Deconstructing Kernel Machines

Mohsen Ali, Muhammad Rushdi†, and Jeffrey Ho

Department of Computer and Information Science and Engineering, University of Florida
†Department of Biomedical and Systems Engineering, Cairo University, Giza, Egypt
moali@cise.ufl.edu, mrushdi@eng.cu.edu.eg, jho@cise.ufl.edu

Abstract. This paper studies the following problem: Given an SVM (kernel)-based binary classifier $\mathbf{C}$ as a black-box oracle, how much can we learn of its internal working by querying it? Specifically, we assume the feature space $\mathbb{R}^d$ is known and the kernel machine has m support vectors such that $d > m$ (or $d >> m$), and in addition, the classifier $\mathbf{C}$ is laconic in the sense that for a feature vector, it only provides a predicted label (± 1) without divulging other information such as margin or confidence level. We formulate the problem of understanding the inner working of $\mathbf{C}$ as characterizing the decision boundary of the classifier, and we introduce the simple notion of bracketing to sample points on the decision boundary within a prescribed accuracy. For the five most common types of kernel function, linear, quadratic and cubic polynomial kernels, hyperbolic tangent kernel and Gaussian kernel, we show that with $\mathbf{O}(dm)$ number of queries, the type of kernel function and the (kernel) subspace spanned by the support vectors can be determined. In particular, for polynomial kernels, additional $\mathbf{O}(m^3)$ queries are sufficient to reconstruct the entire decision boundary, providing a set of quasi-support vectors that can be used to efficiently evaluate the deconstructed classifier. We speculate briefly on the future application potential of deconstructing kernel machines and we present experimental results validating the proposed method.

Keywords: deconstruction, support vector machines, RBF

1 Introduction

This paper proposes to investigate a new type of learning problems we called deconstructive learning. While the ultimate objective of most learning problems is the determination of classifiers from labeled training data, for deconstructive learning, the objects of study are the classifiers themselves. As its name suggests, the goal of deconstructive learning is to deconstruct a given classifier $\mathbf{C}$ by determining and characterizing (as much as possible) the full extent of its capability, revealing all of its powers, subtleties and limitations. Since classifiers in machine learning come in a variety of forms, deconstructive learning correspondingly can be formulated and posed in many different ways. This paper focuses on a family of binary classifiers based on support vector machines [1], and deconstructive learning will be formulated and studied using geometric and algebraic methods without recourse to probability and statistics. Specifically, the (continuous) feature space in which the classifier $\mathbf{C}$ is defined is assumed to be a d-dimensional vector space $\mathbb{R}^d$, and the classifier $\mathbf{C}$ is given as a binary-valued function $\mathbf{C} : \mathbb{R}^d \to \{-1, +1\}$, indicating the class assignment of each feature $\mathbf{x} \in \mathbb{R}^d$. As a kernel machine, $\mathbf{C}$ is specified by a set of m support vectors $\mathbf{y}_1, \cdots, \mathbf{y}_m \in \mathbb{R}^d$ and a kernel function $\mathbf{K}(\mathbf{x}, \mathbf{y})$ such that the decision function $\mathbf{\Psi}(\mathbf{x})$ is given as the sum

$$\mathbf{\Psi}(\mathbf{x}) = \omega_1 \mathbf{K}(\mathbf{x}, \mathbf{y}_1) + \cdots \omega_m \mathbf{K}(\mathbf{x}, \mathbf{y}_m), \tag{1}$$

where $\omega_1, \cdots, \omega_m$ are the weights. With the bias $\mathbf{b}$,

$$\mathbf{C}(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{\Psi}(\mathbf{x}) \le \mathbf{b}, \\ -1 & \text{if } \mathbf{\Psi}(\mathbf{x}) > \mathbf{b}. \end{cases} \tag{2}$$

The classifier $\mathbf{C}$ is also assumed to be laconic in the sense that except for the binary label, it does not divulge any other potentially useful information such as margin or confidence level. With these assumptions, we formulate the problem of deconstructing $\mathbf{C}$ through the following list of four questions (ordered in increasing difficulty)

- Can the kernel function $\mathbf{K}(\mathbf{x}, \mathbf{y})$ be determined?
- Can the subspace $\mathbf{S_Y}$ spanned by the support vectors be determined?
- Can the number m of support vectors be determined?
- Can the support vectors themselves be determined?

Without loss of generality, we will henceforth assume $\mathbf{b} = 1$. Therefore, if the support vectors and the kernel function are known, the weights ω_i can be determined completely given enough points $\mathbf{x}$ on the decision boundary

$$\mathbf{\Sigma} = \{\mathbf{x} \parallel \mathbf{x} \in \mathbb{R}^d,\ \mathbf{\Psi}(\mathbf{x}) = \mathbf{b}\ \}. \tag{3}$$

That is, a kernel machine $\mathbf{C}$ can be completely deconstructed if its support vectors and kernel function are known.

The four questions above are impossible to answer without further quantification on the type of kernel function and the number of support vectors. In this paper, we assume 1) the unknown kernel function belongs to one of the following five types: polynomial kernels of degree one, two and three (linear, quadratic and cubic kernels), hyperbolic tangent kernel and RBF kernel, and 2) the number of support vectors is less than the feature dimension, $d > m$ (or $d >> m$) and they are linearly independent. For most applications of kernel machines, these two assumptions are not particularly restrictive since the five types of kernel are arguably among the most popular ones. Furthermore, as the feature dimensions are often very high and the support vectors are often thought to be a small number of the original training features that are critical for the given classification problem, it is generally observed that $d > m$. With these two assumptions, the method proposed in this paper shows that the first three questions can be answered affirmatively. While the last question cannot be answered for transcendental kernels, we show that using recent results on tensor decomposition (e.g., [2]), a set of quasi-support vectors can be computed for a polynomial kernel that recover the decision boundary exactly.

Given the laconic nature of $\mathbf{C}$, it seems that the only effective approach is to probe the feature space by locating points on the decision boundary $\mathbf{\Sigma}$ and to answer the above questions using local geometric features computed from these sampled points.

More precisely, the proposed algorithm takes the classifier $\mathbf{C}$ and a small number of positive features in $\mathbb{R}^d$ as the only inputs. Starting with these small number of positive features, the algorithm proceeds to explore the feature space by generating new features and utilizing these new features and their class labels provided by $\mathbf{C}$ to produce points on the decision boundary. The challenge is therefore to use only a comparably small number of sampled features (i.e., queries to $\mathbf{C}$) to learn enough about $\mathbf{\Sigma}$ in order to answer the questions, and our main contribution is an algorithm that has complexity (to be defined later) linear in the dimension d of the ambient space.

