Topic Grouper - An Alternative Computational Approach in the Field of Probabilistic Topic Modeling

Daniel Pfeifer, Hochschule Heilbronn (Germany)

July 7, 2016

1 Introduction

This document is work in progress. It presents the basic concepts of a new approach for topic modeling. This new approach, called "Topic Grouper", essentially combines a generative stochastic perspective on topic computation with agglomerative clustering. As opposed to conventional clustering approaches for document collections, Topic Grouper essentially establishes a cluster distance on topics instead of documents.

Topic Grouper starts with one-word-topics, of which there are as many as words in the vocabulary. At every step, it selects the topics "closest" to each other and joins them. This way topics, i.e. sets of words, remain disjunct at every step. The algorithm ends at the point where there is just one topic left – the union of all words in the vocabulary. On the way there, it provides a solution for every possible number of disjunct topics (ranging between the vocabulary size $|V|$ and 1).

The corresponding cluster distance measure is derived by considering documents to be generated via a stochastic process over disjunct topic sets. This is a principal quality that Topic Grouper shares with "purer" statistical approaches such as LDA [Blei et al., 2003] and its variants. The generative model of Topic Grouper lends itself well to apply clustering in reasonably efficient way (see Section 2.1). It is simple and intuitive but differs from the generative model behind LDA and its variants: Since with Topic Grouper, words get hard-assigned to topics, there is no need to statistically infer "word per topic distributions" or "topic per document distributions". Nevertheless, the importance of words in a topic can be determined directly by corresponding word frequencies as given via the corpus. Analogously, the influence of topics per document can determined as well.

1.1 Clustering and Topic Modeling

Document clustering is obviously very popular and has been under consideration since the early days of information retrieval [Salton et al., 1975]. However applying clustering to topics directly, i.e. the latent topics in documents and document collections, seems to be a new approach. [Wallach, 2008] discusses clustering and topic modeling in Chapter 5 but only with regard to clustering

documents – not topics. The authors of [Yin and Wang, 2014] created a clustering approach based on a Dirichlet Multinomial Mixture Model (DMM), but again the focus is on clustering documents – not topics. (The clusters themselves are computed via Gibbs sampling as opposed to standard clustering algorithms.)

1.2 Hyperparameters and LDA

In comparison to LDA (which is the predominant approach in probabilistic topic modeling), Topic Grouper is much less susceptible to the choice of hyperparameters. In fact, in its basic variant, Topic Grouper works without any hyperparameters at all. In contrast, classical LDA mandates the setting of the number of topics n and two vectors $\alpha \in \Re^n$ and $\beta \in \Re^{|V|}$ with $|V|$ being the vocabulary size. Finding "good" values for those LDA-related hyperparameters is anything but trivial. A body of scientific work has been created around this issue including:

- a generalization of LDA to compute n as part of an enhanced generative model [Teh et al., 2004],
- a generalization of LDA to compute α as part of a generative model [Wallach et al., 2009a], also showing that the choice of a symmetric β using a default value is usually sufficient,
- adapting LDA such that it is less susceptible to the choice of α [Tan and Ou, 2010],
- defining measures for hyperparameter search and selection with regard to n [Arun et al., 2010],
- avoiding heavily differing topic probabilities (affecting the choice of α) by filtering stop words or including a background model [Chemudugunta et al., 2006].

1.3 State of Knowledge on Topic Grouper

Using synthetic data sets according to [Tan and Ou, 2010] Topic Grouper has already shown

- to work reasonably efficient on these data sets,
- to offer good solutions for these data sets,
- to offer good solutions in the case of differing topic probabilities (in which case LDA fails when using a symmetric prior α, as is often done in practice),
- to allow for picking the correct (or a reasonable) number of topics via a simple inspection of a so-called "likelihood improvement graph". (The latter can easily be generated during a run of the Topic Grouper algorithm.)

However, Topic Grouper poses still a lot of open questions including (but not limited to)

- how to deal with topical homonyms, i.e. words that should belong to more than one topic because of their multiple meanings,

- whether Topic Grouper is "stable" with regard to "topical homonyms", i.e. whether it produces plausible results, even if topical homonyms are present in documents but are not handled specially as part of the algorithm,
- whether its performance with regard to the commonly used evaluation measure "perplexity" is competitive,
- whether its theoretical complexity (according to Section 2.2) holds for a corresponding implementation,
- whether it actually produces plausible topics as results on typical document collections for evaluation but also on "real world" document collections,
- whether it scales or can be adapted to scale with regard to (very) large document collections.

These issues shall be addressed in the near future.

2 Theory behind Topic Grouper

2.1 Basic Approach

Let

- D be the document set with size $|D|$,
- V be the vocabulary of D with size $|V|$,
- $f_d(w)$ be the frequency of a word $w \in V$ with regard to $d \in D$,
- $f(w) := \sum_{d \in D} f_d(w) > 0$, since otherwise w would not be in the vocabulary,
- $|d| := \sum_{w \in V} f_d(w) > 0$, since otherwise the document would be empty.

Let $T(n) = \{\ t|\ t \subseteq V\}$ be a (topical) partitioning of V such that $s \cap t = \emptyset$ for any $s, t \in T(n)$, $\bigcup_{t \in T(n)} t = V$ and $|T(n)| = n$. Let $t(w)$ be the topic of word w such that $w \in t \in T(n)$.

Our goal is to find the optimal partitioning $S(n)$ over all $T(n)$ for a number of topics n:

$$S(n) := argmax_{T(n)} \prod_{d \in D} \prod_{w \in V, f_d(w) > 0} (p(w|t(w)) \cdot p(t(w)|d))^{f_d(w)}. \quad (1)$$

Let $q(n)$ be the value of the argmax-expression for a corresponding $T(n)$. The idea is that a document $d \in D$ is considered to be generated via a simple stochastic process where a word w in d occurs by

- first sampling a topic t according to a probability distribution $p(t|d)$ (topics per document),
- then sampling a word from t according to a probability distribution $p(w|t)$ (words per topic).

- So the total probability of generating D is proportional to $q(n)$.

The optimal partitioning $S(n)$ consists of n pairwise disjunct subsets of V, whereby each subset is meant to represent a topic. By definition every word w must be in exactly one of those sets. This may help to keep topics more interpretable for humans because they do not overlap with regard to their words. On the other hand, homonymic words can only support one topic, even though it would be justified to keep them in several topics due to multiple contextual meanings. Note that the approach considers a solution for every possible number of topics ranging between $|V|$ and 1.

Let $f_d(t) := \sum_{w \in t} f_d(w)$ be the topic frequency in a document d and $f(t) := \sum_{w \in t} f(w) = \sum_{d \in D} f_d(t)$ be the number of times t is referenced in D via some word $w \in t$. To maximize $q(n)$ we use the maximum likelihood estimation for $p(t(w)|d)$ and $p(w|t(w))$ according to D:

- $p(t(w)|d) \approx f_d(t(w))/|d|$, which is > 0 if $f_d(w) > 0$,
- $p(w|t(w)) \approx f(w)/f(t(w))$, which is always > 0 since $f(w) > 0$.

Unfortunately, constructing the optimal partitionings $\{S(n) \mid n = 1 \dots |V|\}$ is computationally hard. *We suggest a greedy algorithm that constructs sub-optimal partitionings $S'(n)$ instead*, starting with $S'(|V|) := \{\{w\} \mid w \in V\}$ as step $i = 0$. At every step $i = 1 \dots |V| - 1$ the algorithm joins two different topics $s, t \in S'(|V| - (i-1))$ such that $q(|V| - i)$ is maximized along $S'(|V| - i) := S'(|V| - (i-1)) - \{s, t\} \cup \{s \cup t\}$. Alternatively, this may be considered as an *agglomerative clustering approach, where topics not documents form respective clusters.*

For efficient computation we first reorder the terms of $q(n)$ with a focus to topics in the outer factorization:

$$q(n) = \prod_{t \in T(n)} \prod_{d \in D, f_d(t) > 0} (p(t|d)^{f_d(t)} \cdot \prod_{w \in t} p(w|t)^{f_d(w)})$$

We maximize $\log q(n)$ instead of $q(n)$ which is equivalent with respect to the argmax-operator. This leads to

$$\log q(n) = \sum_{t \in T(n)} \sum_{d \in D, f_d(t) > 0} (f_d(t) \cdot \log p(t|d) + \sum_{w \in t} f_d(w) \cdot \log p(w|t)) \approx \sum_{t \in T(n)} h(t)$$

with the maximum likelihood estimation $h(t) :=$

$$\sum_{d \in D, f_d(t) > 0} f_d(t) \cdot (\log f_d(t) - \log |d|) + \sum_{w \in t} f(w) \cdot \log f(w) - f(t) \cdot \log f(t). \quad (2)$$

Using these formulas the join of of two (disjunct) topics $s, t \in S'(n)$ results in

$$\log q(n-1) \approx \log q(n) + \Delta h_n,$$

$$\Delta h_n := argmax_{s,t \in T(n)} \Delta h(s,t),$$

$$\Delta h(s,t) := h(s \cup t) - h(s) - h(t)$$

which makes the computation of $q(n-1)$ more efficient. If Topic Grouper is perceived as a clustering approach, then $\Delta h(s,t)$ may be considered the distance measure between two clusters s and t.

