Kernelized Locality Sensitive Hashing

Daniel Speyer and Geelon So
{dls2192,geelon}@columbia.edu

December 15, 2017

Abstract

In this paper, we consider the motivation and theory behind kernel methods, as we seek to describe some insight and intuition as to how one might *kernelize* existing algorithms. We ground our discussion by focusing on locality-sensitive hashing (LSH)—we examine the obstacles to kernelizing LSH and describe the steps one can take to overcome them. Finally, we suggest possible directions future research might find interesting to pursue.

1 Introduction

Suppose we wish to search flickr for images similar to an uploaded image, or NCBI's database for bacteria similar to the one found in our patient's throat. We will need a rather unusual sort of database to do this with any efficiency.

Often, we use locality sensitive hashing (LSH) to solve such approximate nearest-neighbor problems. But these hashing families are defined on very specific spaces. In this paper, we will describe how, with an appropriate similarity function, we can extend LSH to these more unusual databases by kernel methods. Let us begin by reviewing (1) LSH, (2) kernels, and (3) similarity measures.

1.1 LSH

Locality sensitive hashing is a general solution for searches in which the desired key is near or nearest to the query, rather than equal. It is not as efficient as tree-based solutions for $\mathbb{R}^n$ with small n, but unlike those solutions, it maintains its performance in high dimensions or in arbitrary metric spaces.

Most near search problems can be reduced to cr-near search. Suppose we have a database of points $\{x_1, \dots, x_n\}$ drawn from some arbitrary metric space $\mathcal{X}$ and some query q also from $\mathcal{X}$, plus constants $c, r \in \mathbb{R}$. A cr-near search seeks either an x within distance cr of q or a statement that there are none within r. It is generally best to think of r as part of the query and c as the acceptable level of error.

To use LSH for problems like this we will need a family of randomly chosen hashing functions $h_1, h_2, \dots$ that partitions $\mathcal{X}$ into some small finite space, all following two key inequalities:

$$\begin{aligned} ||x_1 - x_2|| < r &\implies \Pr[h(x_1) = h(x_2)] \geq p_1 \\ ||x_1 - x_2|| > cr &\implies \Pr[h(x_1) = h(x_2)] \leq p_2 \end{aligned}$$

Note that we may require an arbitrary number of randomly-generated hashing functions; this family is often denoted by $\mathcal{H}$ in

the LSH literature, but we'll be reserving that symbol for Hilbert spaces later.

In the simplest example, $\mathcal{X}$ is a high dimensional Hamming space and the hash functions sample random coordinates. This may provide useful intuition for thinking about hash functions.

Once we have a source of functions and a gap between p_1 and p_2, we can widen the gap by taking n functions and concatenating their outputs (getting p_1^n and p_2^n). We can then deal with too low a p_1^n by taking m repetitions and searching all of them (getting mp_1^n and mp_2^n). This solves the cr-nearest neighbor (cr-NN) problem with arbitrarily high probability.

Once we've solved the cr-NN problem, we can use that to solve the aproximate nearest neighbor (ANN) problem: finding a point in our database no more than c times farther from the query than the closest point. The efficiency of this search is $O(n^\rho)$ where $\rho = \log(1/p_1)/\log(1/p_2)$ and is bounded by a function of c. For $\mathbb{R}^n$ and ℓ_2, $\rho > 1/c^2$. [OWZ2009]

1.2 Kernels

In this section, we will motivate and describe the theory of kernel methods. We often develop mathematics and algorithms for objects that explicitly encode a lot of structure. For example, LSH has been developed for Hamming spaces, finite-dimensional vector spaces, n-spheres, and so on. These objects are particularly well-suited for analysis, as the very way we *represent* them directly showcases their *properties*. Often, the harmony between representation and structure empowers analysis and enables elegance.

But we can't expect harmony always. Nonetheless, might we be able to extend these methods to objects that have implicit structure?

Kernel methods are one way of approaching such situations. Take any object $\mathcal{X}$—just an abstract set, say cat photos on flickr. But perhaps it has implicit structure. How might we access this structure? Very naturally, we can encode this structure into a mapping from that abstract set to a structure-rich mathematical object.

We often do this without thinking, for example, assigning numbers to objects in order to compare them—a mapping of the form $\mathcal{X} \to \mathbb{R}$. Indeed, at a high level, we can consider different mathematical objects as the 'canonical' idealization of certain types of structure; then, functions from $\mathcal{X}$ to these objects allow us to 'pull back' or instantiate that structure within our specific $\mathcal{X}$.

Of course, the structure we care about with respect to LSH is *similarity*, one idealization of which is the *(real) inner product*. In fact, with kernel methods, we will be able to tightly bind any abstract object to an inner product space, provided the object has the right notion of similarity. From there, our knowledge about inner product spaces will provde us new insight and new algorithms. Let us take a moment to reexamine the inner product on finite-dimensional real vector spaces, before generalizing to infinite-dimensional vector spaces and arbitrary sets.

