FINAL

Support Vector Machines and Kernel Fisher Discriminants: A Case Study using Electronic Nose Data

Dennis DeCoste and Michael C. Burl
Machine Learning Systems Group
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive; Pasadena, CA 91109
{decoste,burl}@aig.jpl.nasa.gov

Alan Hopkins and Nathan S. Lewis
Department of Chemistry
California Institute of Technology
M/S 127-72
Pasadena, CA 91125
nslewis@caltech.edu

ABSTRACT
Kernel methods provide a promising new family of algorithms for machine learning and data mining applications. In particular, kernel-based nonlinear classifiers such as support vector machines (SVMs) and kernel fisher discriminants (KFDs) have been found to work well in practical problems. In addition, there are methods for training these algorithms on large-scale data sets making them very suitable for use in data mining. In this paper, we evaluate the performance of SVMs and KFDs on a dataset generated with a conducting polymer composite-based electronic nose. The ability of SVM and KFD classifiers to correctly identify the functional class (category) of a chemical based on its electronic nose signature is evaluated and compared against other more traditional methods, including nearest neighbors and linear Fisher discriminants. Tradeoffs between the different kernel methods and performance relative to more traditional methods are discussed.

Keywords
support vector machines, kernel Fisher discriminant, classification, electronic nose

1. INTRODUCTION
Arrays of polymer films embedded with conductive or resistive material have attracted significant attention as "electronic noses." Unlike traditional "lock-and-key" approaches to vapor sensing, in which a detector is very specific to a particular analyte, the polymer-based detectors used here are broadly-tuned so that a given detector responds to many vapors and a single vapor causes a response in many detectors. Only by analyzing the pattern of responses across the array of detectors can specific analytes be identified or discriminated from chemically similar compounds. As part of an ongoing scientific research project between JPL and Caltech [2], we have been studying the suitability of kernel-based methods for various classification tasks involving data from the electronic nose. In this paper, new results using support vector machines (SVMs) and kernel Fisher discriminants (KFD) to learn to predict the category (e.g., alcohol or hydrocarbon) of a previously unseen (unsniffed?) chemical are presented. We will discuss how these results illustrate some of the practical decisions and tradeoffs required to apply SVM and KFD methods and contrast their performance against other traditional methods (i.e. nearest-neighbors and linear Fisher discriminants).

2. ELECTRONIC NOSE
The Caltech electronic nose consists of an array of polymer films embedded with conductive or resistive material. Sorption of a vapor into the polymer films causes physical swelling, which leads to a change in the DC electrical resistance of the film. The DC resistance across each of the films in the array is sampled at approximately uniformly-spaced sample times. The resistance values are digitized with an A-to-D converter. For the experiments reported here, the raw time-series data were converted to vector form by computing the relative change in resistance in each channel compared to the pre-exposure baseline. The raw time-series response of the electronic nose to a given analyte thus becomes a d-dimensional vector where d is the number of channels (polymer films).

All analyte exposures were performed using a computer-controlled vapor generation and control system that regulates the identity, concentration, exposure time, and flow rate of the analyte above the detectors [18]. Between exposures, clean air is passed through the system to remove any residue from the previous exposure. Analytes are presented to the system in a randomized order to prevent biases in the results. For a broad range of concentrations and analytes, the electronic nose arrays behave like a linear system. Increasing or decreasing the concentration of an analyte produces a proportional increase or decrease in the signature, and the response to mixtures of analytes is approximately the weighted average of the response to the individual analytes [18].

