Clustering via String Rectification Kernels

Ben Shank
Graduate Student
Columbia University
New York, NY 10027

Abstract Herein are presented three novel kernels and a pipeline for autogenous clustering of string-based data with obscured a priori interrelationships. The kernels presented improve upon a current natural language processing (NLP) kernel, the String Subsequence Kernel (SSK) [3], in terms of computational efficiency and semantic discrimination. Strictly non-probabilistic kernel methods for string-based data like the SSK employ weighted counting schemes regarding the presence of not necessarily contiguous subsequences within input data, hindering implementation efforts due to the large implicit feature space generated and a combinatorial explosion of complexity. We will show how some simple representational changes can produce kernels that can outperform the SSK within the context of nonlinear dimensionality reduction an example from genetics.

1 Introduction

Extracting tangible, autogenous relationships between members of a collection of string-based data intimately depends on the representation of the collection. Fixed-length strings present themselves with a natural representation as finite sequences of symbols taken from a finite alphabet. However as we probe even further into the structure of these objects, we find them to be more mathematically subtle and complicated than intuition initially gives way. Indeed, the Kleene closure of a finite alphabet with empty string, under the operation of concatenation, is an example of the free monoid, with which further modification opens the floodgates to the study of Kleene algebras. Details can be found in [5]. We will not pursue an algebraic approach in this paper, though the author remains curious about such an avenue.

It is precisely the goal of a kernel method to capture the behavior of the objects within the problem domain as members of a linear space equipped with a dissimilarity measure, thereby lending themselves to the utility of linear algebra. This space is referred to as the feature space, and is usually relatively high-dimensional with respect to the problem domain. Kernel methods rely on the kernel trick and Mercer's theorem [1,2], which together practically allow one to view any conveniently defined symmetric and positive-semidefinite map of two objects in the problem domain as the inner product of their feature space representations. One of the most important consequences of the kernel trick is that the computation of inner products in feature space can be done without actually computing the intermediate feature space representations of the input data. This all begs the question of how exactly to choose a suitable kernel for collections of fixed-length strings that have some intuitive a priori interrelationships. For example, how would we find a suitable kernel for the clustering and exposition of similarity between several strains of an organism's genome?

NLP kernels can be used in nonlinear dimensionality reduction to attack clustering problems like this effectively. One such kernel is the String Subsequence Kernel (SSK) presented by Lodhi et al. [3]. In its basic form, it has complexity on the order of $O\left(|\mathcal{A}|^n\right)$, where n is the length of an input string and $|\mathcal{A}|$ is the cardinality of the finite alphabet $\mathcal{A}$. While there are improvements to efficiency are also developed in [3], a need for faster, more reliable kernels still exists, especially in light of large input corpora.

Definitions. We will now adopt some notational conventions and from this point assume all subsequences and strings are finite unless otherwise specified. We will also proceed with the understanding that subsequences are not necessarily contiguous sequences of characters whereas strings are necessarily contiguous.

If u is a subsequence of an l-length string s we will write $u \sqsubseteq s$. Note that it is certainly true that $s \sqsubseteq s$. We understand that there exists an ordered list of indices $\mathbf{i} = \{i_1, \ldots, i_n\}$ such that $u_1 = s_{i_1}, \ldots, u_n = s_{i_n}$, which we denote $u = s[\mathbf{i}]$. Observe that $i_1 < \ldots < i_n$ and $i_n \leq l$. Finally, let the length of a subsequence u be given by $|u| = i_n - i_1 + 1$.

For a $D \times N$ matrix A, denote the number of elements in A as $(\!|A|\!) = DN$. Additionally, let $\overrightarrow{A}$ be the $(\!|A|\!) \times 1$ vector obtained by stacking each of the N columns of A on top of one another. Lastly, denote $\langle \cdot, \cdot \rangle$ as the standard inner product.