Moreover, we can reuse $h(s)$ and $h(t)$ from step $n-1$ in order to compute $h(s \cup t)$ efficiently as follows: Regarding expression (2) from above, let $i(t) := \sum_{d \in D} \sum_{w \in t} f_d(w) \cdot \log f(w) = \sum_{w \in t} f(w) \cdot \log f(w)$. We have $f_d(s \cup t) = f_d(s) + f_d(t)$, $f(s \cup t) = f(s) + f(t)$ and $i(s \cup t) = i(s) + i(t)$, so

$$h(s \cup t) = \sum_{d \in D, f_d(s \cup t) > 0} f_d(s \cup t) \cdot (\log f_d(s \cup t) - \log |d|) +$$

$$i(s \cup t) - f(s \cup t) \cdot \log f(s \cup t) =$$

$$\sum_{d \in D, f_d(s)+f_d(t) > 0} (f_d(s) + f_d(t)) \cdot (\log(f_d(s) + f_d(t)) - \log |d|) +$$

$$i(s) + i(t) - (f(s) + f(t)) \cdot \log(f(s) + f(t)).$$

Considering the algorithm, the terms $i(u)$ and $f(u)$ with $u = s, t$ have already been computed in step n with regard to h, so the computation of all sums over words w can be avoided with respect to $h(s \cup t)$. This is essential for the runtime complexity and practical performance.

2.2 Algorithm Sketch and Computational Complexity

- The algorithm starts with step 0 by computing all possible joins of distinct topics $s, t \in S'(|V|)$ with $|s| = |t| = 1$, saving the best possible join partner for each topic. The corresponding complexity is in $O(|V|^2 \cdot |D|)$. ...
- At each step $i > 0$ the algorithm joins two topics $s, t \in S'(|V| - (i-1))$ and recomputes the best possible join partner for $s \cup t$ with $|V| - i - 1$ potential join partners available. Moreover, it adjusts the best possible join partners for those topics $u \in S'(|V| - (i-1))$ with $u \neq s, t$ whose best possible join partner was s or t at step $i-1$. The expected number of such topics u is in $O(1)$. So, the total complexity is $O(|V|^2 \cdot |D|)$.

2.3 Topics per Document Distribution Prior

This extension hasn't proven to be useful for synthetic data sets (see Section 3.1) but may be it will do well for document collections.
In many cases we would like to give those solutions a preference that keep the number of topics per document small. Regarding the topics per document distribution $p(t|d)$, the entropy measure $-\sum_t p(t|d) \cdot \log p(t|d)$ reaches its maximum for the uniform distribution and decreases for peaked, non-uniform distributions. So, in order to favor peaked distributions, we optimize with respect to $lq'(n) = \lambda \cdot e(n) + \log q(n)$ and instead of $\log q(n)$ using

$$e(n) := \sum_{t \in T(n)} \sum_{d \in D, p(t|d) > 0} p(t|d) \cdot \log p(t|d) \approx \sum_{t \in T(n)} g_n(t)$$

and with the maximum likelood estimation

$$g_n(t) := \sum_{d \in D, f_d(t) > 0} f_d(t)/|d| \cdot (\log f_d(t) - log|d|).$$

In this context, $\lambda \geq 0$ is a weighting factor that determines the impact of $e(n)$ during optimization steps. In analogy to $q(n)$ and with $h'(t) := h(t) + \lambda \cdot g_n(t)$, we obtain

$$h'_n(t) = \sum_{d \in D, f_d(t) > 0} (1 + \lambda/|d|) \cdot f_d(t) \cdot (\log f_d(t) - \log |d|) + i(t) - f(t) \cdot \log f(t).$$

In analogy to $\log q(n)$ the join of two distinct topics $s, t \in T(n)$ then results in

$$lq'(n-1) \approx lq'(n) + \Delta h'(s,t),$$
$$\Delta h'(s,t) := h'(s \cup t) - h'(s) - h'(t).$$

2.4 Optimization During Initialization and Document Sampling

During initialization (step 0) we have $t = \{w\}$ and so

$$h'(\{w\}) = \sum_{d \in D, f_d(w) > 0} (1 + \lambda/|d|) \cdot f_d(w) \cdot (\log f_d(w) - \log |d|).$$

We also compute the best possible join partner $s = \{v\}$ for some $t = \{w\}$ and so

$$h'(\{v,w\}) =$$
$$\sum_{d \in D, f_d(v) + f_d(w) > 0} (1 + \lambda/|d|) \cdot (f_d(v) + f_d(w)) \cdot (\log(f_d(v) + f_d(w)) - \log |d|) +$$
$$f(v) \cdot \log f(v) + f(w) \cdot \log f(w) - (f(v) + f(w)) \cdot \log(f(v) + f(w)).$$

The first sum in this expression is problematic because we would have to iterate over the document set to compute it. Using an inverted index, we can avoid considering documents with $f_d(v) = 0$ and $f_d(w) = 0$.

To further reduce the overhead, we may use sampled document subsets $E \subseteq D$ in order to estimate the first term's value:

$$|D|/|E| \cdot \sum_{d \in E, f_d(v) + f_d(w) > 0} (1 + \lambda/|d|) \cdot (f_d(v) + f_d(w)) \cdot (\log(f_d(v) + f_d(w)) - \log |d|)$$

This way, if we use fixed-sized sample subsets E for all $h'(t)$, the approach's complexity becomes independent of the size of the document set $|D|$ being reduced to $O(|V|^2 \cdot |E|)$.

2.5 Dealing with Topical Homonyms

We consider a word w as a *topical homonym*, if it is semantically justified to add w to more than one topic due w's multiple meanings. Hereby, each meaning of w may fit one of several semantically distinct topics. E.g., the word "play" may suit a topic t_1 revolving around playing musical instruments but also another topic t_2 revolving around playing board games.

So far, topical homonyms do not fit the approach because it supports disjunct topics only. Maybe worse, the relevance of a topical homonym for several topics may cause such topics to be joined, even though those they are semantically quite different. To deal with a topical homonym w we adapt the approach such that w is

1. recognized at early steps, i.e., when w is still in a two-word-topic $t = \{w, q\}$,
2. then removed from consideration during the run altogether such that w does not influence the join process in an undesired way,
3. readded after the run to all topics (across steps) that would contain w, had it not been removed before,
4. potentially readded to other topics that should contain w just as well due to w's alternative meanings.

Surprisingly, the algorithm can be well accommodated for this: Let s be the best possible join partner for t at step n of the algorithm and let $|s| \leq 2$. The homonym check uses the measure

$$g_s(w) := 1/|s| \cdot \sum_{v \in s} h(\{w\} \cup s)$$

and the ratio

$$u_s(w, q) := |g_s(w) - g_s(q)|/(|g_s(w)| + |g_s(q)|).$$

The algorithm first checks if $u_s(w, q) > \epsilon$ where ϵ is a parameter to control homonym detection sensitivity. If $u_s(w, q) > \epsilon$ holds, then either w and s or q and s should not end up in a joint topic because according to h they hardly co-occur. Note that the algorithm has already chosen s and $t = \{w, q\}$ as best join partners so either w and s co-occur or else q and s co-occur. If $g_s(w) > g_s(q)$ then q and s have less in common and so w must be the potential homonym. Otherwise q must be the potential homonym.

According to this check and according to 2) from above, the potential homonym is removed from its topic t. Also, s and t are *not* joined. All stored values with regard to the computation of h are adjusted accordingly (as if the homonym had not been in the vocabulary) except for document sizes $|d|$. The latter would require to recompute all currently stored best join candidates, which would be too in efficient. It the number of topical homonyms is small, the resulting error remains small and does not affect the course of the join process much.

We end up with resulting unjoined topics of size ≤ 2. Since the algorithm starts with topics of size 1, every word will be in topics of size 2 at some and step will necessarily undergo the homonym check.

2.6 Topic Coherence Prior

Nice idea but turned out to be way too inefficient... Informally speaking, a topic is *coherent* if the words of a topic tend to *all* pairwise co-occur (frequently) in documents of the collection. So far, the approach may lead to topics, which may not be coherent: For example, the a topic $\{south, korea\}$ may be joined with $\{africa\}$ because both "korea" and "africa" co-occur with "south" although "korea" and "africa" do not co-occur in documents.

Our goal is to design a prior $co(t)$ that captures topic coherency but is also efficiently computable as part of $h(t)$ from Section **??**. We define coherence $co(t)$ of a topic t as follows:

$$co(t) := \prod_{d \in D} (\min_{w \in t} f_d(w) + 1)/(\max_{w \in t} f_d(w) + 1).$$

We therefore have:

$$co(t) \in (0 \dots 1], co(\{w\}) = 1, s \subseteq t \Rightarrow co(s) \geq co(t).$$

The latter equation is easy prove as the following holds for arbitrary sets M, N with elements ≥ 0:

$$M \subseteq N \Rightarrow \min M \geq \min N, \max M \leq \max N \Rightarrow \min(M{+}1)/\max(M{+}1) \geq \min(N{+}1)/\max(N{+}1)$$

This applies to every factor of co and so to the whole product.

