Clustering the Google Distance with Eigenvectors and Semidefinite Programming

Jan Poland and Thomas Zeugmann
Division of Computer Science
Hokkaido University
N-14, W-9, Sapporo 060-0814, Japan
{jan,thomas}@ist.hokudai.ac.jp

Abstract

Web mining techniques are becoming increasingly popular and more accurate, as the information body of the World Wide Web grows and reflects a more and more comprehensive picture of the humans' view of the world. One simple web mining tool is called the *Google distance* and has been recently suggested by Cilibrasi and Vitányi. It is an information distance between two terms in natural language, and can be derived from the "similarity metric", which is defined in the context of Kolmogorov complexity. The Google distance can be calculated from just counting how often the terms occur in the web (page counts), e.g. using the Google search engine. In this work, we compare two clustering methods for quickly and fully automatically decomposing a list of terms into semantically related groups: Spectral clustering and clustering by semidefinite programming.

1. Introduction

The Google distance has been suggested by Cilibrasi and Vitányi [2] as a semantical distance function on pairs of words or terms. For instance, for most of today's people, the terms "Claude Debussy" and "Béla Bartók" are much tighter related than "Béla Bartók" and "Michael Schumacher".

The World Wide Web represents parts of the world we live in as a huge collection of documents, mostly written in natural language. By just counting the relative frequency of a term or a tuple of terms, we may obtain a *probability* of this term or tuple. From there, one may define conditional probabilities and, by taking logarithms, complexities. Li et al. [7] have proposed a distance function based on Kolmogorov complexity, which can be used for other complexities, such as those derived from the WWW frequencies.

Spectral clustering is an increasingly popular method for analyzing and clustering data by using only the matrix of pairwise similarities. It was invented more than 30 years ago for partitioning graphs (see e.g. [11] for a brief history). Formally, spectral clustering can be related to approximating the normalized min-cut of the graph defined by the adjacency matrix of pairwise similarities [15]. Finding the exactly minimizing cut is an NP-hard problem.

The Google distances can be transformed to similarities by means of a suitable kernel. However, as such a transformation potentially introduces errors, since in particular the kernel has to be chosen appropriately and the clustering is quite sensitive to this choice, it seems natural to work directly on the distances. Then, the emerging graph-theoretical criterion is that of a *maximum cut.* Optimizing this cut is again NP-hard, but can be approximated with *semidefinite programming* (SDP).

The main aim of this work is to compare spectral clustering and clustering by SDP, both of which are much faster than the computationally expensive phylogenetic trees used by [2]. We will show how state-of-the-art techniques can be combined in order to achieve quite accurate clustering of natural language terms with surprisingly little effort.

There is a huge amount of related work, in computer science, in linguistics, as well as in other fields. Text mining with spectral methods has been for instance studied in [3]. A variety of statistical similarity measures for natural language terms has been listed in [13]. For literature on spectral clustering, see the References section and the references in the cited papers.

The paper is structured as follows. In the next section, we introduce the similarity metric, the Google distance, and spectral clustering as well as clustering by SDP, and we shall state our algorithms. Section 3 describes the experiments and their results. Finally, in Section 4 we discuss the results obtained and give conclusions.

2. Theory

2.1. Similarity Metric and Google Distance

We start with a brief introduction to Kolmogorov complexity (see [8] for a much deeper introduction). Let us fix a universal Turing machine (which one we fix is not relevant, since each universal machine can interpret each other by using a "compiler" program of constant length). For concreteness, we assume that its program tape is binary, such that all subsequent logarithms referring to program lengths are w.r.t. the base 2. The output alphabet is ASCII or UTF-8, according to which character set we are actually using. (For simplicity, we shall always use the term ASCII in the following, which is to be replaced by UTF-8 if necessary.) Then, the (prefix) Kolmogorov complexity of a character string x is defined as

$$K(x) = \text{length of the shortest self-delimiting program generating } x.$$

By the requirement "self-delimiting" we ensure that the programs form a prefix-free set and therefore the Kraft inequality holds, i.e.,

$$\sum_x 2^{-K(x)} \leq 1 \ ,$$

where x ranges over all ASCII strings.

The Kolmogorov complexity is a well-defined quantity regardless of the choice of the universal Turing machine, up to an additive constant.