Sampling points on $\mathbf{\Sigma}$ can be accomplished easily using bracketing, the same idea used in finding the root of a function (e.g., [3]). Given a pair of positive and negative features (PN-pair), the intersection of $\mathbf{\Sigma}$ and the line segment joining the two features cannot be empty, and bracketing allows at least one such point on $\mathbf{\Sigma}$ to be determined up to any prescribed precision. Using bracketing as the main tool, the first two questions can be answered by exploring the geometry of $\mathbf{\Sigma}$ in two different ways. First, the decision boundary $\mathbf{\Sigma}$ is given as the implicit surface of the multi-variate function, $\mathbf{\Psi}(\mathbf{x}) = \mathbf{b}$. With high-dimensional features, it is difficult to work directly with $\mathbf{\Sigma}$ or $\mathbf{\Psi}(\mathbf{x})$; instead, the idea is to examine the intersection of $\mathbf{\Sigma}$ with a two-dimension subspace formed by a PN-pair. The locus of such intersection is in fact determined by the kernel function, and by computing such intersection, we can ascertain the kernel function on this two-dimensional subspace. For the second question, the answer is to be found in the normal vectors of the hypersurface $\mathbf{\Sigma}$. Using bracketing, the normal vector at a given point on $\mathbf{\Sigma}$ can be determined, again in principle, up to prescribed precision. From the parametric forms of the kernel functions, it readily follows that the normal vectors of $\mathbf{\Sigma}$ are generally quite well-behaved in the sense that they either belong to the kernel subspace $\mathbf{S_Y}$ spanned by the support vectors or they are affine-translations of the kernel subspace $\mathbf{S_Y}$. For the former, a quick application of singular value decomposition immediately yields the kernel subspace $\mathbf{S_Y}$, and for the latter, the kernel subspace $\mathbf{S_Y}$ can be computed via a rank-minimization problem that can be solved (in many cases) as a convex optimization problem with the nuclear norm. If we define the complexity of the algorithm as the required number of sampled points in the feature space, it will be shown that the complexity of the proposed method is essentially $\mathbf{O}(dm)$ as it requires $\mathbf{O}(m)$ normal vectors to determine the m-dimensional kernel subspace and $\mathbf{O}(d)$ points to determine the normal vector at a point in $\mathbb{R}^d$. The constant depends on the number of steps used for bracketing, and if the features are assumed to be drawn from a bounded subset in $\mathbb{R}^d$, this constant is then independent of the dimension d.

We note that for a polynomial kernel of degree D, its decision function $\mathbf{\Psi}(\mathbf{x})$ is a degree-D polynomial with d variables. Therefore, in principle, $\mathbf{C}$ can be deconstructed by fitting a polynomial of degree D in $\mathbb{R}^d$ given enough sampled points on $\mathbf{\Sigma}$. However, this solution is in general not useful because it does not extend readily to transcendental kernels. Furthermore, the number of required points is in the order of d^D, and correspondingly, a direct polynomial fitting would require the inversion of a large dense (Vandermonde) matrix that is in the order of $d^D \times d^D$. With a moderate dimension of $d = 100$ and $D = 3$, this would require 10^6 points and the inversion of a $10^6 \times 10^6$ dense matrix. Our method, on the other hand, encompasses both the transcendental and polynomial kernels and at the same time, it avoids the direct polynomial fitting in $\mathbb{R}^d$ and has the overall complexity that is linear in d, making it a truly practical algorithm.

We conclude the introduction with a brief discussion on the potential usefulness of deconstructive learning, providing several examples that illustrate its significance in terms of its future prospects for theoretical development as well as practical applications. The geometric approach taken in this paper shares some visible similarities with low-dimensional reconstruction problems studied in computational geometry [4], and in fact, it is partially inspired by various 3D surface reconstruction algorithms studied in computational geometry (and computer vision) [5] [6]. However, due to the high dimensionality of the feature space, deconstructive learning offers a brand new setting that is qualitatively different from those low-dimensional spaces studied in computational geometry and various branches of geometry in mathematics. High dimensionality of the feature space has been a hallmark of machine learning, a realm that has not be actively explored by geometers, mainly for the lack of interesting examples and motivation. Perhaps deconstructive learning's emphasis on the geometry of the decision boundary in high dimensional space and its connection with machine learning could provide stimulating examples or even counterexamples unbeknown to the geometers, and therefore, provide the needed motivation for the development of new type of high-dimensional geometry [7].

On the practical side, we believe that deconstructive learning can provide a greater flexibility to the users of AI/machine learning products because it allows the users to determine the full extent of an AI/ML program/system, and therefore, create his/her own adaptation or modification of the given system for specific and specialized tasks. For example, once a kernel machine has been deconstructed, it can be subject to various kinds of improvement and upgrade in terms of its application scope, runtime efficiency and others. Imagine a kernel machine that was originally trained to recognize humans in images. By deconstructing the kernel machine and knowing its kernel type and possibly its support vectors, we can improve and upgrade it to a kernel machine that also recognizes other objects such as vehicles, scenes and other animals. The actual process of upgrading the kernel machine can be managed using existing methods such as incremental SMV or online SVM [8] [9], and at the same time, its efficiency can be improved using, for example, suitable parallelization. This provides the users with the unprecedent capability of modifying a kernel machine without access to its source code, something that to the best of our knowledge has not been studied or reported in the machine learning literature. As the kernel machines are often the main workhorse of many existing machine learning programs/systems, the ability to deconstruct a given kernel machine should have other surprising and interesting consequences and applications unforeseen at this point. Furthermore, in the context of adversarial learning [10] [11], deconstructive learning allows a kernel machine to be defeated and its deficiencies revealed. For example, how would an UAI reviewer know that a submitted binary code of a paper really does implement the algorithm proposed in the paper, not some clever implementation of a kernel machine? Deconstructive learning proposed in this paper offers a possible solution without the need to ask for the source code[1]. For more interesting examples in this direction, we leave it to the reader's imagination. Finally,

[1] Asking for source code is certainly not an affordable panacea for all tech problems.

perhaps the most compelling reason (to the authors) for studying deconstructive learning is inscribed by the famous motto uttered by David Hilbert more than eighty years ago: we must know and we will know! Indeed, when presented with a black-box classifier (especially the one with great repute), we have found the problem of determining the secret of its inner working by simply querying it both fascinating and challenging, a problem with its peculiar elegance and charm.

