phrases we mix documents that would otherwise end up in different (smaller) clusters. This adjustment was unfortunately necessary to keep the structure of results similar to the ground truth set and evaluation metrics applicable.

The values of thresholds and parameters used in k-means and DKM were set to their best-guess values determined manually before the experiment (see Table 1). We then clustered the ground truth set of documents five times (for visualizations) and a hundred times (for statistical significance analysis) for each possible combination of the following variable elements:

- *sample size* — size of the sample of documents from the original data set,
- *feature type* — type of feature weighting formulas used for feature selection; we used mutual information (*mi*), discounted mutual information (*mid*), plain *tfidf* and its version downplaying the count of terms in the document by taking a square root of the *tf* component (*tfidf2*),
- *feature vector length* — length of the document feature vector, set to 30, 50, 70 or 100 elements.

Table 1. Values of thresholds used in the experiment.

Threshold	k-means	DKM
maximum reassignments $r_{\min}$	20	20
minimal global objective function increase τ	0.001	0.001
minimum documents allocated to a pattern phrase	n/a	10
minimum documents in a final cluster	n/a	5

For every run of the experiment we compared the clustered result against the ground truth partitioning, calculating the following clustering quality indicators: average cluster purity, average normalized entropy, average F-measure and average contamination measure. Entropy and F-measure are widely used and need no further explanation (see Cheng et al., 2005; Dhillon, Fan and Guan, 2001, for details, for example). Cluster purity gives the average ratio of the dominating class in each cluster to the cluster size. The interpretation of values of these measures is given below each figure (Figs. 6–8). Contamination measure for cluster k_i is defined as the number of pairs of objects found in the same cluster k_i, but not coexisting in any of the original partitions, divided by the worst case scenario — maximum number of such bad pairs in k_i, see Weiss (2006). For pure clusters, cluster contamination measure equals 0, for an even mix of documents from all original partitions the measure equals 1.

5.3. The results

We actually expected descriptive k-means to be slightly worse at clustering documents from a predefined collection because the DCF approach should not group documents for which there are no sensible descriptions (even if these documents are similar to each other) and were quite surprised when it turned out that the experiment results show just the opposite — an improvement in quality for most aspects of the analysis.

Clustering quality

Fig. 6 presents average contamination for each weighting scheme used for feature selection. Surprisingly, DKM in both variants is less contaminated than the baseline k-means. This is confirmed by two other quality metrics[6] — average purity (Fig. 7) and average entropy (Fig. 8). In these metrics, however, k-means is slightly better when mutual information (or rather pointwise mutual information) is used for feature weighting. This phenomenon has an good explanation. MI boosts low-frequency terms (Manning and Schütze, 1999), if such terms are selected to represent the cluster centroid then they find little support in the set of candidate labels. We can find the evidence of these suspicions in Fig. 9 —

[6]The F-measure turned out not to be a good choice because it was biased by the resulting cluster size.

the average cluster size for candidates selected among frequent phrases is much lower than with noun phrases, whose selection is not related to frequency of occurrence. This disproportion does not occur with other weighting schemes.

Another reason why DKM came out so good in cluster quality metrics is most likely because it tends to produce more compact, narrow-topic clusters. In all configurations the number of unclustered remaining documents was quite large. This may be, in contrast to the initial feeling, an advantage of the algorithm — a cluster composed of many topics is rarely intuitive and requires more verbose description than a compact, well defined smaller cluster.

Fig. 10 illustrates an average size of a cluster (number of documents inside a cluster), depending on the size of the feature vector. Descriptive k-means produces smaller groups of documents (remember that k-means basically assigns all documents to their closest cluster, so the average cluster size remains constant for a given k and number of input documents). Note that increasing the size of the sample affects the average cluster size gradually. This again indicates that DKM tends to produce smaller, but more accurate clusters (is conservative in document allocation).

The average number of clusters (Fig. 11) shows that descriptive k-means indeed, as explained previously, reduces the number of clusters k — the number of output groups depended mostly on the length of document vectors (which influence cluster centroid calculation and finally selection of pattern phrases). Interestingly, the size of the sample had no visible influence on the number of clusters — this confirms previous results reported in literature (Goswami, Jin and Agrawal, 2004).