2 The String Subsequence Kernel

The SSK is an interesting and principled kernel presented in [3] that seeks to describe similarity between two strings as essentially a weighted count of all subsequences each string shares. Variations such as the k-mer formulation also presented in [3], alternate notation and naming conventions presented in [6], and finally approximations such as the λ-pruning approach described in [7] all derive from the following formulation. We will not fully investigate these variations since our approach branches purposefully away from the SSK, however we will use the SSK as a basis for comparison. We now motivate the definition of the SSK with an example.

Example. Suppose we have two input strings under the twenty-six character, lowercase English alphabet such that $s =$ pet and $t =$ jet. We wish to compute the similarity of these two strings via the accumulation of shared subsequences of length 1 and 2. First we will introduce a decay factor, $\lambda \in (0, 1] \subset \mathbb{R}$, that will penalize long stretches of non-contiguity. When a subsequence matches, we will tally the result by taking λ to the power of the contiguous length of the matched subsequence. If there is no match, then we count the result as zero. This is easily visualized by way of a table:

	p	j	e	t	p-e	p-t	e-t	j-e	j-t
pet	λ^1	0	λ^1	λ^1	λ^2	λ^3	λ^2	0	0
jet	0	λ^1	λ^1	λ^1	0	0	λ^2	λ^2	λ^3

Computing the SSK, K_l, for subsequence of length l is done by multiplying these intermediate computations row-wise:

$$K_1(s,t) = (\lambda \times 0) + (0 \times \lambda) + (\lambda \times \lambda) + (\lambda \times \lambda)$$

$$K_2(s,t) = (\lambda^2 \times 0) + (\lambda^3 \times 0) + (\lambda^2 \times \lambda^2) + (0 \times \lambda^2) + (0 \times \lambda^3)$$

so that $K_1(s,t) = 2\lambda^2$, $K_2(s,t) = 2\lambda^4$, and

$$K = 2\lambda^2 + 2\lambda^4.$$

One can create variations by playing with the value of λ and taking different linear combinations of K_l for various l, as kernels of this form are closed under addition (see [4]).

It is immediately clear that for large strings and alphabets, this direct computation is impractical. To see why, we will now formally define the feature space and feature mappings involved in the SSK definition.

Let a finite alphabet $\mathcal{A}$ and its Kleene closure $\mathcal{A}^*$ be given. Define the feature mapping

$$\phi_u(s) = \sum_{u \sqsubseteq s} \lambda^{|u|}$$

so that

$$K_n(s,t) = \sum_{u \in \mathcal{A}^n} \phi_u(s)\phi_u(t)$$

which is precisely the kernel we computed by hand in the previous example. One can see here that the complexity of computing K_n is $O\left(|\mathcal{A}|^n\right)$ since effectively every possible subsequence of length n must be tested for inclusion within s and t. This definition implies a massive feature space with much sparsity since each ϕ_u represents the presence of each and every possible subsequence over $\mathcal{A}$. Furthermore, if we assume the probability of selecting any particular character from $\mathcal{A}$ is independent and uniform over the cardinality of $\mathcal{A}$, then the probability of selecting a string of length l from this alphabet is $\prod_1^l Pr(a) = |\mathcal{A}|^{-l}$. Thus for long strings, the probability of finding non-zero components in the exhaustive feature space implied by the SSK is very small, which also adds to the problematic nature of computing the kernel.

Fortunately a more efficient implementation of the SSK is given in [3] that uses dynamic programming. The details will not be explained here, yet we use this implementation for our experiments.

3 The String Rectification Kernel

We seek an alternate string representation that will allow us to leverage contemporary computational resources without incurring the expensive overhead associated with the SSK. Let a finite alphabet $\mathcal{A}$ be given. Suppose its characters are ordered and the indices for each are obtained by the indicator map $I(c) = \alpha \in \mathbb{Z}$ if the character c occupies the α^{th} position in the alphabet.

