Supervised Learning by Training on Aggregate Outputs

David R. Musicant, Robert Atlas, Janara M. Christensen, Jamie F. Olson, Jeffrey M. Rzeszotarski, Emma R. D. Turetsky

Abstract—Supervised learning is a classic data mining problem where one wishes to be able to predict an output value associated with a particular input vector. We present a new twist on this classic problem where, instead of having the training set contain an individual output value for each input vector, the output values in the training set are only given in aggregate over a number of input vectors. This new problem arose from a particular need in learning on mass spectrometry data, but could easily apply to situations when data has been aggregated in order to maintain privacy. We provide a formal description of this new problem for both classification and regression. We then examine how k-nearest neighbor, neural networks, support vector machines, and decision trees can be adapted for this problem.

Index Terms—Data mining, classifier design and evaluation.

1 Introduction

Supervised learning is a classic data mining problem where one wishes to be able to predict an *output value* associated with a particular *input vector*. A predictor is constructed via the use of a *training set*, which consists of a set of input vectors with a corresponding output value for each. This predictor can then be used to predict output values for future input vectors where the output values are unknown. If the output data is continuous, this task is referred to as *regression*. If the output data is nominal, this task is referred to as *classification* [1].

An important application that we are currently engaged in is the analysis of single particle mass spectrometry (SPMS) data. A typical SPMS dataset consists of a series of mass spectra, each one of which is associated with an aerosol particle. One important goal to the SPMS community is to be able to use the mass spectrum associated with an individual aerosol particle to be able to predict some other quantity associated with that particle. For example, we are interested in being able to identify the quantity of *black carbon* (BC) associated with an individual aerosol particle. While SPMS data provides detailed mass spectra for individual aerosol particles, these spectra are not quantitative enough to allow one to precisely identify the amount of BC present in a particle. Alternatively, filter-based machines can collect quantitative BC data, but in aggregate form. These devices allow one to determine how much BC has been observed in aggregate over a particular time span, typically on the order of an hour. SPMS data, on the other hand, may be collected for individual particles (non-aggregate data) many times per minute. Our goal is to be able to predict the quantity of BC present in an individual aerosol particle. Therefore, the mass spectra associated with individual particles become our input vectors, and the quantities of BC become our output values. This would appear to be a traditional regression problem, but there is a key difference. The challenge that we face is that we do not have a single output value for each input vector. Instead, each output value in our training set represents the sum of the true output values (which are unknown to us) for a series of input vectors in our training set. We wish to learn a predictor that will still produce individual output values for each new input vector, even though the training set only contains these output values in aggregate.

We present a new framework for this supervised learning problem, which we refer to as the *aggregate output learning problem*. We develop both a classification and a regression version, and show how a number of popular supervised learning algorithms can be adapted to build predictors on data of this sort. While we arrived at the need for this algorithm from a particular scientific application, this framework would easily apply to training data which has been aggregated for purposes of privacy preservation, or in any situation where one can label a training set by making estimates on totals or proportions as opposed to labeling individual instances.

There are two key contributions of this paper. First, we present a formal framework for this machine learning problem that describes both a classification and a regression version of the problem. Second, we adapt four classic algorithms, namely k-nearest neighbor, neural networks, support vector machines, and decision trees,

- *D. Musicant (dmusican@carleton.edu), R. Atlas, J. Rzeszotarski, and E. Turetsky are with the Department of Computer Science, Carleton College, Northfield, MN 55057.*
- *J. Christensen is with the Department of Computer Science & Engineering at the University of Washington, Seattle, WA, 98195.*
- *J. Olson is with Carnegie Mellon University, Institute for Software Research, Pittsburgh, PA, 15213*

for use under this scenario and show that they perform the task well. Whereas this problem seems to have been considered with relatively specialized algorithms and/or additional assumptions (see section 2), our particular approach adapts traditional straightforward and well-known supervised learning algorithms with no extra assumptions on test set knowledge.

In the next section, we examine related work on this subject. We then follow it by precisely defining the aggregate output learning problem, and then present the details for how a number of standard supervised learning algorithms can be adapted to work for this problem. Finally, we present experiments on a number of datasets to show the efficacy of our approach.

Age	Income	Weight	Savings?
50	75000	220	
30	56000	180	1,900,000
50	60000	170	
48	40000	150	
22	45000	160	
25	50000	180	400,000
23	38000	165	
24	61000	190	

Age	Income	Weight	Savings?
40	48000	170	?
29	60000	180	?
57	18000	195	?

TABLE 1
Sample regression training and test sets. In the training set, output values are known only in aggregate.

2 PREVIOUS WORK

The aggregate output learning problem certainly bears some similarities with other well-known learning problems, but has significant differences as well. The unsupervised learning problem has no output values at all, whereas the supervised learning problem has a distinct output value for each individual input vector in the training set [2], [3]. Semi-supervised learning [4], [5] describes a scenario somewhat between the two, where some of the input vectors have associated output values (as in supervised learning), but others do not (as in unsupervised learning). This is rather distinct from our aggregate output learning problem where *all* input vectors are associated with an output value, but multiple input vectors map to the same output value which represents the sum of the actual (unknown) output values. Moreover, our problem is different from the others described above in that the granularity of the output values is different in the training set than in the test set. In the training set, output values are aggregated over multiple input vectors. In the test set, each input vector has its own output value.

Multiple-instance learning [6] is similar in a number of ways to the aggregate output learning problem. In multiple-instance learning, collections of input vectors also have only a single output value, but this output value is chosen to be positive only if at least one of the underlying unknown output values is positive. This contrasts with out scenario, where the output values are a sum or a count.

Other forms of aggregate learning seem to have been examined, though they differ from the form we discuss here. Arminger et. al. propose using log-linear models to disaggregate data with a similar model to ours, but their framework is based on the idea of filling in missing data in the training set. Their approach does not seem to generalize easily to a test set [7]. Quadrianto et. al. [8] present an approach that requires knowledge of the class distribution in the test set, an assumption that is not part of our framework. McGrogan et. al. consider an approach where the training set contains an individual output value for each input vector, but this output value is an aggregate over multiple measurements [9]. Yang et. al. look at learning with aggregates only in order to preserve privacy [10]. An approach by Chen et. al. examines learning from multiple aggregate tables each derived from an underlying common dataset [11]. This probabilistic approach is different from our scenario in this paper in that we do not assume that the input vectors within an individual aggregation collection share feature values. This last difference also contrasts our approach with a number of ideas from the statistical literature such as iterative proportional fitting [12].

Finally, we note that a portion of this paper has appeared previously as a conference publication [13].

We now move on to provide a formal framework for the problems that we consider.