An interesting study, which we have recently undertaken (preliminary results using nearest neighbor classification appeared in [2]), involves learning to classify analytes into the appropriate functional group based on their electronic nose signature. Five functional groups or chemical families (alcohols, alkyl halides, aromatics, hydrocarbons, and esters) were used for our experiments. Within each class, 15 members were chosen for a total of 75 different chemicals. These were presented to an electronic nose containing 40 polymer-based sensors (two copies of 20 different polymers). The analytes were presented to the nose in groups of eight because the physical setup of the gas dispensation system has eight bubblers. A total of 80 sniffs (ten of each analyte) in randomized order was obtained from each set of eight and then a new set of eight analytes was swapped in. Each group of eight analytes contained two members of four classes so that temporal effects would not bias the results (e.g., if all alcohols were sniffed in the morning and the temperature was cooler then, it might introduce an artificial bias into the separability of alcohols from the other classes). Each sniff produced a multivariate time series which was converted to vector form. Responses of polymer "twins" (duplicates of the same polymer type) were averaged to produce 20-dimensional vectors for each sniff. This data was then used by various kernel-based and traditional classification algorithms in a leave-one-chemical-out (LCO) cross-valdiation. Note that for a given test example, all other sniffs of the same compound were sequestered from the training set. In other words, we wanted to determine if a sniff of methanol could be used to classify it as an alcohol *without* having previously smelled methanol, but perhaps having smelled ethanol, butanol, cyclopentanol, etc.

3. KERNEL METHODS

Recently, many traditional linear methods have been generalized to corresponding nonlinear forms using *Mercer kernels*. Examples include Principal Component Analysis [17], k-means clustering [16] nearest-neighbors [16], and Fisher discriminants [13]. Further, completely new kernel-based methods such as SVM classification [1] and SVM regression [19] have been introduced.

Consider an ℓ-by-D data matrix (X) of examples. A (Mercer) kernel $K(x_i, x_j)$ implicitly projects the two given examples from D-dimensional input space into some (possibly infinite-dimensional) feature space and returns their dot product in that feature space. That is, it computes

$$K(x_i, x_j) \equiv \phi(x_i) \cdot \phi(x_j) \equiv \phi(x_i)' \phi(x_j), \qquad (1)$$

for some mapping function ϕ, but without explicitly computing the coordinates of the projected vectors. In this way, kernels allow large non-linear feature spaces to be explored while avoiding the curse of dimensionality.

The simplest kernel is the *linear* kernel, implemented as a simple dot product:

$$K(u, v) = u \cdot v \equiv \sum_{i=1}^{d} u_i \cdot v_i. \qquad (2)$$

As explained later, kernel methods give the same results as their traditional linear equivalents when linear kernels are used, but will typically be much slower. This cost arises from operating on some matrices of size ℓ-by-ℓ that are only of size D-by-D in traditional linear methods.

The *polynomial* kernel is defined by a non-linearly squashed dot product of the following form:

$$K(u, v) = (u \cdot v + r)^d, \qquad (3)$$

with polynomial degree parameter d. Varying the continuous offset parameter r changes the relative weighting of the (implicit) terms in the non-linear polynomial feature space. We will refer to instances of this kernel as "POLY d r".

One of the most popular kernels is the *radial basis function* (RBF) kernel:

$$K(u, v) = e^{\frac{-||u-v||^2}{2\sigma^2}}, \qquad (4)$$

with variance parameter σ, giving another non-linear squash of the dot product of the two examples. [1] We will refer to instances of this kernel as "RBF g", where $g = \frac{1}{2\sigma^2}$.

In this paper we will focus on two specific kernel methods, SVMs and KFDs, as described below.

3.1 Support Vector Machine (SVM)

Given an ℓ-by-ℓ kernel matrix K (computed from the ℓ-by-D data matrix (X)), an ℓ-by-1 labels vector (y), and a "soft margin" regularization parameter $(C > 0)$, training a binary SVM classifier traditionally consists of the following Quadratic Programming (QP) dual formulation:

minimize:

$$\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{\ell} \alpha_i$$

subject to:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0,$$

where ℓ is the number of training examples, y_i is the label (+1 for positive example, -1 for negative) for the i-th training example (x_i), and $K(x_i, x_j)$ denotes the value of the kernel function for i-th and j-th examples of X.

Note that once the kernel matrix is computed, the SVM itself is independent of both the input dimensionality (D) and the (implicit) feature dimensionality (in kernel space). In this way, SVMs are said to overcome the curse of dimensionality.

The vector of alphas α (of length ℓ) is the solution to the above QP problem. Of significant practical benefit is that there are no local optima for this QP (unlike, say, neural networks).