If x and y are ASCII strings and x^* and y^* are their shortest (binary) programs, respectively, we can define $K(y|x^*)$, which is the length of the shortest self-delimiting program generating y where x^*, the program for x, is given. $K(x|y^*)$ is computed analogously. Thus, we may follow [7] and define the *universal similarity metric* as

$$d(x,y) = \frac{\max\big\{K(y|x^*), K(x|y^*)\big\}}{\max\big\{K(x), K(y)\big\}} \quad (1)$$

We can interprete $d(x,y)$ as an approximation of the ratio by which the complexity of the more complex string decreases, if we already know how to generate the less complex string. The universal similarity metric is almost a metric according to the usual definition, as it satisfies the metric (in)equalities up to order $1/\max\big\{K(x), K(y)\big\}$.

Given a collection of documents like the World Wide Web, we define the probability of a term or a tuple of terms by counting relative frequencies. That is, for a tuple of terms $X = (x_1, x_2, \ldots, x_n)$, where each term x_i is an ASCII string, we set

$$p^{\text{www}}(X) = p^{\text{www}}(x_1, x_2, \ldots, x_n) = \tag{2}$$

$$\frac{\#\text{ web pages containing all } x_1, x_2, \ldots, x_n}{\#\text{ relevant web pages}}.$$

Conditional probabilities can be defined likewise as

$$p^{\text{www}}(Y|X) = p^{\text{www}}(Y \cup X)/p^{\text{www}}(X) \ ,$$

where X and Y are tuples of terms and $\cup$ denotes the concatenation. Although the probabilities defined in this way do not satisfy the Kraft inequality, we may still define complexities

$$K^{\text{www}}(X) \ = \ -\log\big(p^{\text{www}}(X)\big) \text{ and}$$

$$K^{\text{www}}(Y|X) \ = \ K^{\text{www}}(Y \cup X) - K^{\text{www}}(X) \ .$$

Then we use (1) in order to define the *web distance* of two ASCII strings x and y, following [2], as

$$d^{\text{www}}(x, y) = \frac{K^{\text{www}}(x \cup y) - \min\big\{K^{\text{www}}(x), K^{\text{www}}(y)\big\}}{\max\big\{K^{\text{www}}(x), K^{\text{www}}(y)\big\}} \tag{3}$$

Since we use Google to query the page counts of the pages, we also call d^{www} the Google distance. Since the Kraft inequality does not hold, the Google distance is quite far from being a metric, unlike the universal similarity metric above.

A remark concerning the "number of relevant web pages" in (2) is mandatory here. This could be basically the number of all pages indexed by Google. But this quantity is not appropriate for two reasons: First, since some months ago there seems to be no way to directly query this number. Hence, the implementation by [2] used a heuristic to estimate this value, which however yields inaccurate results. Second, not all web pages are really relevant for our search. For example, billions of Chinese web pages are irrelevant if we are interested in the similarity of "cosine" and "triangle." They would be relevant if we were searching for the corresponding Chinese terms. So we use a different way to fix the relevant database size. We add the search term "the" to each query, if we are dealing with English language. This is one of the most frequent words used in English and therefore gives a reasonable restriction of the database. The database size is then the number of occurrences of "the" in the Google index. Similarly, for our experiments with Japanese, we add the term の ("no" in hiragana) to all queries.

2.2. Spectral Clustering

Consider the block diagonal matrix

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Its top two eigenvectors, i.e., the eigenvectors associated with the largest two eigenvalues, are $[1\ 1\ 0]^T$ and $[0\ 0\ 1]^T$. That is, they separate the two perfect clusters represented by this similarity matrix. In general, there are conditions under which the top k eigenvectors of the similarity matrix or its Laplacian result in a good clustering, even if the similarity matrix is not perfectly block diagonal [9, 14]. In particular, it was observed in [4] that the transformed data points given by the k top eigenvectors tend to be aligned to lines in the k-dimensional Euclidean space, therefore the kLines algorithm is appropriate for clustering. In order to get a complete clustering algorithm, we therefore only need to fix a suitable kernel function in order to proceed from a distance matrix to a similarity matrix.