We further define m-limited coherence $co^m(t)$ with $m \in \aleph$. Assuming that $t = s \cup r$ is the result of a former join of the topics r, s in the steps of Topic Grouper:

$$co^m(t) := co(t) \; if \; |t| \leq m \; and \; co^m(t) := \min\{co^m(r), co^m_l(s)\} \; else.$$

3 Evaluation

3.1 Synthetic Data Sets

3.1.1 Data Set According to [Tan and Ou, 2010]

For comparison with other work and as a first test of Topic Grouper, we used a synthetic data set, which is generated exactly as described in [Tan and Ou, 2010] Section III.A: It consists of 400 words equally devided into 4 topics. The words are represented by numbers, such that $0 \dots 99$ belongs to topic 1, $100 \dots 199$ to topic 2 and so on.

The word per topic distribution $p(w|t)$ for each topic t is drawn independently for each topic from a Dirichlet distribution with a symmetric prior $\alpha = 1/100$. There are 6000 documents, each consisting of 30 word occurrences. The topic per document distribution $p(t|d)$ is drawn independently for each document from a Dirichlet distribution with the prior $\alpha = (5, 0.5, 0.5, 0.5)^\top$, where topic 1 with $\alpha_1 = 5$ is meant to represent a typical "stop word topic", which is more likely than other topics.

To generate a word occurrence in a document d, its topic is first drawn from $p(t|d)$. Once the topic is known, the word is drawn via $p(w|t)$. The procedure is repeated for every word occurrence of each document.

We ran Topic Grouper and LDA on the resulting data set.[1] To run LDA, we preset the number of documents $n = 4$, a symmetric $\alpha = 1.5$ and $\beta = 0.5$. The Gibbs sampler-based LDA implementation was set to perform 1000 iterations. Just as reported in [Tan and Ou, 2010] LDA fails to recover the original topics. Figure 1 shows the top ten words along with their probabilities as computed for each topic using LDA – all of these words actually belong to the stop word topic 1 of the synthetic data set. (So, this LDA-based solution is "useless".)

We adapted the LDA-library jLDADMM such that it can deal with an asymmetric prior α and ran it as above except that α was set to $(5, 0.5, 0.5, 0.5)^\top$ (as provided during the generation of the synthetic data set). The result in Figure 2 shows that under these conditions, LDA can indeed recover all topics appropriately. The problem is obviously, that presetting α correctly is not possible as easily for real world data but is rather a part of the problem to be

[1] We used jLDADMM as available under https://github.com/datquocnguyen/jLDADMM.

```
Topic0: 98(0.017555) 43(0.017203) 33(0.01693) 61(0.016871) 70(0.016637) 47(0.016559) 49(0.016559) 69(0.016129) 11(0.015738) 60(0.015386)
Topic1: 33(0.013984) 30(0.013522) 45(0.012326) 61(0.012326) 60(0.012163) 18(0.012055) 58(0.011783) 63(0.011484) 46(0.011076) 68(0.010831)
Topic2: 46(0.01505) 39(0.013703) 38(0.012759) 98(0.011843) 97(0.011628) 67(0.011493) 8(0.011277) 27(0.011169) 1(0.011008) 10(0.010792)
Topic3: 12(0.01236) 7(0.012221) 8(0.012221) 77(0.011971) 11(0.011943) 30(0.01186) 49(0.011276) 61(0.011109) 1(0.010469) 93(0.010469)
```

Figure 1: Result of running LDA on the synthetic data set as described above with a symmetric prior $\alpha = 1.5$ (Generated with class `SymmetricLDAGibbsTester`.)

```
Topic0: 61(0.01685) 30(0.016791) 98(0.016699) 39(0.016657) 58(0.016062) 60(0.01581) 18(0.015785) 27(0.01576) 33(0.015735) 49(0.015651)
Topic1: 148(0.018013) 172(0.017943) 158(0.016956) 116(0.016039) 176(0.015969) 184(0.015757) 157(0.015264) 119(0.015052) 109(0.014982) 111(0.014982)
Topic2: 368(0.019179) 364(0.018681) 317(0.017969) 324(0.017969) 315(0.017898) 307(0.016902) 379(0.016546) 304(0.016475) 301(0.016048) 318(0.015834)
Topic3: 219(0.01746) 273(0.01746) 208(0.016719) 212(0.016496) 200(0.016422) 283(0.016274) 233(0.0162) 262(0.016125) 206(0.015977) 235(0.015977)
```

Figure 2: Result of running LDA on the synthetic data set as described above with an asymmetric prior $\alpha = (5, 0.5, 0.5, 0.5)^\top$ (Generated with class `AsymmetricLDAGibbsTester`.)

analyzed. This situation is also discussed in [Wallach et al., 2009a] along with a suggested extension of LDA that integrates out α or alternatively recommends a hyperparameter search over α. While the extension was reported to cause high computational complexity, the algorithm for hyperparameter search was not discussed in detail (see [Wallach, 2008] for details?).

We also ran the basic version of Topic Grouper (no hyperparameters required): Figure 3 shows the values of Δh at each join step (Y axis) with respect to the number of topics n (X axis). Notice that the graph is computed from right to left because the algorithm decrements the number of topics. The Δh_n values decrease from right to left because the model has less and less topics available to generate or "explain" the documents. As one can see, there is a clear drop at $n = 4$ which indicates that this is a good choice. In addition, Figure 4 shows $\Delta h_{n-1}/\Delta h_n$ for $|T(n)| < 105$ which highlights sudden changes from step to step and thus supports the choice n.

The first four lines of Figure 5 show the top ten words as computed for each topic via Topic Grouper at $n = 4$. In this case, the words per topic are sorted by the frequency in the document collection (as shown in parentheses). All words were assigned correctly. Although not shown, this also holds for most lower ranked words. Errors occur at lower frequencies usually because those frequencies are not significant anymore (i.e., they do not correspond to their true generative probabilities due to the small size of the data set).

The last line of Figure 5 shows the counts $f(t_i)$ for each of the computed topics $t_1, \ldots, t_4$. The frequencies closely match the ratios of the prior $\alpha = (5, 0.5, 0.5, 0.5)^\top$ used at the generation of data set.

Note that we have tried many other variations when parametrizing the generation of the synthetic data set including other document sizes (that also vary across documents), larger document collections, more topics, other α-priors an so on. The results reported above are representative in a sense that all data set variations led to similar results and conclusions.

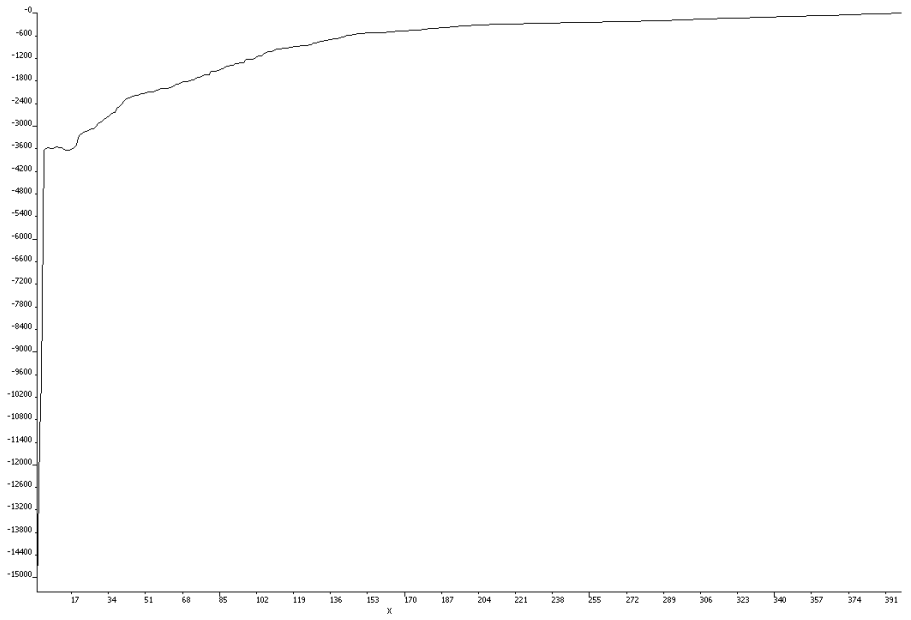

Figure 3: Δh per number of topics n when running Topic Grouper on the synthetic data set as described above (Generated with class `OptimizedTGTester`.)