3 PROBLEM FORMULATIONS

3.1 Regression

Table 1 shows a sample dataset for this framework. Suppose that we are given a training set that consists of input vectors $\{\mathbf{x}_{ci}\}$, where each $\mathbf{x}_{ci}$ has an *unknown* real-valued output value y_{ci}. We are given, however, a set of aggregate output values y_c, where we know that for a fixed c, $y_c = \sum_i y_{ci}$. (The subscript c indicates an aggregate collection, and the subscript i identifies a particular data point within that collection.) The goal is the same as for the traditional regression problem. We wish to learn a predictor f, using the training set only, that performs well on unseen test data drawn from the same source. In other words, for a test input vector $\mathbf{x}$ with output value y, we desire $f(\mathbf{x}) \approx y$. (The precise criterion varies with the learning algorithm.) Note that f operates on a single input vector and produces a single output value, though the training set contains output values aggregated over multiple input vectors. The test set contains unaggregated output values.

3.2 Classification

Though regression is perhaps the more natural approach for thinking about the aggregate output learning prob-

Age	Income	Weight	Beer over Wine?
50	75000	220	
30	56000	180	3 Yes
50	60000	170	1 No
19	2000	150	
32	60000	160	
35	90000	180	1 Yes
60	85000	165	3 No
53	92000	190	

Age	Income	Weight	Beer over Wine?
40	48000	170	?
29	60000	180	?
57	18000	195	?

TABLE 2
Sample classification training and test sets. In the training set, classifications are known only in aggregate.

lem, we present a classification version of it as well.

First, we point out that the traditional classification problem, in general, allows each input vector to belong to one of an arbitrary number of classes. We constrain ourselves in this paper to the binary classification problem, where each input vector belongs to one of two possible classes.

Table 2 shows a sample dataset for our framework. Suppose that we are given a training set of input vectors where each has an *unknown* output value (also known as the class label). This output value is a "yes" or a "no," depending on to which class its corresponding input vector belongs. The training set is divided into collections of input vectors where aggregate output values are known for each collection. More precisely, we suppose that our training set consists of input vectors $\mathbf{x}_{ci}$, where the subscript c indicates to which collection the input vector belongs, and the subscript i identifies a particular input vector within that collection. We further suppose that we are given a set of aggregate output values y_c and $\bar{y}_c$, where for an individual collection c, y_c is the number of "yes" values and $\bar{y}_c$ is the number of "no" values. The goal is the same as for the traditional classification problem. We wish to learn a predictor f, using the training set only, that performs well on unseen test data presumably drawn from the same source. In other words, for a set of test input vectors $\{\mathbf{x}_j\}$ with unknown output values $\{y_j\}$, we desire $f(\mathbf{x}_j) = y_j$ often. (The precise definition of "often" varies with the learning algorithm.) Note that f operates on a single input vector and produces a single output value, though the training set contains output values aggregated over multiple input vectors.

4 Algorithm Updates

We now describe in detail how four popular supervised learning techniques can be updated for the aggregate output learning problem in classification and regression scenarios. k-nearest neighbor is an exceedingly simple algorithm to use, and its adaptation is similarly straightforward. Support vector machines require the formulation of a quadratic programming optimization problem, and so we present new reformulations of these optimization problems to address our scenario. Neural networks traditionally use the backpropagation algorithm [1], [14] to find the local minimum for a quadratic loss optimization problem. We present modifications to the classic backpropagation algorithms to handle our new problem, and similarly show how radial basis networks can be adapted for the regression problem. Decision trees typically use a particular criterion to determine on which feature and values to split a node. We suggest new criteria adapted for building decision trees in the aggregate scenario.

It should be noted that we make no claims that these new approaches are optimal techniques for solving the aggregate output learning problem. We believe that one of the main contributions of this paper is the formulation of this new problem for the community as well as for our particular application, and therefore we present a first effort at solving this problem via modifications to traditional algorithms.

4.1 k-Nearest Neighbor: Classification

The k-nearest neighbor algorithm is a remarkably simple yet effective approach for classification [2], [3]. The algorithm requires the choice of a parameter k, representing the number of neighbors to be used, and a distance metric for quantifying the difference between input examples. In the traditional supervised learning approach, for a particular test input vector its k nearest neighbors in the training set are determined. As each of these nearest neighbors already has an output value, the test point is assigned the output value represented by a majority of the nearest neighbors.

Our adaptation of k-nearest neighbor, in both the classification and regression case, is exceedingly straightforward and perhaps obvious. We present it here as an instructive example to help reinforce our general approach before addressing the more complex algorithms. We also present it for experimental comparison purposes.

For the classification version of the aggregate output learning problem, for a particular test input vector one can find a set of k nearest neighbors in the training set in precisely the same way as one does for the traditional algorithm. The difference is that each nearest neighbor $\mathbf{x}_{ci}$ belongs to a set of training input vectors for which only an aggregate y_c and $\bar{y}_c$ is known. We resolve this by creating an artificial output value for each nearest neighbor $\mathbf{x}_{ci}$, which is defined as the proportion of "yes" classifications for the entire collection to which it belongs. In other words, if we define n_c to be the number of input vectors $\mathbf{x}_{ci}$ in aggregate collection c, we define for each input vector $\mathbf{x}_{ci}$ an artificial output value of $\tilde{y}_{ci} = \frac{y_c}{n_c}$. We then average $\tilde{y}_{ci}$ over all k nearest neighbors to produce our estimated y for our test point.

If this is greater than one-half, we classify the test point as a "yes"; otherwise, we classify it as a "no." Note that if one has knowledge that leads one to believe that the test set has a different ratio of "yes" to "no" than the training set does, one can change the threshold accordingly.

This approach is mathematically equivalent to thinking of "yes" as a 1, "no" as a 0, and applying the regression technique described below.

4.2 k-Nearest Neighbor: Regression

The k-nearest neighbor algorithm can also be used for traditional regression problems. Instead of using a majority rule amongst the neighbors, the output values from all nearest neighbors are averaged together.

Adapting the regression form of k-nearest neighbor for our aggregate output learning problem is quite similar to the adaptation we do for classification. Again, we point out that k-nearest neighbor is a particularly simplistic approach: we demonstrate this for instructional purposes before proceeding to more complex algorithms. For a particular test input vector one can find a set of k nearest neighbors in the training set in precisely the same way as one does for the traditional algorithm. The difference is that each nearest neighbor $\mathbf{x}_{ci}$ belongs to a set of training input vectors for which only an aggregate y_c is known. We resolve this by creating an artificial output value for each nearest neighbor $\mathbf{x}_{ci}$, which is defined as the average of the output value across the entire aggregate collection to which it belongs. In other words, if we define n_c to be the number of input vectors $\mathbf{x}_{ci}$ in aggregate collection c, we define an artificial output value $\tilde{y}_{ci} = \frac{y_c}{n_c}$. We then average $\tilde{y}_{ci}$ over all k nearest neighbors to produce our estimated y for our test point.