For the kernel, we use the Gaussian $k(x, y) = \exp\big(-\frac{1}{2}d(x, y)^2/\sigma^2\big)$. We may use a globally fixed kernel width σ, since the Google distance (3) is scale invariant. In the experiments, we compute the mean value of the entries of the distance matrix D and then set $\sigma = mean(D)/\sqrt{2}$.

In this way, the kernel is most sensitive around $mean(D)$.

The final spectral clustering algorithm for a known number of clusters k is stated below. We shall discuss in the experimental section how to estimate k if the number of clusters is not known in advance.

Algorithm Spectral clustering of a word list
Input: word list $X = (x_1, x_2, \dots, x_n)$, number of clusters k
Output: clustering $c \in \{1 \dots k\}^n$

```
1. for $x, y \in X$, compute Google relative frequencies $p^{\text{www}}(x)$, $p^{\text{www}}(x,y)$
2. for $x, y \in X$, compute complexities $K^{\text{www}}(x)$, $K^{\text{www}}(x,y)$
3. compute distance matrix
       $D = \left(d^{\text{www}}(x,y)\right)_{x,y \in X}$
4. compute $\sigma = mean(D)/\sqrt{2}$
5. compute similarity matrix
       $A = \left(\exp(-\frac{1}{2} d^{\text{www}}(x,y)^2/\sigma^2)\right)$
6. compute Laplacian $L = S^{-\frac{1}{2}} A S^{-\frac{1}{2}}$, where $S_{ii} = \sum_j A_{ij}$ and $S_{ij} = 0$ for $i \neq j$
7. compute top $k$ eigenvectors $V \in \mathbb{R}^{n \times k}$
8. cluster $V$ using kLines [4]
```

2.3. Semidefinite programming

Given a weighted graph $G = (V, D)$ with vertices $V = \{x_1, \dots, x_n\}$ and edge weights $D = \{d_{ij} \geq 0 \mid 1 \leq i, j \leq n\}$ which express pairwise distances, a *k-way-cut* is a partition of V into k disjoint subsets $S_1, \dots, S_k$. Here k is assumed to be given. We define the predicate $A(i,j) = 0$ if $\exists \ell [1 \leq \ell \leq k,\ 1 \leq i, j \leq n$ and $i, j \in S_\ell]$ and $A(i,j) = 1$, otherwise. The weight of the cut $(S_1, \dots, S_k)$ is defined as

$$\sum_{i,j=1}^{n} d_{ij} A(i,j) \ .$$

The *max-k-cut* problem is the task of finding the partition that maximizes the weight of the cut. It can be stated as follows: Let $a_1, \dots, a_k \in \mathcal{S}^{k-2}$ be the vertices of a regular simplex, where

$$\mathcal{S}^d = \{x \in \mathbb{R}^{d+1} \mid \|x\|_2 = 1\}$$

is the d-dimensional unit sphere. Then the inner product $a_i \cdot a_j = -\frac{1}{k-1}$ whenever $i \neq j$. Hence, finding the max-k-cut is equivalent to solving the following integer program:

$$\begin{aligned} \text{IP:} \quad & \text{maximize} \tfrac{k-1}{k} \sum_{i<j} d_{ij}(1 - y_i \cdot y_j) \\ & \text{subject to} y_j \in \{a_1, \dots, a_k\} \\ & \text{for all } 1 \leq j \leq n. \end{aligned}$$

Frieze and Jerrum [5] propose the following semidefinite program (SDP) in order to relax the integer program:

$$\begin{aligned} \text{SDP :} \quad & \text{maximize} \tfrac{k-1}{k} \sum_{i<j} d_{ij}(1 - v_i \cdot v_j) \\ & \text{subject to } v_j \in \mathcal{S}^{n-1} \\ & \text{for all } 1 \leq j \leq n \text{ and} \\ & v_i \cdot v_j \geq -\tfrac{1}{k-1} \text{ for all } i \neq j \\ & \text{(necessary if } k \geq 3). \end{aligned}$$