1.2.1 Finite-Dimensional Inner Product Spaces

Definition 1. Let V be a (possibly infinite) vector space over $\mathbb{R}$. An *inner product* on V is a bilinear map $K : V \times V \to \mathbb{R}$ that is *symmetric* and *positive-definite*. We call the pair (V, K) an *inner product space*.

Symmetry and positive-definiteness are two of the most obvious requirements for any self-respecting function that calls itself a measure of similarity:

1. the similarity between two points x and

y better not depend on the order we specify them, so we require $K(x, y) = K(y, x)$,

2. a nonzero object should be similar to itself, so if $x \neq 0$, then $K(x, x) > 0$.

These two simple intuitions of a 'similarity measure' then precisely define the conditions of an inner product on V.[1] Allow us to be (perhaps overly) rigorous in the following discussion, for it is better to bore in the finite case than to confuse in the general case.

For concreteness, let V be an n-dimensional real vector space with a fixed basis. With any $x, y \in V$, denote by $\langle x, y\rangle$ the *formal dot product* of x and y as an algebraic construction.[2] Then, as K is a bilinear form, there exists a matrix $\mathbf{K}$ where

$$K(x, y) = \langle x, \mathbf{K}y\rangle. \tag{1}$$

We claim that because K is symmetric and positive-definite, we actually have:

$$K(x, y) = \langle \mathbf{K}^{1/2}x, \mathbf{K}^{1/2}y\rangle. \tag{2}$$

The symmetry condition on the inner product K forces the matrix $\mathbf{K}$ to be symmetric; we may appeal to the spectral theorem on symmetric matrices: V has an orthonormal eigenbasis with respect to $\mathbf{K}$. Consider an eigenvalue $\mathbf{K}v = \lambda v$. Because K is positive-definite, we know:

$$K(v, v) = \langle v, \mathbf{K}v\rangle = \lambda\langle v, v\rangle > 0.$$

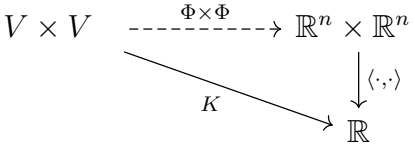
Since the dot product of a vector with itself in any basis is nonnegative, this implies that the eigenvalue λ is positive; so in its eigenbasis, $\mathbf{K}$ is diagonal with positive terms on its diagonal—it follows that $\mathbf{K}^{1/2}$ exists, and thus we obtain Equation 2.

We deduce that there exists a linear map $\Phi : V \to \mathbb{R}^n$ (spoiler: we call Φ a *feature map*) where the inner product on V directly corresponds to the usual dot product on $\mathbb{R}^n$. That is,

$$K(x, y) = \langle \Phi(x), \Phi(y)\rangle. \tag{3}$$

In other (ridiculously abstract) words, Equation 3 precisely says:

Proposition 2. *Let V be an n-dimensional real vector space. If $K : V \times V \to \mathbb{R}$ is an inner product, then there exists a map $\Phi : V \to \mathbb{R}^n$ such that the following diagram commutes:*



where $\langle \cdot, \cdot \rangle$ is the standard dot product on $\mathbb{R}^n$.

This is the main payoff to all this formality: we may view the pair $(\mathbb{R}^n, \langle \cdot, \cdot \rangle)$ as the canonical n-dimensional real inner product space. So, no matter which inner product space (V, K) we want to study, we're in fact guaranteed the existence of an isomorphism Φ to the familiar inner product space $\mathbb{R}^n$, and

[1] For now, we can just view bilinearity as a necessary condition to ensure that K respects the real numbers as an algebraic object. So, in some sense, bilinearity is not intrinsic to K; rather, it is a condition *induced* by the field $\mathbb{R}$. Later on, when we generalize, we won't have a notion of linearity on arbitrary sets $\mathcal{X}$; however, there is a natural 'completion' of $\mathcal{X}$, a vector space $\mathcal{H}$. As before, the linear structure will be induced by $\mathbb{R}$.

[2] One point of confusion may be that $\langle \cdot, \cdot \rangle$ often denotes an inner product—usually unproblematic as the dot product is in fact the inner product *induced by the choice of basis* on V. However, an abstract vector space has no canonical basis; thus, it has no canonical inner product. In particular, here, the inner product corresponding to the dot product is in general unrelated to the inner product K. We will reconcile these two views at Equation 3, which shows that every inner product corresponds to the dot product in an appropriate basis. Conversely, as every dot product is an inner product, if V is an abstract finite-dimensional real vector space, *specifying an inner product on V is equivalent to specifying a basis on V.*

we may freely interchange the two objects in our minds.