We also performed limited statistical analysis of significance between differences in clustering quality for the three algorithms involved. Even though the distribution of original data is unknown, we assumed a sufficiently high number of samples to use a test of difference of means between two populations (Koronacki and Mielniczuk, 2001). The difference for entropy, contamination and purity was statistically significant for $\alpha = 0.05$, suggesting that DKM with any phrase extraction method had a better result than raw k-means. Interestingly, there was no statistically significant difference between the two DKM variants (we expected noun phrases to be more "accurate" and thus improve upon frequent phrases). Full tables comparing the results of statistical analysis are available in Weiss (2006).

Subsampling the input

Assuming cluster centroids remain similar, they should select the same pattern phrases, which, in turn, should allocate identical final content of clusters. And indeed, only the smallest sample (2000 documents) had a different quality characteristic — for samples of 5000, 7000 and 9000 documents the results were very much alike.

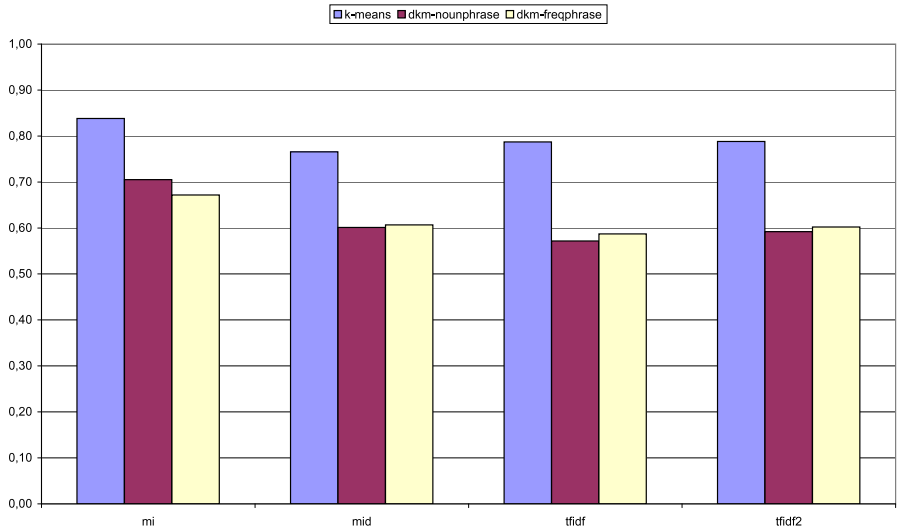


Figure 6. Average cluster contamination depending on the feature type (higher values indicate more contaminated clusters).

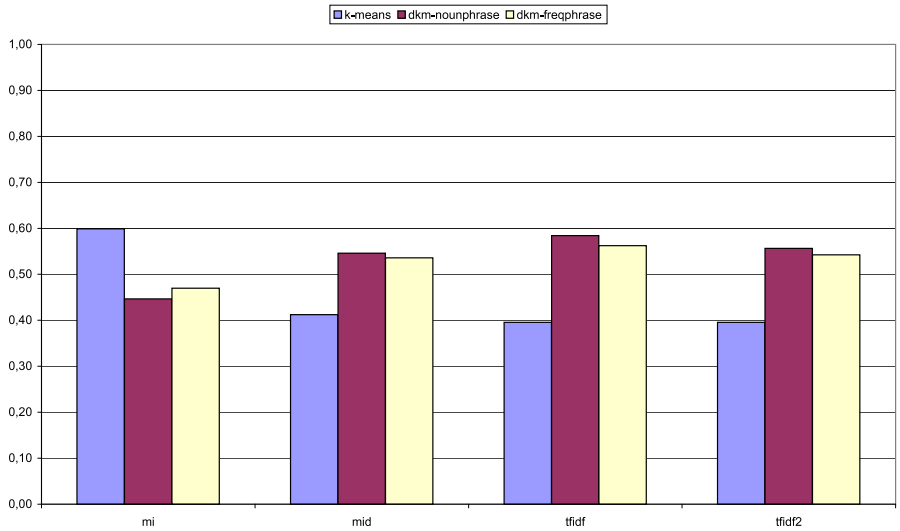


Figure 7. Average cluster purity depending on the feature type (higher values indicate purer clusters).