This approach is mathematically equivalent to preprocessing the training set by assigning to each point an output value equal to the average output value for its aggregate collection, and proceeding with the traditional k-nearest neighbor algorithm. A considerable deficiency with using k-nearest neighbor in this way is that the algorithm does not allow the learner flexibility in distributing the aggregate output value for a collection unevenly throughout the points in that collection. This issue is addressed well, however, by our other approaches.

4.3 Neural Network: Classification

We first describe the traditional neural network for classification [1], [14] to establish our notation. We will be changing some of the notation that we used in the k-nearest neighbor case as we define the neural networks, as our primary goal is to make each algorithm as simple and understandable as possible. We are given an input vector $\mathbf{x}$, where x_i represents the i-th component of the vector $\mathbf{x}$. Each input dimension connects to a series of hidden nodes with weights w_{ij}. An activation function $g(\cdot)$, typically a sigmoid in practice, is used to process the output of each node. The output of each hidden node is denoted as a_j. Each hidden node, in turn, connects to a series of output nodes with weights v_{jk}. The output from each output node o_k is similarly preprocessed with the same activation function $g(\cdot)$. A sigmoid typically has a constant threshold parameter. We adopt the usual strategy of representing that parameter via an artificial input dimension of constant value [1], [14]. More precisely, the output for a particular output node o_k is determined as $o_k = g(\sum_j v_{jk} a_j)$, and the output for a particular hidden node a_j is determined as $a_j = g(\sum_i w_{ij} x_i)$.

For the aggregate output learning problem, we wish to retain this traditional neural network. The test set will consist of individual points, so a network of this form makes sense. The training procedure must differ, however, since the training set contains outputs for aggregate collections instead of for individual points. We therefore derive an update to the traditional backpropagation algorithm [1], [14].

The traditional backpropagation algorithm requires, for a particular input vector $\mathbf{x}$, the comparison of an output o_k with the actual output y_k. Neural networks allow for multiple outputs, which we have not addressed in our framework or other algorithms in this paper. Since including multiple outputs is straightforward for neural networks, we leave this possibility in as we work though our derivations. We therefore define y_k to be the k-th desired output. The key difference we face from the traditional approach is that we only have an output y_k for an aggregate collection that contains a number of input vectors. For a particular aggregate collection, we will use the notation $x_{\ell i}$ to represent the ℓ-th input vector's i-th component. Similarly, we represent the output of each hidden node for each input vector as $a_{\ell j} = g(\sum_i w_{ij} x_{\ell i})$, and output k for an entire aggregate collection as $o_k = \sum_\ell g(\sum_j v_{jk} a_{\ell j})$. This is similar to the traditional approach, but the presence of the summation over the subscripts ℓ requires us to develop modifications to backpropagation to handle this new learning problem.

In order to determine the optimal weights, we start with the output layer. For a particular aggregate collection, the error is defined as:

$$E = \frac{1}{2}\sum_k (y_k - \sum_\ell g(\sum_j v_{jk} a_{\ell j}))^2 = \frac{1}{2}\sum_k e_k^2 \quad (1)$$

where we define e_k to be the difference between the expected output and the actual. Denoting fixed indices by capital letters, we find the gradient by taking the partial derivative with respect to a particular weight v_{JK} as:

$$\begin{aligned}\frac{\partial E}{\partial v_{JK}} &= -e_K \sum_\ell \frac{\partial}{\partial v_{JK}} g(\sum_j v_{jK} a_{\ell j}) \\ &= -e_K \sum_\ell g'(\sum_j v_{jK} a_{\ell j}) a_{\ell J}\end{aligned} \quad (2)$$

We therefore define the propagation update rule for v_{JK} as:

$$v_{JK} := v_{JK} + \alpha e_K \sum_\ell g'(\sum_j v_{jK} a_{\ell j}) a_{\ell J} \quad (3)$$

where α is a learning rate parameter [14]. Similarly, we

can derive an update rule for each weight w_{IJ} as:

$$\begin{aligned}\frac{\partial E}{\partial w_{IJ}} &= -\sum_k [e_k \sum_\ell \frac{\partial}{\partial w_{IJ}} g(\sum_j v_{jk} a_{\ell j})] \\ &= -\sum_k [e_k \sum_\ell g'(\sum_j v_{jk} a_{\ell j}) \frac{\partial}{\partial w_{IJ}} \sum_j v_{jk} a_{\ell j}] \\ &= -\sum_k [e_k \sum_\ell g'(\sum_j v_{jk} a_{\ell j}) \frac{\partial}{\partial w_{IJ}} \sum_j v_{jk} g(\sum_i w_{ij} x_{\ell i})] \\ &= -\sum_k [e_k \sum_\ell g'(\sum_j v_{jk} a_{\ell j}) \sum_j v_{jk} g'(\sum_i w_{ij} x_{\ell i}) \\ &\qquad \times \frac{\partial}{\partial w_{IJ}} \sum_i w_{ij} x_{\ell i}] \\ &= -\sum_k [e_k \sum_\ell g'(\sum_j v_{jk} a_{\ell j}) v_{Jk} g'(\sum_i w_{iJ} x_{\ell i}) x_{\ell I}]\end{aligned} \tag{4}$$

We therefore define the propagation update rule for w_{IJ} as

$$\begin{aligned}w_{IJ} := w_{IJ} + \alpha \sum_k [e_k \sum_\ell g'(\sum_j v_{jk} a_{\ell j}) \\ \times \quad v_{Jk} g'(\sum_i w_{iJ} x_{\ell i}) x_{\ell I}]\end{aligned} \tag{5}$$

With these updated rules, we proceed in a similar fashion to traditional neural network backpropagation. For each aggregate collection, we determine the output from the neural network for each point, aggregate, and compare with the expected output for that collection. The backpropagation update rules then indicate how to update the weights. We iterate over the dataset repeatedly until the error changes by less than a predefined threshold. The key difference between these update rules and the traditional ones is the appropriate usage of the summations over ℓ, which represent the multiple points in an aggregate collection.

In contrast with the k-nearest neighbor approach described earlier, this approach does not require the total aggregate output for each collection to be averaged evenly across that collection in some fashion. Instead, this approach only attempts to constrain the total output across a collection to approximate that in the training set.

