The Annals of Applied Statistics
2011, Vol. 5, No. 4, 2403–2424
DOI: 10.1214/11-AOAS495

PROTOTYPE SELECTION FOR INTERPRETABLE CLASSIFICATION

BY JACOB BIEN[1] AND ROBERT TIBSHIRANI[2]

Stanford University

This paper is dedicated to the memory of Sam Roweis

Prototype methods seek a minimal subset of samples that can serve as a distillation or condensed view of a data set. As the size of modern data sets grows, being able to present a domain specialist with a short list of "representative" samples chosen from the data set is of increasing interpretative value. While much recent statistical research has been focused on producing sparse-in-the-variables methods, this paper aims at achieving sparsity in the samples.

We discuss a method for selecting prototypes in the classification setting (in which the samples fall into known discrete categories). Our method of focus is derived from three basic properties that we believe a good prototype set should satisfy. This intuition is translated into a set cover optimization problem, which we solve approximately using standard approaches. While prototype selection is usually viewed as purely a means toward building an efficient classifier, in this paper we emphasize the inherent value of having a set of prototypical elements. That said, by using the nearest-neighbor rule on the set of prototypes, we can of course discuss our method as a classifier as well.

We demonstrate the interpretative value of producing prototypes on the well-known USPS ZIP code digits data set and show that as a classifier it performs reasonably well. We apply the method to a proteomics data set in which the samples are strings and therefore not naturally embedded in a vector space. Our method is compatible with any dissimilarity measure, making it amenable to situations in which using a non-Euclidean metric is desirable or even necessary.

1. Introduction. Much of statistics is based on the notion that averaging over many elements of a data set is a good thing to do. In this paper, we take an opposite tack. In certain settings, selecting a small number of "representative" samples from a large data set may be of greater interpretative value than generating some "optimal" linear combination of all the elements of a data set. For domain specialists, examining a handful of representative examples of each class can be highly

Received April 2010; revised May 2011.

[1]Supported by the Urbanek Family Stanford Graduate Fellowship and the Gerald J. Lieberman Fellowship.

[2]Supported in part by NSF Grant DMS-99-71405 and National Institutes of Health Contract N01-HV-28183.

Key words and phrases. Classification, prototypes, nearest neighbors, set cover, integer program.

informative especially when n is large (since looking through all examples from the original data set could be overwhelming or even infeasible). Prototype methods aim to select a relatively small number of samples from a data set which, if well chosen, can serve as a summary of the original data set. In this paper, we motivate a particular method for selecting prototypes in the classification setting. The resulting method is very similar to Class Cover Catch Digraphs of Priebe et al. (2003). In fact, we have found many similar proposals across multiple fields, which we review later in this paper. What distinguishes this work from the rest is our interest in prototypes as a tool for better understanding a data set—that is, making it more easily "human-readable." The bulk of the previous literature has been on prototype extraction specifically for building classifiers. We find it useful to discuss our method as a classifier to the extent that it permits quantifying its abilities. However, our primary objective is aiding domain specialists in making sense of their data sets.

Much recent work in the statistics community has been devoted to the problem of interpretable classification through achieving sparsity in the *variables* [Tibshirani et al. (2002), Zhu et al. (2004), Park and Hastie (2007), Friedman, Hastie and Tibshirani (2010)]. In this paper, our aim is interpretability through sparsity in the *samples*. Consider the US Postal Service's ZIP code data set, which consists of a training set of 7,291 grayscale (16×16 pixel) images of handwritten digits 0–9 with associated labels indicating the intended digit. A typical "sparsity-in-the-variables" method would identify a subset of the pixels that is most predictive of digit-type. In contrast, our method identifies a subset of the images that, in a sense, is most predictive of digit-type. Figure 6 shows the first 88 prototypes selected by our method. It aims to select prototypes that capture the full variability of a class while avoiding confusion with other classes. For example, it chooses a wide enough range of examples of the digit "7" to demonstrate that some people add a serif while others do not; however, it avoids any "7" examples that look too much like a "1." We see that many more "0" examples have been chosen than "1" examples despite the fact that the original training set has roughly the same number of samples of these two classes. This reflects the fact that there is much more variability in how people write "0" than "1."

More generally, suppose we are given a training set of points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbf{R}^p$ with corresponding class labels $y_1, \ldots, y_n \in \{1, \ldots, L\}$. The output of our method are prototype sets $\mathcal{P}_l \subseteq \mathcal{X}$ for each class l. The goal is that someone given only $\mathcal{P}_1, \ldots, \mathcal{P}_L$ would have a good sense of the original training data, $\mathcal{X}$ and $\mathbf{y}$. The above situation describes the standard setting of a condensation problem [Hart (1968), Lozano et al. (2006), Ripley (2005)].

At the heart of our proposed method is the premise that the prototypes of class l should consist of points that are close to many training points of class l and are far from training points of other classes. This idea captures the sense in which the word "prototypical" is commonly used.

Besides the interpretative value of prototypes, they also provide a means for classification. Given the prototype sets $\mathcal{P}_1, \ldots, \mathcal{P}_L$, we may classify any new $\mathbf{x} \in \mathbf{R}^p$ according to the class whose $\mathcal{P}_l$ contains the nearest prototype:

$$\hat{c}(\mathbf{x}) = \arg\min_{l} \min_{\mathbf{z} \in \mathcal{P}_l} d(\mathbf{x}, \mathbf{z}). \tag{1}$$

Notice that this classification rule reduces to *one nearest neighbors* (1-NN) in the case that $\mathcal{P}_l$ consists of all $\mathbf{x}_i \in \mathcal{X}$ with $y_i = l$.

The 1-NN rule's popularity stems from its conceptual simplicity, empirically good performance, and theoretical properties [Cover and Hart (1967)]. Nearest prototype methods seek a lighter-weight representation of the training set that does not sacrifice (and, in fact, may improve) the accuracy of the classifier. As a classifier, our method performs reasonably well, although its main strengths lie in the ease of understanding why a given prediction has been made—an alternative to (possibly high-accuracy) "black box" methods.

In Section 2 we begin with a conceptually simple optimization criterion that describes a desirable choice for $\mathcal{P}_1, \ldots, \mathcal{P}_L$. This intuition gives rise to an integer program, which can be decoupled into L separate set cover problems. In Section 3 we present two approximation algorithms for solving the optimization problem. Section 4 discusses considerations for applying our method most effectively to a given data set. In Section 5 we give an overview of related work. In Section 6 we return to the ZIP code digits data set and present other empirical results, including an application to proteomics.

2. Formulation as an optimization problem. In this section we frame prototype selection as an optimization problem. The problem's connection to set cover will lead us naturally to an algorithm for prototype selection.

2.1. *The intuition.* Our guiding intuition is that a good set of prototypes for class l should capture the full structure of the training examples of class l while taking into consideration the structure of other classes. More explicitly, every training example should have a prototype of its same class in its neighborhood; no point should have a prototype of a different class in its neighborhood; and, finally, there should be as few prototypes as possible. These three principles capture what we mean by "prototypical." Our method seeks prototype sets with a slightly relaxed version of these properties.

As a first step, we make the notion of neighborhood more precise. For a given choice of $\mathcal{P}_l \subseteq \mathcal{X}$, we consider the set of ε-balls centered at each $\mathbf{x}_j \in \mathcal{P}_l$ (see Figure 1). A desirable prototype set for class l is then one that induces a set of balls which:

(a) covers as many training points of class l as possible,
(b) covers as few training points as possible of classes other than l, and

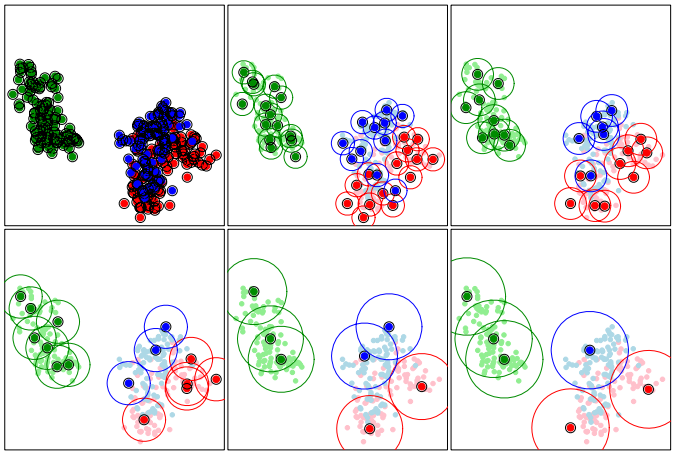

FIG. 1. *Given a value for ε, the choice of $\mathcal{P}_1, \ldots, \mathcal{P}_L$ induces L partial covers of the training points by ε-balls. Here ε is varied from the smallest* (*top-left panel*) *to approximately the median interpoint distance* (*bottom-right panel*).