Widely-used decomposition methods, such as *SMO* [15] and SVM^{light} [9], typically train SVMs (i.e. solve the above QP) in roughly $O(\ell^2)$ time and sub-quadratic space (by computing kernel elements only as required). In our experiments, we use our own implementation [3] of *SMO*, based on [10].

The SVM output classification $F(x)$, for any new example x, can be computed as:

$$G(x) = \sum_{i=1}^{\ell} \alpha_i y_i K(x, x_i), \qquad (5)$$

[1]Where 2-norm defined as $||u - v||^2 \equiv (u \cdot u - 2u \cdot v + v \cdot v)$.

$$F(x) = sign(G(x) - b), \tag{6}$$

The SVM weights (α) over the examples are often rather sparse (typically roughly 5% — 20% are non-zero), making the above output summations somewhat faster in practice than shown above. The special examples x_i for which $0 < \alpha_i \leq C$ are called the *support vectors* (SVs). Retraining the SVM using only the SVs would result in the same α solution.

Let SV^+ represent the set of positive support vector examples and SV^- represent the set of negative SV examples. Similarly, define their corresponding "in-bounds" subsets IN^+ and IN^-, for which $0 < \alpha_i < C$. As is common practice, we compute the scalar bias (b) as midway between the mean of G over IN^+ and the mean of G over IN^-.

A SVM maximizes the *margin* distance between the nearest positive and negative examples (in kernel feature space), which has been shown to lead to excellent generalization performance in many domains [7], for much the same reasons as the similar success of boosting methods [6].

3.2 Kernel Fisher Discriminant (KFD)

The classic linear Fisher discriminant (LFD) for binary classification [5] finds the projection weights (w) that map the data X onto a line such that along that line within-class variance is minimized while between-class variance is maximized.

3.2.1 LFD

Specifically, LFD maximizes the following score J:

$$maximize: \quad J(w) = \frac{w'\ S_B\ w}{w'\ S_W\ w}. \tag{7}$$

The between (S_B) and and within (S_W) components of J are defined as:

$$S_B = (m^+ - m^-)(m^+ - m^-)', \tag{8}$$

$$S_W = \sum_{x_i \in X^-} (x_i - m^-)(x_i - m^-)' + \sum_{x_i \in X^+} (x_i - m^+)(x_i - m^+)', \tag{9}$$

where

$$m^- = \frac{1}{\ell^-} \sum_{x_i \in X^-} x_i, \quad m^+ = \frac{1}{\ell^+} \sum_{x_i \in X^+} x_i, \tag{10}$$

are the D-dimensional mean vectors for the negative (X^-) and positive (X^+) examples, respectively.

The D-dimensional projection weights can be computed in closed-form using:

$$w = S_W^{-1}(m^- - m^+). \tag{11}$$

The LFD classification $f(x)$ for example x is given simply by:

$$g(x) = x'w, \quad f(x) = sign(g(x) - b), \tag{12}$$

where b is a threshold (typically determined on the assumption that the class-conditional denisities of the projected data are Gaussian).

3.2.2 KFD

By substituting $\phi(x_i)$ for each x_i in LFD,, denoting each resulting $\phi(x_i) \cdot \phi(x_j)$ term as kernel element $K(x_i, x_j)$, and using some algebraic simplifications, we get KFD (e.g. [12]):

$$maximize: \quad J(\alpha) = \frac{\alpha'\ Z_B\ \alpha}{\alpha'\ Z_W\ \alpha}. \tag{13}$$

$$Z_B = (\mu^+ - \mu^-)(\mu^+ - \mu^-)', \tag{14}$$

$$Z_W = KK' \tag{15}$$

where

$$\mu^- = \frac{1}{\ell^-} K\ 1^-, \quad \mu^+ = \frac{1}{\ell^+} K\ 1^-, \tag{16}$$

act like (ℓ-dimensional) "mean" vectors [2] and K is the ℓ-by-ℓ kernel matrix with elements:

$$K_{ij} = k(x_i, x_j). \tag{17}$$

The α are computed in closed-form (analogous to w in LFD):