The constraints $v_i \cdot v_j \geq -\frac{1}{k-1}$ are necessary for $k \geq 3$ because otherwise the SDP would prefer solutions where $v_i \cdot v_j = -1$, resulting in a larger value of the objective. We shall see in the experimental part that this indeed would result in invalid approximations. The SDP finally can be reformulated as a convex program:

$$\begin{aligned} \text{CP :} \quad & \text{minimize} \sum_{i<j} d_{ij} Y_{ij} && \text{(4a)} \\ & \text{subject to} Y_{jj} = 1 && \text{(4b)} \\ & \text{for all } 1 \leq j \leq n \text{ and} && \text{(4c)} \\ & Y_{ij} \geq -\tfrac{1}{k-1} \text{ for all } i \neq j && \text{(4d)} \\ & \text{(necessary if } k \geq 3) && \text{(4e)} \\ & \text{and } Y \succeq 0. && \text{(4f)} \end{aligned}$$

Here, $Y \in \mathbb{R}^{n \times n}$ is a matrix, and the last condition $Y \succeq 0$ means that Y is positive semidefinite. Hence, Y will be a kernel matrix. Efficient solvers are available for this kind of optimization problems, such as CSDP [1] or SeDuMi [12]. In order to implement the constraints $Y_{ij} \geq -\frac{1}{k-1}$ with these solvers, actually positive slack variables Z_{ij} have to be introduced together with the equality constraints

$$Y_{ij} - Z_{ij} = -\frac{1}{k-1} .$$

Finally, in order obtain the partitioning from the vectors v_j or the matrix Y, [5] propose to sample k points $z_1, \ldots, z_k$ randomly on $\mathcal{S}^{n-1}$ and assign each v_j to the group by the closest z_i. They show approximation guarantees generalizing those of Goemans and Williamson [6]. In practice however, the approximation guarantee does not necessarily yield a good clustering, and applying the k-means algorithm for clustering the v_j gives better results here. We use the kernel k-means (probably introduced for the first time by [10]) which directly works on the scalar products $Y_{ij} = v_i \cdot v_j$, without need of recovering the v_j. We thus arrive at the following algorithm:

Algorithm SDP clustering of a word list
Input: word list $X = (x_1, x_2, \ldots, x_n)$,
number of clusters k
Output: clustering $c \in \{1 \ldots k\}^n$

```
1. for $x, y \in X$, compute Google relative
     frequencies $p^{www}(x)$, $p^{www}(x,y)$

2. for $x, y \in X$, compute complexities
     $K^{www}(x)$, $K^{www}(x,y)$

3. compute distance matrix

     $$D = \left(d^{www}(x,y)\right)_{x,y \in X}$$

4. solve the SDP by using CP (cf. (4a)
     through (4f))

5. cluster the resulting matrix $Y$ using
     kernel k-means [10]
```

3. Experiments

In this section, we present the experiments of our clustering algorithm applied to four lists of terms. First we show step by step how the algorithm acts on the first data set, which is the following list of 60 English words:

`axiom,average,coefficient,probability, continuous,coordinate,cube, denominator,disjoint, domain, exponent, function,histogram, infinity, inverse, logarithm,permutation, polyhedra, quadratic, random, cancer, abnormal, abscess, bacillus,delirium, betablocker, vasomotor, hypothalamic, cardiovascular, chemotherapy, chromosomal, dermatitis, diagnosis, endocrine, epilepsy, oestrogen, ophthalmic,vaccination, traumatic, transplantation, nasdaq, investor, obligation, benefit,bond, account, clearing, currency, deposit, stock, market, option, bankruptcy,creditor, assets, liability, transactions, insolvent, accrual, unemployment`

The first 20 words are commonly used in mathematics, the next 20 words have been taken from a medical glossary, and the final 20 words are financial terms. The matrix containing the complexities is depicted in Figure 1 (large complexities are white, small complexities black). Clearly, the single complexities on the diagonal are smaller than the pairwise complexities off-diagonal.