(c) is sparse (i.e., uses as few prototypes as possible for the given ε).

We have thus translated our initial problem concerning prototypes into the geometric problem of selectively covering points with a specified set of balls. We will show that our problem reduces to the extensively studied set cover problem. We briefly review set cover before proceeding with a more precise statement of our problem.

2.2. *The set cover integer program.* Given a set of points $\mathcal{X}$ and a collection of sets that forms a cover of $\mathcal{X}$, the set cover problem seeks the smallest subcover of $\mathcal{X}$. Consider the following special case: Let $B(\mathbf{x}) = \{\mathbf{x}' \in \mathbf{R}^p : d(\mathbf{x}', \mathbf{x}) < \varepsilon\}$ denote the ball of radius $\varepsilon > 0$ centered at $\mathbf{x}$ (note: d need not be a metric). Clearly, $\{B(\mathbf{x}_i) : \mathbf{x}_i \in \mathcal{X}\}$ is a cover of $\mathcal{X}$. The goal is to find the smallest subset of points $\mathcal{P} \subseteq \mathcal{X}$ such that $\{B(\mathbf{x}_i) : \mathbf{x}_i \in \mathcal{P}\}$ covers $\mathcal{X}$ (i.e., every $\mathbf{x}_i \in \mathcal{X}$ is within ε of some point in $\mathcal{P}$). This problem can be written as an integer program by introducing indicator variables: $\alpha_j = 1$ if $\mathbf{x}_j \in \mathcal{P}$ and $\alpha_j = 0$ otherwise. Using this notation, $\sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j$ counts the number of times $\mathbf{x}_i$ is covered by a $B(\mathbf{x}_j)$ with $\mathbf{x}_j \in \mathcal{P}$. Thus, requiring that this sum be positive for each $\mathbf{x}_i \in \mathcal{X}$ enforces that $\mathcal{P}$ induces a cover of $\mathcal{X}$. The set cover problem is therefore equivalent to the following integer program:

$$\begin{aligned} &\text{minimize } \sum_{j=1}^{n} \alpha_j \quad \text{s.t.} \quad \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j \geq 1 \qquad \forall \mathbf{x}_i \in \mathcal{X}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \alpha_j \in \{0,1\} \qquad \forall \mathbf{x}_j \in \mathcal{X}. \end{aligned} \tag{2}$$

A feasible solution to the above integer program is one that has at least one prototype within ε of each training point.

Set cover can be seen as a clustering problem in which we wish to find the smallest number of clusters such that every point is within ε of at least one cluster center. In the language of vector quantization, it seeks the smallest codebook (restricted to $\mathcal{X}$) such that no vector is distorted by more than ε [Tipping and Schölkopf (2001)]. It was the use of set cover in this context that was the starting point for our work in developing a prototype method in the classification setting.

2.3. *From intuition to integer program.* We now express the three properties (a)–(c) in Section 2.1 as an integer program, taking as a starting point the set cover problem of (2). Property (b) suggests that in certain cases it may be necessary to leave some points of class l uncovered. For this reason, we adopt a *prize-collecting set cover* framework for our problem, meaning we assign a cost to each covering set, a penalty for being uncovered to each point and then find the minimum-cost partial cover [Könemann, Parekh and Segev (2006)]. Let $\alpha_j^{(l)} \in \{0, 1\}$ indicate whether we choose $\mathbf{x}_j$ to be in $\mathcal{P}_l$ (i.e., to be a prototype for class l). As with set cover, the sum $\sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j^{(l)}$ counts the number of balls $B(\mathbf{x}_j)$ with $\mathbf{x}_j \in \mathcal{P}_l$ that cover the point $\mathbf{x}_i$. We then set out to solve the following integer program:

$$\underset{\alpha_j^{(l)}, \xi_i, \eta_i}{\text{minimize}} \sum_i \xi_i + \sum_i \eta_i + \lambda \sum_{j,l} \alpha_j^{(l)} \quad \text{s.t.}$$

$$\text{(3a)} \qquad \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j^{(y_i)} \geq 1 - \xi_i \qquad \forall \mathbf{x}_i \in \mathcal{X},$$

$$\text{(3b)} \qquad \sum_{\substack{j:\mathbf{x}_i \in B(\mathbf{x}_j) \\ l \neq y_i}} \alpha_j^{(l)} \leq 0 + \eta_i \qquad \forall \mathbf{x}_i \in \mathcal{X},$$

$$\alpha_j^{(l)} \in \{0, 1\} \qquad \forall j, l, \qquad \xi_i, \eta_i \geq 0 \qquad \forall i.$$

We have introduced two slack variables, ξ_i and η_i, per training point $\mathbf{x}_i$. Constraint (3a) enforces that each training point be covered by at least one ball of its own class-type (otherwise $\xi_i = 1$). Constraint (3b) expresses the condition that training point $\mathbf{x}_i$ not be covered with balls of other classes (otherwise $\eta_i > 0$). In particular, ξ_i can be interpreted as indicating whether $\mathbf{x}_i$ does *not* fall within ε of any prototypes of class y_i, and η_i counts the number of prototypes of class other than y_i that are within ε of $\mathbf{x}_i$.

Finally, $\lambda \geq 0$ is a parameter specifying the cost of adding a prototype. Its effect is to control the number of prototypes chosen [corresponding to property (c) of the last section]. We generally choose $\lambda = 1/n$, so that property (c) serves only as a "tie-breaker" for choosing among multiple solutions that do equally well on prop-

erties (a) and (b). Hence, in words, we are minimizing the sum of (a) the number of points left uncovered, (b) the number of times a point is wrongly covered, and (c) the number of covering balls (multiplied by λ). The resulting method has a single tuning parameter, ε (the ball radius), which can be estimated by cross-validation.

We show in the Appendix that the above integer program is equivalent to L separate prize-collecting set cover problems. Let $\mathcal{X}_l = \{\mathbf{x}_i \in \mathcal{X} : y_i = l\}$. Then, for each class l, the set $\mathcal{P}_l \subseteq \mathcal{X}$ is given by the solution to

$$\begin{aligned} &\text{minimize} \sum_{j=1}^{m} C_l(j)\alpha_j^{(l)} + \sum_{\mathbf{x}_i \in \mathcal{X}_l} \xi_i \quad \text{s.t.} \\ &\sum_{j\,:\,\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j^{(l)} \geq 1 - \xi_i \qquad \forall \mathbf{x}_i \in \mathcal{X}_l, \\ &\alpha_j^{(l)} \in \{0,1\} \qquad \forall j, \qquad \xi_i \geq 0 \qquad \forall i : \mathbf{x}_i \in \mathcal{X}_l, \end{aligned} \tag{4}$$

where $C_l(j) = \lambda + |B(\mathbf{x}_j) \cap (\mathcal{X} \setminus \mathcal{X}_l)|$ is the cost of adding $\mathbf{x}_j$ to $\mathcal{P}_l$ and a unit penalty is charged for each point $\mathbf{x}_i$ of class l left uncovered.

3. Solving the problem: Two approaches. The prize-collecting set cover problem of (4) can be transformed to a standard set cover problem by considering each slack variable ξ_i as representing a singleton set of unit cost [Könemann, Parekh and Segev (2006)]. Since set cover is NP-hard, we do not expect to find a polynomial-time algorithm to solve our problem exactly. Further, certain inapproximability results have been proven for the set cover problem [Feige (1998)].[3] In what follows, we present two algorithms for approximately solving our problem, both based on standard approximation algorithms for set cover.

3.1. *LP relaxation with randomized rounding.* A well-known approach for the set cover problem is to relax the integer constraints $\alpha_j^{(l)} \in \{0,1\}$ by replacing it with $0 \leq \alpha_j^{(l)} \leq 1$. The result is a linear program (LP), which is convex and easily solved with any LP solver. The result is subsequently rounded to recover a feasible (though not necessarily optimal) solution to the original integer program.

