Computational topology in text mining.

Hubert Wagner[1], Paweł Dłotko[1] and Marian Mrozek[1]

Institute of Computer Science Jagiellonian University,
`pawel.dlotko, marian.mrozek, hubert.wagner@ii.uj.edu.pl`

Abstract. In this paper we present our ongoing research on applying computational topology to analysis of structure of similarities within a collection of text documents. Our work is on the fringe between text mining and computational topology, and we describe techniques from each of these disciplines. We transform text documents to the so-called vector space model, which is often used in text mining. This representation is suitable for topological computations. We compute homology, using Discrete Morse theory, and persistent homology of the Flag complex built from the point-cloud representing the input data. Since the space is high-dimensional, many difficulties appear. We describe how we tackle these problems and point out what challenges are still to be solved.

Keywords: Computational topology, Computational homology, Flag Complex, Discrete Morse theory, Text mining, Vector space model

1 Introduction and existing work

With the growth of the Internet, efficient and accurate information-retrieval systems are of great importance. Modern search-engines are able to quickly query amounts of data counted in exabytes. Text mining aims at performing more in-depth analysis, revealing some additional knowledge from the data.

Text mining methods often use graph-theoretical approaches [10]. Analysing the connected components of the graph of *similarities* between pairs of documents is a simple example. From a topological perspective, such analysis operates on 1-dimensional *complexes* (only pairs of documents are considered) and gives 0-dimensional topological information.

In general, higher dimensional relationships, i.e. relationships between larger subsets of data, are sometimes used in data-mining. For example, the number of triangles (3-cliques) is an important descriptor of the connectivity of a social or collaborative network [6]. Rather than finding just the *number* of such higher-dimensional elements, we would like to compute their topological structure.

We believe that mining a higher dimensional *topological structure* within a set of text documents can give an important insight into the data. In general, the current state-of-the-art topological methods are incapable of handling large datasets in high dimensions, but efficient methods are being developed [14]. Still, we believe that experimenting with smaller, properly sampled data can give interesting insights. For example, [3] shows that data coming from natural images form a topological Klein bottle.

In the ongoing research, done in cooperation with Google, we use the tools of computational topology to robustly analyse and compare text data. The goal is to find meaningful topological patterns. This information can help understand the global structure of the data. In a longer perspective, this knowledge can be used in conjunction with the standard methods, improving the quality of information-retrieval systems. This is a novel direction, as is the application of computational topology in higher dimensions. In this paper we show how we adapt existing topological methods and how we tackle computational difficulties, exploiting certain properties of the data. The main question we seek to answer is whether the current computational topology algorithms are capable of efficiently handling reasonable amounts of text data.

For an introduction to computational topology see [5]. A paper by Carlson [3] is an important work, which shows that analysis of higher-dimensional data can be meaningful. A number of papers dealing with lower-dimensional spaces exist, but these techniques are hard or impossible to generalize to higher dimensions [11]. A recent paper by Zomorodian [14] deals with building Rips complexes of high dimensional data, which is also part of our computations. A PhD thesis of Lewiner [9] describes the usage of Discrete Morse theory to compute homology groups.

2 Background

2.1 Vector space model

We start with describing a way to map textual data into a representation which allows us to use topological tools. Vector space model is a standard tool in information retrieval and data mining [12]. A *corpus*, i.e. collection of text documents, is mapped into points (or vectors) in $\mathbb{R}^n$. These vectors are the so-called *term-vectors* and each of them represents a single document, as described below. Each dimension in this space corresponds to a single word (or *term*).

With each document in a corpus, we associate a term-vector [12], containing words characteristic of this document. In practice from 10 to 50 words are extracted. While term-vectors do not fully describe the documents, they roughly encapsulate the *topic*. Each term t contained in some document d in corpus D is weighted according to the standard *tf-idf* [12] technique: $w(d,t) = tf(d,t) \cdot idf(t)$, where $tf(d,t)$ is the number of occurrences of word t in document d, and $idf(t) = log\frac{|D|}{|\{d:t\in d\}|}$. Thus, more frequent words in a document are weighted higher but this is offset by the *global* popularity of a given term. By P we denote the array of term-vectors representing all the documents of the corpus D. Each term-vector is associated with a unique integer, which is the *index* of that term-vector in P.