4.4 Neural Network: Regression

In order to use neural networks for regression, we adopt the radial basis function network approach [15]. In this scenario, under the traditional approach we are given an input vector $\mathbf{x}$, where x_i represents the i-th component of the vector $\mathbf{x}$. Each input dimension connects to a series of hidden nodes with weights w_{ij}. The output of each hidden node (denoted as a_j) is the distance between the input vector $\mathbf{x}$ and the associated weights $\mathbf{w_{\cdot j}}$, post-processed by a radial basis function. A linear combination of the outputs from the hidden nodes (with weights v_j) yields the output from the network. We adopt the usual strategy of representing the threshold term in this linear combination via an additional hidden node that always outputs a value of 1 [1].

More precisely, the output o associated with a particular input vector $\mathbf{x}$ is determined as $o = \sum_j v_j a_j$, where the output for a particular hidden node a_j is determined as

$$a_j = e^{\frac{-\sum_i (x_i - w_{ij})^2}{2\sigma^2}} \tag{6}$$

where σ is a parameter. For the aggregate output learning problem, we wish to use this same network structure. As in the classification case, the test set will consist of individual points, so a network of this form makes sense. The training procedure must be reexamined, however, since the training set contains outputs for aggregate collections instead of for individual points.

The traditional approach for training RBF networks is to first choose the weights for the hidden nodes via a clustering algorithm. The output of each hidden node is effectively a measure of how close an input vector is to the point represented by the weights for that node. Therefore, choosing hidden nodes whose weights represent "prototypes" for points in the training set makes sense [15]. This means that this stage of the training process does not need to change at all for our aggregate output learning problem: the difference in our training set and a traditional one lies in the fact that the output values are aggregated, but not the input vectors. It is precisely because clustering is an *unsupervised* procedure that we can leverage it unchanged.

The second stage of training an RBF network, once the weights for the hidden nodes have been determined, is to find optimal values for the weights v_j. We seek to minimize the error over all aggregate collections. This total error can be represented as

$$E = \frac{1}{2} \sum_c (y_c - \sum_{\ell \in c} \sum_j v_j a_{\ell j})^2 \tag{7}$$

where y_c is the output value for aggregate collection c, $a_{\ell j}$ is the output from hidden node a_j associated with the ℓ-th input vector in aggregate collection c, and (with a slight abuse of notation), $\ell \in c$ represents the indices of the data associated with aggregate collection c. The summation over c is understood to run over all aggregate collections.

The weights from the first layer of the network remain fixed, so the terms y_c and $a_{\ell j}$ in the above error are fixed. Optimizing for the best values of v_j is therefore a straightforward unconstrained quadratic optimization problem.

As in the classification case, this new version of an RBF is quite similar to the traditional methodology, and the algorithms for using it are similar. But again, it should be pointed out that this new approach differs from the original in that it does not constrain the output from each individual input vector to match a predetermined output, but rather constrains sums of the outputs from collections of input vectors to match given training set values.

4.5 SVM: Classification

In developing an SVM approach for solving the classification version of the aggregate output learning problem, we observe that the problem is similar in some ways to the *semi-supervised learning problem* [4], [5]. The semi-supervised learning problem consists of both labeled and unlabeled training data, and the goal is to use the unlabeled data to improve classification accuracy over using just labeled data. In our problem, none of the data is labeled individually, but a count of the number of labels of each type is provided for each aggregate

set. These two problems are not the same, but work by Bennett and Demiriz on the semi-supervised problem [4] yields insights on how to appropriately adapt linear SVMs for use with unlabeled data. Our approach heavily leverages their ideas. Therefore, we only consider linear support vector machines in this work.

We again shift our notation slightly for clarity of exposition. The standard SVM for classification [16] is

$$\begin{array}{rl} \min_{(\mathbf{w},b,\boldsymbol{\xi}\geq 0)} & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_i \xi_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \xi_i \quad \geq \quad 1 \end{array} \tag{8}$$

where the subscript i ranges over all training rows, $\mathbf{x}_i$ represents a particular training input vector, and the vector $\mathbf{w}$ and scalar b represent the coefficients of the separating hyperplane. ξ_i is a measure of the error associated with the output from input vector $\mathbf{x}_i$, and C is a user chosen parameter that balances the tradeoff of accuracy against overfitting. y_i is a 1 or a -1, depending on to which class the training point belongs.

For the aggregate output learning problem, we do not know to which class each training point belongs. We do know how many points from each class that there are supposed to be in each aggregate collection, though. Therefore, similar to Bennett and Demiriz [4], we modify the above quadratic program to the following mixed integer program. Here, we use the indicator variable d_i to be a 1 if the point is in class 1 and 0 if the point is in class -1:

$$\begin{array}{rc} \min\limits_{(\mathbf{w},b,\boldsymbol{\xi}\geq 0,\mathbf{z}\geq 0,\boldsymbol{\eta}_c)} & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_i(\xi_i+z_i) + D\sum_c \eta_c \\ \text{s.t.} & \mathbf{w}\cdot\mathbf{x}_i - b + \xi_i + M(1-d_i) \geq 1 \\ & -(\mathbf{w}\cdot\mathbf{x}_i - b) + z_i + Md_i \geq 1 \\ & -\eta_c \leq y_c - \sum_{\ell\in c} d_\ell \leq \eta_c \end{array} \tag{9}$$

The constant M is chosen to be sufficiently large so that if $d_i = 0$, then $\xi_i = 0$ satisfies the first constraint. Similarly, if $d_i = 1$, then $z_i = 0$ satisfies the second constraint. ξ_i and z_i represent the misclassification errors for each point, measured as a traditional SVM would. In this case, however, since we do not know in advance to which class each point belongs, the error is effectively taken to be the minimum error for either of the two classes. The subscript c is used to represent an individual aggregate collection; $\ell \in c$ represents the indices of all input vectors associated with collection c, and y_c represents the actual number of points associated with class 1 for collection c. The term η_c works to ensure that the number of points assigned to each class is consistent with the aggregates provided for the training set. η_c is the difference between the predicted and the actual count of the number of points in class 1 for an aggregate collection c. We therefore sum this error over all points and add it to the objective function, multiplying it by a parameter D to balance the importance of matching the desired aggregate accuracy level for each collection.

The solution to this optimization problem can be found via any mixed integer quadratic programming solver. This approach is somewhat slow, however, and we acknowledge that a faster algorithm can likely be constructed. For example, the semi-supervised work by Bennett and Demiriz was sped up in two fashions. First, they used the popular substitution of $||w||_1$ instead of $\frac{1}{2}||w||_2^2$ in the objective function. This transforms this mixed integer quadratic program into a mixed integer linear program. Furthermore, Fung and Mangasarian [17] reformulated this problem as a concave minimization problem and used a successive linear approximation algorithm. Such an approach might work well here, but is outside the scope of this particular paper. We have therefore chosen to present a formulation as similar as possible to traditional SVMs. Nonetheless, leveraging approximation approaches similar to the ones described above would be worth examining in future work.