$$\alpha = Z_W^{-1}(\mu^- - \mu^+). \tag{18}$$

The projection weights in feature space themselves are then given by:

$$W = \sum_{i=1}^{\ell} \alpha_i \phi(x_i) \tag{19}$$

The KFD classification $F(x)$ for example x follows the same form as for SVMs, except that α_i is no longer restricted to be non-negative and labels y_i no longer appear:

$$G(x) = \phi(x)'W = \sum_{i=1}^{\ell} \alpha_i K(x, x_i), \tag{20}$$

where W is the (implicit) weight vector in kernel feature space, and

$$F(x) = sign(G(x) - b). \tag{21}$$

For a linear kernel, the D-dimensional weights (w) of LFD can be recovered, by "weight folding" KFD's ℓ-dimensional (α):

$$w = \quad W = \sum_{i=1}^{\ell} \alpha_i \phi(x_i) \quad = \sum_{i=1}^{\ell} \alpha_i x_i. \tag{22}$$

This shows that KFD with the linear kernel gives the identical weights as LFD (but is much slower to train).

As in SVM's, some form of regularization for KFD is required in practice. One common approach, which we employ in our experiments here, is to add some regularization scalar to the diagonal of the Z_W. This also prevents inversion problems when Z_W is nearly singular.

[2] 1^+ is the ℓ-by-1 vector which contains ones where the labels vector y has 1's and contains zeros elsewhere. 1^- is similar, but contains ones where target y has -1's.

One remaining issue for KFD is how to compute the threshold bias (b). One approach ([14]), which we employ here is to train a linear SVM, using the projected KFD outputs as (1-dimensional) training data, and use the bias computed by the SVM. Other approaches are described in [12].

In contrast to SVMs, KFDs have recently been shown [12] to roughly maximize the *average margin*, i.e. the distance between the centers of the positive and negative data once they are projected on the Fisher line.

4. CLASSIFICATION EXPERIMENTS

4.1 Nearest Neighbors

For baseline comparisons, we repeat here earlier results [2], using 1-nearest-neighbors. The Euclidean distances of the test example from all members of the reference library were computed. The functional class label of the closest member of the reference library was taken to be the class label of the test example. *For a given test example, all other sniffs of the same compound were sequestered from the reference library.* In other words, we wanted to determine if a sniff of methanol could be used to classify it as an alcohol *without* having previously smelled methanol, but perhaps having smelled ethanol, butanol, cyclopentanol, etc. A confusion matrix showing the results of this experiment is given in Table 1. The first row shows that all members of the alcohol family were correctly classified as alcohols. The second row shows that 83% of the members of the alkyl halide family were correctly classified, with 6.9% of the members confused as aromatics, 0.6% confused as hydrocarbons, and 9.4% confused as esters. Overall, the average correct classification percentage is 77%.

4.2 Handling Multiple Classes

There are several ways to handle multiple (k) classes (in our case, k=five) using binary classifiers. The two that we have explored can be described as "one-vs-rest" and "pair-wise voting".

In one-vs-rest, one learns k classifiers, each deciding if an example is of that class or not. One decides which of the k classes it is by finding which of the k classifiers has the strongest positive output.

In pair-wise voting, one learns $k(k-1)/2$ classifiers, for each pair-wise contest. If a single class unanimously wins all pair-wise contests for an example versus each of the other $k-1$ classes, then its label is assigned to the example. If a unanimous decision cannot be reached, it is treated as a "punt" (i.e. no classification is made).

Due to computational expense, to date we have only tried one-vs-rest for our kernel methods. We have tried both ways for LFD. The results are presented below.

4.3 Kernel Methods

Kernel methods require model selection to select appropriate kernels – both type (e.g. polynomial vs RBF) and parameters (i.e. poly degree or RBF variance level). Ideally, one would do model selection search (e.g. via cross-validation) for each leave-one-chemical-out in our experiment. However, current techniques make that too costly, especially for KDA since no efficient model selection methods have yet been formulated (see [4] and [11] for some recent methods for more efficient SVM model selection).