Let $\{\alpha_1^{*(l)}, \ldots, \alpha_m^{*(l)}\} \cup \{\xi_i^* : i \text{ s.t. } \mathbf{x}_i \in \mathcal{X}_l\}$ denote a solution to the LP relaxation of (4) with optimal value $\mathrm{OPT}_{\mathrm{LP}}^{(l)}$. Since $\alpha_j^{*(l)}, \xi_i^* \in [0,1]$, we may think of these as probabilities and round each variable to 1 with probability given by its value in the LP solution. Following Vazirani (2001), we do this $O(\log|\mathcal{X}_l|)$ times and take the union of the partial covers from all iterations.

[3] We do not assume in general that the dissimilarities satisfy the triangle inequality, so we consider arbitrary covering sets.

We apply this randomized rounding technique to approximately solve (4) for each class separately. For class l, the rounding algorithm is as follows:

- Initialize $A_1^{(l)} = \cdots = A_m^{(l)} = 0$ and $S_i = 0\ \forall i : \mathbf{x}_i \in \mathcal{X}_l$.
- For $t = 1, \ldots, 2\log|\mathcal{X}_l|$:
  (1) Draw independently $\tilde{A}_j^{(l)} \sim \text{Bernoulli}(\alpha_j^{*(l)})$ and $\tilde{S}_i \sim \text{Bernoulli}(\xi_i^*)$.
  (2) Update $A_j^{(l)} := \max(A_j^{(l)}, \tilde{A}_j^{(l)})$ and $S_i := \max(S_i, \tilde{S}_i)$.
- If $\{A_j^{(l)}, S_i\}$ is feasible and has objective $\leq 2\log|\mathcal{X}_l|\text{OPT}_{\text{LP}}^{(l)}$, return $\mathcal{P}_l = \{\mathbf{x}_j \in \mathcal{X} : A_j^{(l)} = 1\}$. Otherwise repeat.

In practice, we terminate as soon as a feasible solution is achieved. If after $2\log|\mathcal{X}_l|$ steps the solution is still infeasible or the objective of the rounded solution is more than $2\log|\mathcal{X}_l|$ times the LP objective, then the algorithm is repeated. By the analysis given in Vazirani (2001), the probability of this happening is less than $1/2$, so it is unlikely that we will have to repeat the above algorithm very many times. Recalling that the LP relaxation gives a lower bound on the integer program's optimal value, we see that the randomized rounding yields a $O(\log|\mathcal{X}_l|)$-factor approximation to (4). Doing this for each class yields overall a $O(K \log N)$-factor approximation to (3), where $N = \max_l|\mathcal{X}_l|$. We can recover the rounded version of the slack variable η_i by $T_i = \sum_{l \neq y_i} \sum_{j : \mathbf{x}_i \in B(\mathbf{x}_j)} A_j^{(l)}$.

One disadvantage of this approach is that it requires solving an LP, which we have found can be relatively slow and memory-intensive for large data sets. The approach we describe next is computationally easier than the LP rounding method, is deterministic, and provides a natural ordering of the prototypes. It is thus our preferred method.

3.2. *A greedy approach.* Another well-known approximation algorithm for set cover is a greedy approach [Vazirani (2001)]. At each step, the prototype with the least ratio of cost to number of points newly covered is added. However, here we present a less standard greedy algorithm which has certain practical advantages over the standard one and does not in our experience do noticeably worse in minimizing the objective. At each step we find the $\mathbf{x}_j \in \mathcal{X}$ and class l for which adding $\mathbf{x}_j$ to $\mathcal{P}_l$ has the best trade-off of covering previously uncovered training points of class l while avoiding covering points of other classes. The incremental improvement of going from $(\mathcal{P}_1, \ldots, \mathcal{P}_L)$ to $(\mathcal{P}_1, \ldots, \mathcal{P}_{l-1}, \mathcal{P}_l \cup \{\mathbf{x}_j\}, \mathcal{P}_{l+1}, \ldots, \mathcal{P}_L)$ can be denoted by $\Delta\,\text{Obj}(\mathbf{x}_j, l) = \Delta\xi(\mathbf{x}_j, l) - \Delta\eta(\mathbf{x}_j, l) - \lambda$, where

$$\Delta\xi(\mathbf{x}_j, l) = \left|\mathcal{X}_l \cap \left(B(\mathbf{x}_j) \setminus \bigcup_{\mathbf{x}_{j'} \in \mathcal{P}_l} B(\mathbf{x}_{j'})\right)\right|,$$

$$\Delta\eta(\mathbf{x}_j, l) = |B(\mathbf{x}_j) \cap (\mathcal{X} \setminus \mathcal{X}_l)|.$$

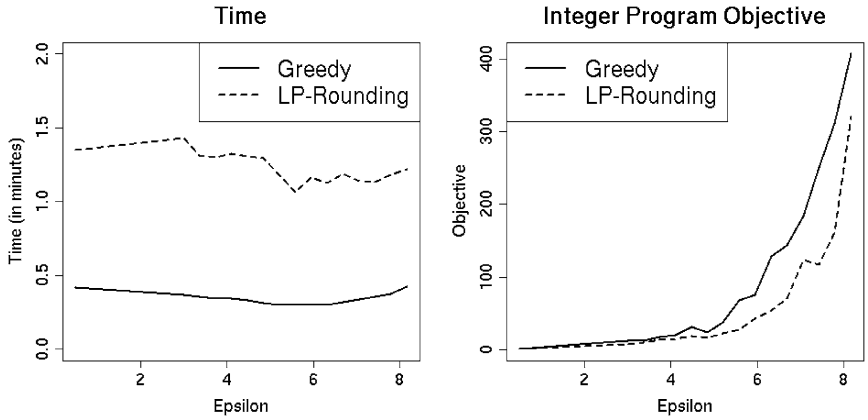

FIG. 2. *Performance comparison of LP-rounding and greedy approaches on the digits data set of Section 6.2.*

The greedy algorithm is simply as follows:

(1) Start with $\mathcal{P}_l = \varnothing$ for each class l.
(2) While $\Delta \operatorname{Obj}(\mathbf{x}^*, l^*) > 0$:

- Find $(\mathbf{x}^*, l^*) = \arg\max_{(\mathbf{x}_j, l)} \Delta \operatorname{Obj}(\mathbf{x}_j, l)$.
- Let $\mathcal{P}_{l^*} := \mathcal{P}_{l^*} \cup \{\mathbf{x}^*\}$.

Figure 2 shows a performance comparison of the two approaches on the digits data (described in Section 6.2) based on time and resulting (integer program) objective. Of course, any time comparison is greatly dependent on the machine and implementation, and we found great variability in running time among LP solvers. While low-level, specialized software could lead to significant time gains, for our present purposes, we use off-the-shelf, high-level software. The LP was solved using the `R` package `Rglpk`, an interface to the GNU Linear Programming Kit. For the greedy approach, we wrote a simple function in `R`.

4. Problem-specific considerations. In this section we describe two ways in which our method can be tailored by the user for the particular problem at hand.

4.1. *Dissimilarities.* Our method depends on the features only through the pairwise dissimilarities $d(\mathbf{x}_i, \mathbf{x}_j)$, which allows it to share in the benefits of kernel methods by using a kernel-based distance. For problems in the $p \gg n$ realm, using distances that effectively lower the dimension can lead to improvements. Additionally, in problems in which the data are not readily embedded in a vector space (see Section 6.3), our method may still be applied if pairwise dissimilarities are available. Finally, given any dissimilarity d, we may instead use $\tilde{d}$, defined by

$\tilde{d}(\mathbf{x}, \mathbf{z}) = |\{\mathbf{x}_i \in \mathcal{X} : d(\mathbf{x}_i, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{z})\}|$. Using $\tilde{d}$ induces ε-balls, $B(\mathbf{x}_j)$, consisting of the $(\lfloor \varepsilon \rfloor - 1)$ nearest training points to $\mathbf{x}_j$.