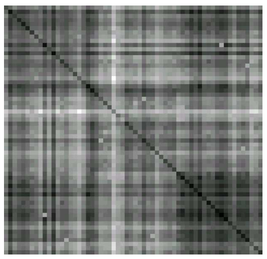
Fig. 1: Complexities

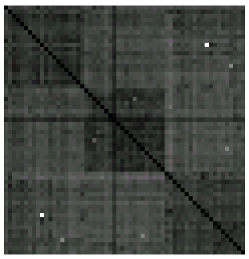
Fig. 2: Distances

In the distance matrix (Figure 2), the

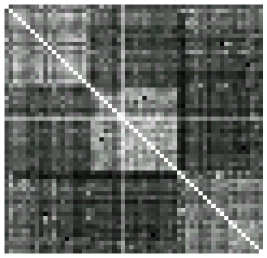

Fig. 3: Similarity

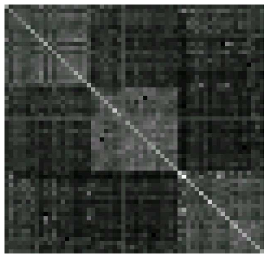

Fig. 4: Laplacian

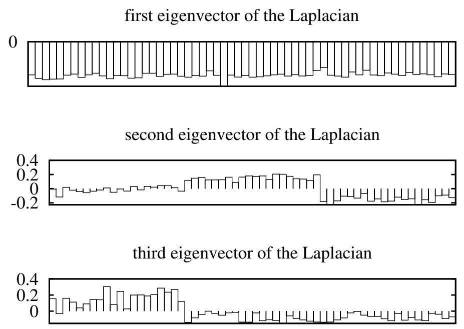

Fig. 5: Eigenvector plot

block structure is visible. After transformation to the Similarity matrix (Figure 3) and Laplacian (Figure 4), the block structure of the matrix becomes very clear. Figure 5 shows the top three eigenvectors of the Laplacian. The first eigenvector having only negative entries seems not useful at all for the clustering (but in fact it is useful for the kLines algorithm). The second eigenvector separates the medical terms (positive entries) from the union of mathematical and financial terms (negative entries). This indicates that the mathematical and financial clusters are closer related than each is related to the medical terms, in a hierarchical clustering we would first split off the medical terms and then divide mathematical and financial terms. The spectral clustering correctly groups all terms except for "`average`", which is assigned to the financial terms instead of the mathematical terms (this is also visible from Figure 5). As `average` also occurs often in finance, we cannot even count this as misclustering. We stress that the clustering algorithm is of course invariant to permutations of the data, i.e. yields the same results if the terms are given in a different order. It is just convenient for the presentation to work with an order corresponding to the correct grouping.

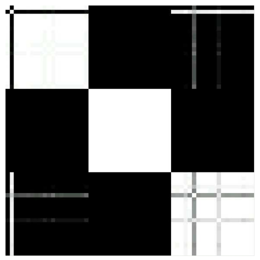

Fig. 6: Kernel matrix after SDP

The same clustering result is obtained from the SDP clustering. The kernel matrix resulting from solving the SDP clearly displays the block structure, again with the exception of the term "`average`".

In case that we do not know the number of clusters k in advance, there is a way to estimate this quite reliably from the eigenvalues of the Laplacian, if the data is not too noisy. Consider again the case of a perfect block diagonal matrix, i.e. all intra-cluster similarities are 1 and all other entries 0. Then the number of non-zero eigenvalues of this matrix is equal to the number of blocks/clusters. If the matrix is not perfectly block diagonal, we may still expect some dominant eigenvalues which are clearly larger than the others. Figure 7 top left shows this for our first example data set. The top three eigenvalues of the Laplacian are dominant, the fourth and all subsequent ones are clearly smaller. (Observe that the smallest eigenvalues are even negative: This indicates that the distances we used do not stem from a metric. Otherwise all eigenvalues should be nonnegative, since the Gaussian kernel is positive definite.)

We propose a simple method for detecting the gap between the dominant eigenvalues and the rest: Tentatively split

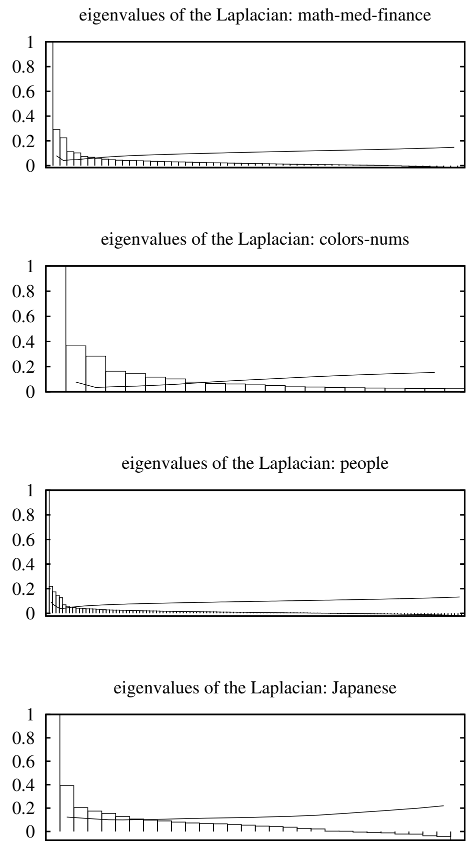

Fig. 7: Plots of the eigenvalues of the Laplacian (bars) and the s.s.e. score for determining the number of clusters (lines) for the four data sets