Furthermore, we are now justified in dropping the distinction between the formal dot product and inner product; let us denote both by $\langle \cdot, \cdot \rangle$.

1.2.2 Generalization: Kernels and Hilbert Spaces

First, we will need to extend the 'similarity measure' from vector spaces V to abstract sets $\mathcal{X}$. In the former, notice that given a basis $\{v_1, \ldots, v_n\}$, the inner product K is fully determined by the collection of values $K(v_i, v_j)$, where i, j range between 1 and n. This follows from bilinearity of K and is equivalent to saying that we can represent the bilinear form K by a matrix of $n \times n$ values $\mathbf{K}$. In some sense, this means that there are only 'n ways' or 'n directions' in which objects of V may be (dis)similar.

This suggests that we can generalize the definition of inner products because we didn't need to start with all of V! Suppose someone secretly had an n-dimensional inner product space (V, K), but just gave us n linearly independent vectors, say $[n] := \{v_1, \ldots, v_n\}$, along with the corresponding restriction of the similarity measure $K : [n] \times [n] \to \mathbb{R}$.

Our view of $[n]$ would just be a collection of n abstract objects, and K just an $n \times n$ matrix over $\mathbb{R}$ (this is often called the *Gram matrix*). Still, we would have produced the same feature map $\Phi_{[n]} : [n] \hookrightarrow \mathbb{R}^n$, albeit restricted to $[n] \subset V$. But in the previous analysis, since V and $\mathbb{R}^n$ are isomorphic via Φ, this implies that $\Phi_{[n]}$ actually *recovers* V by its identification with $\mathbb{R}^n$; it is as though we've discovered the secret that $[n]$ lives inside the space V.

Very naturally, the generalization of K from $[n]$ to $\mathcal{X}$ takes the view that K is a similarity 'matrix' on $\mathcal{X}$ (though we call it a *kernel*):

Definition 3. Let $\mathcal{X}$ be a nonempty set. A *kernel*[3] is a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We say that K is *positive-definite* if for any finite subsets A of $\mathcal{X}$, the corresponding Gram matrix of A is symmetric and positive-definite.

In the finite case, we produced an embedding $\Phi_{[n]} : [n] \to \mathbb{R}^n$ of $[n]$ satisfying Equation 3 (K corresponds to the standard inner product). As we had only n objects, it is clear that we needed an inner product space with at most n dimensions, hence $\mathbb{R}^n$.

However, in the general case, where $\mathcal{X}$ may be infinite, we might need infinite dimensions. Consider the real vector space of functions from $\mathcal{X}$ to $\mathbb{R}$, denoted by $\mathbb{R}^{\mathcal{X}}$ (vector addition and scalar multiplication are defined pointwise). We construct the map

[3] Unfortunately, the terminology *kernel* can be terribly confusing. Within functional analysis, kernels are often interchangably called *kernel maps*, *kernel functions*, and *positive-definite kernels*. Yet, each of these also take on other meanings depending on author and context. 'Kernels' and 'kernel maps' are almost always synonymous. Sometimes, 'kernel functions' denote kernels of the form $K(x, y) = k(x - y)$, in situations where subtraction is defined. And in many instances, when context is clear, the author cares only for positive-definite kernels, so omitting the specifier *positive-definite*.

Furthermore, *kernels* used in this sense are unrelated to the algebraic kernel of a linear map. The terminology comes from the theory of integral operators. In particular, infinite-dimensional function spaces often have inner products of the form:

$$\langle f, g \rangle = \int_{X \times X} K(x, y) f(x) g(y) \, \mathrm{d}x \mathrm{d}y,$$

where the K here performs the analogous role as $\langle x, \mathbf{K}y \rangle$ in Equation 1. Actually, it performs precisely the role we desire in the general case. These maps K were historically called kernels.

$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$ by:

$$\Phi(x) := K(\cdot, x).$$

For short, let $k_x := \Phi(x)$. Then, Φ maps $\mathcal{X}$ into a subset of $\mathbb{R}^{\mathcal{X}}$; let us denote V by the subspace of $\mathbb{R}^{\mathcal{X}}$ that $\Phi(\mathcal{X})$ lives in. That is, $V = \text{span}(\Phi(\mathcal{X}))$ is a linear subspace whose elements $f \in V$ are of the form:

$$f = \alpha_1 k_{x_1} + \cdots + \alpha_n k_{x_n},$$

where $\alpha_i \in \mathbb{R}$ and n ranges over $\mathbb{N}$. We claim that because K is a positive-definite kernel, there exists a well-defined bilinear map on V, corresponding to our desired inner product. Of course, we want the inner product on V to satisfy:

$$\langle k_x, k_y \rangle = K(x, y).$$

But in fact, once we specify this, linearity determines the inner product on general elements $f = \sum_{i=1}^{n_1} \alpha_i k_{x_i}$ and $g = \sum_{j=1}^{n_2} \beta_j k_{y_j}$:

$$\langle f, g \rangle = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \alpha_i \beta_j K(x_i, y_j).$$