4.2. *Prototypes not on training points.* For simplicity, up until now we have described a special case of our method in which we only allow prototypes to lie on elements of the training set $\mathcal{X}$. However, our method is easily generalized to the case where prototypes are selected from any finite set of points. In particular, suppose, in addition to the labeled training data $\mathcal{X}$ and $\mathbf{y}$, we are also given a set $\mathcal{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ of unlabeled points. This situation (known as semi-supervised learning) occurs, for example, when it is expensive to obtain large amounts of labeled examples, but collecting unlabeled data is cheap. Taking $\mathcal{Z}$ as the set of potential prototypes, the optimization problem (3) is easily modified so that $\mathcal{P}_1, \ldots, \mathcal{P}_L$ are selected subsets of $\mathcal{Z}$. Doing so preserves the property that all prototypes are actual examples (rather than arbitrary points in $\mathbf{R}^p$).

While having prototypes confined to lie on actual observed points is desirable for interpretability, if this is not desired, then $\mathcal{Z}$ may be further augmented to include other points. For example, one could run K-means on each class's points individually and add these $L \cdot K$ centroids to $\mathcal{Z}$. This method seems to help especially in high-dimensional problems where constraining all prototypes to lie on data points suffers from the *curse of dimensionality*.

5. Related work. Before we proceed with empirical evaluations of our method, we discuss related work. There is an abundance of methods that have been proposed addressing the problem of how to select prototypes from a training set. These proposals appear in multiple fields under different names and with differing goals and justifications. The fact that this problem lies at the intersection of so many different literatures makes it difficult to provide a complete overview of them all. In some cases, the proposals are quite similar to our own, differing in minor details or reducing in a special case. What makes the present work different from the rest is our *goal*, which is to develop an interpretative aid for data analysts who need to make sense of a large set of labeled data. The details of our method have been adapted to this goal; however, other proposals—while perhaps intended specifically as a preprocessing step for the classification task—may be effectively adapted toward this end as well. In this section we review some of the related work to our own.

5.1. *Class cover catch digraphs.* Priebe et al. (2003) form a directed graph $D_k = (\mathcal{X}_k, E_k)$ for each class k where $(\mathbf{x}_i, \mathbf{x}_j) \in E_k$ if a ball centered at $\mathbf{x}_i$ of radius r_i covers $\mathbf{x}_j$. One choice of r_i is to make it as large as possible without covering more than a specified number of other-class points. A dominating set of D_k is a set of nodes for which all elements of $\mathcal{X}_k$ are reachable by crossing no more than one edge. They use a greedy algorithm to find an approximation to the minimum dominating set for each D_k. This set of points is then used to form the

Class Cover Catch Digraph (CCCD) Classifier, which is a nearest neighbor rule that scales distances by the radii. Noting that a dominating set of D_k corresponds to finding a set of balls that covers all points of class k, we see that their method could also be described in terms of set cover. The main difference between their formulation and ours is that we choose a fixed radius across all points, whereas in their formulation a large homogeneous region is filled by a large ball. Our choice of fixed radius seems favorable from an interpretability standpoint since there can be regions of space which are class-homogeneous and yet for which there is a lot of interesting within-class variability which the prototypes should reveal. The CCCD work is an outgrowth of the Class Cover Problem, which does not allow balls to cover wrong-class points [Cannon and Cowen (2004)]. This literature has been developed in more theoretical directions [e.g., DeVinney and Wierman (2002), Ceyhan, Priebe and Marchette (2007)].

5.2. *The set covering machine.* Marchand and Shawe-Taylor (2002) introduce the *set covering machine* (SCM) as a method for learning compact disjunctions (or conjunctions) of $\mathbf{x}$ in the binary classification setting (i.e., when $L = 2$). That is, given a potentially large set of binary functions of the features, $\mathcal{H} = \{h_j, j = 1, \ldots, m\}$ where $h_j : \mathbf{R}^p \to \{0, 1\}$, the SCM selects a relatively small subset of functions, $\mathcal{R} \subseteq \mathcal{H}$, for which the prediction rule $f(\mathbf{x}) = \bigvee_{j \in \mathcal{R}} h_j(\mathbf{x})$ (in the case of a disjunction) has low training error. Although their stated problem is unrelated to ours, the form of the optimization problem is very similar.

In Hussain, Szedmak and Shawe-Taylor (2004) the authors express the SCM optimization problem explicitly as an integer program, where the binary vector α is of length m and indicates which of the h_j are in $\mathcal{R}$:

$$\begin{aligned} &\underset{\alpha,\xi,\eta}{\text{minimize}} \sum_{j=1}^{m} \alpha_j + D\left(\sum_{i=1}^{m} \xi_i + \sum_{i=1}^{m} \eta_i\right) \quad \text{s.t.} \\ &H_+\alpha \geq 1 - \xi, \qquad H_-\alpha \leq 0 + \eta, \qquad \alpha \in \{0, 1\}^m; \qquad \xi, \eta \geq 0. \end{aligned} \tag{5}$$

In the above integer program (for the disjunction case), H_+ is the matrix with ijth entry $h_j(\mathbf{x}_i)$, with each row i corresponding to a "positive" example $\mathbf{x}_i$ and H_- the analogous matrix for "negative" examples. Disregarding the slack vectors ξ and η, this seeks the binary vector α for which every positive example is covered by at least one $h_j \in \mathcal{R}$ and for which no negative example is covered by any $h_j \in \mathcal{R}$. The presence of the slack variables permits a certain number of errors to be made on the training set, with the trade-off between accuracy and size of $\mathcal{R}$ controlled by the parameter D.

A particular choice for $\mathcal{H}$ is also suggested in Marchand and Shawe-Taylor (2002), which they call "data-dependent balls," consisting of indicator functions for the set of all balls with centers at "positive" $\mathbf{x}_i$ (and of all radii) and the complement of all balls centered at "negative" $\mathbf{x}_i$.

Clearly, the integer programs (3) and (5) are very similar. If we take $\mathcal{H}$ to be the set of balls of radius ε with centers at the positive points only, solving (5) is equivalent to finding the set of prototypes for the positive class using our method. As shown in the Appendix, (3) decouples into L separate problems. Each of these is equivalent to (5) with the positive and negative classes being $\mathcal{X}_l$ and $\mathcal{X} \setminus \mathcal{X}_l$, respectively. Despite this correspondence, Marchand and Shawe-Taylor (2002) were not considering the problem of prototype selection in their work. Since Marchand's and Shawe-Taylor's (2002) goal was to learn a conjunction (or disjunction) of binary features, they take as a classification rule $f(\mathbf{x})$; since our aim is a set of prototypes, it is natural that we use the standard nearest-prototype classification rule of (1).

For solving the SCM integer program, Hussain, Szedmak and Shawe-Taylor (2004) propose an LP relaxation; however, a key difference between their approach and ours is that they do not seek an integer solution (as we do with the randomized rounding), but rather modify the prediction rule to make use of the fractional solution directly.

Marchand and Shawe-Taylor (2002) propose a greedy approach to solving (5). Our greedy algorithm differs slightly from theirs in the following respect. In their algorithm, once a point is misclassified by a feature, no further penalty is incurred for other features also misclassifying it. In contrast, in our algorithm, a prototype is always charged if it falls within ε of a wrong-class training point. This choice is truer to the integer programs (3) and (5) since the objective has $\sum_j \eta_j$ rather than $\sum_j 1\{\eta_j > 0\}$.

5.3. *Condensation and instance selection methods.* Our method (with $\mathcal{Z} = \mathcal{X}$) selects a subset of the original training set as prototypes. In this sense, it is similar in spirit to condensing and data editing methods, such as the *condensed nearest neighbor rule* [Hart (1968)] and *multiedit* [Devijver and Kittler (1982)]. Hart (1968) introduces the notion of the minimal consistent subset—the smallest subset of $\mathcal{X}$ for which nearest-prototype classification has 0 training error. Our method's objective, $\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \eta_i + \lambda \sum_{j,l} \alpha_j^{(l)}$, represents a sort of compromise, governed by λ, between consistency (first two terms) and minimality (third term). In contrast to our method, which retains examples from the most homogeneous regions, condensation methods tend to specifically keep those elements that fall on the boundary between classes [Fayed and Atiya (2009)]. This difference highlights the distinction between the goals of reducing a data set for good classification performance versus creating a tool for interpreting a data set. Wilson and Martinez (2000) provide a good survey of *instance-based learning*, focusing—as is typical in this domain—entirely on its ability to improve the efficiency and accuracy of classification rather than discussing its attractiveness for understanding a data set. More recently, Cano, Herrera and Lozano (2007) use evolutionary algorithms to

perform instance selection with the goal of creating decision trees that are both precise and interpretable, and Marchiori (2010) suggests an instance selection technique focused on having a large hypothesis margin. Cano, Herrera and Lozano (2003) compare the performance of a number of instance selection methods.

