A LEARNING THEORY APPROACH TO SENSOR NETWORKS

SLOBODAN N. SIMIĆ

Abstract. We propose an approach to various sensor network applications via learning theory. As a first step, we put forward a well-known algorithm from learning theory, applying it to environmental monitoring, tracking moving objects and plumes, and localization. We give a brief overview of some of the basic concepts of learning theory and present some preliminary simulation results.

1. Introduction

Recent advances in MEMS, computing, and communication technology have sparked the emergence of massively distributed, wireless sensor networks potentially consisting of thousands of nodes. Each node is able to sense the environment, process the collected data, and communicate with its peers or to an external observer. A steady increase in capabilities of such networks and a continuous decrease of the cost of their production have opened up many new possibilities for applications previously believed to be prohibitively expensive or undoable [ECaGS02]. This, on the other hand, creates great theoretical challenges in distributed signal processing, control, and other areas.

The aim of this report is to propose a unified approach to various sensor network applications as instances of "supervised learning". We focus on what we call *random wireless sensor networks* in which the locations of the nodes are assumed to be randomly distributed in the region of deployment. We assume that in the normal mode of operation, nodes communicate and collaborate only with nearby nodes (those within the communication range) but that a base station, with a more powerful computer on board, can query any node or group of nodes when a need emerges, and perform data fusion. We mainly have in mind the sensor network platform developed at UC Berkeley [CBP+], but our approach applies to a broader framework.

By supervised learning we mean learning from examples, consisting of "input-output" pairs, whereby a system, not programmed in advance, is able to estimate an unknown function and predict its values for inputs not in the "training" set. Understanding the problem of learning arguably leads to understanding intelligence in humans and machines. The problem of dealing with (or learning from) data becomes more and more important in the world flooded with massive amounts of information. Examples abound in biology, particle physics, astronomy, engineering, etc. Instrumenting the physical world with wireless sensors will generate massive

Date: July 8, 2003.

Partially supported by the NSF grant ITR/SI: Societal Scale Information Systems: Technologies, Design and Applications, number EIA-0122599.

amounts of data ("examples") waiting to be processed ("learned from") in an efficient and useful way. This is where, we believe, learning theory comes into play. Learning techniques have been applied in many diverse scenarios which we do not review here (but cf. [PS03]). Our goal is to sketch some basic ideas and discuss how they may apply to random wireless sensor networks. The main learning theory references we rely on are the recent expository papers of Cucker, Poggio, and Smale [CS02, PS03]. We also recommend the fundamental work of Vapnik [Vap98].

We point out that this is a report on work in progress, so the ideas and results presented are preliminary, and are meant to expose parts of the sensor network community to a point of view we find fruitful.

2. Basic notions of learning theory

We now outline some basic ideas of learning theory we will later need, relying heavily on [CS02] and [PS03]. We focus on supervised learning or *learning from examples*, in which systems are trained with a collection of examples, instead of being programmed in advance. By an example we mean an input-output pair representing a sample of the object (such as a function) to be learned. A classical example of learning a physical law by curve fitting to data illustrates the point.

2.1. **Example.** Suppose that a physical law to be learned is an unknown function $f : \mathbb{R} \to \mathbb{R}$ ($\mathbb{R}$ denotes the set of real numbers) of a special form. Assume, for instance, that f is a polynomial of degree N, with coefficients $w_0, \ldots, w_N$. Let $(x_1, y_1), \ldots, (x_m, y_m)$ $(m > N)$ be samples of f, where y_i is obtained by "measuring" f at x_i. If the measurements were exact, we would have $y_i = f(x_i)$, but in reality we expect them to be affected by noise. Learning f amounts to finding the vector of coefficients $w = (w_0, \ldots, w_N)$ such that $f_w(x) = \sum_{i=0}^{N} w_i x^i$ is the best fit for the data. This can be done efficiently by the *method of least squares*, going back to Gauss, which computes w such that

$$\sum_{i=1}^{m} (f_w(x_i) - y_i)^2$$

is minimized. Note that often (such as in the context of random wireless sensor networks) the x_i are random rather then chosen.