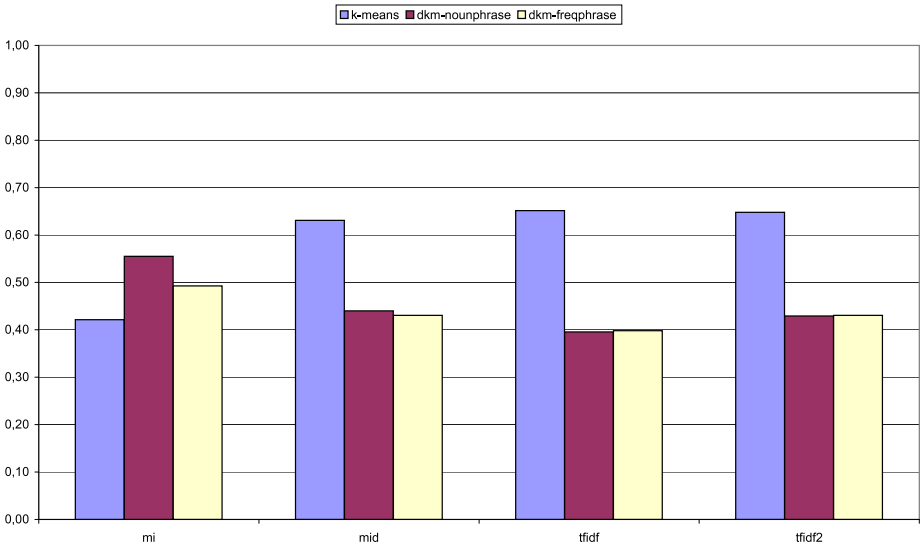


Figure 8. Average entropy depending on the feature type (lower values indicate better clusters).

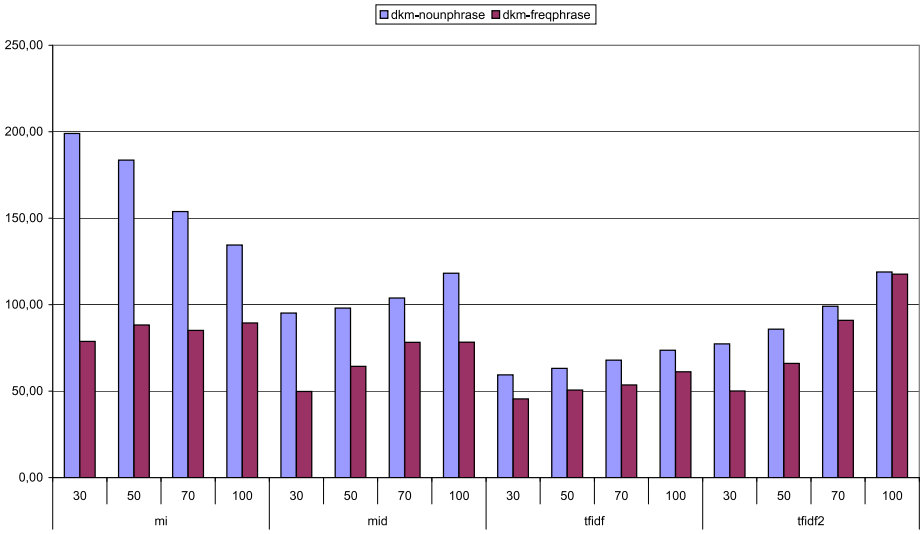


Figure 9. Average size of a cluster depending on the feature type and sample size.