This should feel quite familiar, recalling how the Gram matrix of n linearly independent vectors fully determined the inner product in the finite case. However, we must be a little more careful here, as we are not guaranteed that the collection of k_x's are linearly independent. In particular, suppose that f, as follows, is identically zero:

$$0 \equiv f = \sum_{i=1}^{n} \alpha_i k_{x_i}.$$

That is, $f(x) = 0$ for all $x \in \mathcal{X}$. Then, $\langle f, g \rangle$ better equal 0 for all $g \in V$. Notice that it is sufficient to show that $\langle f, k_x \rangle = \langle k_x, f \rangle = 0$ for each $x \in \mathcal{X}$. And indeed, by assumption

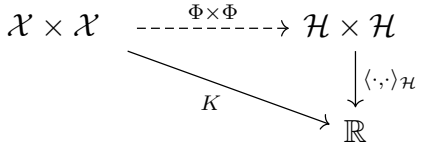
$$\langle k_x, f \rangle = \sum_{i=1}^{n} k_{x_i}(x) = f(x) = 0.$$

This proves that V admits an inner product that is compatible with K in the sense of Equation 3. We will in fact go beyond producing an inner product space V; we can complete the space with respect to the norm induced by the inner product, producing a *Hilbert space*.

That is, let $\mathcal{H}$ be the completion of V by taking equivalence classes of Cauchy sequences on V. We claim that $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ is a collection of functions in the sense that $f(x) < \infty$ for all $f \in \mathcal{H}$ and $x \in \mathcal{X}$.[4] But let us relegate the proof to references, say [P2009, Thm 3.16], for it is not particularly enlightening. However, the consequences are quite important.

First of all, we immediately attain the analog to Proposition 2:

Proposition 4 (Moore-Aronszajn)**.** *Let $\mathcal{X}$ be a set, and $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a positive-definite kernel. Then, there exists a map $\Phi : \mathcal{X} \to \mathcal{H}$ into a Hilbert space such that the following diagram commutes:*



where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product on $\mathcal{H}$.

This is a powerful result because we may now view $\mathcal{X}$ not as an abstract set, but as vectors within a Hilbert space, which carries with it both algebraic and geometric properties that $\mathcal{X}$ now inherits. It is particularly amazing because the only thing we needed required was a positive-definite kernel K to obtain $\mathcal{H}$. And in fact:

[4] Note that in general, it is not the case that the completion of a function space consists of functions. Elements of $L^2(\mathbb{R})$, for example, are equivalence classes of functions.

Fact 5. *The Hilbert space $\mathcal{H}$ induced by K is unique. We call $\mathcal{H}$ the* reproducing kernel Hilbert space (RKHS) *on $\mathcal{X}$ with respect to K. (See [P2009, Prop 3.3]).*

Later, we'll see a characterization: if $\mathcal{H}$ is an RKHS, then $\mathcal{H}$ induces a unique kernel K on $\mathcal{X}$. So, analogous to the finite-dimensional case, we can think of $\Phi(\mathcal{X})$ in $\mathcal{H}$ as the 'canonical form' of $\mathcal{X}$ with respect to the similary measure K.

Despite this, the way we've presented the embedding of $\mathcal{X}$ into $\mathcal{H}$ is somewhat unnatural. While it makes sense to consider the collection of functions k_x, the identification of x with k_x may seem *ad hoc*. But if we venture a bit further into functional analysis, we may obtain a clearer understanding of RKHS's. Along the way, hopefully we can clarify why we might call this identification a feature map.

1.2.3 Feature Maps

Let us once again consider an abstract set $\mathcal{X}$, and a collection of functions over $\mathcal{X}$, say $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$. We may imagine each $f \in \mathcal{H}$ as an observable or a feature, where $f(x)$ gives the *measured value* of the object x by feature f.

As a result, every $x \in \mathcal{X}$ is associated to the Cartesian product of measurements,

$$x \overset{\Psi}{\mapsto} \prod_{f \in \mathcal{H}} f(x). \tag{4}$$

The high-level punchline here will be that $\Psi(x)$ is actually a point in the *dual* of $\mathcal{H}$, following the canonical mapping of $\mathcal{X}$ into the double dual $\mathcal{X}^{**}$. But in the case where $\mathcal{H}$ is a Hilbert space and all the $\Psi(x) \in \mathcal{H}^*$ are bounded (i.e. continuous), then $\mathcal{H}$ is isomorphic to $\mathcal{H}^*$. It turns out that $\Psi(x)$ is naturally identified with k_x, justifying the initial definition $\Phi : x \mapsto k_x$.