In terms of implementation, each term in the corpus can also be assigned a unique number, which represents the term. This is more efficient than storing multiple copies of the string representations of the terms. Term-vector $d \in P$ is compactly stored as a sparse vector: we explicitly represent only the coordinates

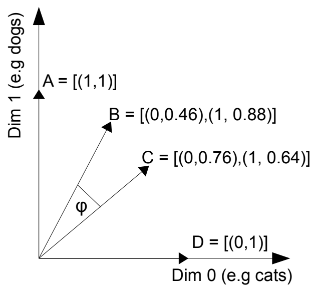

Fig. 1. Example of the vector space model. A two-dimensional space is shown, which means that only two different words are extracted from all documents. The similarity between vectors B, C equals $cos(\varphi) = 0.46 \cdot 0.76 + 0.88 \cdot 0.64 = 0.91$.

with non-zero weights. The actual data-structure representing term-vector d is simply an array of pairs (index of t, $w(d,t)$). See Figure 1 for a simple example. Note that, for brevity, we often identify a document with its term-vector.

Rather than using the Euclidean metric on this space, we use the so-called *cosine similarity measure*. This is a natural choice, as this measure is a standard text mining tool used to compare documents. The similarity between two documents (represented by term-vectors a, b), is given by $sim(a,b) := cos(\angle(a,b)) = \frac{\langle a,b \rangle}{||a||||b||}$. This formula requires computing square roots, which is costly. We will store normalized (according to Euclidean norm) term-vectors and equivalently compute similarity as:

$$sim(a,b) = \langle a,b \rangle$$

Cosine similarity quantifies the closeness of topics of two documents [12]. The values range from 0 (completely unrelated topics) to 1 (identical topic). Note that the constructed space (equipped with the cosine similarity measure) is not a metric space. Later we will use a weight function $d(a,b) := 1 - sim(a,b)$, which is also not a metric, as the triangle inequality is not satisfied.

We have to distinguish between extrinsic (embedding) and intrinsic dimension of the space. In this case, the extrinsic dimension, R, is large, equal to the number of unique words in the dataset, which can reach tens of thousands to several millions in practical applications. It is typically assumed that the intrinsic dimension is significantly lower, which prevents the *curse of dimensionality* from making the computations infeasible. Another important property of data coming from real-world text corpora is that the frequency of word occurrences follow the Zipf distribution [13]. It assumes that the frequency of an r-th most common word is expressed as: $P(r) = \frac{R}{rln(cR)}$, where c is some constant which depends on the corpus. This distribution is far from uniform – intuitively, the most common words apear much more often than the others.

Also, note that due to very high extrinsic dimensionality, the space is very sparse (empty) in practice. Another observation is that the number of keywords extracted from each document is relatively small. In practice, this number is chosen between 10-50. So, for each term-vector, the number of nonzero coordinates is small, compared to the number of zero coordinates. Therefore, the similarity between two randomly chosen term-vectors should be zero most of the time, since the support of these vectors is disjoint. These facts suggest that this space behaves differently than the Euclidean space, where the distance between any two points is finite (intuitively, zero similarity corresponds to infinite distance). In practice, this effect is offset by the Zipf distribution – more popular words increase the number of pairs of documents with nonzero similarity.

The described properties of the data are important, as they reduce the number of large cliques appearing during computations. This makes topological computations based on flag complexes, as described in the following section, more feasible.

2.2 Computational topology

First, we would like to outline the computations we perform. We are interested in computing *homology* and *persistent homology* of the space describing similarities between the documents in a corpus. Representing the textual data in the vector space model yields a point-cloud, allowing us to use topological tools. Starting from the point-cloud we will construct a *simplicial complex* called a *flag complex*, which encodes higher dimensional topological information, and can be viewed as a higher-dimensional analog of a graph. Since the complex can be large, we simplify it, using Discrete Morse theory. This step retains the topological information. Finally, we compute homology on the reduced complex.