5.4. *Other methods*. We also mention a few other nearest prototype methods. K-means and K-medoids are common unsupervised methods which produce prototypes. Simply running these methods on each class separately yields prototype sets $\mathcal{P}_1, \ldots, \mathcal{P}_L$. K-medoids is similar to our method in that its prototypes are selected from a finite set. In contrast, K-means's prototypes are not required to lie on training points, making the method *adaptive*. While allowing prototypes to lie anywhere in $\mathbf{R}^p$ can improve classification error, it also reduces the interpretability of the prototypes (e.g., in data sets where each $\mathbf{x}_i$ represents an English word, producing a linear combination of hundreds of words offers little interpretative value). Probably the most widely used adaptive prototype method is *learning vector quantization* [LVQ, Kohonen (2001)]. Several versions of LVQ exist, varying in certain details, but each begins with an initial set of prototypes and then iteratively adjusts them in a fashion that tends to encourage each prototype to lie near many training points of its class and away from training points of other classes.

Takigawa, Kudo and Nakamura (2009) propose an idea similar to ours in which they select convex sets to represent each class, and then make predictions for new points by finding the set with nearest boundary. They refer to the selected convex sets themselves as prototypes.

Finally, in the main example of this paper (Section 6.2), we observe that the relative proportion of prototypes selected for each class reveals that certain classes are far more complex than others. We note here that quantifying the complexity of a data set is itself a subject that has been studied extensively [Basu and Ho (2006)].

6. Examples on simulated and real data. We demonstrate the use of our method on several data sets and compare its performance as a classifier to some of the prototype methods best known to statisticians. Classification error is a convenient metric for demonstrating that our proposal is reasonable even though building a classifier is not our focus. All the methods we include are similar in that they first choose a set of prototypes and then use the nearest-prototype rule to classify. LVQ and K-means differ from the rest in that they do not constrain the prototypes to lie on actual elements of the training set (or any prespecified finite set $\mathcal{Z}$). We view this flexibility as a hinderance for interpretability but a potential advantage for classification error.

For K-medoids, we run the function `pam` of the `R` package `cluster` on each class's data separately, producing K prototypes per class. For LVQ, we use the functions `lvqinit` and `olvq1` [optimized learning vector quantization 1, Kohonen (2001)] from the `R` package `class`. We vary the initial codebook size to produce a range of solutions.

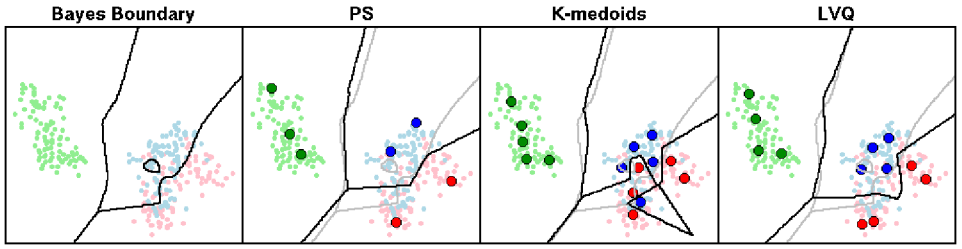

FIG. 3. *Mixture of Gaussians. Classification boundaries of Bayes, our method (PS), K-medoids and LVQ (Bayes boundary in gray for comparison).*

6.1. *Mixture of Gaussians simulation.* For demonstration purposes, we consider a three-class example with $p = 2$. Each class was generated as a mixture of 10 Gaussians. Figure 1 shows our method's solution for a range of values of the tuning parameter ε. In Figure 3 we display the classification boundaries of a number of methods. Our method (which we label as "PS," for prototype selection) and LVQ succeed in capturing the shape of the boundary, whereas K-medoids has an erratic boundary; it does not perform well when classes overlap since it does not take into account other classes when choosing prototypes.

6.2. *ZIP code digits data.* We return now to the USPS handwritten digits data set, which consists of a training set of $n = 7{,}291$ grayscale (16×16 pixel) images of handwritten digits 0–9 (and 2,007 test images). We ran our method for a range of values of ε from the minimum interpoint distance (in which our method retains the entire training set and so reduces to 1-NN classification) to approximately the 14th percentile of interpoint distances.

The left-hand panel of Figure 4 shows the test error as a function of the number of prototypes for several methods using the Euclidean metric. Since both LVQ and K-means can place prototypes anywhere in the feature space, which is advantageous in high-dimensional problems, we also allow our method to select prototypes that do not lie on the training points by augmenting $\mathcal{Z}$. In this case, we run 10-means clustering on each class separately and then add these resulting 100 points to $\mathcal{Z}$ (in addition to $\mathcal{X}$).

The notion of the *tangent distance* between two such images was introduced by Simard, Le Cun and Denker (1993) to account for certain invariances in this problem (e.g., the thickness and orientation of a digit are not relevant factors when we consider how similar two digits are). Use of tangent distance with 1-NN attained the lowest test errors of any method [Hastie and Simard (1998)]. Since our method operates on an arbitrary dissimilarities matrix, we can easily use the tangent distance in place of the standard Euclidean metric. The righthand panel of Figure 4 shows the test errors when tangent distance is used. K-medoids similarly readily accommodates any dissimilarity. While LVQ has been generalized to arbitrary differentiable metrics, there does not appear to be generic, off-the-shelf

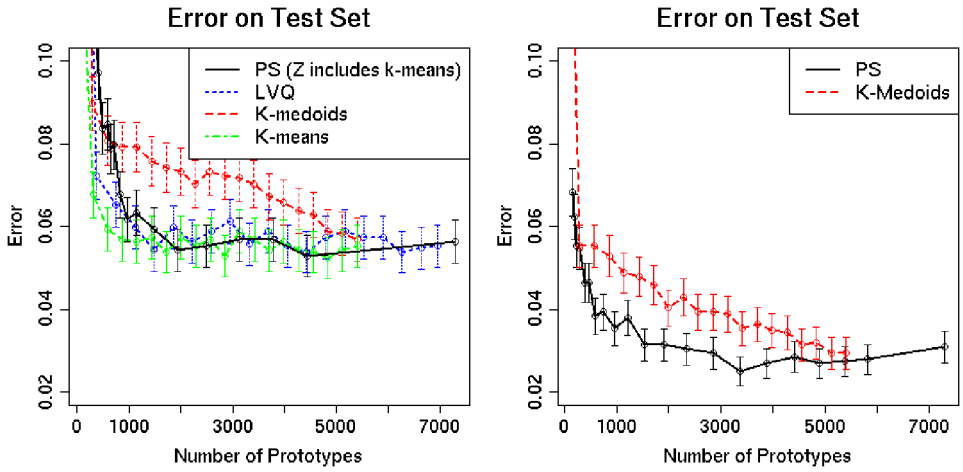

FIG. 4. *Digits data set. Left: all methods use Euclidean distance and allow prototypes to lie off of training points (except for K-medoids). Right: both use tangent distance and constrain prototypes to lie on training points. The rightmost point on our method's curve (black) corresponds to 1-NN.*

software available. The lowest test error attained by our method is 2.49% with a 3,372-prototype solution (compared to 1-NNs 3.09%).[4] Of course, the minimum of the curve is a biased estimate of test error; however, it is reassuring to note that for a wide range of ε values we get a solution with test error comparable to that of 1-NN, but requiring far fewer prototypes.

As stated earlier, our primary interest is in the interpretative advantage offered by our method. A unique feature of our method is that it automatically chooses the relative number of prototypes per class to use. In this example, it is interesting to examine the class-frequencies of prototypes (Table 1).

The most dramatic feature of this solution is that it only retains seven of the 1,005 examples of the digit 1. This reflects the fact that, relative to other digits, the digit 1 has the least variation when handwritten. Indeed, the average (tangent)

TABLE 1
Comparison of number of prototypes chosen per class to training set size

	Digit										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
Training set	1,194	1,005	731	658	652	556	664	645	542	644	7,291
PS-best	493	7	661	551	324	486	217	101	378	154	3,372

[4] Hastie and Simard (1998) report a 2.6% test error for 1-NN on this data set. The difference may be due to implementation details of the tangent distance.