Related Work To the best of our knowledge, there is no previous work on deconstructing general kernel machines as described above. However, [10] studied the problem of deconstructing linear classifiers in a context that is slightly different from ours. This corresponds to linear kernel machines and consequently, their scope is considerably narrower than ours as (single) linear classifiers are relatively straightforward to deconstruct. Active learning (e.g., [12] [13] [14]) shares certain similarities with deconstructive learning ($\mathbf{DL}$) in that it also has a notion of querying a source. However, the main distinction is their specificities and outlooks: for active learning, it is general and relative while for $\mathbf{DL}$, it is specific and absolute. More precisely, for active learning, the goal is to determine a classifier from a concept class with some prescribed (PAC-like) learning error bound using samples generated from the underlying joint distribution of feature and label. In this model, the learning target is the joint distribution and the optimal learned classifier is relative to the given concept class. On the other hand, in $\mathbf{DL}$, the learning target is a given classifier and the classifier defines an absolute partition of the feature space into two disjoint regions of positive and negative features. Furthermore, the classifier is assumed to belong to a specific concept class (e.g., kernel machines with known types of kernel function) such that the goal of $\mathbf{DL}$ is to identify the classifier within the concept class using the geometric features of the decision boundary. In this absolute setting, geometry replaces probability as the joint feature-label distribution gives way to the geometric notion of decision boundary as the main target of learning. In particular, bracketing is a fundamentally geometric notion that is generally incompatible with a probabilistic approach, and with it, $\mathbf{DL}$ possesses a much more efficient and precise tool for exploring the spatial partition of the feature space, and consequently, it allows for a direct and geometric approach without requiring much probability.

2 Preliminaries

Let $\mathbb{R}^d$ denote the feature space equipped with its standard Euclidean inner product, and for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\|\mathbf{x} - \mathbf{y}\|^2 = (\mathbf{x} - \mathbf{y})^\top(\mathbf{x} - \mathbf{y})$. For the kernel machines studied in this paper, we assume their kernel functions are of the following five types:

Linear Kernel	$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top\mathbf{y}$,
Quadratic Kernel	$\mathbf{K}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top\mathbf{y} + 1)^2$,
Cubic Kernel	$\mathbf{K}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top\mathbf{y} + 1)^3$,
Hyperbolic Tangent Kernel	$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \tanh(\alpha\,\mathbf{x}^\top\mathbf{y} + \beta)$,
Gaussian Kernel	$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2})$,

for some constants α, β, σ. We will further refer to the three polynomial kernels and the hyperbolic tangent kernel as the Type-A kernels and the Gaussian kernel as the Type-B kernel. This particular taxonomy is based on their forms that can be generically written as

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = f(\mathbf{x}^\top \mathbf{y}), \qquad \mathbf{K}(\mathbf{x},\mathbf{y}) = g(\|\mathbf{x}-\mathbf{y}\|^2),$$

for some smooth univariate function $f, g : \mathbb{R} \to \mathbb{R}$.

Given the forms of the kernel function, an important consequence is that the decision boundary $\mathbf{\Sigma}$ is determined in large part by its intersection with the kernel subspace $\mathbf{S_Y}$ spanned by the support vectors. More precisely, for $\mathbf{x} \in \mathbb{R}^d$, let $\overline{\mathbf{x}}$ denote the projection of $\mathbf{x}$ on $\mathbf{S_Y}$:

$$\overline{\mathbf{x}} = \arg\min_{y \in \mathbf{S_Y}} \|\mathbf{x} - y\|^2.$$

For Type-A kernel $\mathbf{K}(\mathbf{x},\mathbf{y}) = f(\mathbf{x}^\top \mathbf{y})$, we have $\mathbf{K}(\mathbf{x}, \mathbf{y}_i) = \mathbf{K}(\overline{\mathbf{x}}, \mathbf{y}_i)$ for every support vector $\mathbf{y}_i$. In particular, $\overline{\mathbf{x}}$ is on the decision boundary if and only if $\mathbf{x}$ is. For Type-B kernels, we have (using Pythagorean theorem with $q^2 = \|\mathbf{x}\|^2 - \|\overline{\mathbf{x}}\|^2$)

$$\mathbf{K}(\mathbf{x}, \mathbf{y}_i) = g(\|\mathbf{x} - \mathbf{y}_i\|^2) = g(\|\overline{\mathbf{x}} - \mathbf{y}_i\|^2 + q^2),$$

and with the Gaussian kernel g, we have $g(\|\overline{\mathbf{x}} - \mathbf{y}_i\|^2 + q^2) = e^{-\frac{q^2}{2\sigma^2}} g(\|\overline{\mathbf{x}} - \mathbf{y}_i\|^2)$. It then follows that for any $\mathbf{x} \in \mathbf{\Sigma}$, its projection $\overline{\mathbf{x}}$ on $\mathbf{S_Y}$ must satisfy

$$\mathbf{\Psi}(\overline{\mathbf{x}}) = e^{\frac{q^2}{2\sigma^2}}\, \mathbf{\Psi}(\mathbf{x}) = e^{\frac{q^2}{2\sigma^2}}\, \mathbf{b}.$$

In other words, the decision boundary $\mathbf{\Sigma}$ is essentially determined by the level-sets of $\mathbf{\Psi}(\mathbf{x})$ on the kernel subspace $\mathbf{S_Y}$.

Since the decision boundary $\mathbf{\Sigma}$ is given as the implicit surface $\mathbf{\Psi}(\mathbf{x}) = \mathbf{b}$, a normal vector $\mathbf{n}(\mathbf{x})$ at a point $\mathbf{x} \in \mathbf{S}$ can be given as the gradient of $\mathbf{\Psi}(\mathbf{x})$:

$$\mathbf{n}(\mathbf{x}) = \nabla \mathbf{\Psi}(\mathbf{x}) = \sum_{i=1}^{m} \omega_i \nabla_{\mathbf{x}} \mathbf{K}(\mathbf{x}, \mathbf{y}_i). \tag{4}$$

For the two types of kernels we are interested in, their gradient vectors assume the following forms:

$$\nabla_{\mathbf{x}} \mathbf{K}(\mathbf{x},\mathbf{y}) = f'(\mathbf{x}^\top \mathbf{y})\mathbf{y}, \tag{5}$$

$$\nabla_{\mathbf{x}} \mathbf{K}(\mathbf{x},\mathbf{y}) = 2g'(\|\mathbf{x}-\mathbf{y}\|^2)\,(\mathbf{x}-\mathbf{y}). \tag{6}$$

Using the above formulas, it is clear that for Type-A kernels, the normal vector $\mathbf{n}(\mathbf{x})$ depends on $\mathbf{x}$ only through the coefficients in the linear combination of the support vectors, while for Type-B kernels, $\mathbf{x}$ actually contributes to the vectorial component of $\mathbf{n}(\mathbf{x})$. It will follow that an important element in the deconstruction method introduced below is to exploit this difference in how the normal vectors are computed for the two types of kernels. For example, for a polynomial kernel of degree D, a normal vector at a point $\mathbf{x} \in \mathbf{\Sigma}$ is

$$\mathbf{n}(\mathbf{x}) = \sum_{i=1}^{m} D\,\omega_i\,(\mathbf{x}^\top \mathbf{y}_i + 1)^{D-1}\,\mathbf{y}_i. \tag{7}$$