A finite collection of sets, S, is an abstract *simplicial complex* if for every $t \in S$ and for every $s \subset t$ we have $s \in S$. Every element $t \in S$ is a *simplex* and its *dimension* is defined as $card(t) - 1$. By S_k we denote the k-skeleton of complex S, i.e. all simplices in S with dimension $\leq k$. If $p \subset q$ and $card(q) - card(p) = 1$, we say that p is a *face* of q and q is a co-face of p. *(Co-)boundary* is the set of all (co-)faces of a simplex. A simplex of dimension 0,1,2 is respectively: *a vertex, an edge* and *a triangle.*

An *ϵ-graph* imposed by the similarity measure *sim* on the collection of term-vectors P is defined as $G = (P, E)$, where $E = \{(a, b) \in P \times P \mid 1 - sim(a, b) \leq \epsilon\}$. In other words, edges connect pairs of documents with similarity above certain threshold. In general, for graph $G = (V, E)$, a subset $V' \subset V$ is a *clique* if for every $v_1, v_2 \in V'$, $(v_1, v_2) \in E$. *Flag complex* of graph G is defined as: $S(G) := \{V' \subset V \mid V' \text{ is a clique in } G\}$. In other words, the flag complex of graph G is the maximal simplicial complex having G as its 1-skeleton.

3 Construction of flag complex

The flag complex, as well as the so-called *Vietoris-Rips* complex (see [5]), is a standard tool used to perform topological data analysis [3]. In this section we describe an efficient bottom-up technique to obtain flag complexes. The presented technique avoids the usage of associative data structures, which incur a significant performance penalty. We designed the code to use only vectors (dynamically growing arrays as in the C++ Standard Library) which are fast due to good caching properties.

The complex building phase is similar to the construction of Vietoris-Rips complex presented in [14]. Since in Section 4 we are focused on computing Morse complexes, we require fast access to the (co-)boundary of each simplex, which is not included in the cited paper. We use the name *flag complex* instead of *Vietoris-Rips complex*, as the latter assumes a metric function.

The input to the algorithms presented in this Section is the array P together with the similarity function *sim*. Let us first describe the data structure we use to store simplices. Each simplex s has a vector `vertices` storing the 0-dimensional simplices (which correspond to indices of term-vectors) that belong to s. Moreover, it has a vector `boundary`, containing the faces of s.

During the construction we also use vectors `coboundary` and `neigh`. Vector s.`neigh` contains the vertices adjacent to *all* vertices in s, with the additional property that for each $i \in s$.`neigh` $i > max\{s.\texttt{vertices}\}$. We assume the the entire simplex is *created by* its maximum vertex. Importantly, we exploit this property in the algorithms to ensure that each simplex is created only once (when the maximal vertex is processed). `SimplicialComplex`, stores a vector of pointers to simplices separately for each dimension. Algorithm 1 shows how we build the 1-skeleton of the constructed flag complex.

Algorithm 1 CreateOneSkeleton

```
Input: array $P$ of term-vector, double $\epsilon$
 1: verts = array of simplex*;
 2: for i = 0 to $P$.size do
 3:     verts[i] = new simplex();
 4:     verts[i].vertices = i;
 5: for i = 0 to $P$.size do
 6:     verts[i].neigh = ComputeNeigh(i , $P$ , $\epsilon$);
 7: SimplicialComplex[0] = verts;
 8: edges = array of simplex*;
 9: for for i = 0 to vert.size do
10:     for j = 0 to vert[i].neigh.size do
11:         simplex* edge = new simplex;
12:         edge.boundary = (vert[i], vert[i].neigh[j]);
13:         vert[i].coBoundary.add(edge), vert[i].neigh[j].coBoundary.add(edge);
14:         edge.vertices = (vert[i], vert[i].neigh[j]);
15:         edge.neigh = vert[i].neigh $\cap$ vert[i].neigh[j].neigh;
16:         edges.add(edge);
17: SimplicialComplex[1] = edges;
```

$ComputeNeigh$ algorithm computes the ϵ-neighborhood of a given vertex with a constraint that it returns only vertices with indices higher than the index of the considered vertex. For the time being, we assume that it just iterates through all the vertices, computes the similarity and rejects the vertices corresponding to documents with similarity below the threshold. This makes the complexity of the entire Algorithm 1 quadratic. Since, in practice, the output graph is sparse, the complexity can potentially be reduced. We are investigating methods of computing ϵ-graphs, such as cover trees [2], which are more suitable for this type of data. Some efficient techniques for metric spaces are reviewed in [14].