In general, we are given a set of data $(x_1, y_1), \ldots, (x_m, y_m)$, where x_i are in some *input space* X, usually a closed subset of some $\mathbb{R}^k$, and y_i belong to an *output space* Y which we assume to be $\mathbb{R}$ (in Section 4 we will see what they are in the context of sensor networks).

On the space $X \times Y$ of input-output pairs, we assume there exists a probability measure (cf., [GS97]) ρ which governs the sampling. That is, pairs (x_i, y_i) are randomly picked according to ρ. The measure ρ defines a function $f_\rho : X \to Y$, called the *regression function* of ρ. For each input x, $f_\rho(x)$ is the average (with respect to ρ) value of the output y in $\{(x, y) : y \in Y\}$. The goal is then to

learn (i.e., find a good approximation of) f_ρ from random samples $(x_1, y_1), \ldots, (x_m, y_m)$.

In other words, we would like to find a function $f : X \to Y$ which minimizes the *error*

$$\int_X (f - f_\rho)^2 d\rho_X$$

where ρ_X is the so called marginal measure on X induced by ρ [GS97].

The central question of learning theory is how well f *generalizes*, i.e., "how well it estimates the outputs for previously unseen inputs" [PS03].

We do not explore the mathematics of learning any more deeply, as it requires much more machinery which we do not have time to develop here. The interested reader is referred to [CS02, PS03], and [Vap98] for details.

3. The key algorithm

We now present the key algorithm from [PS03, CS02], which fits the set of "training data" $(x_1, y_1), \ldots, (x_m, y_m) \in X \times Y$ with a function $f : X \to Y$, where X is a closed subset of some $\mathbb{R}^k$ and $Y = \mathbb{R}$. First, some definitions. A function

$$K : X \times X \to \mathbb{R}$$

is called *symmetric* if $K(x, x') = K(x', x)$, for all $x, x' \in X$. It is called *positive definite* if, in addition, for any finite subset $\{p_1, \ldots, p_n\}$ of X, the $n \times n$ matrix with entries $K(p_i, p_j)$ is positive definite. Recall that this means that for all real numbers $c_1, \ldots, c_n$, $\sum_{i,j} c_i c_j K(p_i, p_j) \geq 0$. If K is symmetric, positive definite, and continuos, we will call it a *Mercer kernel*. Examples are

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \tag{1}$$

and

$$K(x, x') = (a^2 + \|x - x'\|^2)^{-\alpha},$$

where $\|\cdot\|$ denotes the 2-norm, and $\alpha, \sigma > 0$.

The algorithm is as follows.

Step 1 *Start with data* $(x_1, y_1), \ldots, (x_m, y_m)$.

Step 2 *Choose a Mercer kernel such as the one in* (1).

Step 3 *Choose a positive real number* γ *and let* $\mathbf{c} = (c_1, \ldots, c_m)$ *be the unique solution to the equation*

$$(m\gamma \mathbf{I} + \mathbf{K})\mathbf{c} = \mathbf{y}, \tag{2}$$

where $\mathbf{I}$ *is the* $m \times m$ *identity matrix,* $\mathbf{K}$ *is the square positive definite matrix with entries* $K(x_i, x_j)$*,* $1 \leq i, j \leq m$*, and* $\mathbf{y}$ *is the column vector with components* $y_1, \ldots, y_m$*.*

Step 4 *Define* $f : X \to Y$ *by*

$$f(x) = \sum_{i=1}^{m} c_i K(x_i, x).$$

Since the matrix $m\gamma\mathbf{I} + \mathbf{K}$ is positive definite, hence nonsingular, equation (2) is well-posed.

So what does the approximating function f look like? If the kernel is Gaussian (1), then f is a weighted sum of Gaussian "blobs" each centered at one of the x_i's.