4.6 SVM: Regression

Developing a version of support vector regression (SVR) for the aggregate output learning problem is considerably simpler than for classification. The standard linear SVR approach [16] is expressed as:

$$\begin{array}{rl} \min_{(\mathbf{w},b,\boldsymbol{\xi}\geq 0,\mathbf{z}\geq 0)} & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_i(\xi_i+z_i) \\ \text{s.t.} & \mathbf{w}\cdot\mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i \\ & y_i - \mathbf{w}\cdot\mathbf{x}_i - b \leq \varepsilon + z_i \end{array} \tag{10}$$

where the subscript i represents a particular training set row, $\mathbf{x_i}$ represents a particular training input vector, and the vector $\mathbf{w}$ and scalar b represent the coefficients of the regression surface. y_i represents the desired output value for each input vector. ξ_i and z_i serve to measure how far the predicted output value is from the actual; the optimization problem ensures that for each i, either ξ_i or z_i is zero depending on whether the predicted value is too small or too large. C is a user chosen parameter that balances the tradeoff of accuracy against overfitting, and ε is a user chosen parameter representing the size of the “zone of insensitivity” within which errors do not contribute.

The aggregate output version of SVR does not have an individual y_i for each input vector $\mathbf{x}_i$. Instead, each aggregate collection contains an individual aggregate output value y_c. Therefore, we can modify the SVR formulation to constrain (with slack) the outputs from all points within an aggregate collection to sum to the appropriate total:

$$\begin{array}{rl} \min_{(\mathbf{w},b,\boldsymbol{\xi}\geq 0,\mathbf{z}\geq 0)} & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_c(\xi_c+z_c) \\ \text{s.t.} & \sum_{\ell\in c}(\mathbf{w}\cdot\mathbf{x}_\ell + b) - y_c \leq \varepsilon + \xi_c \\ & y_c - \sum_{\ell\in c}(\mathbf{w}\cdot\mathbf{x}_\ell + b) \leq \varepsilon + z_c \end{array} \tag{11}$$

The subscript c is used to represent an individual aggregate collection; $\ell \in c$ represents the indices of all input vectors associated with collection c.

As in the classification case, this quadratic program can be solved by any off-the-shelf quadratic programming solver. Considerably faster algorithms for SVR are well known [18], [19], and so one or all of them might be adaptable to work with our formulation here. Similarly,

our approach could likely be adapted to work with nonlinear SVMs.

4.7 Decision Trees: Classification

For traditional classification decision trees, we build a tree that takes an input vector $\mathbf{x}$ and maps it to a particular leaf. Each leaf in the tree is assigned a "yes" or "no" value, which is determined to be the classification value for that input vector. To build the tree, one starts with a root node, and splits it according to some splitting criterion. This divides the data into subsets, which are used to build the lower portions of the tree.

For the aggregated classification algorithm, our goal is the same: we want to create a tree that for each input vector outputs a "yes" or "no" value which is determined by the leaf to which the input vector maps. The challenge is in developing the splitting criterion. Traditional approaches, such as entropy or gini index [20], require knowledge for a particular split of how many "yes" and "no" points in the training set are in each subtree for that split. We know precisely which training points split to which subtree, but we do not know the classification values assigned to each. We know in total how many "yes" and "no" values there are for each aggregate collection; for a particular decision tree split, however, we do not know how individual "yes" and "no" values split. This does not fit well with traditional local splitting criteria that are based on a count of the number of "yes" and "no" values on each side of the split.

We have therefore chosen the following alternative approach. Typically, decision tree algorithms attempt a split, then measure the quality of that split *locally*, i.e. based on only those training points that have traversed the tree to that node. Instead, we measure the quality of a split *globally*: for a potential split in the decision tree, we try all points in the training set on that new decision tree. Every point in the training set thus receives a classification, most of which are likely determined through other portions of the tree than the one under consideration for splitting. Once a classification is determined for every point in the training set, we can aggregate these together for each aggregate collection and calculate an error based on the known aggregate totals. We then repeat for every potential split at a particular node, and choose the split for that node that minimizes total error over the entire dataset.

More specifically, we simplify matters by only allowing the tree to split in a binary fashion: every non-leaf has precisely two children. Given a node under consideration that we wish to split, we try all possible splits over the potential features and values (just as in traditional decision tree algorithms). For each potential split, we produce two children. We then try all four possibilities for labeling these two children with "yes" and "no" values (yes/yes, yes/no, no/yes, no/no). For each possibility, for each aggregate collection, we calculate the error as the squared difference between the number of known "yes" values for that collection and the number of "yes" values for that collection determined by the decision tree. Specifically, if we define y_c, as the known total of "yes" values in collection c, and $\tilde{y}_{ci}$ to be the decision tree predicted classification of data vector $\mathbf{x_{ci}}$ (where a "yes" is a 1 and a "no" is a 0), we wish to minimize the splitting criterion:

$$\sum_c [y_c - \sum_{i \in c} \tilde{y}_{ci}]^2 \tag{12}$$

After all possible binary splits have been tested for that node (across all features, values, and "yes" / "no" combinations), the feature and value combination that produces the smallest possible error is chosen.

The order in which the nodes are chosen to be split can greatly affect how the tree is built. The root node is always split first, but then it has two children. Which of these should be split first? We choose to split next whichever leaf has the most input vectors that map to it. The rationale here is that we are focusing our attention on the portion of the tree that contains the most data, as opposed to tweaking a portion with only a trivial amount of data. Clearly, there are many potential approaches here which might be examined in the future.

Finally, we discuss termination of the algorithm. Traditional decision trees offer many approaches to limit overfitting via pruning and other means [1]. Many of these ideas could be applied to our algorithm as well. Since our purpose here is to illustrate how one can apply decision trees to the aggregate output learning problem, and not to undergo a detailed analysis of various decision tree pruning algorithms, we simply limit the depth of the tree to a fixed depth.

4.8 Decision Trees: Regression

There are a variety of approaches for adapting decision trees to handle regression [21], [20]. Most of them work very similarly to classification decision trees, as described above. At each node, a split is determined based on some sort of local criterion: in the regression scenario, this might be something like mean square error on the output values for the input vectors that traverse the tree to that node. In its simplest case, each leaf is described by a single numeric value. That is the approach we adopt here. More intricate regression tree approaches perform a linear regression at each leaf in order to obtain more accurate values. Although this would be interesting to pursue, our goal here is to demonstrate that the problem can be solved reasonably without considering all possible enhancements.