Once the 1-skeleton of the complex is created, we proceed with the creation of higher dimensional simplexes, as described in Algorithm 2. In terms of computational complexity, the entire constructed flag complex can be exponential in the number of vertices of the input. This is related to the fact, that the total number of cliques is pessimistically exponential. In practice, we are interested in computing the complex only up to a certain, small dimension, which yields polynomial worst-case complexity. The actual performance is heavily dependent on the data.

Algorithm 2 CreateHigherDimensionalSimplices

```
Input: array initial of simplex*, int dim
 1: new_elements = array of simplex*;
 2: for i = 0 to initial.size do
 3:   for j = 0 to initial[i].neigh.size do
 4:     simplex* new_simplex = new simplex();
 5:     new_simplex.neigh = initial[i].neigh $\cap$ initial[i].neigh[j].neigh;
 6:     new_simplex.vertices = initial[i].vertices $\cup$ initial[i].neigh[j]
 7:     initial[i].coboundary.add(new_simplex);
 8:     new_simplex.boundary.add(initial[i]);
 9:     for each bd $\in$ initial[i].boundary do
10:       for each cbd $\in$ bd.coboundary do
11:         if cbd $\neq$ initial[i] and initial[i].neigh[j] $\in$ cbd.vertices then
12:           cbd.coboundary.add(new_simplex);
13:           new_simplex.boundary.add(cbd);
14:     new_elements.add(new_simplex);
15: SimplicialComplex[dim] = new_elements;
```

4 Morse-Flag complexes

In this section we show how to use Discrete Morse theory [7] to compress the Flag complex during its construction. Iterating Morse complex computations (see Algorithm 3) yields the absolute $\mathbb{Z}_2$ homology of the considered complex (see Theorem 1). Exploiting this property, we do not have to generate boundary matrices required for algebraic computations, which tend to be costly in terms memory and time. Note that using $\mathbb{Z}_2$, rather than $\mathbb{Z}$, coefficients simplifies the computations, but prevents us from capturing the so-called *torsion* (see [5]) in homology groups. As in the algorithm described in Section 3, during the construction in dimension n, we need to store $n-1$ and $n-2$ dimensional elements, which enables us to reduce memory usage.

Theorem 1. *Let S be the input complex. We build a Morse complex of S and iterate Morse construction, as long as some Morse pairing exist. Let $|S_n|$ denote the number of n-cells in the final Morse complex. Then dim $H_n(S, \mathbb{Z}_2) = \beta_n(S) = |S_n|$.*

Proof. Since, in general, the Morse complex obtained in the described construction is not a simplicial complex, we use a more general algebraic Morse theory (see Section 11.3, [8]). Compared to the setting of Forman [7], the important difference is that during the matching between cells a and b, where a is a face of b, we want the incidence coefficient ($\kappa(a, b)$, see [8]) to be invertible. However, in case of field coefficients, for a being a face of b we have $\kappa(a, b) \neq 0$, so it is invertible in the field. We can instantly read the Betti numbers if every cell has empty boundary and coboudary (this is a consequence of the definition of homology groups).

Now by contradiction, let us assume that $\beta_n(S) \neq |S_n|$ and there are no more Morse pairings to be made. Therefore, from Morse inequalities, we have that $\beta_n(S) > |S_n|$. If all elements from S_n have empty both boundary and coboundary, then from the previous paragraph we get the contradiction, since $\beta_n(S) > |S_n| = \beta_n(S)$. Therefore some elements form S_n need to have nonempty boundary or coboundary. But then a Morse pairing can be made between these elements, which leads to a contradiction.

□

In Algorithm 3 we describe the compression of the initial complex to a Morse complex. The procedure $doMorsePairings(C, d)$ performs Morse pairings in a complex C under a constraint that the dimension of every element in a pairing is $\leq d$. The procedure $computeMorseComplex(C, V)$ computes the boundary coefficients between critical cells (see [7, 8] for the theory and also [9] for algorithmic details) and removes from C all the non-critical elements (i.e. elements from V). The procedure stops execution when there are no more pairings to be done in the complex. We want to point out that only the $d-2$-dimensional skeleton of the constructed complex is modified, because we need the simplices in dimension $d-1$ and d in order to build the higher dimensional skeleton. This algorithm should be called for each step of construction in Algorithm 2.