For a fixed-length string $s \in \mathcal{A}^l$, write

$$\begin{array}{rccccc} & I(s_1) & I(s_2) & \dots & I(s_l) \\ = & \alpha_1 & \alpha_2 & \dots & \alpha_l. \end{array}$$

Define the $m \times 1$ vectors

$$\mathbb{1}_m = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbb{0}_m = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{with } \mathbb{1}_0 = \mathbb{0}_0 = \varnothing.$$

Now let $\alpha_M = \max(\alpha_j)\, \forall j \in [1, l]$ and define

$$R_s = \begin{bmatrix} \mathbb{1}_{\alpha_1} & \mathbb{1}_{\alpha_2} & \dots & \mathbb{1}_{\alpha_l} \\ \mathbb{0}_{\alpha_M - \alpha_1} & \mathbb{0}_{\alpha_M - \alpha_2} & \dots & \mathbb{0}_{\alpha_M - \alpha_l} \end{bmatrix}.$$

This is an $\alpha_M \times l$ representation matrix we call the *rectification matrix*. We call this process *string rectification*, as it in a sense refines the string into an immediately useful object of linear algebra without sacrificing notions of character order or imposing a particular numerical representation. To see why the latter statement is true, notice that an infinite class of rectification matrices exists by scaling the columns of the basic form. Perhaps

the most defining characteristic of the rectification matrices is simply that each column sum is equal to the alphabet index for each character of the rectified input string, that is,

$$\sum_{i=1}^{\alpha_M} R_{i,j} = I(s_j) \quad \forall j \in [1, l]\,.$$

We now define the String Rectification Kernel (SRK) for two input strings s and t. Suppose that R_s and R_t are respectively $\alpha_{M_s} \times |s|$ and $\alpha_{M_t} \times |t|$. Let $D = \max(\alpha_{M_s}, \alpha_{M_t})$ and $N = \max(|s|, |t|)$. Form the $D \times N$ zero-padded matrices

$$Z_s = \begin{bmatrix} R_s & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \quad \text{and} \quad Z_t = \begin{bmatrix} R_t & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}.$$

Then the SRK is computed in a familiar geometric form as

$$K_R(s,t) = \frac{\left\langle \overrightarrow{Z_s}, \overrightarrow{Z_t} \right\rangle}{\sqrt{\left\langle \overrightarrow{Z_s}, \overrightarrow{Z_s} \right\rangle \left\langle \overrightarrow{Z_t}, \overrightarrow{Z_t} \right\rangle}}.$$

4 Two Boolean Kernels

4.1 Distance Based

Although we will not explore in-depth variations of the basic definition of the SRK, one simple observation is that rectification matrices by construction are boolean matrices. Thus we are free to employ boolean operations on these objects. We will build a simplistic kernel inspired by the Hamming distance between two binary strings, defining a Boolean SRK (BSRK) as

$$K_B(s,t) = \left\langle \mathbb{1}_{DN}^T,\ \overrightarrow{Z_s} \oplus \overrightarrow{Z_t} \oplus 1 \right\rangle / DN$$

where $\oplus$ denotes addition modulo 2. This kernel yields a broad sense of the positions rectified strings share.

4.2 Cross-Comparison Based

Yet another boolean kernel is formulated by considering the $(i,j)^{\text{th}}$ entry of a matrix of the form

$$B_{ij}(s,t) = \delta_{s_i,t_j}$$

where we assume that strings s and t are of equal length and δ is the Kronecker delta.

We will compute this Boolean Compare Kernel (BCK) as

$$K_C(s,t) = \frac{\langle B(s,t), B(s,t) \rangle_F}{\| B(s,s) \|_F \| B(t,t) \|_F}$$

where the subscript F denotes the Frobenius norm and inner product.

5 A Hybrid Clustering Pipeline

To avoid reformulating the general clustering problem, we formed a hybrid clustering pipeline built end-to-end with Minimum Volume Embedding (MVE) [8,9] nonlinear dimensionality reduction and Mean-Shift Clustering (MSC) derived from [10]. We chose to construct the pipeline this way because 1) given the appropriate kernel, MVE is extremely useful in reducing the dimensionality of arbitrary data to one or two dimensions, and 2) MSC has a simple formulation and is robust to identifying clusters in low dimensional, naïve circumstances.

There are a number of parameters that can be altered in both MVE and MSC that substantially affect performance. To this end, we determined the best parameters through experimentation and guided by the theoretical motivations given in the respective literature.