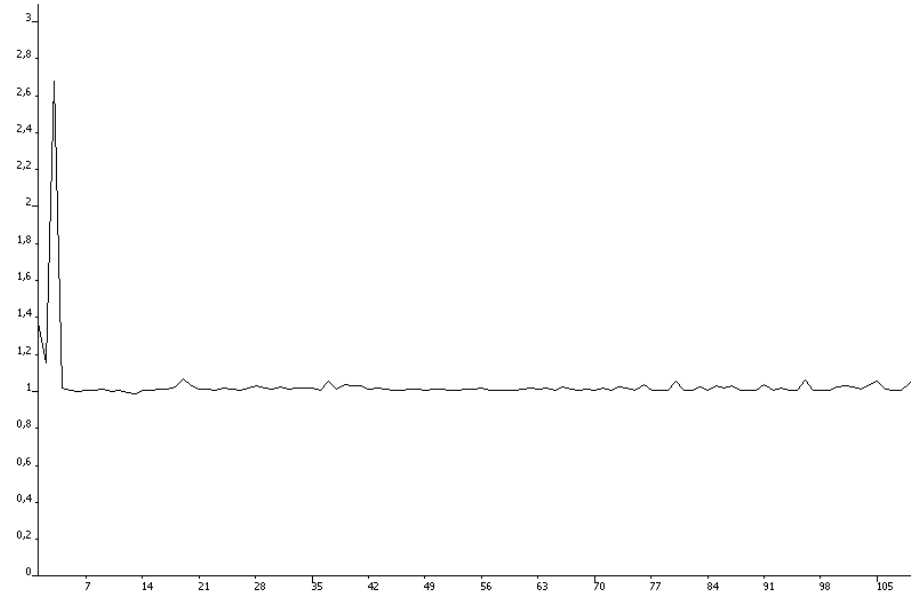

Figure 4: $\Delta h_{n-1}/\Delta h_n$ per number of topics n when running Topic Grouper on the synthetic data set as described above (Generated with class `OptimizedTGTester2`.)

```
[61 (2471), 30 (2421), 39 (2399), 98 (2396), 58 (2317), 18 (2303), 60 (2294), 46 (2285), 33 (2285), 27 (2282) ...
[172 (281), 148 (277), 116 (270), 184 (270), 158 (257), 193 (253), 176 (252), 111 (251), 144 (246), 119 (243) ...
[317 (301), 368 (291), 379 (290), 364 (280), 315 (277), 324 (267), 319 (262), 307 (261), 304 (254), 343 (247) ...
[219 (267), 262 (250), 212 (250), 283 (249), 208 (248), 200 (247), 273 (243), 235 (241), 214 (236), 269 (236),...
Topic frequencies: [138856 14460 13236 13448 ]
```

Figure 5: Result of running Topic Grouper on the synthetic data set as described above (Generated with class `OptimizedTGTester`.)

3.1.2 Data Set with Topical Homonyms

3.2 Perplexity Analysis

It is well known that the optimization of probabilistic topic models via predictive holdout measures (such as perplexity) can be problematic because such optimized models often do not coincide (at all) with human intuition and expectation regarding soundness of topics. This was more closely investigated in [Chang et al., 2009]. Nevertheless related comparisons are useful and well established because they tell about the predictive power of corresponding models regarding given data [Blei et al., 2003, Chen and Wang,].

3.2.1 Perplexity Computation

We apply *perplexity* as discussed in [Blei et al., 2003] to a test collection D_{test}. The measure is defined as follows:

$$perplexity(D_{test}) := \exp(-\sum_{d \in D_{test}} \log p(d) / \sum_{d \in D_{test}} |d|)$$

Using topic modeling the document probability $p(d)$ under a (trained) topic model should be determined as follows:

$$p(d) = (|d|! / \prod_{w \in V, f_d(w) > 0} f_d(w)!) \cdot \prod_{w \in V, f_d(w) > 0} p(w|d)^{f_d(w)},$$

$$p(w|d) = (\sum_{t \in T} p(w|t) \cdot p(t|d)).$$

The expression $|d|! / \prod_{w \in V, f_d(w) > 0} f_d(w)!$ accounts for the underlying bag of words model for documents (where word order is ignored).[2] Note that the size of a test document $|d| = \sum_{w \in V} f_d(w)$ is measured by using with words from the training vocabulary V.

Regarding Topic Grouper, the probability $p(w|d)$ can be estimated as follows on the basis of a partitioning $T(n)$ and the maximum likelihood estimations for $p(t|d)$ and $p(w|t)$ from equation 1:

$$p(w|d) \approx (f(w)/f(t(w)) \cdot f_d(t(w))/|d|).$$

Note that f_d and $|d|$ refer to D_{test} whereas $t(w)$, $f(w)$, $f(t)$ and V belong to the model (i.e., the partitioning $T(n)$ of V into topics) and the training collection D respectively. Also note that the sum over all topics $t \in T$ from above has only one non-zero addend because $p(w|t) = 0$ if $w \notin t$.

[2] It is often ignored in publications because if two approaches are compared (with the same holdout data), the expression turns out to be an identical factor both perplexity results.

The estimation has to extreme cases: If $|T(n)| = 1$, then $p(w|d) \approx (f(w)/\sum_{w \in V} f(w))$ – this is the unigram model. If $|T(n)| = |V|$, then $p(w|d) \approx (f_d(w)/|d|)$, which means word probabilities are estimated directly via the test document (and nothing is learned from the training data).

Unfortunately, computing $p(w|d)$ on the basis of an LDA-model is not as straight-forward because $p(t|d)$ from the above expression is unknown for test documents d. Theoretical solutions to this issue are discussed in [Wallach et al., 2009b]. We tried Mallet's[3] implementation of the "Left-to-Right" algorithm from [Wallach, 2008] in order to compute $\log p(d)$ directly for given a LDA-model and a test document d. For reference, we call this approach "LDA-P-Method 1". Unfortunately Mallet's respective implementation seems to be flawed and returned errors during computation. Apart from this problem, we believe that the approach is generally questionable, since it actually defers a training task (requiring considerable statistical inference efforts) to the evaluation phase.

Another alternative discussed in a forum entry[4] is to hold out a percentage of words from each training document and then to compute $p(d)$ on the held out words (by considering the set of held out words from each training document as a test document). Consequently, each training document is reduced by the corresponding set of held out words. $p(t|d)$ is then obviously known from the LDA model, since d (or at least parts of it) were used during training. The author of the entry argues that the approach is suboptimal and biased because it does not work with "proper" test documents. For reference we call this approach "**LDA-P-Method 2**".

We suggest a third alternative which is to estimate $p(t|d)$ on the basis of all distributions available as part of the generated LDA-model itself, namely $p(w|t)$ and $p(t)$:

$$p(t|d) \approx 1/|d| \cdot \sum_{w \in V} f_d(w) \cdot p(t|w)$$

$$p(t|w) = p(w|t) \cdot p(t)/p(w), p(w) \approx f(w)/\sum_{w \in V} f(w)$$

So $p(t|d)$ is averaged across the words in d and via the "degree to which t belongs to each w". We think this is "fair game" because it makes best use of the model and also, there is no need to create questionable holdout documents as discussed in the forum entry. For reference we call this approach "**LDA-P-Method 3**".

3.2.2 Results on Synthetic Data

Figure 6 depicts a perplexity graph for Topic Grouper regarding the data set from Section 3.1.1 and a randomized holdout set for test with 10% of the size of the entire data set (so only 90% used for training). The X axis represents the number of topics. The perplexity at $n = 4$ is about 13.998.

Figure 7 depicts a perplexity graph for LDA using LDA-P-Method 3 regarding the same data set as in Figure 6. LDA hyperparameters are set (in favour of LDA) to the best matching values, which are $n = 4$, $\alpha = (5, 0.5, 0.5, 0.5)^\top$ and $\beta = 0.5$. The X axis represents the number of iterations. The perplexity is about 15.944 after 400 iterations. (400 iterations were sufficient for convergence.)

[3] See http://mallet.cs.umass.edu.

[4] See http://stats.stackexchange.com/questions/18167/how-to-calculate-perplexity-of-a-holdout-with-latent-dirichlet-allocation.

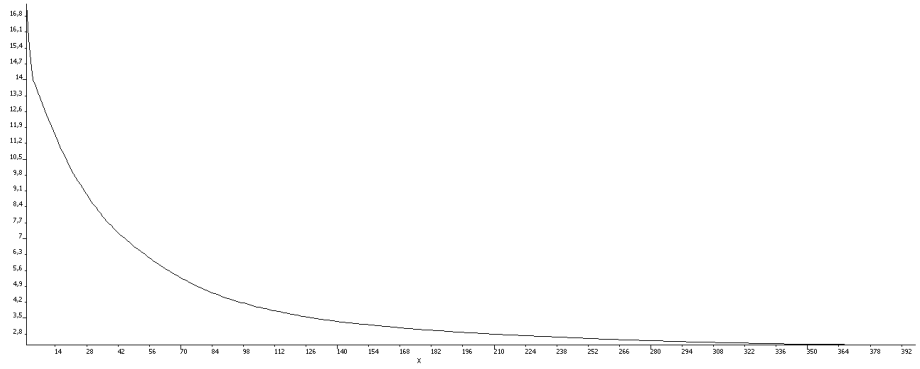

Figure 6: Perplexity as a result of Topic Grouper on the data set from Section 3.1.1 but with a 10% holdout for test (Generated with class `OptimizedTGTesterPM3`.)