Algorithm 3 IteratedComputationOfMorseComplex

```
Input: C - initial complex
Output: C - reduced Morse complex
1: $dim$ = dimension of C;
2: while true do
3:     $n = size(C)$;
4:     List of Morse pairings $V$ = doMorsePairings(C , $dim - 2$);
5:     computeMorseComplex( C,V );
6:     if $size(C) == n$ then
7:         return C;
```

5 Experiments

We have developed and tested a C++ implementation which includes the algorithms outlined above. To compute homology and persistent homology, we use the standard *matrix reduction* method [5], with the *twist* by Chen and Kerber [4]. For experiments we sample the corpus of the English Wikipedia [16], processed using Python library gensim [17].

There are two main parameters of our software. Parameter *dim* controls the maximum dimension of the constructed complex. As a result, homology is computed up to dimension *dim*-1. The second parameter is $\epsilon \in [0, 1]$, which means that only edges (a, b) with $sim(a, b) \geq 1 - \epsilon$ are included in the skeleton. These parameters reduce the amounts of computations.

While in the worst case computing persistent homology takes cubic time, the reduction algorithm is typically assumed to take linear time in practical situations [4]. Judging from our experiments, the behavior is definitely super-linear, probably roughly quadratic (in the size of the complex) for dimensions ≥ 3 (see Figure 2:left). For dimension < 2 the time required to build the complex dominated over the persistence computations. Therefore, for higher dimensions the reduction algorithm is clearly the computational bottleneck. Additionally the number of cells grows super-linearly in the input size, but it is strongly dependent on the chosen ϵ.

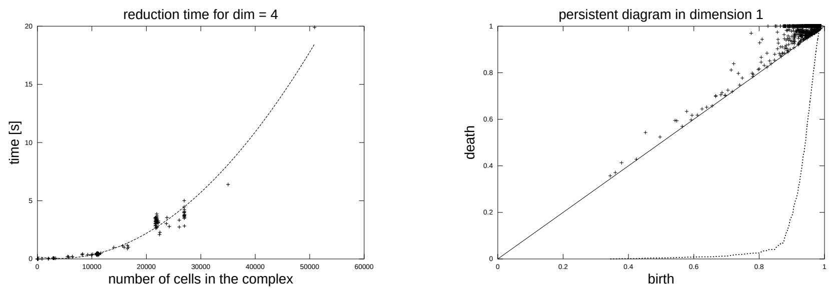

Fig. 2. Left: Runtime (in seconds) of the reduction algorithm for dimension 4. The behaviour appears to be quadratic, which is emphasised by fitting a degree two polynomial. **Right**: Persistence diagram for a complex containing 0.8M cells (we can assume 1 for infinity, since we know that at this point the skeleton would become a complete graph with all the cliques present). The curve in the lower part of the diagram represents a normalized cumulative sum of persistence values. It helps visualize the region of values for which features of relatively large persistence appear.

The observed quadratic behavior is important in the context of finding efficient methods of computing persistent homology. Recent research suggest that simplifying the input complex using discrete Morse theory can increase efficiency.

In practice, the significant advantage is in terms of memory usage [15]. Morse simplification, which exhibits roughly linear behavior, in most cases took more time than computing persistence of the original, unreduced, complex.

The observed quadratic behaviour is a good motivation for further development of algorithms based on discrete Morse theory. In case of textual data, such methods could help increase both the memory and time efficiency of persistence computations. Currently published methods are limited to dimension ≤ 3, but we are aware of ongoing research promising methods working in general dimension.

Our attempt to compute standard homology using discrete Morse theory was not very effective as the reduction factor was only about $10 - 20\%$. This is surprising, since the homology appears to be simple, which would suggest a large reduction factor. We plan to investigate this issue further.

As shown in Figure 2:right, the 1-dimensional topology is quite uninteresting until the filtration value around 0.8. It means that only after introducing edges with similarity ≤ 0.2, do many features of larger persistence start to appear. On the other hand, we measured that the highest-dimensional cells are the most abundant in the complex. It appears that cells rarely cluster to create homological features of non-zero persistence. Note that in our setting a boundary of a single simplex either remains a cycle 'forever' (which actually means it is killed at value 1, so persistence equals $1-$birth) or is filled instantly (zero persistence). Experiments in higher dimensions (but for much smaller datasets) back up this statement.