For MVE, we chose a value of 8 for the k-nearest neighbors and the target dimension was set to 2. All embedding coordinates were scaled to occupy $[-1,1]^2 \subset \mathbb{R}^2$. It was found that no improvement was gained with larger values for the k-nearest neighbors, and smaller values yielded only slightly worse embeddings.

We used MVE in *full* mode due to the expected sparse connectivity of disparate string clusters. This mode relaxes the local isometry constraint formulated in [9]. Originally, inputs from the problem domain x_i and corresponding features $\phi(x_i)$ are forced to be locally isometric, meaning that the there exists a rotation and translation that maps x_i and its k-nearest neighbors onto $\phi(x_i)$ and its neighbors. The modified constraints allow for scaling in feature space, so that our embedding will not necessarily be forced to contract.

For MSC, we ran several variations of MVE over the data sets before finally imposing a Gaussian kernel windowing parameter of 0.01. The bandwidth parameters used ranged from 0.001 to 0.01. Larger values of either parameter yield softer discrimination between string clusters, whereas in our case, we wanted to err on the side of caution with our clustering, so that relatively small changes in the embedding coordinates represent subtle semantic changes over the original input strings.

6 Experimental Results

We tested the clustering performance using the SSK, SRK, BSRK, and BCK kernels within the hybrid clustering pipeline described in **§5**. Performance was measured subjectively, against the a priori human-labeled data sets.

6.1 A Model Organism Haplotype Distribution

Associating genetic loci to complex phenotypes across strains of a model organism is often cast into a genome-wide study of single nucleotide polymorphism (SNP) variations to measured phenotypes. For our purposes,

one may regard SNPs as nucleotides along an organism's genome that "wobble" reliably between two values over generations of breeding. Nature often packages sequences of SNPs into logical units within an organism's genome called *haplotypes*. A biologist may be interested in determining an organism's haplotype distribution for several reasons ranging from tracing ancestry to breaking down a genotype-phenotype association problem.

We applied our hybrid clustering pipeline to the SNP genotype data of fifteen strains of mice, as provided by Perlegen [12]. Each cluster obtained represents a particular haplotype. The input data sets were divided into groups based on predicted haplotype block boundaries formed from prior analysis by Frazer, et al [11]. This was an important decision since it is another task altogether to identify the haplotype block boundaries. Much of that task is biologically inspired and beyond the scope of this paper.

Note that for this dataset, we have that $\mathcal{A} = \{\text{A,T,C,G,N}\}$, where A, T, C, and G represent the four chemical nucleotide bases and N is an error indicator character that represents a non-sequenced nucleotide.

We now introduce a good baseline dataset consisting of a haplotype block length of fifteen nucleotides. The fact that there are fifteen strains as well is coincidental. One can observe from Figure 1 that upon inspection there are six well-defined haplotypes:

$$H_1 = \{6\}\,, \quad H_2 = \{7\}\,, \quad H_3 = \{8\}\,, \quad H_4 = \{15\}$$
$$H_5 = \{1, 2, 3, 4, 5\}$$
$$H_6 = \{9, 10, 11, 12, 13, 14\}\,.$$

It is the goal of our pipeline to autogenously reproduce these groupings, and we can see that is successfully does so with well-formed, low-error data such as this.

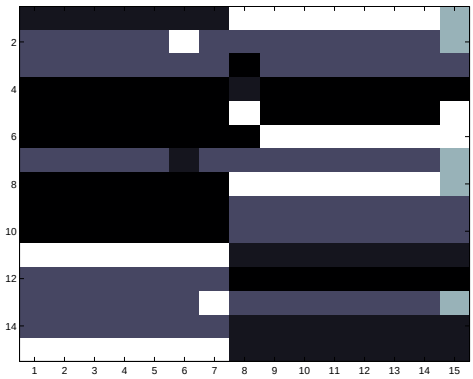

Figure 1: The raw input string map with strains along the x-axis and the nucleotide values color coded along the y-axis. Here, teal is identified with N, black with A, white with T, dark gray with C, and light gray with G.