Our approach to this problem is a variation on the classification algorithm described above. In precisely the same manner, we build the tree by starting with the root node, splitting, finding leaf regression values, and repeating with whichever leaf has the most input vectors in it. The one key aspect from classification that fails for regression is the repeated attempts at trying all four

yes/no combinations for each pair of leaves. In the regression case, where each leaf receives a numeric value, the number of possibilities is infinite. Therefore, we choose an alternate approach where we assign regression values to all leaves at once rather than determining them individually. We can formulate this as an optimization problem, where the regression values are determined by minimizing a sum of squares.

Specifically, for a given tree if we define k_{cl} as the number of points in aggregate collection c that are directed to leaf l, r_l as the regression value for leaf l, and y_c as the known aggregate regression value of aggregate collection c, then we wish to find regression values r_l that optimize:

$$\min_{(\mathbf{r})} \quad \sum_c [\ (\sum_l k_{cl} \ r_l) - y_c]^2 \tag{13}$$

This minimization problem chooses leaf regression values that minimize aggregate error over the entire training set. It should be noted that this global approach means that every time a new split is added to the tree, regression values in all leaves are updated.

One problem we observed with the above is that for practical datasets, the above minimization problem tends to overfit. We are minimizing over leaf regression values. For a binary tree of depth 5, there are 32 such variables. For a dataset of 500 data points, where each aggregate collection contains 10 data points, there are only 50 aggregate collections. Minimizing 32 variables with only 50 collections can easily lead to overfit solutions. Therefore, we introduced regularization in a similar fashion to ridge regression [22]. This resulted in the following minimization problem:

$$\min_{(\mathbf{r})} \quad \sum_c [(\sum_l k_{cl}(r_0 + r_l)) - y_c]^2 + C \sum_l (r_l^2) \tag{14}$$

In this regularized algorithm, we use r_0 to set a baseline regression value for all leaves. (This is similar to the intercept term in traditional linear regression, which is typically not included in the minimization term when regularization is undertaken.) The other leaf regression values become *offsets* from r_0. With our introduction of $\sum_l (r_l^2)$, the magnitude of the r_l values can be reduced using C as a balancing factor. This regularization term serves to reduce overfitting when assigning regression values. Equation (14) is a quadratic minimization problem with no constraints, and thus can be solved analytically.

We acknowledge that there are many approaches to regularization; for example, we could have used a lasso regression sort of approach instead [22]. Such experimentation is outside the scope of this paper, but is important future work.

5 Experimental Results

5.1 Overview

The main contributions of this paper are in proposing our new machine learning problem, and in posing some initial attempts at solving it. All four of the algorithms proposed here are natural generalizations of well-known algorithms. In some sense, it is somewhat unclear as to what purpose experiments would serve. Since we pose a new learning problem, there are no other algorithms in the literature which are appropriate for comparison purposes. We can compare these algorithms with each other, but it is well known that different algorithms perform better on different datasets. However, we do see for both regression and classification a simpler "experimental control" technique which might be appropriate for comparison purposes, which we describe later.

As we are currently unable to release our SPMS data, we present in this paper regression data from three well-known publicly available datasets. The first, *auto-mpg* [23], predicts miles per gallon based on seven variables including cylinders, origin, model year, and acceleration. Although the dataset included a car name variable, we ignored it because it was not numeric. The second dataset, *housing* [23], is the classic "Boston Housing Data." This dataset is used to predict the median value of owner-occupied homes in Boston based on thirteen variables including the crime rate, the non-retail business acreage, the average number of rooms, and so on. The final dataset, *cpu-small* [24], measures the portion of time (%) that cpus run in user mode based on fourteen variables. These include number of reads between system memory and user memory, number of writes between system memory and user memory, number of system read calls per second, and so on. There were 398 instances in the auto-mpg dataset, 506 in the housing dataset, and 300 in the cpu-small dataset (we chose a small subset of size 300 from the original, which had 8192 examples).

For the classification experiments, we similarly use three datasets. The first, *breast-cancer-wisconsin* [23], contains data used to determine whether a tumor is malignant or benign. This set contains 11 features, the last of which is a binary classification of the tumor. We ignore the first feature, a subject identification number, since it is not relevant to our analysis. The other features consist of integer values between one and ten that approximate continuous diagnostic measurements. *Dermatology* [23] is our second dataset. This contains 33 features for determining the type of Eryhemato-Squamous Disease in patients. All values are between zero and three except for one which describes the age of the patient. We similarly normalized the patient age feature to a value between zero and three. There are six different disease classifications in the original data, but our techniques only support binary classification; therefore, the data was altered so that the classification is either Psoriasis (the most common of the disease classifications) or not Psoriasis. The final dataset, *ionosphere* [23], classifies radar returns from an antenna installation as either good or bad, with good indicating a structure in the ionosphere and bad showing no such structure. There are 34 features corresponding to values returned by analysis of the radar signals. Values for each are continuous and between -1 and 1, and each

point has a binary classification. There are 699 points in *breast-cancer-wisconsin*, 358 in *dermatology*, and 351 in *ionosphere*.

None of the above datasets have aggregate outputs. They are traditional datasets in that they contain an output value for each input vector. Therefore, we use them for experiments by creating aggregate training sets. After separating training data from test data under a cross-validation framework, we group together multiple input vectors in the training set and aggregate their output values together. In the regression case, we simply add the regression values; in the classification case, we count the numbers of positive and negative examples. This transforms the training set into one appropriate for the aggregate output learning problem, and leaves us with a traditional test set for measuring success. This technique for creating artificial aggregate datasets gives us the capability to run multiple experiments, each with different characteristics. Specifically, we vary the dataset in two different ways, each of which could potentially influence the performance of an aggregate output learning algorithm.

First, we vary the *size* of the aggregate sets, i.e., the number of rows in the original dataset whose outputs are summed to form each aggregate set. For the traditional supervised learning problem, all aggregate sets are of size 1. Note that the set size is essentially an upper bound on how much information is lost due to aggregation. For simplicity, all aggregate sets that we generate for a particular dataset have the same size (except for possibly the last one).

Second, we vary the amount of *randomness* in the aggregation. In order for us to be able to learn anything from aggregate data, we have been making the assumption that points within an aggregate set are somewhat related. If this assumption were invalid, the problem would seem unsolvable; we would end up with aggregate sets where each collection of input vectors varied over the range of the dataset, and each aggregate output value would therefore be approximately the same. Said differently, we assume that each aggregate set is different from the others in a way that provides structure to help us learn from the data. We therefore vary the level of disorder in the data for experimental purposes in order to measure this effect. To do so, the original data is first sorted by output value. Individual data points in the training set (input vectors and output values together) are randomly chosen in pairs and swapped. After a number of random pairs have been swapped, the points are taken in order starting from the top of the dataset to create the aggregate groups. Large numbers of swaps therefore correspond to relatively random aggregate groups, whereas low numbers of swaps correspond to highly ordered aggregate groups. The "randomness" value seen in our experimental results refers to the number of pairs that we randomly swapped before aggregating the data.