The weights c_i are such as to minimize the error on the training set. The parameter σ from (1) controls the degree of smoothing, noise tolerance, and generalization. For instance, as $\sigma \to 0$, f becomes just a "look-up table" that gives the exact value of y when $x = x_i$, but does not generalize to other values of the input. The parameter γ ensures that the equation (2) is well-posed: the larger the $m\gamma$, the better the condition number of equation (2). On the other hand, if γ were zero, $f(x_i)$ would equal y_i exactly. It turns out that there exists a *best* γ which solves this trade-off problem.

This algorithm has been used in a number of problems. It performs well in a variety of applications involving regression as well as binary classification. Systems like (2) have been studied since Gauss; algorithms for efficient its solutions form one of the most developed fields of numerical analysis (cf., [PS03]).

We illustrate the algorithm with an example.

3.1**. Example.** Let us apply the above algorithm to a training set of m points (x_i, y_i), where the inputs x_i are random, generated by the uniform distribution on the square $Q = [-5, 5]^2$. The output y_i is a Gaussian random variable with mean $f_*(x_i)$ and variance Σ^2, where $f_*(x) = \sin \|x\|/\|x\|$. We want to estimate the (unknown) function f_* given noisy data. Figure 1 shows how the graph of the estimate changes as $m = 100, 200, 500$. Figure 2 illustrates how the estimate changes as we vary γ.

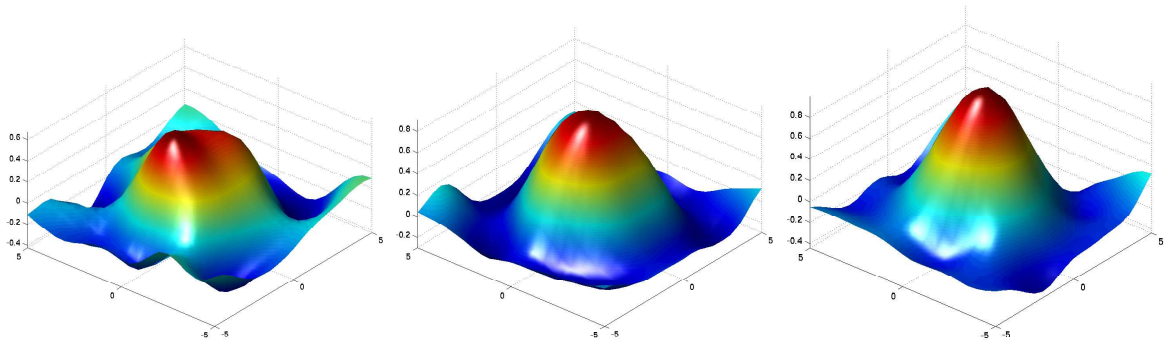

FIGURE 1. Estimates of f_* for $m = 100, 200, 500$, $\gamma = 0.001$, $\Sigma^2 = 0.01$, and $\sigma^2 = 2$.

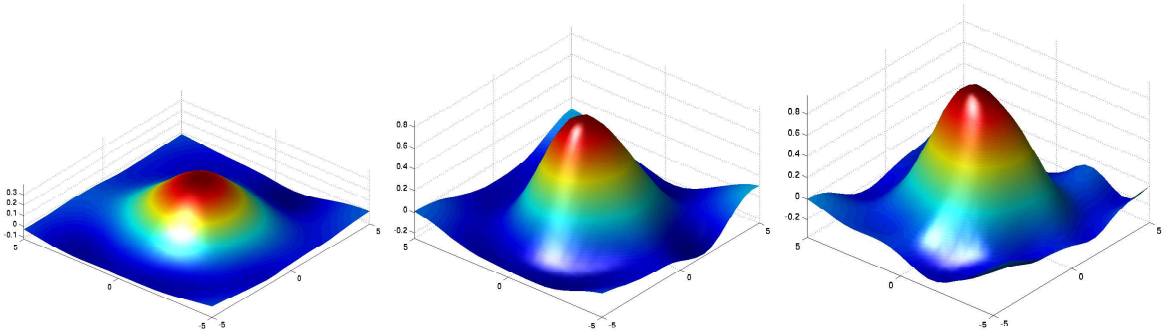

FIGURE 2. Estimates of f_* for $\gamma = 0.1, 0.01, 0.001$, $m = 100$, $\Sigma^2 = 0.01$ and $\sigma^2 = 2$.