Thus, we simply selected kernels for SVM and KFD based on which worked well when randomly partitioning the data set into training and validation sets. This is likely to be sub-optimal, since we thus selected one kernel to use regardless of which chemical class is being left out in turn in the final test experiment.

It is also possible that this approach is slightly contaminated, since some final test chemicals occur in training sets during this process. However, we only did this model selection search to determine "reasonable" kernels to use. The actual model weights in that feature space (e.g. the SVM's or KFD's α) are trained in the final test experiments with no knowledge of the hold-out chemicals.

Tables 2 and 3 show the results for the best kernel selected for SVMs and KFDs, respectively.

Note that we found KFD worked best with the unnormalized polynomial kernel ($K(u,v) = (u \cdot v + 10)^3$) whereas SVM worked best with the normalized version: $K(u,v) = \frac{1}{2^3}(u \cdot v + 1)^3$, where all u and v are 2-normed (unit length) versions of the original data. This result is consistent with observations made elsewhere that SVMs seem to work best when feature vectors are normalized [8]. KFD apparently does not benefit from such normalization, due to the way Fisher discriminants use the covariance matrix explicitly.

The SVM appears to work significantly better. We believe part of the reason may be because one-versus-rest approaches to multi-class problems are not particularly suitable for Fisher discriminants. The next section shows that indeed, at least for LFD, pair-wise classifiers seems to be better than one-versus-rest. In fact, the one-vs-rest LFD result shows that our KFD result is no better than that.

4.4 Linear Fisher Discriminants

Tables 4 and 5 show the results for the linear Fisher discrimnation, using one-vs-rest and pairwise approaches, respectively, on our multi-class classification problem.

5. CONCLUSIONS

Our results show that kernel methods offer some promise for challenging real-world tasks such as our chemical functional class problem. However, we are still working on several important issues. One is conducting more comprehensive model selection to more accurately (and fairly) determine the best kernels to use for each chemical hold-out experiment. We are also studying the tradeoffs of one-vs-rest and pair-wise approaches to multi-class problems such as ours. And we are looking into the best way to use soft (probabilistic) target labels. For example, scientists recently provided us with fractional assignments of the chemicals to the five groups. We suspect that the current ceiling of test performance (near 85%) may be exceeded once we more fairly account for the fast that many of these chemicals do not fall exclusively in only one of the five groups.

1NN: 0.77	alcohol	alkyl halide	aromatic	hydrocarbon	ester
alcohol	1.000	0.000	0.000	0.000	0.000
alkyl halide	0.000	0.831	0.069	0.006	0.094
aromatic	0.000	0.267	0.527	0.200	0.007
hydrocarbon	0.037	0.019	0.213	0.688	0.044
ester	0.047	0.147	0.000	0.018	0.788

Table 1: LCO confusion matrix for 1NN classification of compounds into functional classes. Each functional class contained fifteen compounds with 10 or in a few cases 20 sniffs each. For a given test example, all other examples of the same compound were withheld from the reference library (i.e. LCO = Leave-Chemical-Out). Average correct LCO classification rate = 0.77.

SVM: 0.82	alcohol	alkyl halide	aromatic	hydrocarbon	ester
alcohol	1.000	0.000	0.000	0.000	0.000
alkyl halide	0.013	0.688	0.062	0.000	0.237
aromatic	0.000	0.113	0.713	0.120	0.053
hydrocarbon	0.031	0.062	0.050	0.856	0.000
ester	0.059	0.100	0.000	0.000	0.841

Table 2: LCO confusion matrix for SVM trained one-vs-rest (kernel='poly 3 .1' C=100). Average correct rate = 0.821, total runtime = 868.9 secs. (No-holdout training rate = 0.921.)

KFD: 0.79	alcohol	alkyl halide	aromatic	hydrocarbon	ester
alcohol	1.000	0.000	0.000	0.000	0.000
alkyl halide	0.000	0.606	0.062	0.131	0.200
aromatic	0.067	0.047	0.687	0.133	0.067
hydrocarbon	0.025	0.050	0.075	0.850	0.000
ester	0.059	0.124	0.006	0.000	0.812

Table 3: LCO confusion matrix for KFD trained one-vs-rest (kernel='POLY 3 10'). Average correct rate = 0.792, total runtime = 1919.8 secs. (No-holdout training rate = 0.935.)