Before elaborating on this, let us provide a visual example; perhaps a bit of concreteness will convince the reader why we should even care about the association in Equation 4 in the first place.

Imagine $\mathcal{X}$ is a set of abstract cats, while $\mathcal{H} = \{f_1, \ldots, f_n\}$ represents the pixels within the lens of a camera. Given a cat x, the value $f_i(x)$ gives the color the ith pixel on the camera. Then, the product $\Psi(x)$ in Equation 4 represents the *image* of the cat—the collection of colors all the camera pixels see as the camera takes the picture of cat x.

Here, it makes sense to call the mapping in Equation 4 the *feature map*, for it maps the cat to a collection of features, specifically its colors at different locations. In general, given any function $f : \mathcal{X} \to \mathbb{R}$, we may dually view points of x as functionals, $\mathrm{ev}_x : \mathcal{H} \to \mathbb{R}$, where

$$\mathrm{ev}_x(f) := f(x).$$

In functional analysis, we say that ev_x is the *evaluation at x*, and if it is continuous, then we view it as an element of a larger set of continuous functionals on $\mathcal{H}$, the *continuous dual*, $\mathcal{H}^*$.[5] Let us return to kernels. But whereas in the previous section, the feature map fell out of the construction almost incidentally, here, we begin with it: $\Psi(x) = \mathrm{ev}_x$.

In particular, we equivalently define an RKHS (equivalency will become clear):

Definition 6. Let $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$ be a Hilbert space. It is an RKHS if for all $x \in \mathcal{X}$, the evaluation functional ev_x is continuous.

That is, $\mathcal{H}$ is an RKHS if $\Psi(\mathcal{X}) \subset \mathcal{H}^*$. As made intuitive by the cat images example, $\Psi(\mathcal{X})$ is the representation of $\mathcal{X}$ seen through the lens of features $\mathcal{H}$. And if those features are quite discerning, for any distinct objects $x, y \in \mathcal{X}$, we might expect that their differences will be measurable; there exists some

[5] The earlier point that $\mathcal{H}$ is a collection of functions over $\mathcal{X}$ is equivalent to saying that the evaluation functionals are continuous.

$f \in \mathcal{H}$ such that $f(x) \neq f(y)$. When this is the case, we say that $\mathcal{H}$ *separates points*, and it follows that:

Proposition 7. *The mapping $\Psi : \mathcal{X} \to \mathcal{H}^*$ is injective if and only if $\mathcal{H}$ separates points.*

So in many cases, we may think of Ψ as an embedding of $\mathcal{X}$ into $\mathcal{H}^*$. But then, Riesz representation theorem states that $\mathcal{H} \cong \mathcal{H}^*$. This is a powerful isomorphism, for it implies that every x, which corresponds to a unique $\Psi(x) \in \mathcal{H}^*$, then corresponds to some unique point k_x in $\mathcal{H}$ such that:

$$\Psi(x) = \langle \cdot, k_x \rangle_{\mathcal{H}}.$$

We call k_x the *reproducing kernel for the point* x, and for consistency, we can let $\Phi : x \mapsto k_x$. But via the isomorphism, we may afford to drop the distinction between Ψ and Φ.

Now, as vectors in a Hilbert space, the points k_x inherits the similarity measure from the inner product on $\mathcal{H}$, producing a positive-definite kernel on $\mathcal{X}$:

$$K(x, y) = \langle k_y, k_x \rangle_{\mathcal{H}}.$$

We call K the *reproducing kernel of* $\mathcal{H}$, for $\mathcal{H}$ uniquely determines K. Recalling from before—every kernel K uniquely determines an $\mathcal{H}$—we now see that these two views of RKHS's we've provided are equivalent. And whenever we are given a set $\mathcal{X}$ with a positive-definite kernel, we can reason as though we know that they belong to a Hilbert space $\mathcal{H}$.

1.2.4 In Practice

In many cases, one important limitation is the infeasibility of constructing Φ. And especially in a computational setting, if $\mathcal{H}$ is infinite dimensions, representation of its vectors becomes even more problematic.

Still, we can still take advantage of the existence of Φ; sometimes we can reason about properties of $\mathcal{X}$ as points in $\mathcal{H}$, but pull the analysis back through K.

In the rest of the paper, we'll look at how this may be done for LSH.

1.3 Similarity Measures

LSH generally applies to distance metrics, hence the inequalities discussing $||x_1 - x_2||$. Kernels generally apply to similarity metrics. How do we combine these ideas?