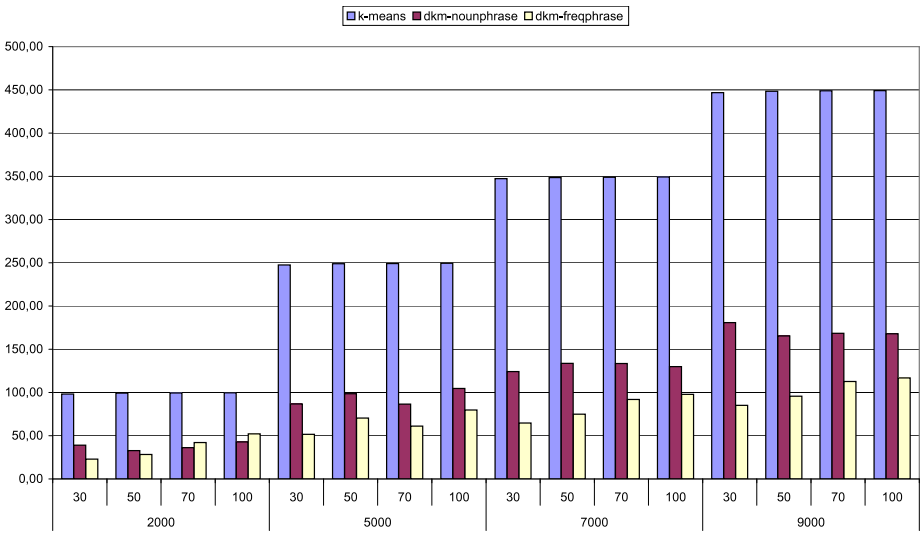


Figure 10. Average size of a cluster depending on the number of features and sample size.

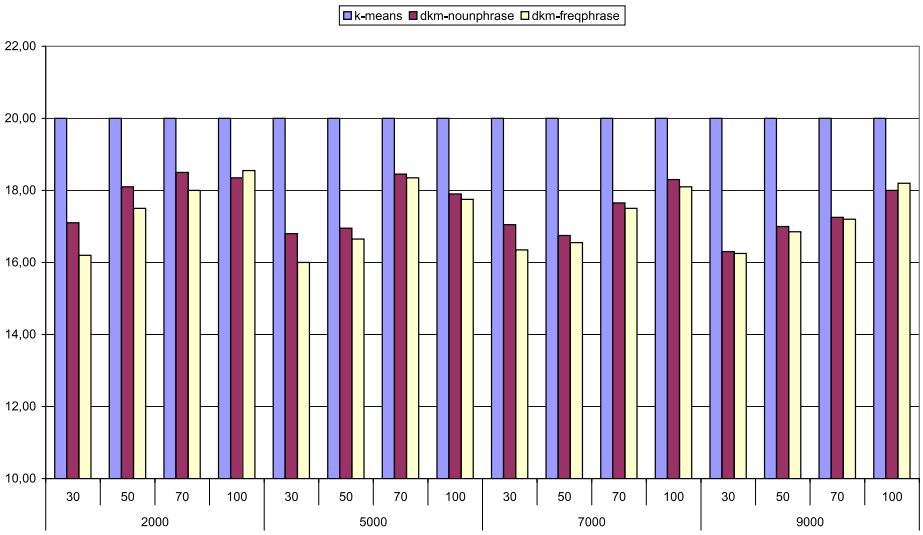


Figure 11. Average number of clusters depending on the number of features and sample size.

An interesting aspect to investigate would be to see how stable the set of pattern phrases is depending on the size of the sample. An experimental validation of this question would require ways of detecting how pattern phrases change (or rather: how their rank changes in a list of results returned for a Boolean query issued to the index of candidate labels). This is an interesting direction for future research, but we have not collected the data to be able to answer this question at the time of this writing.

A few words about the quality of labels

Although it was not among the experiment objectives, we manually inspected cluster labels of several clustered instances. For example, topmost labels from a cluster most likely corresponding to a group called soc.religion.christian were: *Lord Jesus Christ*, *salvation through our Lord Jesus Christ*, *God does not exist*, *existence of God*, *grace of God*. Another sample showed cluster labels concerned with the Muslim/ Jewish communities: *Palestinians, Israel*, *Israel Gaza*, *Serbs, Croats and Muslims*, *Israeli Jews*, *Bosnian Serbs and Bosnian Muslims*.