As a special case, for linear kernel $D = 1$, we have

$$\mathbf{n}(\mathbf{x}) = \sum_{i=1}^{m} \omega_i \, \mathbf{y}_i,$$

that is independent of $\mathbf{x}$. For the Gaussian kernel, we have

$$\mathbf{n}(\mathbf{x}) = \sum_{i=1}^{m} -\frac{\omega_i}{\sigma^2} \exp(-\frac{\|\mathbf{x}-\mathbf{y}_i\|^2}{2\sigma^2}) \, (\mathbf{x}-\mathbf{y}_i). \tag{8}$$

3 Deconstruction Method

The deconstruction algorithm requires two inputs: 1) an SVM-based binary classifier $\boldsymbol{\Psi}(\mathbf{x})$ that uses one of the five kernel types indicated above, and 2) a small number of positive and negative features. The algorithm uses the small number of input features to generate other pairs of positive and negative features. For a pair $\mathbf{p}, \mathbf{n}$ of positive and negative features (a PN-pair), we can be certain that the line segment joining $\mathbf{p}, \mathbf{n}$ must intersect the decision boundary in at least one point. Using bracketing, we can locate one such point $\mathbf{x}$ on the decision boundary within any given accuracy, i.e., we can use bracketing to obtain a PN-pair $\mathbf{p}, \mathbf{n}$ such that $\|\mathbf{p} - \mathbf{n}\| < \epsilon$ for some prescribed $\epsilon > 0$. With a small enough ϵ, the midpoint between $\mathbf{p}, \mathbf{n}$ can be considered approximately as a sampled point $\mathbf{x}$ on $\boldsymbol{\Sigma}$ and its normal vector can then be estimated. The algorithm proceeds to sample a collection of points and their normals on the decision boundary $\boldsymbol{\Sigma}$, and using this information, the algorithm first computes the kernel subspace $\mathbf{S}_{\mathbf{Y}}$ and this step separates the Type-A kernels from the Type-B kernels (Gaussian kernel). The four Type-A kernels can further be identified by computing the intersection of $\boldsymbol{\Sigma}$ with a few randomly chosen two-dimensional subspaces. These two steps provide the affirmative answers to the first three questions in the introduction. For polynomial kernels, we can determine a set of quasi-support vectors that provide the exact recovery of the decision boundary $\boldsymbol{\Sigma}$. However, no such results for the two transcendental kernels are known at present and we leave its resolution to future research.

3.1 Bracketing

Given a PN-pair, $\mathbf{p}, \mathbf{n}$, the decision boundary must intersect the line segment joining the two features. Therefore, we can use bracketing, the well-known root-finding method (e.g., [3]), to locate the point on $\boldsymbol{\Sigma}$. Note that bracketing does not require the function value, only its sign. This is compatible with our classifier $\mathbf{C}$ that only gives binary values ± 1. In particular, if we bisect the interval in each step of bracketing, the length of the interval is halved at each iteration, and for a given precision requirement $\epsilon > 0$, the number of steps required to reach it is in the order of $|\log \epsilon|$. If we further assume that the features are generated from a bounded subset of $\mathbb{R}^d$ (which is often the case) with diameter less than K, then for any PN-pair $\mathbf{p}, \mathbf{n}$, bracketing terminates after at most

$$\log_2 K - \log_2 \epsilon + 1 \tag{9}$$

steps, a number that is independent of the ambient dimension d.

3.2 Estimating Normal Vectors

Given the pair $\mathbf{p}, \mathbf{n}$, let $\bar{\mathbf{p}}, \bar{\mathbf{n}}$ denote the two points near $\mathbf{\Sigma}$ after the bracketing step and $\mathbf{x}$ denote their midpoint. To estimate the normal vector at $\mathbf{x}$, we use the fact that the (unknown) kernel function is assumed to be smooth and $\mathbf{\Sigma}$ is a level-surface of the decision function $\mathbf{\Psi}(\mathbf{x})$ that is a linear combination of smooth functions. Consequently, a randomly chosen point on $\mathbf{\Sigma}$ is almost surely non-singular [15] in that it has a small neighborhood in $\mathbf{\Sigma}$ that can be well-approximated using a linear hyperplane (its tangent space) in $\mathbb{R}^d$. Accordingly, we will estimate the normal vector at $\mathbf{x}$ by linearly fitting a set of points on $\mathbf{\Sigma}$ that belong to a small neighborhood of $\mathbf{x}$. More specifically, we chose a small $\delta > \epsilon > 0$ and generate PN-pairs on the sphere centered at $\mathbf{x}$ with radius δ. Using bracketing and the convexity of the ball enclosed by the sphere, we obtain PN-pairs that are near $\mathbf{\Sigma}$ and no more than δ away from $\mathbf{x}$. Taking the midpoint of these PN-pairs, we obtain a set of randomly generated $\mathbf{O}(d)$ points on $\mathbf{\Sigma}$. We linearly fit a $(d-1)$-dimensional hyperplane to these points and the normal vector is then computed as the eigenvector associated to the smallest eigenvalues of the normalized covariance matrix. The result can be further sharpened by repeating the step over multiple δ and taking the (spherical) average of the unit normal vectors. However, in practice, we have observed that good normal estimates can be consistently obtained using one small $\delta \approx 10^{-3}$ (with $\epsilon = 10^{-6}$) and $2d$ sampled points[2].

3.3 Determining Kernel Subspace $\mathbf{S_Y}$

To determine the kernel subspace $\mathbf{S_Y}$, we will use the formulas for the normal vectors given in Equations 5 and 6. Assume that we have sampled $s > m$ points on $\mathbf{\Sigma}$ and their corresponding normal vectors. Let $\mathbb{N}, \mathbf{X}$ denote the following two matrices

$$\mathbf{X} = [\mathbf{x}_1\, \mathbf{x}_2 \ldots \mathbf{x}_s], \qquad \mathbb{N} = [\mathbf{n}_1\, \mathbf{n}_2 \ldots \mathbf{n}_s] \tag{10}$$

that horizontally stack together the points $\mathbf{x}_i$ and their normal vectors $\mathbf{n}_i$, respectively. If all $\mathbf{n}_i$ are correctly recovered (without noise), we have the following:

- For Type-A kernels, $\mathbf{n}_i \in \mathbf{S_Y}$, i.e., $\mathbf{n}_i$ is a linear combination of the support vectors.
- For Type-B kernels, $\mathbf{n}_i \in \gamma_i \mathbf{x}_i + \mathbf{S_Y}$, for some $\gamma_i \in \mathbb{R}$, i.e., $\mathbf{n}_i - \gamma_i \mathbf{x}_i \in \mathbf{S_Y}$.