The simplest case is when the data is already on the n-sphere with the usual ℓ_2 distance:

$$\begin{aligned} ||x_1 - x_2|| &= \sqrt{\langle x_1 - x_2, x_1 - x_2 \rangle} \\ &= \sqrt{\langle x_1, x_1 \rangle + \langle x_2, x_2 \rangle - 2\langle x_1, x_2 \rangle} \\ &= \sqrt{2(1 - \langle x_1, x_2 \rangle)}. \end{aligned}$$

In this case, saying $||x_1 - x_2|| < r$ as we did in our original inequality is equivalent to saying $\langle x_1, x_2 \rangle > 1 - r^2/2$. This means the c term becomes squared, which works to our benefit when constructing an ANN apgorithm.

Not every distance metric can be converted into a similarity metric as cleanly. If ℓ_1 distance could be converted into a similarity metric and thence to an ℓ_2 distance without sacrificing cr-near properties, such a transform could be used to perform cr-near neighbor searches with ρ comparable to ℓ_2 cr-NN, but this has been proven impossible [OWZ2009]

One can convert a distance to a similarity using general formulas such as $k(x_1, x_2) = 1 - d(x_1, x_2)^2/\text{diam}(\mathcal{X})^2$ (analogous to the unit sphere) or $k(x_1, x_2) = \exp(-d(x_1, x_2))$ (if we wish to operate elegantly without needing to find the maximum distance), but these are not guaranteed to have nice cr-NN properties.

Often it is better to construct a new metric. For example, the usual distance metric in genetic sequences is string-edit distance, usually with indels costing twice as much as

changes. This is too similar to ℓ_1 to likely permit a good conversion. However, one can also count k-mer frequency and take a normalized inner product (or, for longer k-mers, Hamming distance subtracted from 4^k). These are proper similarity metrics which predict evolutionary homology and analogy of function almost as well as string edit distance [MWS+2012]. The true test of similarity metric quality is an empirical one.

Similarly in images, you can describe distance as the length of the series of edits to transform one image into the other (where each step is visually simple), but this is often not the best predictor of human-judged similarity. A more popular technique is to extract a set of features by convolution with random Gaussian or sinusoidal filters and use some form of normalized inner product. [ZBMM2006]

For a more general solution, one could train a deep neural net on pairs of objects that are and are not "essentially" the same. For example, two photos of the same cat from different angles are essentially the same, but photos of different cats are not. The final output of the net f would then be the probability that two objects are essentially the same. If f is not symmetric, we can fix that by using $f'(x, y) = (f(x, y) + f(y, x))/2$. And if the neural net uses a logistic activation function, that will guarentee $0 < f < 1$ so the function will be trivially positive definite as well.

In general, because the conditions for similarity metrics (symmetry and positive-definiteness) were so intuitively motivated, it is likely that in many cases when we have some notion of similarity, we'll be able to produce a valid positive-definite kernel.

2 KLSH

Now we are ready to put the pieces together into KLSH. Suppose we have a set of objects $\mathcal{X}$ on which the only defined function is a simularity metric $K : \mathcal{X} \times \mathcal{X} \to [0, 1]$ (with $K(x, x) = 1$). In the most common example, $\mathcal{X}$ is a collection of images and K is whatever kernel the computer vision community recommends.

Note that $\mathcal{X}$ is *not* generally a vector space. We cannot add two images, nor take their inner product. We cannot generate a random image drawn from a Gaussian distribution. These things are not defined. Furthermore, we might be unable to find a function Φ that would map the image into a vector space. We know one exists, but we do not have it, nor do we know any interesting properties of the Hilbert space it maps into.

Nevertheless, we would like to apply LSH techniques that are defined in the kernel space to search problems in the object space.

Specifically, we will start by applying LSH of the unit sphere (which is where we are, by the $K(x, x) = 1$ definition earlier) by hyperplane carving.

2.1 Hyperplane Carving

The simplest way to do LSH on a unit sphere in $\mathbb{R}^n$ with inner product similarity is to carve the sphere in half with a random hyperplane. This is equivalent to taking a vector perpendicular to that hyperplane (traditionally called g) and taking the sign of its inner product with the vector in question. The vector g must be drawn at random from a spherical Gaussian distribution. This means that regardless of x_1 and x_2, whether $h(x_1) = h(x_2)$ is determined by whether g falls into the region which makes it so, where the size of that region is determined by $\langle x_1, x_2 \rangle$.

2.2 Attempt at KLSH

Let's try to kernelize the ball-carving LSH method. Given $\mathcal{X}$ and a normalized positive-

definite kernel K,[6] one very natural way is to consider the embedding $\Phi(\mathcal{X}) \subset \mathcal{H}$ inside the RKHS.