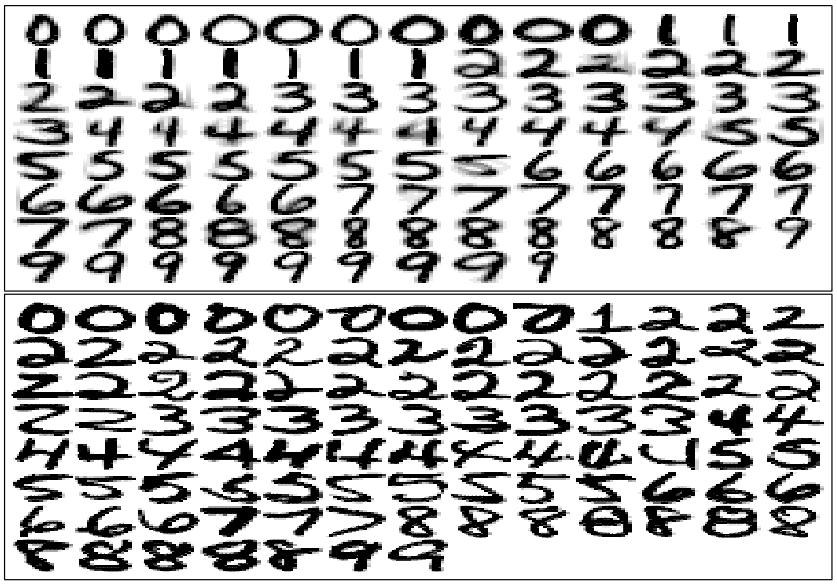

FIG. 5. (*Top*) *centroids from* 10*-means clustering within each class*. (*Bottom*) *prototypes from our method* (*where* ε *was chosen to give approximately* 100 *prototypes*). *The images in the bottom panel are sharper and show greater variety since each is a single handwritten image.*

distance between digit 1's in the training set is less than half that of any other digit (the second least variable digit is 7). Our choice to force all balls to have the same radius leads to the property that classes with greater variability acquire a larger proportion of the prototypes. By contrast, K-medoids requires the user to decide on the relative proportions of prototypes across the classes.

Figure 5 provides a qualitative comparison between centroids from K-means and prototypes selected by our method. The upper panel shows the result of 10-means clustering within each class; the lower panel shows the solution of our method tuned to generate approximately 100 prototypes. Our prototypes are sharper and show greater variability than those from K-means. Both of these observations reflect the fact that the K-means images are averages of many training samples, whereas our prototypes are single original images from the training set. As observed in the 3,372-prototype solution, we find that the relative numbers of prototoypes in each class for our method adapts to the within-class variability.

Figure 6 shows images of the first 88 prototypes (of 3,372) selected by the greedy algorithm. Above each image is the number of training images previously uncovered that were correctly covered by the addition of this prototype and, in parentheses, the number of training points that are miscovered by this prototype. For example, we can see that the first prototype selected by the greedy algorithm, which was a "1," covered 986 training images of 1's and four training images that were not of 1's. Figure 7 displays these in a more visually descriptive way: we

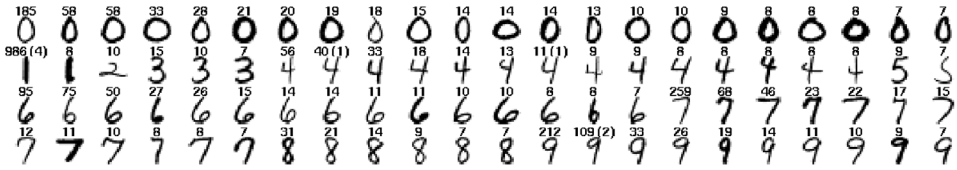

FIG. 6. *First* 88 *prototypes from greedy algorithm. Above each is the number of training images first correctly covered by the addition of this prototype* (*in parentheses is the number of miscovered training points by this prototype*).

have used multidimensional scaling to arrange the prototypes to reflect the tangent distances between them. Furthermore, the size of each prototype is proportional to the log of the number of training images correctly covered by it. Figure 8 shows a complete-linkage hierarchical clustering of the training set with images of the 88 prototypes. Figures 6–8 demonstrate ways in which prototypes can be used to graphically summarize a data set. These displays could be easily adapted to other domains, for example, by using gene names in place of the images.

The left-hand panel of Figure 9 shows the improvement in the objective, $\Delta\xi - \Delta\eta$, after each step of the greedy algorithm, revealing an interesting feature of the solution: we find that after the first 458 prototypes are added, each remaining prototype covers only one training point. Since in this example we took $\mathcal{Z} = \mathcal{X}$ (and since a point always covers itself), this means that the final 2,914 prototypes

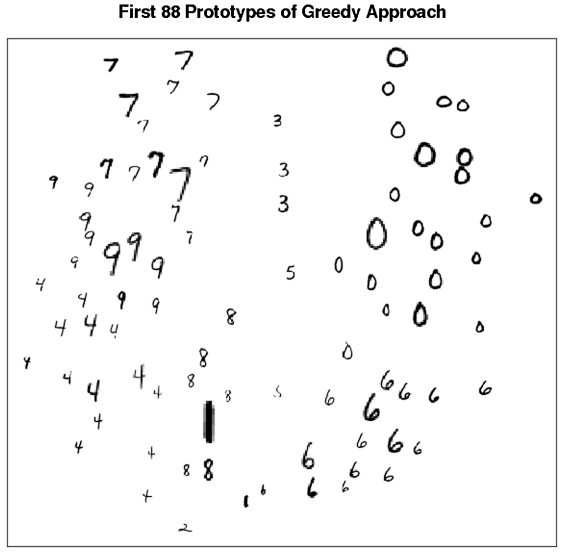

FIG. 7. *The first* 88 *prototypes* (*out of* 3,372) *of the greedy solution. We perform MDS* (R *function* `sammon`) *on the tangent distances to visualize the prototypes in two dimensions. The size of each prototype is proportional to the log of the number of correct-class training images covered by this prototype.*

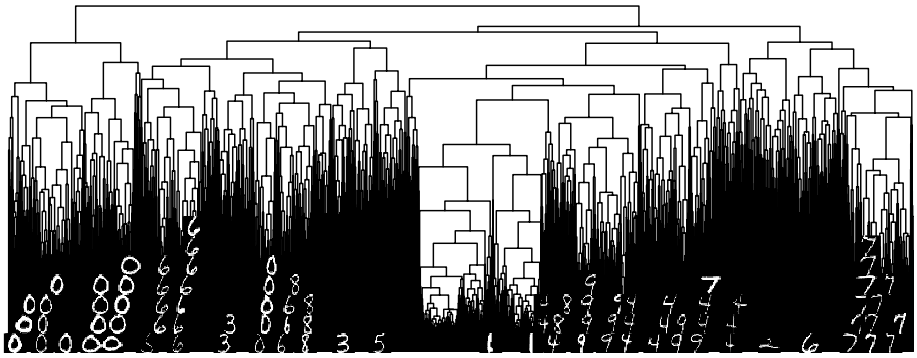

FIG. 8. *Complete-linkage hierarchical clustering of the training images* (*using* `R` *package* `glus` *to order the leaves*). *We display the prototype digits where they appear in the tree. Differing vertical placement of the images is simply to prevent overlap and has no meaning.*

were chosen to cover only themselves. In this sense, we see that our method provides a sort of compromise between a sparse nearest prototype classifier and 1-NN. This compromise is determined by the prototype-cost parameter λ. If $\lambda > 1$, the algorithm does not enter the 1-NN regime. The right-hand panel shows that the test error continues to improve as λ decreases.