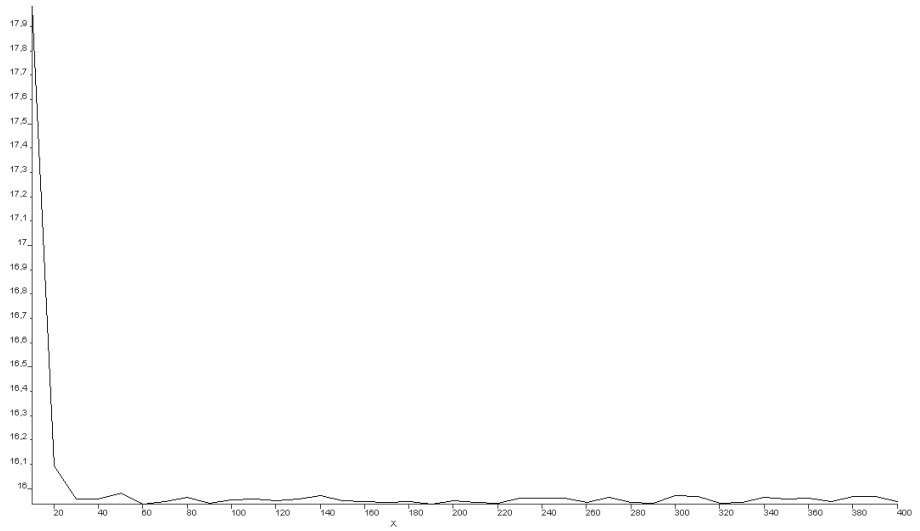

Figure 7: Perplexity as a result of LDA on the data set from Section 3.1.1 but with a 10% holdout for test (using LDA-P-Method 3) (Generated with class `AsymmetricLDAGibbsTesterPM3`.)

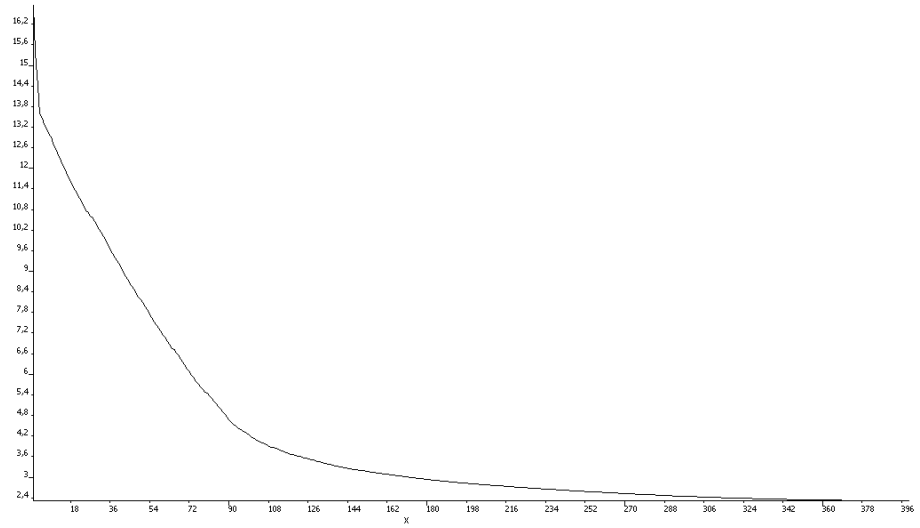

Figure 8: Perplexity as a result of Topic Grouper on the data set from Section 3.1.1 but with a 10% holdout for test. The holdout is constructed according to LDA-P-Method 2 (Generated with class `OptimizedTGTesterPM2`.)

We also tested LDA-P-Method 2 and constructed the holdout set and training set accordingly. Figures 8 and 9 show the corresponding results on the data set from Section 3.1.1 but with 300 instead of 30 words per documents instead. (We had to increase the document size so that the holdout documents end up to have a reasonable size of about 30 words.) The perplexity for Topic Grouper is about 13.589 at $n = 4$ and about 15.417 for LDA after 400 iterations.

3.2.3 Results on AP Corpus Extract

We ran Topic Grouper and LDA on an extract of the AP Corpus as available under https://www.cs.princeton.edu/ blei/lda-c/index.html. As preprocessing steps we performed stop word filtering, Porter stemming and kept only terms which occur at least ten times in the document collection, which resulted in $|V| = 5575$ and $|D| = 2946$.

Regarding LDA, we first performed a hyperparameter search for symmetric α and β values using a randomized 10% holdout according to LDA-P-Method 2. To do so we ran LDA with a fixed number of $n = 50$ topics and 300 iterations using the Gibbs sampler. We searched the area $\alpha \in [0 \dots 100]/n$, $\beta \in [0 \dots 5]$. (We first ensured that 300 iterations were sufficient by inspecting respective perplexity convergence.)

Figure 10 illustrates how the hyperparameter space was searched towards an exponentially shrinking hyperparameter area with decreasing perplexity. Figure 11 shows the corresponding perplexity at each step. It also highlights how sensitive LDA is with regard to hyperparameter selection as perplexity values vary between 225 and 80 for the given data points.

Using the optimized values for α and β, we ran LDA and Topic Grouper on the AP Corpus extract as described above but with a varying number of topics. Figure 12 shows the corresponding results: Topic Grouper outperforms

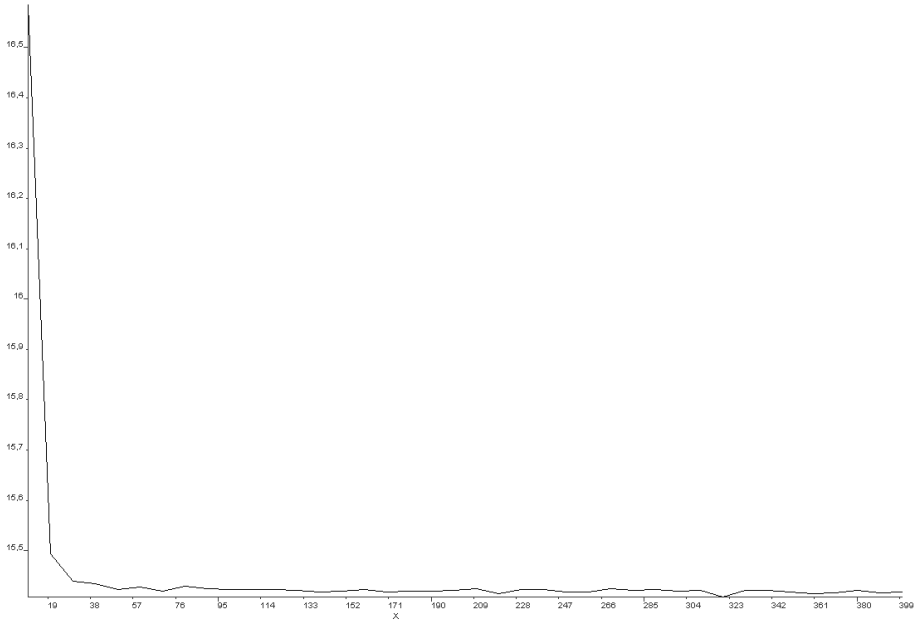

Figure 9: Perplexity as a result of LDA on the data set from Section 3.1.1 but with a 10% holdout for test (using LDA-P-Method 2). The holdout is constructed according to LDA-P-Method 2 (Generated with class `AsymmetricLDAGibbsTesterPM2`.)

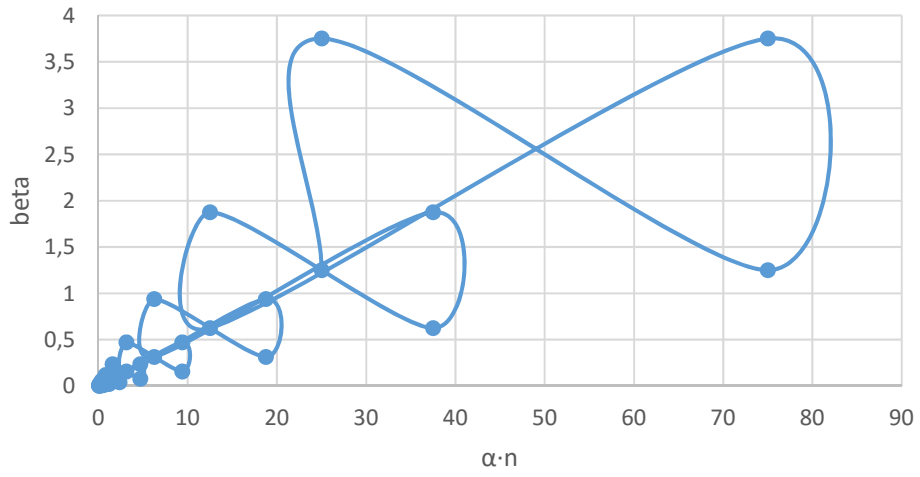

Figure 10: Hyperparameter search for α and β using LDA an the AP Corpus extract (Generated with class `APExtractAlphBetaOptGibbsTester`.)