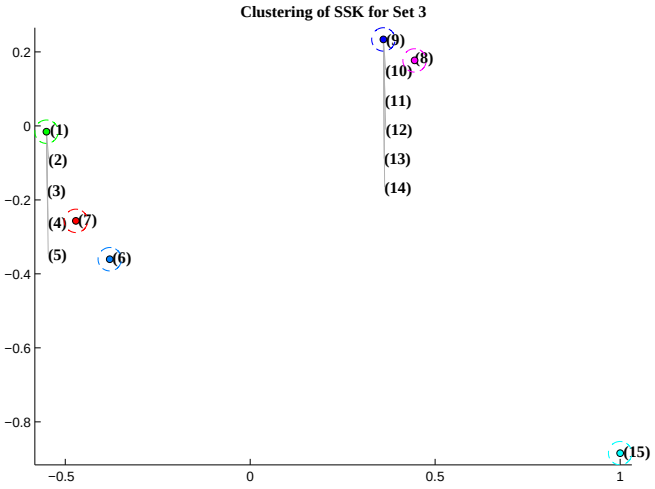

Figure 2: Results using the SSK with $\lambda = 0.25$, subsequence lengths of $n = 1, 2, 3, 4$, and MSC bandwidth of 0.001. This embedding is of good quality and correctly produces the six haplotype clusters, yet there is a slight lack of discrimination between strings differing by a small number of characters.

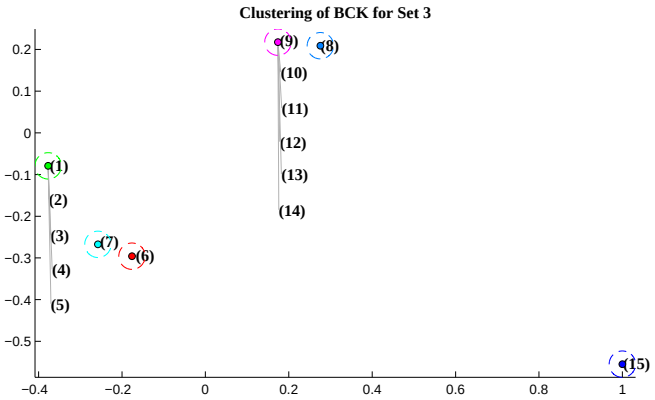

Figure 3: Results using the BCK with MSC bandwidth of 0.001. Note the similarity between the SSK embedding and this one, only the BCK took less than ten seconds to compute, whereas the SSK took over one minute.

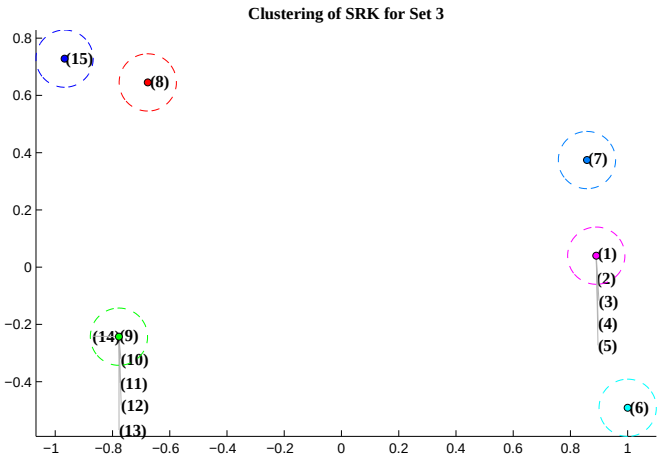

Figure 4: Results using the SRK and MSC bandwidth of 0.01 with noted improvement in the semantic segregation. This kernel took less than ten seconds to compute.