after the second eigenvalue, compute the means of the eigenvalues in the two groups "dominant" and "non-dominant" (ignore the top eigenvalue, which is always much larger), and calculate the sum square error (s.s.e.) of all eigenvalues w.r.t. their means. Compute this s.s.e. score also for the split after the third eigenvalue, the fourth eigenvalue and so forth. Choose the split with the lowest score. We have depicted the s.s.e. scores in Figure 7 by solid lines. For the math-med-finance data set, the minimum score is at the correct number of $k = 3$ clusters.

Note that this method works only for the spectral clustering, there is no obvious corresponding algorithm for the SDP clustering.

The next data set is the "colors-nums" data set from [2]:

`purple, three, blue, red, eight, transparent, black, white, small, six, yellow,seven, fortytwo, five, chartreuse, two, green, one, zero, orange, four`

Although the intended clustering has two groups, colors and numbers (where "`small`" is supposed to be a number and "`transparent`" a color), the eigenvalues of the Laplacian in Figure 7 bottom left indicate that there are three clusters. Indeed, in the final spectral clustering, "`fortytwo`" forms a singleton group, and "`white`" and "`transparent`" are misclustered as numbers. Clustering with SDP gives a slightly different result: Here, best results are obtained with $k = 2$ clusters, in which case only "`fortytwo`" is wrongly assigned to the colors.

The next data set,

`Takemitsu, Stockhausen, Kagel, Xenakis, Ligeti, Kurtag, Martinu, Berg, Britten, Crumb,Penderecki, Bartok, Beethoven, Mozart, Debussy, Hindemith, Ravel, Schoenberg, Sibelius,Villa-Lobos, Cage, Boulez, Kodaly, Prokofiev, Schubert, Rembrandt, Rubens, Beckmann, Botero, Braque, Chagall, Duchamp, Escher, Frankenthaler, Giacometti, Hotere, Kirchner, Kandinsky, Kollwitz, Klimt, Malevich, Modigliani, Munch, Picasso, Rodin, Schlemmer, Tinguely, Villafuerte, Vasarely, Warhol, Rowling, Brown, Frey, Hosseini, McCullough, Friedman, Warren, Paolini, Oz, Grisham, Osteen, Gladwell, Trudeau, Levitt, Kidd, Haddon, Brashares, Guiliano, Maguire, Sparks, Roberts, Snicket, Lewis, Patterson, Kostova, Pythagoras, Archimedes, Euclid, Thales, Descartes, Pascal, Newton, Lagrange, Laplace, Leibniz, Euler, Gauss, Hilbert, Galois, Cauchy, Dedekind, Kantor, Poincare, Godel, Ramanujan, Wiles, Riemann, Erdos, Thomas Zeugmann, Jan Poland, Rolling, Stones, Madonna,Elvis, Depeche, Mode, Pink, Floyd, Elton, John, Beatles, Phil, Collins,` `Toten, Hosen, McLachan, Prinzen, Aguilera, Queen, Britney, Spears, Scorpions, Metallica, Blackmore, Mercy`

consists of five groups of each 25 (more or less) famous people: composers, artists, last year's bestseller authors, mathematicians (including the authors of the present paper), and pop music performers. We deliberately did not specify the terms very well (except for our own much less popular names), in this way the algorithm could "decide" itself if "`Oz`" meant one of the authors Amos and Mehmet Oz or one of the pop music songs with Oz in the title. The eigenvalue plot in Figure 7 top right shows clearly five clusters. From the 125 names, 9 were not clustered into the intended groups using the spectral method. The highest number of incorrectly clustered names (4 misclusterings) occurred in the least popular group of the mathematicians (but our two names were correctly assigned). We also observed that the spectral clustering gets disproportionately harder when the number of clusters increases: Clustering only the first 50, 75, and 100 names gives 0, 2, and 5 clustering errors, respectively. We also tried clustering the same data set w.r.t. the Japanese web sites in the Google index, this gave 0, 1, 4, and 16 clustering errors for the first 50, 75, 100, and 125 names, respectively.