These observations suggest that features of non-zero persistence capture, let us call it, semi-similar sets of documents. By that we mean sets of documents which are similar enough to create a, say, p-dimensional cycle, but there are no additional documents similar enough to fill this cycle (for a given similarity threshold). Consequently, persistence can be viewed as the measure of discrepancy between the inter-similarity of a set of documents, and and its certain superset.

Analysis of the 1-dimensional persistence suggests two explanations of the low number of features of non-zero persistence. 1) The similarities are very strong, many large cliques appear and most lower-dimensional features have zero persistence. 2) The similarities are strong only locally - almost all appearing cycles are boundaries of a single simplex, so their persistence is 0 (if they are killed). To verify this hypothesis experiments in higher (> 4) dimensions should be run. The behaviour of persistence in higher dimensions might also be different, which makes it interesting to check.

6 Summary

The main purpose of this paper is to challenge the current computational topology tools with large text datasets represented within the vector-space model. The experimental results show that these methods lack efficiency. Specifically, the algorithm for persistent homology exhibits quadratic behaviour in the size of the constructed complex, which prevents our approach from scaling for realistic amounts of data. Interestingly, to the best of our knowledge, this is the first dataset exhibiting quadratic scaling, which comes from an application.

The experiments we were able to conduct did shed some light on the lower-dimensional topological structure of the dataset. In our future research we would like to answer some of the questions posed in the previous section as well as try to verify the efficiency of different computational methods.

Acknowledgments

All authors are supported by Google Research Awards programme. H.W. is also supported by a Foundation for Polish Science IPP Programme "Geometry and Topology in Physical Models". Both M.M. and P.D. are partially supported by Polish MNSzW, Grant N N201 419639, P.D. also by Grant Nr IP 2010 046370.

References

1. R. Baeza-Yates, B. Ribeiro-Neto, *Modern information retrieval*, Reading, MA: Addison-Wesley Longman. p. 192 (1999).
2. A. Beygelzimer, S. Kakade, J. Langford, *Cover trees for nearest neighbor*, Proc. of ICML '06 (2006).
3. G. Carlson, *Topology and Data*, Bulletin of the AMS, 46(2), pp. 255308 (2009).
4. C. Chen, M. Kerber, *Persistent homology computation with a twist*, 27th European Workshop on Computational Geometry (EuroCG 2011) (2011).
5. H. Edelsbrunner, J. L. Harer, *Computational Topology. An Introduction.* Amer. Math. Soc., Providence, Rhode Island (2010).
6. A-X. Feng, C-H. Fu, X-L. Xu, A-F. Liu, H. Chang, D-R. He, G-L. Feng, *An Empirical Investigation on Important Subgraphs in Cooperation-Competition networks*, Science (2011).
7. R. Forman, *A User's Guide To Discrete Morse Theory*, Sminaire Lotharingien de Combinatoire, Vol. B48c, pp. 1-35 (2002).
8. D. Kozlov, *Combinatorial Algebraic Topology*, Springer 2007.
9. T. Lewiner, *Geometric discrete Morse complexes*, PhD Thesis (2005)
10. X. Polanco and E.S. Juan, *Text Data Network Analysis Using Graph Approach*, Proc. of InSciT, pp. 586-592 (2006).
11. V. Robins, P. J. Wood, A. P. Sheppard, *Theory and Algorithms for Constructing Discrete Morse Complexes from Grayscale Digital Images*, IEEE Trans. Pattern Anal. Mach. Intell. 33, pp. 1646-1658 (2011).
12. G. Salton, A. Wong, C.S. Yang, *A vector space model for automatic indexing*, Commun. ACM 18(11), pp. 613–620 (1975).
13. G.K. Zipf, *Human Behavior and the Principle of Least Effort*, Cambridge, MA: Addison-Wesley (1949).
14. A. Zomorodian, *Fast construction of the Vietoris-Rips complex*, Computers & Graphics, Vol. 34, Nr. 3, pp. 263-271 (2010).
15. D. Günther, J. Reininghaus, H. Wagner, I. Hotz, *Memory Efficient Computation of Persistent Homology for 3D Image Data using Discrete Morse Theory*, Sibgrapi 2011, Maceio, Brazil (2011).
16. *English Wikipedia corpus http://dumps.wikimedia.org/enwiki/.*
17. *Gensim Library http://radimrehurek.com/gensim/.*