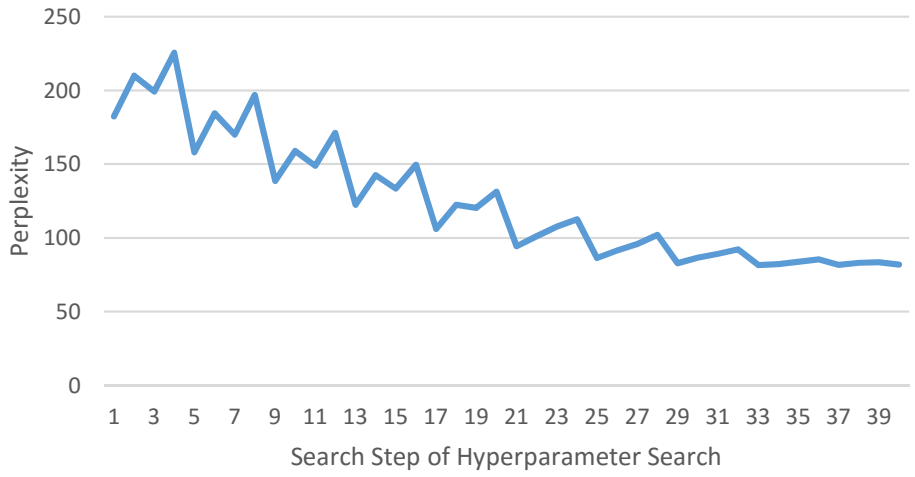

Figure 11: Perplexity values for hyperparameter search from Figure 10 (Generated with class `APExtractAlphBetaOptGibbsTester`.)

LDA slightly even after hyperparameter optimization for LDA. Please note that although LDA offers low perplexity under these conditions the resulting topics under optimized hyperparameters *are rather inconclusive*. E.g. given $n = 50$ topics, the word "said" appears 24 times in the top ten topic words under LDA (so in almost 50% of all topics). This is not the case for the topics returned from Topic Grouper. (Text: Maybe adding a figure here to illustrate this?)

Considering the next to last paragraph from Section 3.1.1 one could have the idea of using Topic Grouper as a preprocessing step for LDA in order to find reasonable hyperparameters values for α (ratios). We tried this on the AP Corpus extract but neither did it improve perplexity of LDA nor did it seem to substantially improve the quality of resulting topics.

3.3 Result Visualization and Inspection

Since Topic Grouper is essentially an agglomerative clustering approach, all respective visualization techniques become available. E.g., Figure 13 shows a corresponding dendrogram for the data set from Section 3.1.1, but with only 40 instead of 400 words (just to simplify the diagram).

As opposed to LDA, Topic Grouper returns a hierarchical topic model by design. This quality can be used to do interactive "drill downs" from larger to smaller topics assuming that larger topics form some kind semantic abstraction of contained smaller topic and also, contained topics are specializations of containing topics. To get an idea of the meaning of a topic, every topic can be represented by its top-most frequent words on every containment level. Analyzing results this way may give users additional insight into the nature of a document collection's inherent topics.

We created corresponding mind map diagrams by exporting computation results of Topic Grouper to the mind map tool FreeMind[5]. (To do so, we

[5]See http://freemind.sourceforge.net.

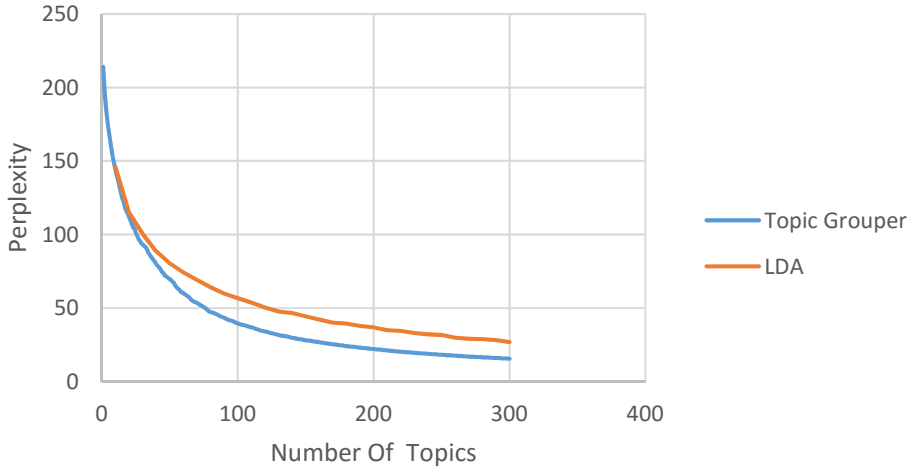

Figure 12: Perplexity values for LDA and Topic Grouper on AP corpus extract with optimized hyperparameters for LDA (Generated with classes `APExtractMultiTopicGibbsInDocHoldoutTester` and `APExtractMultiTopicTGInDocHoldoutTester`.)

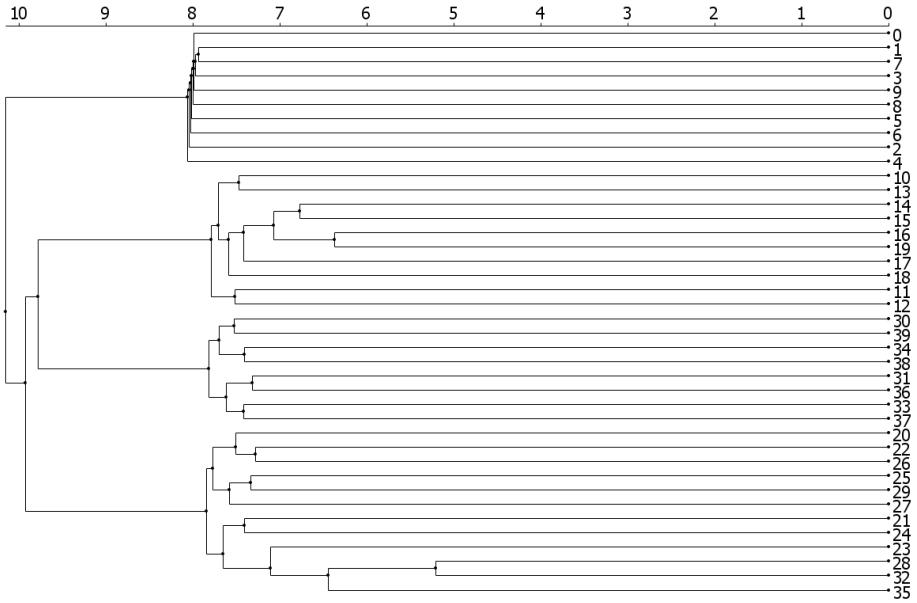

Figure 13: Dendrogram as a result of Topic Grouper on a synthetic data set according to Section 3.1.1 but with only 40 words; the X axis depicts the values of $\log -\Delta h$ per step n (Generated with class `DendroGramDemo`.)

simply generated appropriate XML files, which are FreeMind's standard file format.) More frequent word sets are shaded in blue (because they tend to be less relevant, e.g. when thinking of stop words), whereas less frequent word sets are shaded in red. Moreover, tree nodes are marked with a yellow flag if $\Delta h_{n-1}/\Delta h_n$ exceeded an adjustable threshhold (in this case set to 1.2) (see also Figure 4 for reference). The yellow flags indicate joins that lead to "unusually high cost", which is a hint that two related topics have distinct semantics.

Note that corresponding topics may occur further up or deeper down in the mind map. This makes the visualization technique rather independent of a specific fixed number of topics n to be considered at a time (as is custom with LDA). The appeal to the user is rather: "Look where the yellow flags are and see if you can make sense of the topics described via the words at the corresponding nodes." Figure 14 presents a corresponding mind map in analogy to Figure 13.

For a more realistic mind map, we ran Topic Grouper on a subset of the (supposedly well known) Reuters 21578 data set[6]. We used a subset of 13476 documents, performed stemming (using a Porter stemmer), kept stop words and removed all terms with a frequency less than 50, which resulted in $|V| = 2343$. Figure 15 shows an extract of the resulting mind map. (Many detailed nodes have been collapsed after inspection to obtain a reasonably sized image.)

3.4 Performance

We evaluated the performance of a Java implementation of Topic Grouper using the synthetic data set generator from Section 3.1.1 with a symmetric $\alpha = 50/n$ (with n being the number of generated topics). We measured the computation time of the algorithm when reducing the number of topics from $|V|$ to 1 via the following dimensions (while keeping parameters of respective other dimensions fixed):

1. number of documents $|D|$,
2. fixed document size $|d|$,
3. number of words generated per topic (this influences the size of the vocabulary $|V|$),
4. number of generated topics n (this influences the size of the vocabulary $|V|$),

We used a regular PC environment (Intel i7 processor, 2.2 GHz, 16 GB RAM, Windows 10, Java 1.6 runtime) and minimized algorithmic output to writing the above mentioned parameter values and measured computation times to a file. The task of generating the documents, i.e. word vectors, were not included in the time measurements. (This is a preprocessing task that has to be done for all probabilistic topic modeling approaches in more or less the same way.)