6.3. *Protein classification with string kernels.* We next present a case in which the training samples are not naturally represented as vectors in $\mathbf{R}^p$. Leslie et al. (2004) study the problem of classification of proteins based on their amino acid sequences. They introduce a measure of similarity between protein sequences called the *mismatch kernel*. The general idea is that two sequences should be considered similar if they have a large number of short sequences in common (where two short sequences are considered the same if they have no more than a specified number of mismatches). We take as input a 1,708 $\times$ 1,708 matrix with K_{ij}

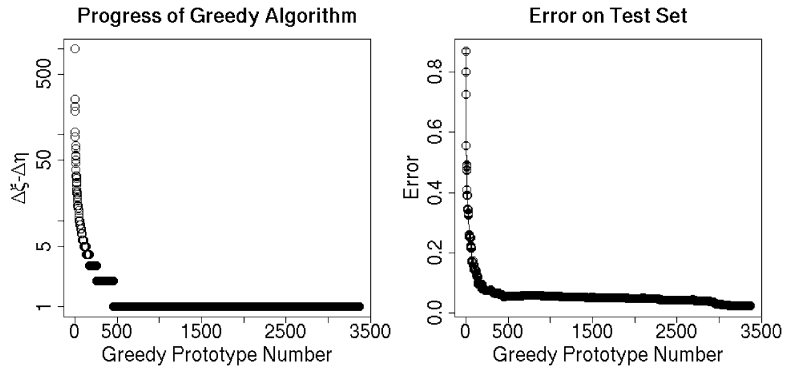

FIG. 9. *Progress of greedy on each iteration.*

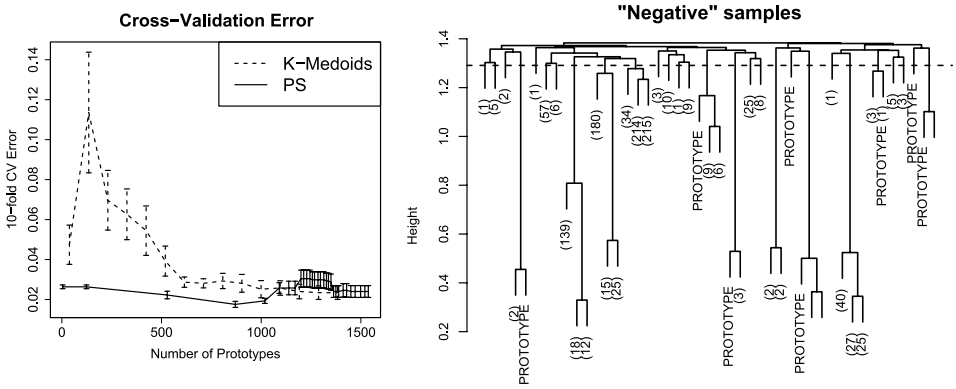

FIG. 10. *Proteins data set. Left*: *CV error* (*recall that the rightmost point on our method's curve corresponds to* 1-*NN*). *Right*: *a complete-linkage hierarchical clustering of the negative samples. Each selected prototype is marked. The dashed line is a cut at height* ε. *Thus*, *samples that are merged below this line are within* ε *of each other. The number of "positive" samples within* ε *of each negative sample*, *if nonzero*, *is shown in parentheses*.

containing the value of the normalized mismatch kernel evaluated between proteins i and j [the data and software are from Leslie et al. (2004)]. The proteins fall into two classes, "Positive" and "Negative," according to whether they belong to a certain protein family. We compute pairwise distances from this kernel via $D_{ij} = \sqrt{K_{ii} + K_{jj} - 2K_{ij}}$ and then run our method and K-medoids. The left panel of Figure 10 shows the 10-fold cross-validated errors for our method and K-medoids. For our method, we take a range of equally-spaced quantiles of the pairwise distances from the minimum to the median for the parameter ε. For K-medoids, we take as parameter the fraction of proteins in each class that should be prototypes. This choice of parameter allows the classes to have different numbers of prototypes, which is important in this example because the classes are greatly imbalanced (only 45 of the 1,708 proteins are in class "Positive"). The right panel of Figure 10 shows a complete linkage hierarchical clustering of the 45 samples in the "Negative" class with the selected prototypes indicated. Samples joined below the dotted line are within ε of each other. Thus, performing regular set cover would result in every branch that is cut at this height having at least one prototype sample selected. By contrast, our method leaves some branches without prototypes. In parentheses, we display the number of samples from the "Positive" class that are within ε of each "Negative" sample. We see that the branches that do not have protoypes are those for which every "Negative" sample has too many "Positive" samples within ε to make it a worthwhile addition to the prototype set.

The minimum CV-error (1.76%) is attained by our method using about 870 prototypes (averaged over the 10 models fit for that value of ε). This error is identical to the minimum CV-error of a support vector machine (tuning the cost parameter) trained using this kernel. Fitting a model to the whole data set with the selected

TABLE 2
10-*fold CV* (*with the* 1 *SE rule*) *on the training set to tune the parameters* (*our method labeled "PS"*)

Data		**1-NN/ℓ_2**	**1-NN/ℓ_1**	**PS/ℓ_2**	**PS/ℓ_1**	***K*-med./ℓ_2**	***K*-med./ℓ_1**	**LVQ**
Diabetes	*Test Err*	28.9	31.6	24.2	26.6	32.0	34.4	25.0
($p = 8, L = 2$)	*# Protos*	512	512	12	5	194	60	29
Glass	*Test Err*	38.0	32.4	36.6	47.9	39.4	38.0	35.2
($p = 9, L = 6$)	*# Protos*	143	143	34	17	12	24	17
Heart	*Test Err*	21.1	23.3	21.1	13.3	22.2	24.4	15.6
($p = 13, L = 2$)	*# Protos*	180	180	6	4	20	20	12
Liver	*Test Err*	41.7	41.7	41.7	32.2	46.1	48.7	33.9
($p = 6, L = 2$)	*# Protos*	230	230	16	13	120	52	110
Vowel	*Test Err*	2.8	1.7	2.8	1.7	2.8	4.0	24.4
($p = 10, L = 11$)	*# Protos*	352	352	352	352	198	165	138
Wine	*Test Err*	3.4	3.4	11.9	6.8	6.8	1.7	3.4
($p = 13, L = 3$)	*# Protos*	119	119	4	3	12	39	3

value of ε, our method chooses 26 prototypes (of 45) for class "Positive" and 907 (of 1,663) for class "Negative."

6.4. *UCI data sets*. Finally, we run our method on six data sets from the UCI Machine Learning Repository [Asuncion and Newman (2007)] and compare its performance to that of 1-NN (i.e., retaining all training points as prototypes), K-medoids and LVQ. We randomly select 2/3 of each data set for training and use the remainder as a test set. Ten-fold cross-validation [and the "1 standard error rule," Hastie, Tibshirani and Friedman (2009)] is performed on the training data to select a value for each method's tuning parameter (except for 1-NN). Table 2 reports the error on the test set and the number of prototypes selected for each method. For methods taking a dissimilarity matrix as input, we use both ℓ_2 and ℓ_1 distance measures. We see that in most cases our method is able to do as well as or better than 1-NN but with a significant reduction in prototypes. No single method does best on all of the data sets. The difference in results observed for using ℓ_1 versus ℓ_2 distances reminds us that the choice of dissimilarity is an important aspect of any problem.

7. Discussion. We have presented a straightforward procedure for selecting prototypical samples from a data set, thus providing a simple way to "summarize" a data set. We began by explicitly laying out our notion of a desirable prototype set, then cast this intuition as a set cover problem which led us to two standard approximation algorithms. The digits data example highlights several strengths. Our method automatically chooses a suitable number of prototypes for each class. It

is flexible in that it can be used in conjunction with a problem-specific dissimilarity, which in this case helps our method attain a competitive test error for a wide range of values of the tuning parameter. However, the main motivation for using this method is interpretability: each prototype is an element of $\mathcal{X}$ (i.e., is an actual hand drawn image). In medical applications, this would mean that prototypes correspond to actual patients, genes, etc. This feature should be useful to domain experts for making sense of large data sets. Software for our method will be made available as an `R` package in the `R` library.

APPENDIX: INTEGER PROGRAM (3)'S RELATION TO PRIZE-COLLECTING SET COVER

CLAIM. *Solving the integer program of* (3) *is equivalent to solving* L *prize-collecting set cover problems*.