Note that γ_i depends on $\mathbf{x}_i$ and the two statements can be readily checked using Equations 4 - 6. Therefore, the kernel subspace $\mathbf{S_Y}$ can be recovered, for Type-A kernels, using Singular Value Decomposition (SVD). Specifically, let $\mathbb{N} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ denote the singular value decomposition of $\mathbb{N}$. There are precisely m nonzero singular values and $\mathbf{S_Y}$ is spanned by the first m columns of $\mathbf{U}$. For Type-B, a slight complication arises because we must determine s constants $\gamma_1, \cdots, \gamma_s$ such that the span of the following matrix is $\mathbf{S_Y}$:

$$\mathbb{N} - \mathbf{X}\mathit{\Gamma} \equiv [\mathbf{n}_1\, \mathbf{n}_2 \ldots \mathbf{n}_s] - [\gamma_1 \mathbf{x}_1\, \gamma_2 \mathbf{x}_2 \ldots \gamma_s \mathbf{x}_s], \tag{11}$$

where $\mathit{\Gamma}$ is a diagonal matrix with γ_i as its entries. Note that in general, $\mathbb{N}, \mathbf{X}$ are of full-rank $\min(d, s)$, and we are trying to find a set of γ_i such that the above matrix has

[2] We note that for sufficiently small δ, the angular error of the estimated normal is approximately in the order of $\tan^{-1}(\frac{\epsilon}{2\delta})$.

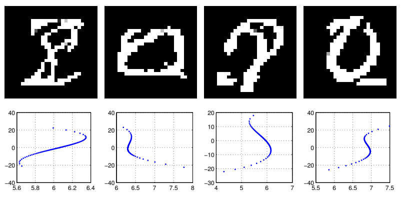


Fig. 1: **Intersections of Σ and two-dimensional affine subspaces** An SVM using the cubic kernel is trained on MINST dataset. **Top Row:** Midpoints of PN-pairs near the decision boundary Σ after bracketing. **Bottom Row:** Sampled polynomial curves given the intersections of the decision boundary with two-dimensional affine subspaces containing the images above.

rank $m < s$. However, for a generically chosen set of $\mathbf{x}_1, \cdots, \mathbf{x}_s$, the rank of $\mathbb{N} - \mathbf{X}\Gamma$ is at least m because the support vectors are linearly independent. Therefore, γ_i can be determined via the following rank-minimization problem

$$\arg\min_{\gamma_i} \mathbf{Rank}([\mathbf{n}_1\, \mathbf{n}_2 \ldots \mathbf{n}_s] - [\gamma_1\mathbf{x}_1\, \gamma_2\mathbf{x}_2 \ldots \gamma_s\mathbf{x}_s]). \tag{12}$$

As is well-known, a convex relaxation of the above problem uses the nuclear norm $\|\cdot\|_*$ (sum of singular values) as the surrogate

$$\arg \min_{\text{diagonal } \Gamma} \|\mathbb{N} - \mathbf{X}\Gamma\|_*, \tag{13}$$

and there are efficient algorithms for solving this type of convex optimization problem [16]. We note that for Type-A kernels, the rank is minimized at $\gamma_1 = \cdots = \gamma_s = 0$. In both cases, the span of $\mathbb{N} - \mathbf{X}\Gamma$ gives the kernel subspace $\mathbf{S_Y}$. As the support vectors are assumed to be linearly independent, the dimension of $\mathbf{S_Y}$ then gives the number of support vectors. For noisy recovery, the above method requires the standard modification that uses the significant gap between singular values as the indicator. For Type-A kernels, this is applied to the SVD decomposition of $\mathbb{N}$ directly, and for Type-B kernels, this is applied to the SVD decomposition of $\mathbb{N} - \mathbf{X}\Gamma$ with Γ determined by the nuclear norm minimization.

3.4 Determining Kernel Type

For determining the four Type-A kernels, we will examine the locus of the intersection of the decision boundary with a two-dimensional affine subspace containing a point close to the decision boundary. More specifically, let $\mathbf{x}_+, \mathbf{x}_-$ denote a PN-pair that is sufficiently close to the decision boundary Σ. We can randomly generate a two-dimensional subspace containing $\mathbf{x}_+, \mathbf{x}_-$ by, for example, taking the subspace $\mathbf{A}$ formed by $\mathbf{x}_+, \mathbf{x}_-$ and the origin. For a generic two-dimensional subspace $\mathbf{A}$, its intersection with Σ is a one-dimensional curve, and the parametric form of this curve is determined by the (yet unknown) kernel function. See Figure 1. Take a polynomial kernel of degree D as an example. By its construction, the intersection of the decision boundary and the affine subspace $\mathbf{A}$ is nonempty, and the locus of the intersection formed a curve in $\mathbf{A}$ that satisfies a polynomial equation of degree D. This can be easily seen as follows: take $\mathbf{x}_+$ as the origin on $\mathbf{A}$ and choose (arbitrary) orthonormal vectors $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^d$ such that the triplet $\mathbf{x}_+, \mathbf{U}_1, \mathbf{U}_2$ identifies $\mathbf{A}$ with $\mathbb{R}^2$. Therefore, any point $p \in \mathbf{A}$ can be uniquely identified with a two-dimensional vector $\mathbf{p} = [\mathbf{p}_1, \mathbf{p}_2] \in \mathbb{R}^2$ as

$$p = \mathbf{x}_+ + \mathbf{p}_1\mathbf{U}_1 + \mathbf{p}_2\mathbf{U}_2.$$

If $\mathbf{p} \in \mathbf{A}$ is a point in the intersection of $\mathbf{A}$ with the decision boundary $\mathbf{\Psi}(\mathbf{p}) = \mathbf{b}$, we have

$$\sum_{i=1}^{m} w_i((\mathbf{x}_+^\top \mathbf{Y}_i + \mathbf{p}_1\mathbf{U}_1^\top \mathbf{Y}_i + \mathbf{p}_2\mathbf{U}_2^\top \mathbf{Y}_i)1)^D = \mathbf{b}, \tag{14}$$

which is a polynomial of degree D in the two variables $\mathbf{p}_1, \mathbf{p}_2$. Therefore, to ascertain the degree of the polynomial kernel, we can (assuming $D < 4$)

– Sample at least nine points on the intersection of the decision boundary and $\mathbf{A}$.
– Fit a bivariate polynomial of degree D to the points. If the fitting error is sufficiently small, this gives an indication that the polynomial kernel is indeed of degree D.

We note that up to a multiplicative constant, a bivariate cubic polynomial in $\mathbb{R}^2$ has nine coefficients and this gives the minimum number of points required to fit a cubic polynomial. In addition, since the degree of the polynomial is invariant under any linear transform, this shows that the choice of the two basis vectors is immaterial. The advantage of the reduction from $\mathbb{R}^d$ to $\mathbb{R}^2$ is considerable as it implies that the complexity of this step is essentially independent of the ambient dimension d. For a transcendental kernel such as the hyperbolic tangent kernel, the locus of the intersection is generally not a polynomial curve and this can be detected by the curve-fitting error. Although, in principle, one affine subspace $\mathbf{A}$ is sufficient to distinguish between four Type-A kernels (as shown by the above equation), in practice, due to various issues such as possible degeneracy of the polynomial curve and the curve fitting error, we randomly sample several affine subspaces and use a majority voting scheme to determine the kernel type.