LFD: 0.80	alcohol	alkyl halide	aromatic	hydrocarbon	ester
alcohol	0.931	0.056	0.000	0.000	0.013
alkyl halide	0.000	0.719	0.081	0.075	0.125
aromatic	0.007	0.027	0.713	0.133	0.120
hydrocarbon	0.031	0.031	0.062	0.875	0.000
ester	0.076	0.165	0.006	0.000	0.753

Table 4: LCO confusion matrix for LFD trained one-vs-rest (i.e. KFD with kernel='linear'). Average correct rate = 0.799, total runtime = 17.4 secs. (No-holdout training rate = 0.927.)

LFDp: 0.84	alcohol	alkyl halide	aromatic	hydrocarbon	ester
alcohol	0.988	0.000	0.000	0.000	0.000
alkyl halide	0.000	0.831	0.069	0.006	0.000
aromatic	0.000	0.067	0.660	0.133	0.013
hydrocarbon	0.000	0.006	0.138	0.831	0.019
ester	0.053	0.006	0.000	0.006	0.871

Table 5: LCO confusion matrix for LFD with pairwise voting. Average correct rate = 0.836. Note: rows do not sum to 1 because deadlocks between the different pairwise classifiers are treated as "punts".

6. ACKNOWLEDGEMENTS

The research described in this paper was carried about at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

7. REFERENCES

[1] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[2] M. C. Burl, S. Briglin, B. Doleman, A. Hopkins, A. Matzger, D. N. Ortiz, A. Schaffer, S. Upchurch, T. Vaid, and N. S. Lewis. Mining the detector responses of a conducting polymer composite-based electronic nose. In *First SIAM Int. Conf. on Data Mining*, April 2000.

[3] D. DeCoste and B. Schölkopf. Training invariance support vector machines. *Machine Learning*, 2001. In press.

[4] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, August 2000.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[6] A. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

[7] I. Guyon. Online SVM application list, 2000. (See http://www.clopinet.com/isabelle/Projects/SVM/applist.html.).

[8] R. Herbrich and T. Graepel. A PAC-bayesian margin bound for linear classifiers: Why SVMs work. *Advances in Neural Information System Processing (NIPS) 13*, 2001.

[9] T. Joachims. Making large-scale support vector machine learning practical, 1999. In *Advances in Kernel Methods: Support Vector Machines* [?].

[10] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. Technical Report CD-99-14, Dept. of Mechanical and Production Engineering, National University of Singapore, 1999.

[11] J.-H. Lee and C.-J. Lin. Automatic model selection for support vector machines. Technical report, Dept. of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, November 2000. Implementation at [?]. Online at http://www.csie.ntu.edu.tw/ cjlin/papers/modelselect.ps.gz.

[12] S. Mika, G. Rätsch, and K.-R. Müller. A mathematical programming approach to the kernel fisher algorithm. In *Advances in Neural Information Processing Systems (NIPS) 13*, 2001.

[13] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

[14] S. Mika, A. Smola, and B. Schölkopf. An improved training algorithm for kernel Fisher discriminants. In T. Jaakkola and T. Richardson, editors, *Artificial Intelligence and Statistics*, pages 98 – 104, San Francisco, CA, 2001. Morgan Kaufmann. Also: Microsoft Research TR-2000-77.

[15] J. Platt. Fast training of support vector machines using sequential minimal optimization, 1999. In *Advances in Kernel Methods: Support Vector Machines* [?].

[16] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical report no. 44, Max-Planck-Institut for Biologische Kybernetik, Tübingen, Dec 1996.

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

[18] E. Severin, B. Doleman, and N. Lewis. An investigation of the concentration dependence and response to analyte mixtures of carbon black-insulating organic polymer composite vapor detectors. *Anal. Chem.*, 72:658–668, 2000.

[19] A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 2000.