4. Applications to sensor networks

Most applications of sensor networks can be formulated as learning about the environment through examples. Usually, we are interested in a particular feature of the environment which can be represented by a vector-valued function defined on the region on which our sensor network is deployed. Let X be that region. If a sensor network is deployed on the ground, we can take X to be a closed subset of $\mathbb{R}^2$, or, if the region is significantly curved (e.g., mountainous), a 2-dimensional surface in $\mathbb{R}^3$. If the sensor network is airborne or deployed in water, we can take $X \subset \mathbb{R}^3$.

Data generated by the sensor network are of the desired form $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$ is usually the (estimated) location of the i^{th} sensor node and y_i is an ℓ-tuple of real numbers representing measurements of the i^{th} node. Since the framework from Section 3 requires the outputs to be scalar, we decompose each y_i into scalar components $y_i = (y_i^1, \ldots, y_i^\ell)$, and work with data sets $(x_i, y_i^j)_{i=1}^m$, for $j = 1, \ldots, \ell$, separately. So, without loss, we can assume that our sensor network outputs are scalar.

Recall that points (x_i, y_i) are picked from $X \times Y$ according to some unknown probability measure ρ, which represents the noise in the measurements. What we would like to estimate (or learn) is the regression function f_ρ of ρ, which in our context is some feature of the environment represented by the outputs $y_i \approx f_\rho(x_i)$.

The main problems in applying the above algorithm to sensor networks are: 1) how much computation should be *distributed* to individual nodes, and how much of it should be *centralized*?; 2) which kernel should we choose?; 3) which values of γ and σ are best? We only discuss 1), as 2) and 3) are beyond the scope of this paper.

Centralized computation requires a lot of possibly long-range communication with a base station, which is energy consuming, but it may allow for more intensive data processing by a more powerful computer at the base station. On the other hand, distributed or localized computation at each node means a reduction in long-range communication, but the amount of computation that each node can do may be significantly limited. This is the usual trade-off between communication and computation in sensor networks. However, since transmitting

a bit of information is still an order of magnitude more expensive than computing it, we opt for a more distributed alternative, assuming that nodes can do a non-trivial amount of computation (such as solving reasonably large systems of linear equations).

Our general approach, which could be called *distributed sensorweb learning*, can be summarized as follows:

Each node S generates its data set by communicating with its neighbors and applies the algorithm to it, to obtain its own estimate f_S.

We call a node a *neighbor* of S if it can establish a bi-directional communication link with S. Denote by $\mathcal{U}_S$ the communication neighborhood of S, the subset of X such that all nodes located in $\mathcal{U}_S$ are the neighbors of S. The estimate f_S then only applies to $\mathcal{U}_S$. What happens if the communication neighborhoods of two nodes S and S' overlap - which estimate do we accept in the intersection $\mathcal{U}_S \cap \mathcal{U}_{S'}$? That is, if a need for it arises, how should the base station merge the estimates f_S and $f_{S'}$? We postpone this question to future work and for now propose the following approach. Assume the sensor network receives a query for the estimated value of the unknown function at some location $x \in X$. The base station finds a node S which is closest to x, queries S for $f_S(x)$, and takes that value as the desired estimate.

Let $S_1, \ldots, S_N$ be sensor nodes deployed in a region X, and denote the position of S_i by x_i. It often makes most sense to assume that the x_i's are independent uniformly distributed random variables on X, but other distributions are possible in particular situations as well.