3.5 Complexity Analysis and Exact Recovery of $\mathbf{\Sigma}$

The steps outlined above essentially aim to ascertain the parametric form of the decision boundary $\mathbf{\Sigma}$ using a (relatively) small number of sampled points on $\mathbf{\Sigma}$. We note that the bracketing error in general can be explicitly controlled, and there are only two steps above that incur uncertainty: the normal estimate and the nuclear norm relaxation of the rank minimization problem. Our approach of using the local linear approximation to estimate the normal vector at a point is the standard one common in computational geometry and machine learning (e.g., [17,18] [19]), and the nuclear norm relaxation is the standard convex relaxation for the original NP-hard rank minimization problem [20]. A complete complexity analysis of the proposed algorithm would require detailed probabilistic estimates pertaining to these two steps, and although there are partial and related results scattered in the literature (e.g.,[20] [21]), we are unable to provide a definitive result at this point. Instead, we present a simple complexity analysis below under the assumption that these two steps can be determined exactly, i.e., the convex relaxation using the nuclear norm gives the same result as the original rank minimization problem.

The computational complexity can be defined as the number of features (not necessarily only on the decision boundary) in $\mathbb{R}^d$ sampled during the process and this number is the same as the number of queries to the classifier $\mathbf{C}$. From the above, it is clear that to determine the m-dimensional kernel subspace, at least $\mathbf{O}(m)$ sampled normals are required, i.e., $\mathbb{N}$ has at least m columns. Furthermore, to determine each normal vector at a given point $\mathbf{x}$ $\mathbf{O}(d)$ number of points are required, as the ambient dimension is d. Therefore, the total complexity is $\mathbf{O}(dm)$. The multiplicative constant here, as can be readily seen, is bounded by the maximum number of steps required for the bracketing, and this number is independent of the dimensions d, m, provided the features are drawn from a bounded subset of $\mathbb{R}^d$ (Equation 9).

Once the kernel subspace $\mathbf{S_Y}$ and the kernel type are determined, this allows us to focus on the intersection $\mathbf{\Sigma} \cap \mathbf{S_Y}$. In the case $m << d$, this reduction from $\mathbf{\Sigma} \subset \mathbb{R}^d$ to $\mathbf{\Sigma} \cap \mathbf{S_Y} \subset \mathbf{S_Y}$ is computationally significant. In particular, for polynomial kernels, we can sample $\mathbf{O}(m^D)$ points on $\mathbf{\Sigma} \cap \mathbf{S_Y}$ to reconstruct the polynomial $\mathbf{\Psi}(\mathbf{x})$ on $\mathbf{S_Y}$. At this point, $\mathbf{\Psi}(\mathbf{x})$ is a degree-D polynomial in m variables, and using recent results on tensor decomposition (e.g., [2] [22])[3], we can decompose $\mathbf{\Psi}(\mathbf{x})$ (more precisely, its homogenized version)

$$\mathbf{\Psi}(\mathbf{x}) = \sum_{i=1}^{r} \ell_i(\mathbf{x})^D, \tag{15}$$

where $\ell_1, \cdots, \ell_r$ are linear (homogeneous) polynomials. The smallest integer r for such decomposition gives the rank of the (homogeneous) polynomial (as a symmetric tensor) and in general, such decomposition is also possible for r greater than the rank. If we write the linear polynomials (after de-homogenization) as $\ell_i(\mathbf{x}) = \mathbf{z}_i^\top \mathbf{x} + 1$ for some vector $\mathbf{z}_i$, it is tempting to infer $\mathbf{z}_i$ as the support vector $\mathbf{y}_i$ from the above equation. However, because the non-uniqueness of the decomposition, $\mathbf{z}_i \neq \mathbf{y}_i$ in general. Nevertheless, $\mathbf{z}_i$ do act as if they are support vectors in the sense that the evaluation of the polynomial $\mathbf{\Psi}(\mathbf{x})$ becomes computationally trivial using the above decomposition. For polynomial kernels, the recovery of these quasi-support vectors $\mathbf{z}_i$ then determines the decision boundary $\mathbf{\Sigma}$ exactly, essentially completing the deconstruction process. Although the general algorithms for tensor decomposition [2] [22] require some mathematical machinery, the special case of quadratic kernels (degree-two polynomials) can be readily solved using eigen-decomposition of a symmetric matrix (the details are provided in the supplemental material). For transcendental kernels, no similar results are known at present. Although the reduction from $\mathbf{\Sigma} \subset \mathbb{R}^d$ to $\mathbf{\Sigma} \cap \mathbf{S_Y} \subset \mathbf{S_Y}$ offers the possibility of reconstructing the decision boundary in $\mathbf{S_Y}$, due to the nature of the transcendental functions, the details are considerably more difficult than the polynomial case, and we leave its resolution to future research.

4 Experiments

We present two sets of experiments in this section. The first set of experiments evaluates various components of the proposed method and the second set of experiments applies

[3] Algorithm 5.1 in `http://arxiv.org/pdf/0901.3706v2.pdf`, the archived version of [2].

the proposed method to explicitly deconstruct a kernel machine and subsequently improve it using incremental SVM [9].

4.1 Evaluation of the Deconstruction Algorithm

We present two experiments using kernel machines whose support vectors are randomly generated (first experiment) and support vectors trained using real image data (second experiment). We remark that there is no qualitative differences between deconstructing kernel machines with randomly-generated support vectors and deconstructing kernel machines trained with real data since, in both cases, the kernel function and decision function (Eq 1) are the same. Using randomly-generated kernel machines allow us to study the behavior of the deconstruction algorithm over a much wider range of support vector configuration, demonstrating its accuracy and robustness. In the first set of experiments, we set the feature dimension $d = 30$, and we randomly generate 12 support vectors. For determining the kernel type, we sample 25 points close to the decision boundary $\mathbf{\Sigma}$ and at each point, we compute the intersection of $\mathbf{\Sigma}$ and a two-dimensional subspace. We fit a quadratic and then a cubic polynomial to these points, and the smallest degree giving an error below some threshold value is declared as the degree of the kernel. However, if in both cases the fitting errors are greater than the threshold value, the kernel is declared to be a Gaussian kernel at this location. This is repeated at 25 sampled locations and a majority vote is used to determine the kernel type. Once the kernel type is determined, we use SVD to determine the dimension of the kernel subspace $\mathbf{S_Y}$ and the subspace itself. For the Gaussian kernel, the nuclear-norm minimization is performed before using SVD to locate the subspace $\mathbf{S_Y}$. In this experiment, we sample $s = 100$ points on the decision boundary in order to form the matrices $\mathbb{N}, \mathbf{X}$ and the tolerance in the bracketing step is set at 10^{-6}. Let $\overline{\mathbf{S_Y}}$ denote the kernel subspace computed by our method. We use the principal angles [23] between the two subspaces $\mathbf{S_Y}, \overline{\mathbf{S_Y}}$ as the metric for quantifying the error.