Clustering with SDP gives better results here: 0, 0, 1, and 4 clustering errors for the first 50, 75, 100, and 125 names, respectively.

The last data set consists of 20 Japanese terms from finance and 10 Japanese terms from computer science (taken from glossaries):

依頼, 為替, 営業, 円高, 株価, 環境, 金利, 景気, 雇用, 購入, 財政, 株価, 落札, 輸出, 税金, 売上高, 破綻, 流動性, 有価証券, 販路, 回路, 画像処理, 画像圧縮, 関数, 近似, 係数, 形式, 論理, 実験, 算術.

The eigenvalue plot in Figure 7 does not clearly indicate the correct number of $k = 2$ clusters. However, when using $k = 2$, only the term "環境" (which means "environment") is non-intendedly grouped with the computer science words by the spectral clustering. SDP clustering gives the same result here.

The computational resources required by our clustering algorithms are much lower than those needed by Cilibrasi and Vitányi's algorithm [2]. Their clustering tries to optimize a quality function, a task which is NP hard. The approximation costs hours even for small lists of $n = 20$ words. On the other hand, our spectral clustering's naive time complexity is cubic $O(n^3)$ in the number of words. This can be improved to $O(n^2k)$ by using Lanczos method for computing the top k eigenvectors.

On our largest "people" data set with $n = 125$, our spectral clustering needs about 0.11sec on a 3GHz Pentium4 processor with the ATLAS library.

Solving the SDP in order to approximate max-cut is more expensive. The respective complexity is $O(n^3 + m^3)$ (cf. [1]), where m is the number of constraints. If $k = 2$, then $m = n$ and the overall complexity is cubic. However, for $k \geq 3$, we need $m = O(n^2)$ constraints, resulting in an overall computational complexity of $O(n^6)$. On our largest "people" data set with $n = 125$, our SDP clustering needs about 2544sec.

4. Discussion and Conclusions

We have shown that it needs surprisingly little effort, just a few queries to a popular search engine together with some state-of-the-art methods in machine learning, to automatically separate lists of terms into clusters which make sense. We have focused on unsupervised learning in this paper, but for other tasks such as supervised learning, appropriate tools are available as well (e.g. SVM). Our methods are theoretically quite well founded, basing on the theories of Kolmogorov complexity on the one hand and graph cut criteria and spectral clustering or semidefinite programming on the other hand. The SDP clustering is the more direct approach, as it needs less steps. Also, it yields slightly better results. However, it is computationally more expensive than spectral clustering.

References

[1] B. Borchers and J. G. Young. Implementation of a primal-dual method for sdp on a shared memory parallel architecture. March 27, 2006.

[2] R. Cilibrasi and P. M. B. Vitányi. Automatic meaning discovery using Google. Manuscript, CWI, Amsterdam, 2006.

[3] I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the 7th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining(KDD)*, pages 269–274, 2001.

[4] I. Fischer and J. Poland. New methods for spectral clustering. Technical Report IDSIA-12-04, IDSIA / USI-SUPSI, Manno, Switzerland, 2004.

[5] A. Frieze and M. Jerrum. Improved algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, 1997.

[6] M. X. Goemans and D. P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2SAT. In *STOC '94: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 422–431. ACM Press, 1994.

[7] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitányi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004.

[8] M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer, 2nd edition, 1997.

[9] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, 2001.

[10] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[11] D. A. Spielman and S. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *IEEE Symposium on Foundations of Computer Science*, pages 96–105, 1996.

[12] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(12):625–653, 1999.

[13] E. L. Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *HLT-NAACL 2003: Main Proceedings*, pages 244–251, Edmonton, Alberta, Canada, 2003.

[14] U. von Luxburg, O. Bousquet, and M. Belkin. Limits of spectral clustering. In *Advances in Neural Information Processing Systems (NIPS) 17.* MIT Press, 2005.

[15] S. X. Yu and J. Shi. Multiclass spectral clustering. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 313–319. IEEE Computer Society, 2003.