We now focus on particular sensor network applications.

4.1. **Environmental monitoring.** In this case, we want to estimate some unknown scalar environmental function f_{env} defined on X, such as temperature, air pressure, humidity, light, etc (cf., [MPS+02]). We can assume that the "output" of the measurement y_i at the node S_i is a normally distributed random variable with mean $f_{\text{env}}(x_i)$.

We may also want to estimate a vector field F, such as the gradient of temperature, representing the heat flow in the environment [SS03]. This can be done be breaking the task up into estimating the scalar components of the field by each node.

4.2. **Plume tracking.** Suppose we want to track a plume of hazardous gas slowly moving through the region X (cf., [CG02, NM03]). Let A be the set representing the extent of the plume at a time instant t_0. Let f_A be the *indicator function* of A, i.e., $f_A(x) = 1$ if $x \in A$, $f_A(x) = 0$ otherwise. Estimating A amounts to estimating f_A. Each node S_i measures the concentration of the gas at its location and outputs $y_i = 1$ if the concentration is larger than some threshold τ, or $y_i = 0$ otherwise.

Alternatively, S_i can output its numerical measurement and the network can estimate the concentration function $C : X \to \mathbb{R}$. An estimate of A is then the set of all points x where $C(x) > \tau$.

4.3. **Tracking of moving objects.** Suppose an unknown set of objects $\mathcal{O}$ (such as people, animals, or vehicles) are moving through the region X. At various time instants we would like

to estimate their positions and hence trajectories. Let $\delta > 0$ be a desired estimation accuracy. We can think of each object in $\mathcal{O}$ not as a point-mass, but a "smeared point", or a disk of radius δ, and apply the key algorithm.

Denote by $\mathcal{O}_\delta$ the subset of X consisting of points x such that the distance from x to at least one object in $\mathcal{O}$ is $< \delta$ at a fixed time instance t_0. Then learning the indicator function $f_{\mathcal{O}_\delta}$ amounts to estimating the positions of objects in $\mathcal{S}$ with accuracy δ. The inputs x_i are positions of sensor nodes, while the outputs y_i are $+1$ or 0, according to whether at least one element of $\mathcal{O}$ is within the distance δ from S_i.

4.4. **Localization.** We can approach the problem of localization of nodes in a similar way. Suppose a certain number of nodes (e.g., beacons) know their positions with some accuracy. Often the first task the sensor network has to solve is to localize (i.e., estimate the positions of) the remaining nodes (cf. [SS02a, SS02b]). Let S be a node with unknown position $p \in X$, and let $\delta > 0$ be the desired accuracy. Then S can communicate with its neighbors which are aware of their positions and gather data (x_i, y_i), where x_i is the position of a known neighbor, $y_i = 1$ if the distance between that neighbor and S is $< \delta$, and 0 otherwise. Learning from the given data, S obtains an estimate of the indicator function of the disk centered at p with radius δ. We can take a point where the estimate reaches its maximum to be the position of S.

5. Simulation results

We present some preliminary simulation results in Matlab. We used the centralized version of the algorithm with Gaussian kernel 1.

Figure 3 compares the graphs of the "true" temperature function $T(x)$ for $x \in [-5, 5]^2$ (right) and its estimate (left). We took $T(x) = \exp(-\|x\|^2)$ and assumed the output y_i is a Gaussian random variable with mean $T(x_i)$ and variance 0.001. The number of nodes is $m = 100$, $\sigma^2 = 0.5$, and $\gamma = 0.0001$.