The quality of cluster labels was, as we generally suspected, satisfactory. Again, note that we relied on the candidate cluster label extraction method to provide the best candidates possible, it was not our intention to measure if frequent phrases or noun phrases were truly comprehensive.

6. Related work

We could divide the related work into two sections — one devoted to clustering algorithms in general (there are a great number of papers here) and another focused on cluster labeling (very few attempts to tackle the problem). The first attempt to combine these two threads was perhaps the *suffix tree clustering* algorithm, presented in Zamir and Etzioni (1999). A follow-up work by Guli and Ferragina (2004) on the *SnakeT* project showed how to avoid certain limitations of STC and use non-contiguous phrases (so-called approximate sentences) for labeling clusters.

Among the approaches with goals similar to DCF, but different in implementation, we should mention the *clustering with committees* algorithm, Pantel and Lin (2002), or Kummamuru et al. (2004) where authors present a search results clustering algorithm which tries to associate documents with a single unique concept. Hotho and Stumme (2002) and Hotho, Staab and Stumme (2003) build a "conceptual clustering" system that refines cluster descriptions using concept lattices. Their cluster descriptions are still single words but they use a large thesaurus and formal concept analysis to avoid repetitions and synonyms in cluster keywords. Applications of concept-lattices to compacting textual information and labeling clusters for display on mobile devices were presented by Carpineto and Romano (2004).

Traces of the pattern-phrase idea can be found in Zeng et al. (2004) — authors perform an interesting experiment with supervised training of a cluster label selection procedure (the classifier function is much like that of cluster centroids in the DCF — select good cluster labels from a larger set).

Sanderson and Croft (1999) present a completely different, yet related approach to exploring document collections. Instead of clustering input documents, they start with salient terms and phrases taken from predefined queries to a document collection and expand this set with a technique called *local context analysis*.

7. Summary and conclusions

We presented the goals and requirements of descriptive clustering — a subclass of the general clustering problem where the clusters are meant to be suitable for use in document browsing interfaces and thus require comprehensive, intuitive cluster labels. We then outlined the proposed solution to the descriptive clustering problem — the description comes first approach — and described how a model of cluster centroids can be combined with potential cluster label discovery to form clear, compact and properly labeled clusters.

We demonstrate a concrete implementation of the DCF on an example of descriptive k-means: a modification of the k-means clustering algorithm. We also describe how certain engineering tricks can be employed to make the algorithm scale to large document collections.

Finally, we provide the results of a computational experiment in which we quantitatively compare the clustering quality of both the modified and the baseline k-means algorithm. The experimental results show that descriptive k-means slightly increases clustering quality. From the way the algorithm is constructed we also speculate that it provides compact and comprehensive cluster labels with intuitive relationship between the content of a cluster and its description.

Directions for the future work are quite broad. The DCF approach relies heavily on the quality of cluster label candidates. Frequent phrase extraction is very convenient here because of its efficiency, but has several disadvantages like word-order dependence, for example. Any method of determining good quality potential cluster labels would most likely improve the overall clustering quality as well. It has been already mentioned that the number of clusters k could be determined from the input data rather than being required in advance. An alternative to estimating k could be to replace k-means with a different clustering algorithm capable of producing cluster centroid representation suitable for the merging phase without explicit knowledge of k. Finally, measuring the quality of the produced cluster labels in terms of comprehensibility, clarity and compactness remains a challenging and unsolved problem.

Acknowledgement

We are very grateful to anonymous reviewers for their constructive suggestions and comments. This work was supported by the Polish Ministry of Science and Higher Education under Grant No. 3 T11C 05227.

A. Terminology related to Lucene

In the paper we refer to certain terms and concepts that are very specific to the information retrieval library Lucene. Below we provide definitions and highlight differences compared to the classic VSM model.

Lucene index Lucene index can contain several things: raw content of added documents, an inverted index of terms, each term associated with a list of documents it occurred in, and a lists of terms and their counts for each document. A Lucene index is therefore a combination of a typical inverted index of terms with on-disk sparse representation of document vectors and a database of document content.