In particular, because K is normalized, all the points $\Phi(x)$ fall onto the unit sphere in $\mathcal{H}$, so it seems that we just need to ball-carve the sphere in $\mathcal{H}$. In the usual LSH situation in $\mathbb{R}^n$, we drew a Gaussian $g \sim \mathcal{N}(0,1)^n$ and hashed according the function:

$$h(x) = \text{sign } \langle x, g \rangle.$$

So it seems that we just need to replicate this within $\mathcal{H}$ instead of $\mathbb{R}^n$:

$$h(x) = \text{sign } \langle \Phi(x), g \rangle_{\mathcal{H}},$$

where g is drawn according to a Gaussian in $\mathcal{H}$. But here, we run into two problems:

1. if $\mathcal{H}$ is infinite-dimensional, it does not have a canonical Gaussian distribution,
2. if we don't have Φ, how can we compute the inner product of $\Phi(x)$ with g?

Let's take a closer look at why there is no canonical Gaussian distribution—in fact, this will provide us with some heuristic to think about how Φ might embed $\mathcal{X}$ into $\mathcal{H}$. From that insight, we will be able to reduce the infinite-dimensional problem down to a finite-dimensional one, which we essentially already know how to solve.

2.3 Using the Kernel

The main barrier is that we still don't have the explicit feature map, so drawing Gaussians as we usually would is not helpful—we must work indirectly through the kernel map. A secondary barrier is that $\mathcal{N}(0, I)$ is not defined if $\mathcal{H}$ is infinite-dimensional, but we will assume finite dimension for now and come back to the infinite case later.

The clever solution that [K2012] came up with relies on the central limit theorem. Suppose $\Phi(\mathcal{X})$ has mean μ and covariance Σ. After drawing t i.i.d. samples, $\Phi(X_1), \ldots, \Phi(X_t)$, CLT implies that the random variable:

$$\Sigma^{-1/2} Z := \Sigma^{-1/2} \left(\sqrt{t} \cdot \frac{1}{t} \sum_{i=1}^{t} \Phi(X_i) - \mu \right)$$

will converge to a standard Gaussian as t approaches ∞.

Let z be a realization of Z. Then, it follows that we could ball-carve using the hash:

$$h(x) = \text{sign } \left\langle \Phi(x), \Sigma^{-1/2} z \right\rangle.$$

Let's examine the term $\left\langle \Phi(x), \Sigma^{-1/2} z \right\rangle$; as before, λ_i, v_i are the eigenvalue/vectors of Σ. Additionally, for clarity, we will assume that $\mu = 0$, as it is a just small technical detail to center the data. Then, the inner product term expands out to be:

$$\langle \Phi(x), \Sigma^{-1/2} z \rangle = \sum_{i=1}^{d_\Phi} \frac{1}{\sqrt{\lambda_i}} \langle \Phi(x), v_i \rangle \langle v_i, z \rangle.$$

We still can't quite compute this because we don't have the λs and d_Φ might be infinite. So we truncate the operation to N eigenvalues, making a finite sum regardless of $\mathcal{H}$. These eigenvalues can be approximated using a size N random sample S from $\Phi(\mathcal{X})$ by taking the kernel matrix for the sample and manipulating it [S1998]. This both resolves our foundational concerns about defined gaussians and gives us a formula we can

[6] One can normalize any positive-definite kernel $K(x, y)$ by:

$$K'(x, y) = \frac{K(x, y)}{\sqrt{K(x, x) K(y, y)}}$$

.

compute, specifically [K2012]:

$$h(x) = \text{sign } \langle \Phi(x), \Sigma_S^{-1/2} z \rangle = \text{sign } \left(\sum_{i=1}^{t} w_i \cdot K(x_i, x) \right), \quad (5)$$

where $w = \hat{K}^{-1/2} \mathbb{1}_S$, $\hat{K}$ is the centered kernel matrix for S, and $\mathbb{1}_S$ just the indicator vector for the sample S in $\mathcal{X}$. By randomly choosing different subcollections of t points from within S, we obtain our family of random hashes for our arbitrary set $\mathcal{X}$.

We have dropped terms, specifically $\sum_{i=N+1}^{d_\Phi} \frac{1}{\sqrt{\lambda_i}} \langle \Phi(x), v_i \rangle \langle v_i, z \rangle$. Informally, we can say the resultant error is bounded by λ_{N+1}, because if some combination of later terms exceeded it, that would be the new v_{N+1}. Since $\lambda_{N+1} < \lambda_N$ which we have, we can bound the error as we go. If the error is unacceptably large, we can increase N and re-compute. For a formal treatment, see [J2015]'s supplemental materials.

Conceptually, we are taking the first N principle components of $\Phi(\mathcal{X})$ to serve as an approximation, and for this reason the technique is sometimes called "KPCA".

3 Data-Dependent KLSH

3.1 Data-Dependent LSH

Random hyperplanes are not the most efficient form of LSH for a unit sphere. They achieve only $\rho = 1/c$ while the best-known technique achieves $\rho = 1/(2c^2 - 1)$ [AR15]. This is better than the theoretical limit and is made possible by basing the hashing functions on the data.