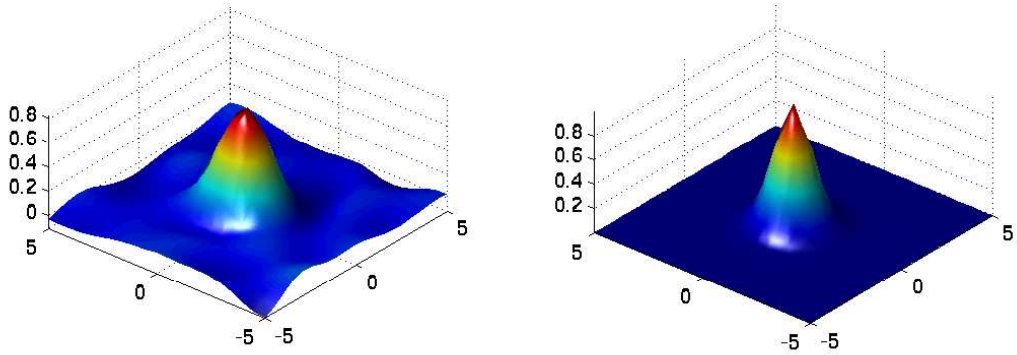

Figure 3. Temperature estimate with $m = 100$ nodes (left). The "true" temperature is shown on the right.

Figure 4 shows two estimates of the set $A = \{(u, v) : |v| < 1\}$ using $m = 200$ and 400 nodes, with $\sigma^2 = 2$ and $\gamma = 0.001$. Pictured are the graphs of the estimates of the indicator function of A.

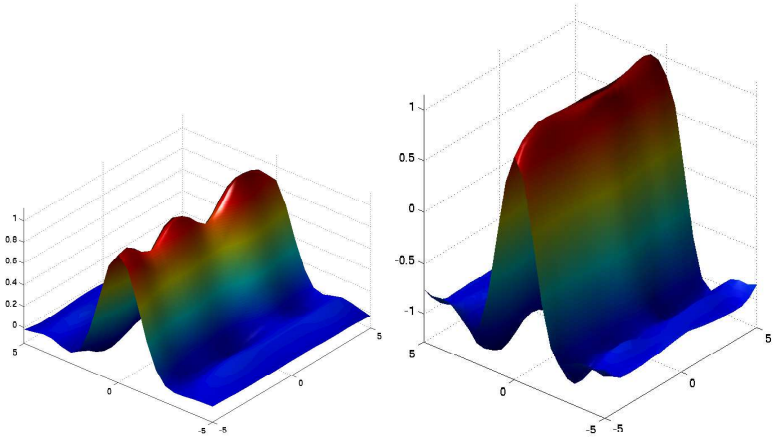

FIGURE 4. Estimates of the indicator function of the set $A = \{(u, v) : |v| < 1\}$ using $m = 200$ and 400 nodes, with $\sigma^2 = 2$ and $\gamma = 0.001$.

Figure 5 shows a localization estimate with $\sigma^2 = 2$, $\gamma = 0.0001$ and $\delta = 0.9$. We assumed the "unknown" position of node S is $(0, 0)$; S has m neighbors ($m = 30, 60$), each aware of its own position. The figure shows the graph of the function estimating the indicator function of the disk of radius δ centered at the origin. The point where it achieves its maximum (which is ideally 1) can be taken as the estimate of the position of S.

6. Conclusion

We propose an approach to various sensor network applications as problems in supervised learning theory. We presented a learning algorithm and some preliminary simulation results, arguing that, as the capabilities of sensor nodes grow, the use of powerful learning techniques, with special needs sensor networks impose taken into account, can be benefit the design of efficient algorithms for diverse applications.

References

[CBP+] D. Culler, E. Brewer, K. Pister, S. Sastry, and D. Wagner, `http://webs.cs.berkeley.edu`.

[CG02] K. K. Chintalapudi and R. Govindan, *Localized edge detection in sensor fields*, Tech. Report 02-773, University of Southern California, 2002, available at http://www.cs.usc.edu/tech-reports/technical-reports.html.

[CS02] F. Cucker and S. Smale, *On the mathematical foundations of learning*, Bulletin of the American Mathematical Society **39** (2002), no. 1, 1–49.