As stated above, there are no algorithms that we know of that make sense to compare our new algorithms with, since the aggregate output learning problem is new. We can, however, compare to the following simple technique. Assuming that we start with an aggregate training set (which is, of course, the problem which we are trying to solve), we create a new training set where the input vectors are the same, but each input vector is assigned its own individual output value. For the regression problem, this value is the average of the known aggregate output value for the collection; for the classification problem, this value is the ratio of positive classifications over the total number of points in the collection. In either case, this new dataset thus resembles a traditional supervised learning regression dataset, and thus traditional regression algorithms can be used on it. Of course, using traditional algorithms overconstrains the problem, as any such algorithm will try to find a predictor that matches each input vector in the training set individually to the average output value for its collection. Nonetheless, this technique requires no new algorithms, and so it seems as though it is a worthy experimental control. It should be noted that in testing our regression approaches, the control is a regression algorithm; in testing our classification approaches, the control is *also* a regression algorithm. Note that the control approach for our decision tree experiments does not use "standard" approaches, because our approach and philosophy differs considerably from the traditional one. We discuss this in more detail in section 5.5. Finally, we note that an alternative control approach might seem to be to aggregate the input vectors within each aggregate set together in order to match the aggregated output values. This would work for training purposes, but not for testing, where the goal is to do prediction of output values for individual input vectors.

For all experimental results, five-fold cross-validation was used. (We were running enough experiments that the savings in time over ten-fold cross-validation was convenient.) All fields for the input values in all datasets were normalized by subtracting the mean and dividing by the standard deviation. Regression accuracy was measured via mean squared error, whereas classification accuracy was measured by the fraction of correct test classifications ("test set accuracy").

In looking at our experimental results, it is tempting to compare test set accuracies across algorithms, e.g. to compare the results from neural networks with k-nearest neighbor. We emphasize that such comparisons are not valid. Our purpose in running these experiments is to show how our technique varies with different aggregate set sizes, and with varying amounts of randomness among the collections. Therefore, we pick a simple set of parameters for each algorithm, and generally leave them fixed throughout the experiments (we discuss these in more detail below). All of these algorithms have considerable capability for being tweaked to improve the results. One could try a variety of values for k for the k-nearest neighbor algorithm, for example, or one could

size	randomness 0	25	50	100	200	500	2000
auto-mpg: aggregate algorithm							
2	0.15	0.22	0.23	0.32	0.36	0.46	0.40
5	0.15	0.24	0.31	0.47	0.63	0.70	0.75
10	0.15	0.26	0.33	0.52	0.72	0.85	0.90
20	0.15	0.27	0.35	0.57	0.83	0.92	0.95
housing: aggregate algorithm							
2	0.25	0.28	0.32	0.41	0.47	0.50	0.51
5	0.25	0.30	0.38	0.53	0.70	0.80	0.79
10	0.25	0.32	0.41	0.58	0.77	0.89	0.88
20	0.25	0.33	0.43	0.60	0.81	0.93	0.94
cpu-small: aggregate algorithm							
2	0.25	0.31	0.37	0.40	0.52	0.50	0.52
5	0.26	0.34	0.43	0.60	0.76	0.76	0.76
10	0.29	0.38	0.48	0.73	0.88	0.88	0.87
20	0.42	0.52	0.63	0.82	0.92	0.95	0.95

Fig. 1. k-nearest neighbor, regression mean square error.

size	randomness 0	25	50	100	200	500	2000
breast-cancer-wisconsin: aggregate algorithm							
2	0.97	0.97	0.96	0.96	0.94	0.92	0.90
5	0.97	0.97	0.95	0.94	0.88	0.80	0.76
10	0.97	0.97	0.95	0.94	0.85	0.67	0.67
20	0.97	0.96	0.95	0.93	0.84	0.66	0.66
dermatology: aggregate algorithm							
2	1.00	1.00	0.99	1.00	0.96	0.94	0.96
5	1.00	1.00	0.99	0.92	0.81	0.73	0.76
10	1.00	0.98	0.96	0.91	0.73	0.70	0.70
20	1.00	0.99	0.97	0.82	0.71	0.69	0.69
ionosphere: aggregate algorithm							
2	0.84	0.84	0.83	0.84	0.82	0.81	0.84
5	0.85	0.83	0.82	0.76	0.70	0.68	0.71
10	0.84	0.80	0.79	0.74	0.69	0.66	0.66
20	0.84	0.81	0.81	0.72	0.66	0.64	0.64

Fig. 2. k-nearest neighbor, classification accuracy.

vary the number of hidden nodes in the neural network. In order for comparisons across algorithms to be valid, we would have had to make an attempt to optimize parameters across all algorithms to get the best possible results. We have not done so. Again, our purpose is to be able to look at each algorithm and observe its behavior on variations of the data, and to make comparisons with a control version of the algorithm with a similar set of parameters. Comparisons within the support vector machine results, for example, are completely valid and worthwhile. Comparing the SVM results with the neural network results does not make sense because we have not optimized the parameters for either appropriately.

5.2 k-Nearest Neighbor

First, we present the results from k-nearest neighbor, where we fixed $k = 5$ for all experiments. Note that for k-nearest neighbor, the control method is mathematically equivalent to our algorithm (see the ends of sections 4.1 and 4.2). We thus only provide one set of experiments for k-nearest neighbor (regression and classification), whereas for the other algorithms, we show two.

Figure 1 shows the results from running k-nearest neighbor for regression on our three datasets. We see that regression error increases as aggregate set size increases, which makes sense. As the aggregate set size increases, we are throwing more information out of the training set. Similarly, we see regression error increase as the amount of randomness in the aggregate sets increases. This makes sense as well. For highly ordered aggregate sets, within each aggregate set the (unknown) output values are quite similar to each other. Replacing each with the average for the aggregate set is a good approximation in this case. On the other hand, for highly random aggregate sets, assigning each point an output value which is the average of its aggregate set makes considerably less sense.