PROOF. Observing that the constraints (3b) are always tight, we can eliminate $\eta_1, \ldots, \eta_n$ in (3), yielding

$$\underset{\alpha_j^{(l)}, \xi_i, \eta_i}{\text{minimize}} \sum_i \xi_i + \sum_i \sum_{\substack{j : \mathbf{x}_i \in B(\mathbf{z}_j) \\ l \neq y_i}} \alpha_j^{(l)} + \lambda \sum_{j,l} \alpha_j^{(l)} \quad \text{s.t.}$$

$$\sum_{j : \mathbf{x}_i \in B(\mathbf{z}_j)} \alpha_j^{(y_i)} \geq 1 - \xi_i \qquad \forall \mathbf{x}_i \in \mathcal{X},$$

$$\alpha_j^{(l)} \in \{0, 1\} \qquad \forall j, l, \qquad \xi_i \geq 0 \qquad \forall i.$$

Rewriting the second term of the objective as

$$\sum_{i=1}^{n} \sum_{\substack{j : \mathbf{x}_i \in B(\mathbf{z}_j) \\ l \neq y_i}} \alpha_j^{(l)} = \sum_{j,l} \alpha_j^{(l)} \sum_{i=1}^{n} 1\{\mathbf{x}_i \in B(\mathbf{z}_j), \mathbf{x}_i \notin \mathcal{X}_l\}$$

$$= \sum_{j,l} \alpha_j^{(l)} |B(\mathbf{z}_j) \cap (\mathcal{X} \setminus \mathcal{X}_l)|$$

and letting $C_l(j) = \lambda + |B(\mathbf{z}_j) \cap (\mathcal{X} \setminus \mathcal{X}_l)|$ gives

$$\underset{\alpha_j^{(l)}, \xi_i}{\text{minimize}} \sum_{l=1}^{L} \left[\sum_{\mathbf{x}_i \in \mathcal{X}_l} \xi_i + \sum_{j=1}^{m} C_l(j) \alpha_j^{(l)} \right]$$

s.t. for each class l:

$$\sum_{j : \mathbf{x}_i \in B(\mathbf{z}_j)} \alpha_j^{(l)} \geq 1 - \xi_i \qquad \forall \mathbf{x}_i \in \mathcal{X}_l,$$

$$\alpha_j^{(l)} \in \{0, 1\} \qquad \forall j, \qquad \xi_i \geq 0 \qquad \forall i : \mathbf{x}_i \in \mathcal{X}_l.$$

This is separable with respect to class and thus equivalent to L separate integer programs. The lth integer program has variables $\alpha_1^{(l)}, \ldots, \alpha_m^{(l)}$ and $\{\xi_i : \mathbf{x}_i \in \mathcal{X}_l\}$ and is precisely the prize-collecting set cover problem of (4). $\square$

Acknowledgments. We thank Sam Roweis for showing us set cover as a clustering method, Sam Roweis, Amin Saberi, Daniela Witten for helpful discussions, and Trevor Hastie for providing us with his code for computing tangent distance.

REFERENCES

ASUNCION, A. and NEWMAN, D. J. (2007). UCI Machine Learning Repository. Univ. California, Irvine, School of Information and Computer Sciences.

BASU, M. and HO, T. K. (2006). *Data Complexity in Pattern Recognition*. Springer, London.

CANNON, A. H. and COWEN, L. J. (2004). Approximation algorithms for the class cover problem. *Ann. Math. Artif. Intell.* **40** 215–223. MR2037478

CANO, J. R., HERRERA, F. and LOZANO, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation* **7** 561–575.

CANO, J. R., HERRERA, F. and LOZANO, M. (2007). Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. *Data and Knowledge Engineering* **60** 90–108.

CEYHAN, E., PRIEBE, C. E. and MARCHETTE, D. J. (2007). A new family of random graphs for testing spatial segregation. *Canad. J. Statist.* **35** 27–50. MR2345373

COVER, T. M. and HART, P. (1967). Nearest neighbor pattern classification. *Proc. IEEE Trans. Inform. Theory* **IT-11** 21–27.

DEVIJVER, P. A. and KITTLER, J. (1982). *Pattern Recognition*: *A Statistical Approach*. Prentice Hall, Englewood Cliffs, NJ. MR0692767

DEVINNEY, J. and WIERMAN, J. C. (2002). A SLLN for a one-dimensional class cover problem. *Statist. Probab. Lett.* **59** 425–435. MR1935677

FAYED, H. A. and ATIYA, A. F. (2009). A novel template reduction approach for the K-nearest neighbor method. *IEEE Transactions on Neural Networks* **20** 890–896.

FEIGE, U. (1998). A threshold of $\ln n$ for approximating set cover. *J. ACM* **45** 634–652. MR1675095

FRIEDMAN, J. H., HASTIE, T. and TIBSHIRANI, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* **33** 1–22.

HART, P. (1968). The condensed nearest-neighbor rule. *IEEE Trans. Inform. Theory* **14** 515–516.

HASTIE, T. and SIMARD, P. Y. (1998). Models and metrics for handwritten digit recognition. *Statist. Sci.* **13** 54–65.

HASTIE, T., TIBSHIRANI, R. and FRIEDMAN, J. (2009). *The Elements of Statistical Learning*: *Data Mining*, *Inference*, *and Prediction*, 2nd ed. Springer, New York. MR2722294

HUSSAIN, Z., SZEDMAK, S. and SHAWE-TAYLOR, J. (2004). The linear programming set covering machine. *Pattern Analysis, Statistical Modelling and Computational Learning*.

KOHONEN, T. (2001). *Self-Organizing Maps*, 3rd ed. *Springer Series in Information Sciences* **30**. Springer, Berlin. MR1844512

KÖNEMANN, J., PAREKH, O. and SEGEV, D. (2006). A unified approach to approximating partial covering problems. In *Algorithms—ESA* 2006. *Lecture Notes in Computer Science* **4168** 468–479. Springer, Berlin. MR2347166

LESLIE, C. S., ESKIN, E., COHEN, A., WESTON, J. and NOBLE, W. S. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics* **20** 467–476.

LOZANO, M., SOTOCA, J. M., SÁNCHEZ, J. S., PLA, F., PKALSKA, E. and DUIN, R. P. W. (2006). Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces. *Pattern Recognition* **39** 1827–1838.

MARCHAND, M. and SHAWE-TAYLOR, J. (2002). The set covering machine. *J. Mach. Learn. Res.* **3** 723–746.

MARCHIORI, E. (2010). Class conditional nearest neighbor for large margin instance selection. *IEEE Trans. Pattern Anal. Mach. Intell.* **32** 364–370.

PARK, M. Y. and HASTIE, T. (2007). L_1-regularization path algorithm for generalized linear models. *J. R. Stat. Soc. Ser. B Stat. Methodol.* **69** 659–677. MR2370074

PRIEBE, C. E., DEVINNEY, J. G., MARCHETTE, D. J. and SOCOLINSKY, D. A. (2003). Classification using class cover catch digraphs. *J. Classification* **20** 3–23. MR1983119

RIPLEY, B. D. (2005). *Pattern Recognition and Neural Networks*. Cambridge Univ. Press, New York.

SIMARD, P. Y., LE CUN, Y. A. and DENKER, J. S. (1993). Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems* 50–58. Morgan Kaufmann, San Mateo, CA.

TAKIGAWA, I., KUDO, M. and NAKAMURA, A. (2009). Convex sets as prototypes for classifying patterns. *Eng. Appl. Artif. Intell.* **22** 101–108.

TIBSHIRANI, R., HASTIE, T., NARASIMHAN, B. and CHU, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proc. Natl. Acad. Sci. USA* **99** 6567–6572.

TIPPING, M. E. and SCHÖLKOPF, B. (2001). A kernel approach for vector quantization with guaranteed distortion bounds. In *Artificial Intelligence and Statistics* (T. Jaakkola and T. Richardson, eds.) 129–134. Morgan Kaufmann, San Francisco.

VAZIRANI, V. V. (2001). *Approximation Algorithms*. Springer, Berlin. MR1851303

WILSON, D. R. and MARTINEZ, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning* **38** 257–286.

ZHU, J., ROSSET, S., HASTIE, T. and TIBSHIRANI, R. (2004). 1-norm support vector machines. In *Advances in Neural Information Processing Systems* 16 (S. Thrun, L. Saul and B. Schölkopf, eds.). MIT Press, Cambridge, MA.

DEPARTMENT OF STATISTICS
STANFORD UNIVERSITY
SEQUOIA HALL
390 SERRA MALL
STANFORD, CALIFORNIA 94305
USA
E-MAIL: jbien@stanford.edu

DEPARTMENTS
OF HEALTH, RESEARCH, AND POLICY
AND STATISTICS
STANFORD UNIVERSITY
SEQUOIA HALL
390 SERRA MALL
STANFORD, CALIFORNIA 94305
USA
E-MAIL: tibs@stanford.edu