[ECaGS02] D. Estrin, D. Culler, and K. Pister andd G. Sukhatme, *Connecting the physical world with pervasive networks*, IEEE Pervasive Computing (2002), 59–69.

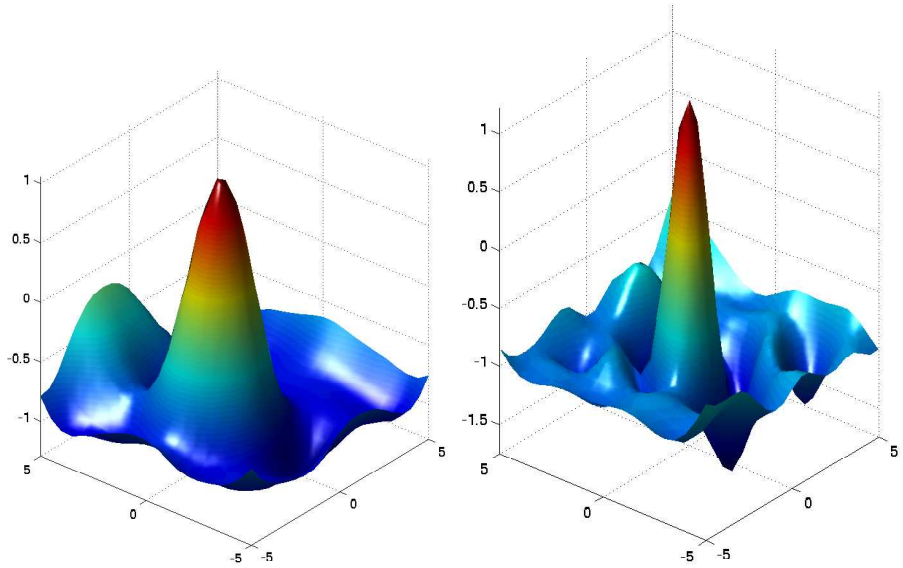

FIGURE 5. A position estimate using $m = 30$ and 60 nodes in $[-5, 5]^2$, with $\sigma^2 = 2, \gamma = 0.0001, \delta = 0.9$.. The correct position is at $(0, 0)$.

[GS97] G.R. Grimmett and D.R. Strizaker, *Probability and random processes*, 2nd ed., Oxford University Press, 1997.

[MPS+02] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, *Wireless sensor networks for habitat monitoring*, First ACM Workshop on Wireless Sensor Networks and Applications, WSNA'02 (Atlanta, GA), September 2002.

[NM03] R. Nowak and U. Mitra, *Boundary estimation in sensor networks: theory and methods*, Proceedings of the 2^{nd} International Workshop on Information Processing in Sensor Networks, IPSN'03 (PARC, Palo Alto, CA) (F. Zhao and L. Guibas, eds.), April 2003.

[PS03] T. Poggio and S. Smale, *The mathematics of learning: dealing with data*, Notices of the American Mathematical Society **50** (2003), no. 5, 537–544.

[SS02a] S. N. Simić and S. Sastry, *A distributed algorithm for localization in random wireless networks*, Discrete Applied Mathematics (2002), submitted for publication.

[SS02b] ———, *Distributed localization in wireless ad hoc networks*, Tech. Report Technical Memorandum no. UCB/ERL M02/26, UC Berkeley, 2002.

[SS03] ———, *Distributed environmental monitoring using radom sensor networks*, Proceedings of the 2^{nd} International Workshop on Information Processing in Sensor Networks, IPSN'03 (PARC, Palo Alto, CA) (F. Zhao and L. Guibas, eds.), April 2003.

[Vap98] V. Vapnik, *Statistical learning theory*, John Wiley & Sons, 1998.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, UNIVERSITY OF CALIFORNIA, BERKELEY, CA 94720

E-mail address: simic@eecs.berkeley.edu