Lucene query model, Boolean query Lucene uses a combination of the vector space model and the Boolean model to execute queries against its indexes. The Boolean model is used to narrow the set of documents to only those that match a Boolean expression given between query terms. So, for example, "Clockwork OR Orange" would select a subset of documents containing any of the query terms, whereas "Clockwork AND Orange" would select a subset of documents with both these terms. The vector space model is used to sort the selected subset of documents based on the relevance score between the query (translated to a vector of terms) and each document.

phrase query A phrase query seeks for documents, in which all the query terms were present and at subsequent positions with respect to the query. For example a phrase query for "Clockwork Orange" would yield documents which contained the term "Clockwork", followed by the term "Orange". The criterion on the order of terms in the query phrase can be relaxed by adding a non-negative *slop factor* (explained below).

slop factor A phrase query with a non-negative slop factor retrieves documents containing all terms contained in the query, but allows certain degree of reordering of query terms and injection of other words in between these terms. The slop factor controls how mangled the phrase can be to still consider the document relevant to the query. Technically, slop factor is defined as a difference in positions between two terms maximally displaced from their original positions in the phrase. For a query p, containing ordered terms $t = t_1, t_2, \ldots, t_n$ and a document d with terms from the query at indices $d_1, d_2, \ldots, d_n$, slop factor is defined as:

$$\mathrm{slop}(p, d) = \max\big(\underset{i}{\forall}(d_i - i)\big) - \min\big(\underset{i}{\forall}(d_i - i)\big). \tag{4}$$

For example, consider a phase query with four terms: "a b c d". The positions of each term in this phrase can be written down as shown in Fig. 12(a). If we have two short documents in the index, one containing reordered query terms: "a c b d", and the other containing non-phrase term injected inside: "a b z c d", the calculations would proceed as shown in Figs. 12(b) and 12(c).

term	a	b	c	d
i	0	1	2	3

(a) The query and its term positions.

term	a	c	b	d
d_i	0	1	2	3
$d_i - i$	0	-1	1	0

$\rightarrow$ slop $= 1 - (-1) = 2$

(b) Coefficients for the slop factor for document "a c b d".

term	a	b	x	c	d
d_i	0	1	2	3	4
$d_i - i$	0	0	-	1	1

$\rightarrow$ slop $= 1 - 0 = 1$

(c) Coefficients for the slop factor for document "a b z c d".

Figure 12. Example calculation of document "sloppiness" for query "a b c d" and two different documents.

References

Abney, S. (1991) Parsing by Chunks. In: R.C. Berwick, S.P. Abney and C. Tenny, eds., *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer Academic Publishers, 257–278.

Arthur, D. and Vassilvitskii, S. (2006) On the worst case complexity of the k-means method. *22nd Annual ACM Symposium on Computational Geometry.*

Baeza-Yates, R.A. and Ribeiro-Neto, B.A. (1999) *Modern Information Retrieval*. ACM Press, Addison-Wesley.

Bekkerman, R., Raghavan, H., Allan, J. and Eguchi, K. (2007) Interactive clustering of text collections according to a user-specified criterion. In: *Proceedings of IJCAI-07, the 20th International Joint Conference on Artificial Intelligence,* Hyderabad, India.

Carpineto, C. and Romano, G. (2004) Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO. *Journal of Universal Computer Science* **10** 8, 985–1013.

Cheng, D., Vempala, S., Kannan, R. and Wang, G. (2005) A Divide-And-Merge Methodology for Clustering. In: *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Baltimore, Maryland, USA, ACM Press, 196–205.

DEAN, J. and GHEMAWAT, S. (2004) Mapreduce: Simplified data processing on large clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*.

DHILLON, I.S. and MODHA, D.S. (2001) Concept decompositions for large sparse text data using clustering. *Machine Learning* **42** 1–2, 143–175.

DHILLON, I.S., FAN, J. and GUAN, Y. (2001) Efficient Clustering of Very Large Document Collections. In: V.K.R. Grossman, C. Kamath and R. Namburu, eds., *Data Mining for Scientific and Engineering Applications*, Kluwer Academic Publishers.