The first step is to replace the hyperplane-carving hash function with a cap-carving one. Instead of checking the sign of $\langle g, q \rangle$, we test if it is greater than $d^{1/4}$. And instead of simply reporting "no" if not, we keep picking gs according to some deterministic psuedorandom rule until we find one that is.

If the data is spread evenly over the sphere, then we expect a very small fraction of points in each cap. To speak somewhat imprecisely, the inner product is the sum of the product in each co-ordinate. Each of those products has expected value zero (by symmetry) and a very small variance (because of the cap on the total length). Summing many of them drives the variance down further and allows us to apply a Chernoff bound.

The low fraction of points means that if a query and a database point are in the same cap, it's probably because they're close to one another. Note that we are assuming the data is sparse compared to the possibility space.

This allows us to get $\rho = 1/(2c^2 - 1)$, but only if the data is random or close to random.

3.1.1 Approximate Evenness

What we mean by "close to random" is that no cap exceeds its expected number of points by more than a constant factor.

This is not something we are likely to have. First, a $[0, 1]$-ranged similarity function will map all points into the "upper right" orthant. More worryingly, we can expect any real-world data to be extremely clumpy.

It is possible to extract a reasonable number of "lumps" and leave an approximately even residue by taking a small number of test points from the dataset, computing their similarities to all other points and noticing if any are close to many. This is a slightly superlinear time preperatory step that tells us about troublesome balls centered on known datapoints.

The standard technique is then to remove those balls and process them separately, by ignoring their sphere-segment structure and slicing them into new spheres. While complex, the operation is time-efficient. Unfortunately, the process of sphere-slicing a ball likely would not be possible in kernel space.

3.2 Smaller Caps

What might be possible would be to first carve the dense regions with smaller caps, then the rest of the ball with regular size ones. If we increase the threshold on $\langle g, q \rangle$ to decrease the diameter of the cap proportionally to the diameter of the ball, this should get us something close to the correct number of points.

It may still be necessary to recurse on dense regions inside dense regions, but no more so than in the original algorithm.

Unlike sphere carving, this should be possible to implement in kernel space. The points can be drawn by taking the center (a point in the database!), adding a normal vector resized to the ball's radius, and normalizing the result. These operations are straightforward to reduce to computable kernel operations.

We don't have space to fully develop this idea, which we have neither proved nor tested empirically. But, from everything we can see, there have been no attempts to combine kernelized and data-dependent LSH, so we are optimistic about this approach.

References

[A1950] Aronszajn, Nachman. *Theory of reproducing kernels.* Transactions of the American mathematical society 68.3 (1950): 337-404.

[AR15] Andoni, Alexandr and Razenshteyn, Ilya. *Optimal Data-Dependent Hashing for Approximate Near Neighbors.* CoRR abs/1501.01062. 2015

[E2016] Eldredge, Nathaniel *Analysis and probability on infinite-dimensional spaces.* arXiv preprint arXiv:1607.03591 (2016).

[G2013] Gretton, Arthur. *Introduction to RKHS, and some simple kernel algorithms.* Adv. Top. Mach. Learn. Lecture Conducted from University College London (2013).

[H2008] Hofmann, Thomas, Bernhard Schlkopf, and Alexander J. Smola. *Kernel methods in machine learning.* The annals of statistics (2008): 1171-1220.

[J2015] Jiang, Ke, Qichao Que, and Brian Kulis. *Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[K2012] Kulis, Brian, and Kristen Grauman. *Kernelized locality-sensitive hashing.* IEEE Transactions on Pattern Analysis and Machine Intelligence 34.6 (2012): 1092-1104.

[L2012] Lifshits, Mikhail. *Lectures on Gaussian processes.* Lectures on Gaussian Processes. Springer Berlin Heidelberg, 2012. 1-117.

[MWS+2012] Mahmood K, Webb GI, Song J, Whisstock JC, Konagurthu AS. *Efficient large-scale protein sequence comparison and gene matching to identify orthologs and co-orthologs.* Nucleic Acids Research. 2012;40(6).

[OWZ2009] O'Donnell, Ryan and Wu, Yi and Zhou, Yuan. *Optimal lower bounds for locality sensitive hashing (except when q is tiny).* CoRR abs/0912.0250. 2009.

[P2009] Paulsen, Vern I., and Mrinal Raghupathi. *An introduction to the theory of reproducing kernel Hilbert spaces.* Vol. 152. Cambridge University Press, 2016.

[S1998] Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Mller. *Nonlinear component analysis as a kernel eigenvalue problem.* Neural computation 10.5 (1998): 1299-1319.

[ZBMM2006] Zhang and Berg and Maire and Malik, *SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), 2006, pp. 2126-2136.