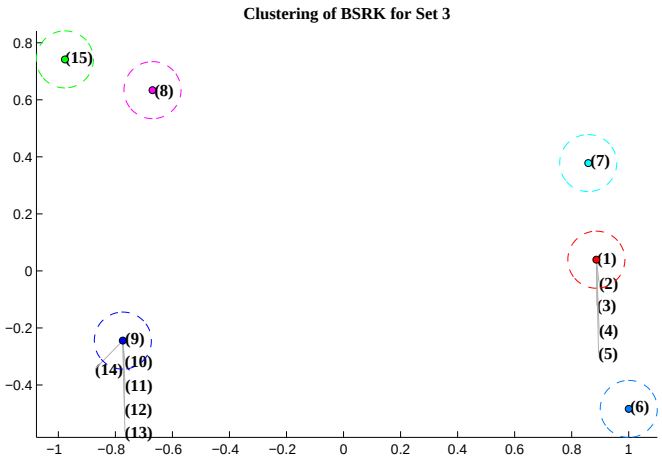

Figure 5: Results using the BSRK and MSC bandwidth of 0.01 with almost no noted improvement in the semantic segregation from the SRK. This kernel took less than ten seconds to compute.

We can see that the clustering performance in all cases is good, but the SRK and BSRK are both faster and produce the greatest semantic separation in the embedding coordinates, which makes the application of MSC more meaningful. These results were consistently observed for several members of this particular data set, with the expected decrease in semantic segregation as the number of N-type SNPs present in the data increased.

Perhaps the most surprising aspect of these experiments is the qualitative similarity between the embeddings produced using the SSK and the BCK. This similarity was observed in six of the experimental SNP datasets used. Further investigation is needed to understand this relationship and unfortunately exceeds the scope of this paper.

7 Discussion and Future Work

While we focused on the autogenous clustering of string-based data, a noted extension to the pipeline proposed in **§5** would involve using the clusters obtained to train a multi-class support vector machine (SVM).

We have demonstrated that there are more simple and effective means to approach NLP clustering problems like the one presented in this paper. While the SSK has an interesting, principled definition, we have shown that satisfactory results can be attained without its built-in complexity. Further work must be done to identify the general mathematical mechanisms in play that correspond to the empirical performance improvements noted herein.

Loosely speaking, the kernels we presented in this paper have essentially mapped the input domain to smaller yet informatively sufficient feature spaces than the SSK. It may be that a well-defined optimization problem exists which captures the tradeoff between the dimensionality of the feature space and its inherent ability to express the natural and most meaningful relationships present in the data. Making the latter notion precise would be a cardinal task.

References

[1] A. Schlkopf, et al. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 1998.

[2] B. Schlkopf. The Kernel Trick for Distances. *NIPS*, pp. 301-307, 2000.

[3] H. Lodhi, et al. Text Classification Using String Kernels. *Journal of Machine Learning Research* 2, pp. 419-444, 2002.

[4] D. Haussler. Convolution Kernels on Discrete Structures. *Technical Report UCSC-CRL-99-10*, University of California in Santa Cruz, Computer Science Department, July 1999.

[5] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979. Print.

[6] A. Martins. String Kernels and Similarity Measure for Information Retrieval. *Technical Report*, Priberam, Lisbon, Portugal, 2006.

[7] A. Seewald and F. Kleedorfer. Lambda Pruning: An Approximation of the String Subsequence Kernel. *Technical Report TR-2005-13*, Austrian Research Institute for Artificial Intelligence, Vienna, 2005.

[8] B. Shaw and T. Jebara. Minimum Volume Embedding. *Proceedings of the Conference on Artificial Intelligence and Statistics*, 2007.

[9] K. Q. Weinberger, et al. Learning a Kernel Matrix for Nonlinear Dimensionality Reduction. *Proceedings of the Twenty First International Conference on Machine Learning*, pp. 839-846, Banff, Canada, 2004.

[10] K. Fukunaga and L. D. Hostetler. The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition. *IEEE Transactions on Information Theory* 21 (1), pp. 32-40, 1975.

[11] K. A. Frazer, et al. A Sequence-Based Variation Map of 8.27 million SNPs in Inbred Mouse Strains. *Nature* 448, pp. 1050-3, 2007.

[12] "Mouse SNP and Genotype Data Download." Perlegen Sciences, Inc. 23 Nov. 2010. ⟨http://mouse.perlegen.com/mouse/index.html⟩