FERRAGINA, P. and GULLI, A. (2004) The Anatomy of SnakeT: A Hierarchical Clustering Engine for Web-Page Snippets. In: *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy. Lecture Notes in Computer Science* **3202**, Springer, 506–508.

GOSWAMI, A., JIN, R. and AGRAWAL, G. (2004) Fast and Exact Out-of-Core K-Means Clustering. In: *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK*. IEEE Computer Society, 83–90.

HOTHO, A., STAAB, S. and STUMME, G. (2003) Explaining Text Clustering Results Using Semantic Structures. In: *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia. Lecture Notes in Computer Science* **2838**, Springer, 217–228.

HOTHO, A. and STUMME, G. (2002) Conceptual Clustering of Text Clusters. In: *Proceedings of FGML Workshop, Special Interest Group of German Informatics Society, Hannover, Germany*, 37–45.

KORONACKI, J. and MIELNICZUK, J. (2001) *Statystyka dla studentów kierunków technicznych i przyrodniczych*. Wydawnictwa Naukowo-Techniczne WNT.

KUMMAMURU, K., LOTLIKAR, R., ROY, S., SINGAL, K. and KRISHNAPURAM, R. (2004) A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results. In: *Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA*. ACM Press, 658–665.

LARSSON, J.N. (1999) *Structures of String Matching and Data Compression*. PhD thesis, Department of Computer Science, Lund University.

MANNING, C.D., RAGHAVAN, P. and SCHÜTZE, H. (2008) *Introduction to Information Retrieval*. Cambridge University Press, (to appear).

MANNING, C.D. and SCHÜTZE, H. (1999) *Foundations of Statistical Natural Language Processing*. MIT Press.

OSIŃSKI, S., STEFANOWSKI, J. and WEISS, D. (2004) Lingo: Search results clustering algorithm based on Singular Value Decomposition. In: M.A. Kłopotek, S.T. Wierzchoń, and K. Trojanowski, eds., *Proceedings of the International IIS: Intelligent Information Processing and Web Mining Con-*

ference (Zakopane, Poland). *Advances in Soft Computing*, Springer, 359–368.

Osiński, S. and Weiss, D. (2005) A concept-driven algorithm for clustering search results. *IEEE Intelligent Systems* **20** (3), 48–54.

Pantel, P. and Lin, D. (2002) Document Clustering With Committees. In: *Proceedings of the 25th ACM International Conference on Research and Development in Information Retrieval, Tampere, Finland*. ACM Press, 199–206.

Pelleg, D. and Moore, A. (1999) Accelerating Exact k-means Algorithms with Geometric Reasoning. In: *Knowledge Discovery and Data Mining*, 277–281.

Salton, G. (1989) *Automatic Text Processing – The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley.

Sanderson, M. and Croft, B. (1999) Deriving Concept Hierarchies from Text. In: *Proceedings of the 22nd ACM International Conference on Research and Development in Information Retrieval, Berkeley, USA*. 206–213.

Schütze, H. and Silverstein, C. (1997) Projections for Efficient Document Clustering. In: *Proceedings of the 20th ACM International Conference on Research and Development in Information Retrieval, Philadelphia, PA, USA*. ACM Press, 74–81.

Ukkonen, E. (1995) On-Line Construction of Suffix Trees. *Algorithmica* **14** 3, 249–260.

Weiss, D. (2006) *Descriptive Clustering as a Method for Exploring Text Collections*. PhD thesis, Poznań University of Technology, Poznań, Poland.

Zamir, O. and Etzioni, O. (1999) Grouper: A Dynamic Clustering Interface to Web Search Results. *Computer Networks* **31**, 11–16, 1361–1374.

Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y. and Ma, J. (2004) Learning to Cluster Web Search Results. In: *Proceedings of the 27th ACM International Conference on Research and Development in Information Retrieval, Sheffield, United Kingdom*, ACM Press, 210–217.

Zhang, T., Damerau, F. and Johnson, D. (2002) Text Chunking Based on a Generalization of Winnow. *Journal of Machine Learning Research* **2**, 615–637.