Figure [**?**] confirms the theoretical complexity with respect to a linear dependency on the number of documents $|D|$. Figure [**?**] confirms that computation time does not depend on the document size. Figures [**?**] and [**?**] confirm that computation time depends on $|V|^2$ – but note that the constant factor of the quadratic component is small.

[6] See http://www.daviddlewis.com/resources/testcollections/reuters21578/.

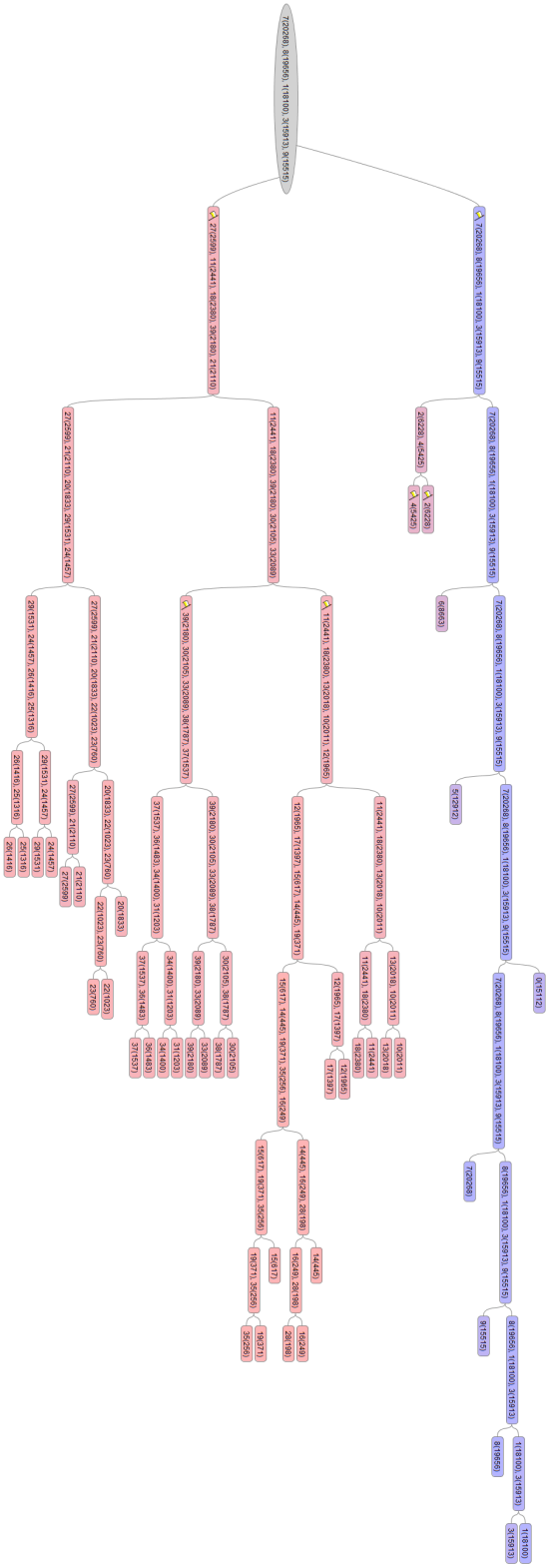

Figure 14: Mind map as a result of Topic Grouper on a synthetic data set according to Section 3.1.1 but with only 40 words (Generated with class `MindMapDemo`.)

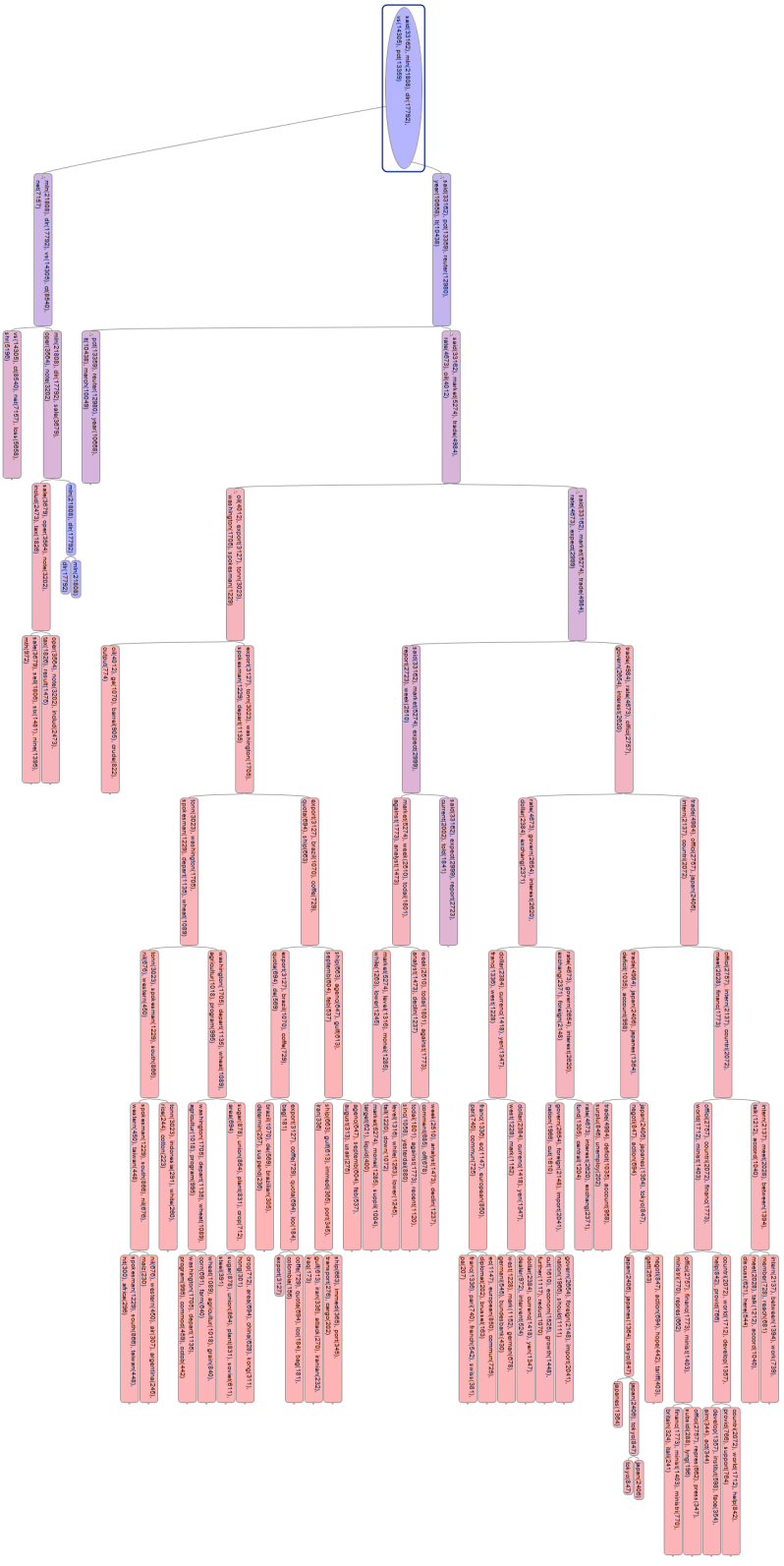

Figure 15: Mind map as a result of Topic Grouper on a subset of the Reuters 21578 data set. (Generated with class `MindMapDemoReuters`.)

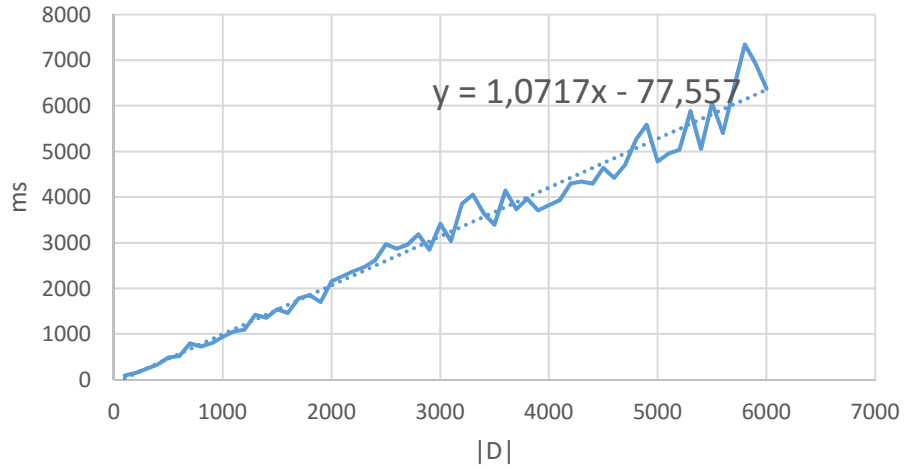

Figure 16: Topic Grouper computation time with regard to 1) from text, $|V| = 100$ ($n = 10$, 10 words per topic) and $|d| = 10000$ including a linear regression trend line (Generated with class `TGPerformanceTester`.)