Summary The gap between the singular values of $\mathbb{N}$ is an important indicator of the dimension of the kernel subspace, and it is affected by the accuracy of the normals. Figure 2 shows the effect in terms of the radius δ used in computing the normals, showing the expected result that the ratio of δ/ϵ is directly related to the accuracy of the recovered normals (larger ratios provide more accuracy). For determining the kernel type, the specificity for the polynomial kernels is close to 100% with the specificity of approximately 80% for the Gaussian kernel (and hyperbolic tangent kernel). This can be attributed to the majority voting scheme used in assigning the kernel type, and we leave it as important future work for designing more robust criteria. The accuracy of the recovered kernel subspaces is shown in Figure 3 and 4a. The first figure shows the means and variances of the (cosine of) twelve principal angles, taken over one hundred randomly generated kernel machines using polynomial kernels. Note that $\cos^{-1}(0.99)$ is approximately $8°$ and this gives a good indication of the accuracy. In the second figure, the twelve principal angles computed before and after the rank-minimization are shown, indicating the correctness and necessity of performing rank-minimization. Finally, each deconstruction makes between $60,000$ and $70,000$ queries to the classifier, and on a typical 3Ghz machine, it takes no more than a few minutes to complete the deconstruction process. Since the algorithm is readily parallelizable (which would be important for deconstruction in high-dimensional feature spaces), a full parallelized and

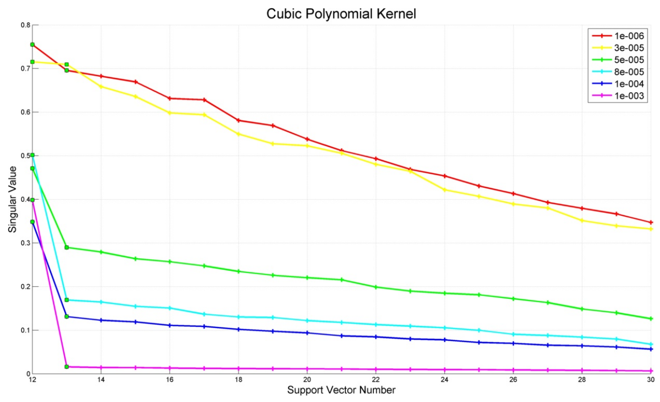


Fig. 2: Singular Values of the normal matrix $\mathbb{N}$ for different choices of δ. The kernel machine uses a polynomial cubic kernel with 12 support vectors. The expected gaps between the 12^{th} and 13^{th} singular values are indicated by the green markers. Note that for a fixed tolerance $\epsilon = 10^{-6}$, the optimal $\delta = 10^{-3}$. For smaller δ without changing ϵ accordingly, the estimated normals become less accurate. (**Image best viewed in color**)

optimized implementation can be expected to shorten the running time considerably, perhaps in the range of only a few seconds.

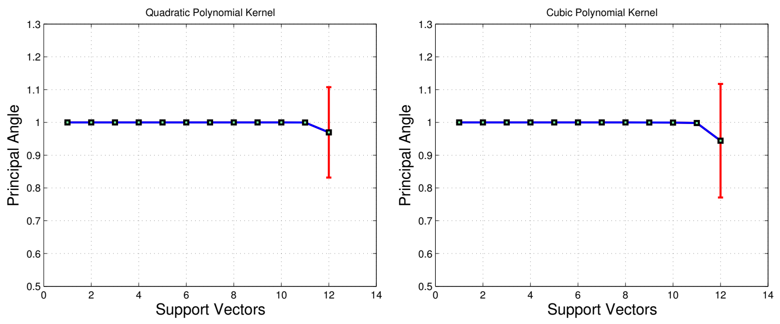
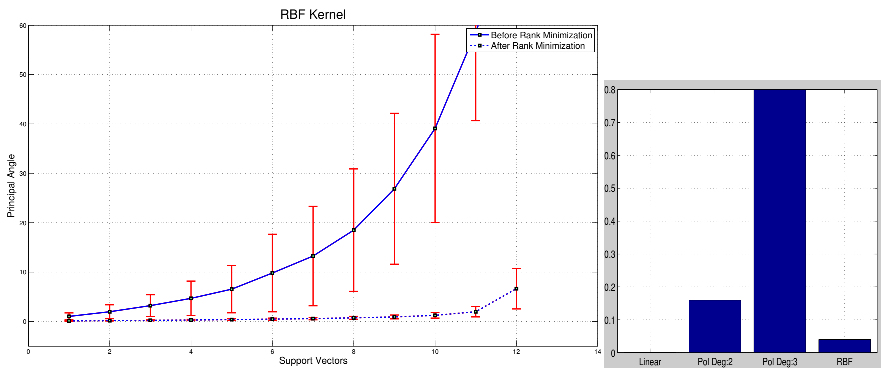


Fig. 3: Means and variances of the cosines of the twelve principal angles between $\overline{\mathbf{S_Y}}$ and $\mathbf{S_Y}$. Means and variances are taken over one hundred independent deconstruction results for kernel machines with twelve support vectors using a polynomial kernel (Quadratic kernel on the left and cubic kernel on the right). (**best viewed in color**)

In the second experiment, we train a kernel machine with cubic polynomial kernel using 1000 images from MNIST dataset [24]. The positive class consists of images of the digit 2 and the negative class consists of $0, 5, 7, 8$. The trained kernel machine has 275 support vectors. Figure 1 displays the intersections of the decision boundary with several two-dimensional affine subspaces, noticing the superpositions of the images of 2 with images of other digits. In this experiment, we randomly generate 200 two-dimensional affine subspaces and for each subspace, its vote on the type of kernel is determined as above. Figure 4b shows the distribution of votes, clearly indicating the correct result. For this experiment, the gap in the singular values of $\mathbb{N}$ indicates the cor-



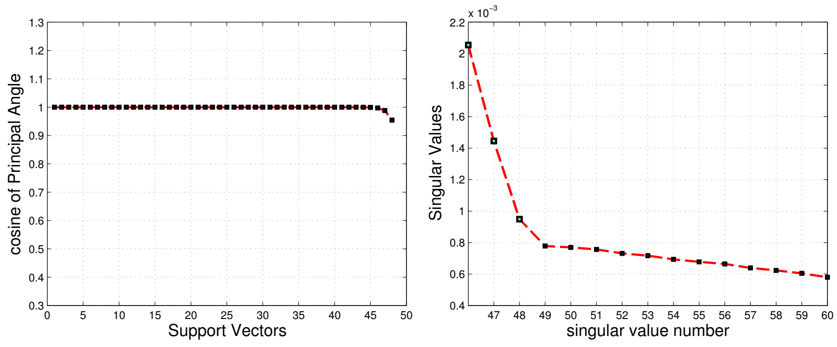
(a) (b)

Fig. 4: **Left:** Means and variances of the the twelve principal angles between $\overline{\mathbf{S_Y}}$ and $\mathbf{S_Y}$. Means and variances are taken over one hundred independent deconstruction results for kernel machines with twelve support vectors using a Gaussian kernel. The principal angles before and after rank minimization are shown. (**best viewed in color**). **Right:Distribution of Votes on Kernel Type** For a cubic kernel machine trained on 1000 MNIST images, the distribution of votes on kernel type for 200 randomly sampled two-dimensional affine subspaces. The correct result is clearly indicated.

rect dimension of the kernel subspace (275) and the kernel subspace is also successfully recovered.

4.2 Kernel Machine Upgrade Without Source Code

In the second experiment, we demonstrate the possibility of upgrading a kernel machine without access to the kernel machine's source code. As outlined in the introduction, we apply the deconstruction algorithm to deconstruct the kernel machine. This step provides us with the kernel type and quasi-support vectors (for a polynomial kernel machine). For the subsequent upgrade (or update), we use the incremental SVM algorithm [9] to retrain the kernel machine given the new training data. Specifically, we first train a kernel machine using MNIST dataset: images of digit 1 as positive samples and the negative training samples comprise the remaining digits except 8. Dimensionality reduction is applied to the images using PCA to a feature space of dimension 60. An SVM with quadratic kernel is trained on these training samples, resulting in 97.30% true positive detection rate and 99.17% true negative detection rate on the test dataset. The initial kernel machine has 48 support vectors. During deconstruction, the kernel subspace is recovered using 800 sampled normal vectors. Let $\mathbb{N}$ denote the matrix obtained by horizontally stacking together the normal vectors and $\mathbb{N} = USD$, its SVD decomposition. The plot of the singular values is shown in Figure 5b and the significant gap between the 48th and 49th singular values indicate the correct dimension (and the number of support vectors). The principle angles between the kernel subspace estimated by the first 48 columns of U and the ground-truth is shown in Figure 5a. Once the kernel subspace is recovered, we proceed to recover the quasi-support vectors. The kernel machine defined by the quasi-support vectors should be a good approximation of the original kernel machine and this is shown in Table 1a, where we compare the



(a) (b)

Fig. 5: **Left:**Cosines of the principal angles between the recovered kernel subspace and the ground-truth kernel subspace. **Right:**Singular values of the matrix $\mathbb{N}$. The gap between 48th and 49th singular values is significant as the gaps among the remaining singular values are substantially smaller. The correct dimension of the kernel subspace (and the number of support vectors) is 48.

classification results using the recovered kernel machine and the original one. In this example, the results as expected are quite similar, with the recovered kernel machine actually performing slightly better. Once we have recovered the quasi-support vectors, we next proceed to upgrade the kernel machine. The task is to upgrade a kernel machine that recognizes only digit 1 to a kernel machine that recognizes digits 1 and 8. The classification results for the initial and upgraded kernel machines are tabulated in Table 1b. As shown in the table, before the upgrade, the original kernel machine performs poorly on the images of digit 8 and for the upgraded machine, both digits can now be successfully classified.

	Quasi-SV Machine outcome		Original Machine outcome	
	+ve	-ve	+ve	-ve
Positive	100.00%	00.00%	97.30%	2.70%
Negative	3.73%	96.27%	0.83%	99.17%

(a)

	Classification Rate	
	Original Machine	Upgraded Machine
Digit 1	97.30%	100.00%
Digit 8	00.00%	92.31%
Negative	99.17%	97.93%

(b)

Table 1: **Left:**Confusion matrices for the original kernel machine and the kernel machine defined by the recovered quasi-support vectors. Both machines are tested on the same test dataset. **Right:**Comparisons of classification results for the original kernel machine and the upgraded kernel machine.

5 Conclusion

We have introduced the novel notion of deconstructive learning and proposed an algorithm for deconstructing kernel machines. Preliminary experimental results have demonstrated both the viability and effectiveness of the proposed method. Although much work remains for the future, the results presented in this paper serve as a small first step in understanding the full implication and potential of deconstructive learning.

References

1. V. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.
2. J. Brachat, P. Comon, B. Mourrain, and E. Tsigaridas, "Symmetric tensor decomposition," *Linear Algebra Appl*, vol. 433, no. 11-12, pp. 1851–1872, 2010.
3. M. Heath, *Scientific Computing*. The McGraw-Hill Companies, Inc, 2002.
4. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer, 2010.
5. T. Dey, *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge University Press, 2006.
6. B. Horn, *Robot Vision*. MIT Press, 2010.
7. G. Carlsson, "Topology and data," *Bulletin of the American Mathematical Society*, vol. 46, no. 2, pp. 255–308, 2009.
8. G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances in neural information processing systems*, pp. 409–415, 2001.
9. C. P. Diehl and G. Cauwenberghs, "SVM incremental learning, adaptation and optimization," in *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 4. IEEE, 2003, pp. 2685–2690.
10. D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 641–647.
11. M. Torkamani and D. Lowd, "Convex adversarial collective classification," in *Proc. Int. Conf. Machine Learning (ICML)*, 2013.
12. S. Dasgupta, "Analysis of a greedy active learning strategy," in *In Advances in Neural Information Processing Systems*, 2004.
13. M. Balcan, A. Beygelzimer, and J. Langford, "Agnostic active learning," in *Proc. Int. Conf. Machine Learning (ICML)*, 2006.
14. M. Balcan, A. Broder, and T. Zhang, "Margin-based active learning," *Learning Theory*, pp. 35–50, 2007.
15. M. Hirsch, *Differential Topology*. Springer, 1997.
16. Y. Nesterov and A. Nemirovski, "Some first-order algorithm for l1/nuclear norm minimization," *Acta Numerica*, 2014.
17. S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
18. M. Belkin and P. Niyogi, "Semi-supervised learning on riemannian manifolds," *Machine learning*, vol. 56, no. 1-3, pp. 209–239, 2004.
19. S. Rifai, Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller, "The manifold tangent classifier." in *NIPS*, 2011, pp. 2294–2302.
20. E. Candes and B. Recht, "Exact matrix completion via convex optimization," *Commun. ACM*, vol. 55, no. 6, pp. 111–119, 2012.
21. M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *The Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.
22. E. Ballico and A. Bernardi, "Decompoision of homogeneous polynomials with low rank," *Mathematische Zeitschrift*, vol. 271, no. 3–4, pp. 1141–1149, 2012.
23. G. Golub and C. V. Loan, *Matrix Computation*. John Hopkins University Press, 1996.
24. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.