Running Topic Grouper on the AP Corpus extract from Section 3.2.3 on the above described PC environment took about 0.23 hours. We also ran Topic Grouper on $|D| = 16,000$ documents of the (entire) AP Corpus with $|V| = 13383$. This took 4.39 hours and required about 850 MB of memory. Both enables to roughly estimate (non-parallelized) runtimes t in milliseconds for the Topic Grouper implementation on the given PC environment:

$$t(|V|, |D|) \approx \gamma \cdot (0.0000055 \cdot |V|^2 \cdot |D|).$$

A direct performance comparison with LDA is difficult because there exist different computational approaches for it. Regarding Gibbs sampling and for a given set of topics n, α and β the cost of one Gibbs iteration is $O(n \cdot \sum_{d \in D} |d|)$. Typically one runs at between 300 and 5000 iterations for convergence. Moreover, the cost must be multiplied with the times the algorithm is run in order to do hyperparameter optimization which in practice, often requires interleaved manual result inspection.

4 Implementation

A basic Java implementation of Topic Grouper is available under https://github.com/pfeiferd/TopicGrouperJ. Required libraries must be included by running Maven[7] on `./pom.xml`. The library `lingpipe-3.9.3.jar` cannot be downloaded from an online Maven archive. It must therefore be registered in your local maven repository using `./bin/updateRepository.bat` or `./bin/updateRepository.sh` respectively (prior to running `./pom.xml`). The libary itself is included in the project under `./lib`.

The main methods of classes in the package `org.hhn.topicgrouper.figures` can be used to generate *all* the figures of this document. (The unqualified Java

[7] See https://maven.apache.org/.

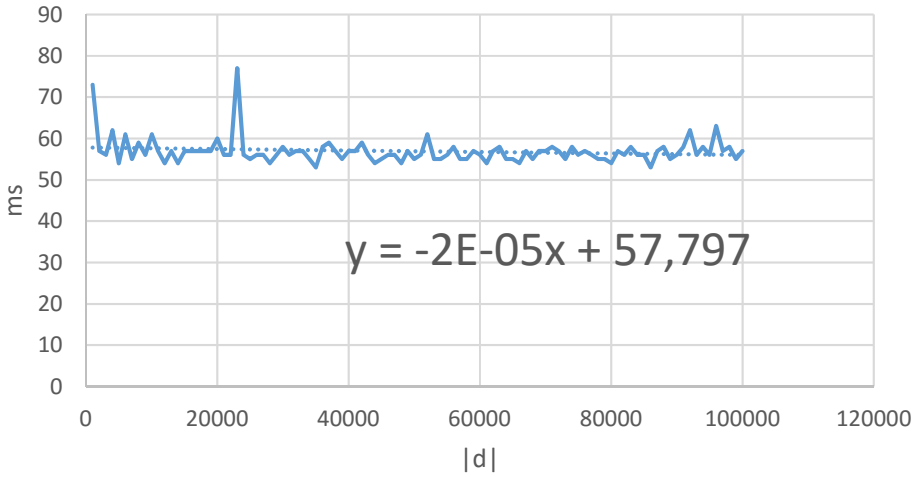

Figure 17: Topic Grouper computation time with regard to 2) from text, $|V| = 100$ ($n = 10$, 10 words per topic) and $|D| = 100$ including a linear regression trend line (Generated with class `TGPerformanceTester`.)

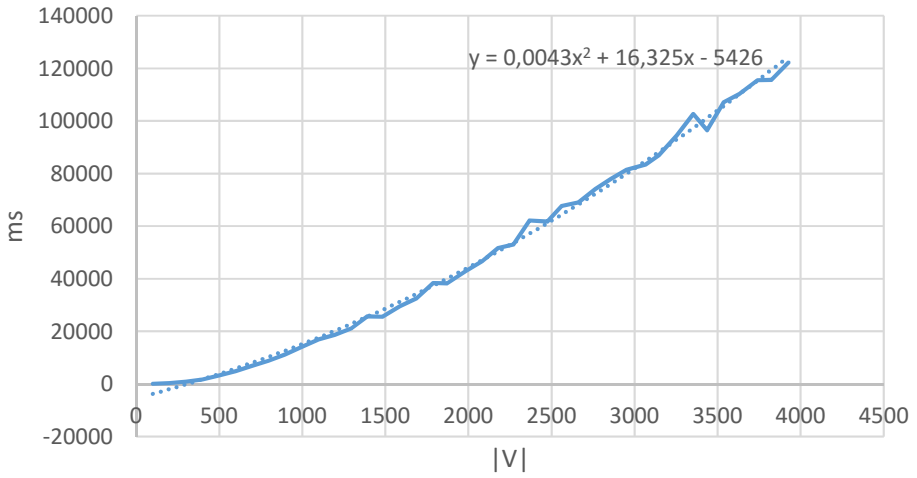

Figure 18: Topic Grouper computation time with regard to 3) from text, $n = 10$, $|d| = 1000$ and $|D| = 100$ (ranging words per topic), including a quadratic regression trend line (Generated with class `TGPerformanceTester`.)

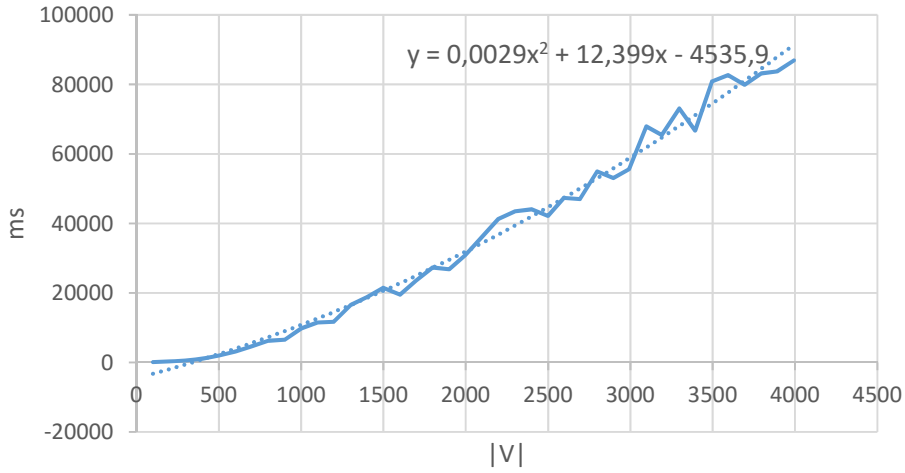

Figure 19: Topic Grouper computation time with regard to 3) from text, 10 words per topic, $|d| = 10000$ and $|D| = 100$ (ranging number of topics n) including a quadratic regression trend line (Generated with class `TGPerformanceTester`.)

name of a corresponding class is mentioned in each figure's caption.) All output files can be found in the folder `./target` (if not present, it must be created first). LDA outputs its topics in files like `./target/*.topWords`. The mind maps are generated as XML files like `./target/*.mm` and can be opened with the open source tool FreeMind for display. Other graphs and charts appear in a window while running Topic Grouper (or LDA) and are updated during the run. Textual outputs of Topic Grouper appear on the command line or in a files like `./target/*.txt`.

To run programs related to the Reuters 21578 data set, the data set files must first be downloaded and placed in the folder `./src/test/resources/reuters21578`. There is a read-me file in that folder, that explains the details.

References

[Arun et al., 2010] Arun, R., Suresh, V., Veni Madhavan, C. E., and Narasimha Murthy, M. N. (2010). *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I*, chapter On Finding the Natural Number of Topics with Latent Dirichlet Allocation: Some Observations, pages 391–402. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022.

[Chang et al., 2009] Chang, J., Boyd-Graber, J., Wang, C., Gerrish, S., and Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In *Neural Information Processing Systems*.

[Chemudugunta et al., 2006] Chemudugunta, C., Smyth, P., and Steyvers, M. (2006). Modeling general and specific aspects of documents with a probabilistic topic model. In *NIPS*, pages 241–248. MIT Press.

[Chen and Wang,] Chen, S. and Wang, Y. Latent dirichlet allocation.

[Griffiths and Steyvers, 2004] Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl. 1):5228–5235.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620.

[Tan and Ou, 2010] Tan, Y. and Ou, Z. (2010). Topic-weak-correlated latent dirichlet allocation 1.

[Teh et al., 2004] Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2004). Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101.

[Wallach, 2008] Wallach, H. M. (2008). *Structured Topic Models for Language*. PhD thesis, University of Cambridge.

[Wallach et al., 2009a] Wallach, H. M., Mimno, D. M., and McCallum, A. (2009a). Rethinking lda: Why priors matter. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *NIPS*, pages 1973–1981. Curran Associates, Inc.

[Wallach et al., 2009b] Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. (2009b). Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1105–1112, New York, NY, USA. ACM.

[Yin and Wang, 2014] Yin, J. and Wang, J. (2014). A dirichlet multinomial mixture model-based approach for short text clustering. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 233–242, New York, NY, USA. ACM.