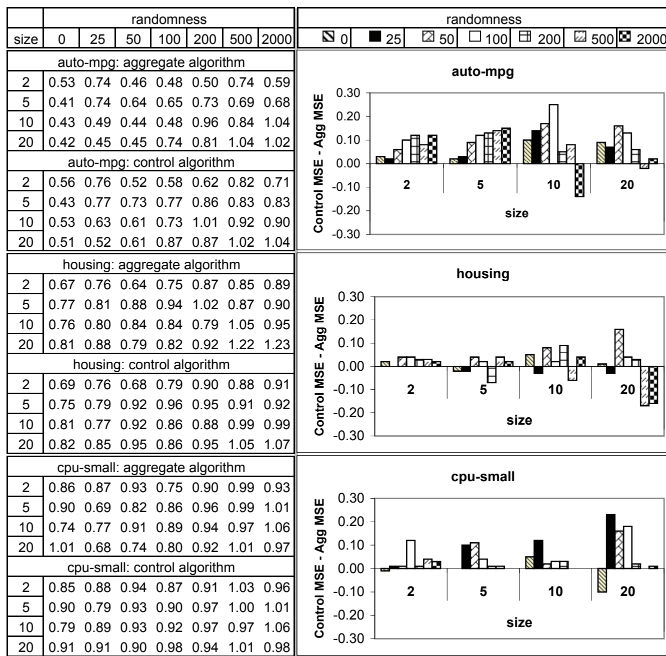
Figure 2 shows the results for k-nearest neighbor on the three classification datasets. As with the regression results, these represent the sort of effect that we intuitively expect data aggregation to have. The best results occur at little randomness. Because training data was at first aggregated in order of classification, results are almost identical at low randomness regardless of aggregate set size (if a collection contains only positive classifications, for instance, then our algorithm assigns a positive classification to each point, and no data is lost). As randomness increases, different aggregate collection sizes react differently. The accuracy of smaller group sizes decreases slowly. For larger group sizes, on the other hand, accuracy quickly approaches a base level corresponding to the frequency of the more common of the two binary classifications for each dataset, as the algorithm assigns the more common classification to every point.

As discussed earlier, these k-nearest neighbor results are a worthwhile benchmark for understanding our experimental techniques, but it is the results for neural networks, SVMs, and decision trees that illustrate the power of our approach.

5.3 Neural Networks

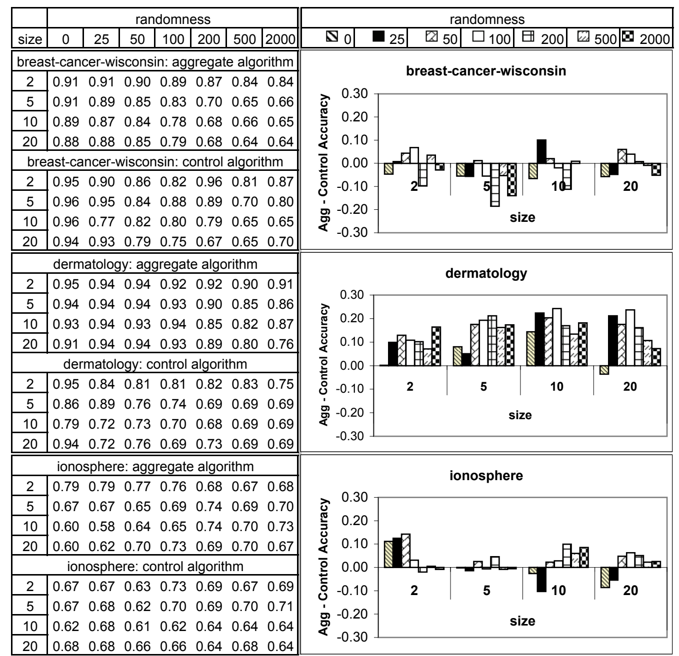
For the neural network regression experiments, we used a learning rate of 1×10^{-5} and a convergence tolerance of 0.001. The hidden nodes were determined via k-means clustering on the input vectors, and for each cluster σ was determined to be the average distance from each point in that cluster to the center. We arbitrarily fixed the number of hidden nodes at 12.

By comparing the differences between the aggregate algorithm and the control algorithm for a given dataset and technique, the patterns are quite clear. For the neural network regression results shown in Figure 3, we see that our aggregate algorithm generally outperforms the control algorithm. (In the bar chart, a bar going up indicates that the aggregate algorithm performs better than control. We adopt this convention throughout our experimental results.) The differences are most pronounced for moderate randomness values. For highly ordered data, our aggregate algorithm performs similarly to (and occasionally worse than) the control algorithm. As in the k-nearest neighbor experiments, this makes sense; when



size	randomness 0	25	50	100	200	500	2000
auto-mpg: aggregate algorithm							
2	0.53	0.74	0.46	0.48	0.50	0.74	0.59
5	0.41	0.74	0.64	0.65	0.73	0.69	0.68
10	0.43	0.49	0.44	0.48	0.96	0.84	1.04
20	0.42	0.45	0.45	0.74	0.81	1.04	1.02
auto-mpg: control algorithm							
2	0.56	0.76	0.52	0.58	0.62	0.82	0.71
5	0.43	0.77	0.73	0.77	0.86	0.83	0.83
10	0.53	0.63	0.61	0.73	1.01	0.92	0.90
20	0.51	0.52	0.61	0.87	0.87	1.02	1.04
housing: aggregate algorithm							
2	0.67	0.76	0.64	0.75	0.87	0.85	0.89
5	0.77	0.81	0.88	0.94	1.02	0.87	0.90
10	0.76	0.80	0.84	0.84	0.79	1.05	0.95
20	0.81	0.88	0.79	0.82	0.92	1.22	1.23
housing: control algorithm							
2	0.69	0.76	0.68	0.79	0.90	0.88	0.91
5	0.75	0.79	0.92	0.96	0.95	0.91	0.92
10	0.81	0.77	0.92	0.86	0.88	0.99	0.99
20	0.82	0.85	0.95	0.86	0.95	1.05	1.07
cpu-small: aggregate algorithm							
2	0.86	0.87	0.93	0.75	0.90	0.99	0.93
5	0.90	0.69	0.82	0.86	0.96	0.99	1.01
10	0.74	0.77	0.91	0.89	0.94	0.97	1.06
20	1.01	0.68	0.74	0.80	0.92	1.01	0.97
cpu-small: control algorithm							
2	0.85	0.88	0.94	0.87	0.91	1.03	0.96
5	0.90	0.79	0.93	0.90	0.97	1.00	1.01
10	0.79	0.89	0.93	0.92	0.97	0.97	1.06
20	0.91	0.91	0.90	0.98	0.94	1.01	0.98

Fig. 3. Neural networks, regression mean square error.



size	randomness 0	25	50	100	200	500	2000
breast-cancer-wisconsin: aggregate algorithm							
2	0.91	0.91	0.90	0.89	0.87	0.84	0.84
5	0.91	0.89	0.85	0.83	0.70	0.65	0.66
10	0.89	0.87	0.84	0.78	0.68	0.66	0.65
20	0.88	0.88	0.85	0.79	0.68	0.64	0.64
breast-cancer-wisconsin: control algorithm							
2	0.95	0.90	0.86	0.82	0.96	0.81	0.87
5	0.96	0.95	0.84	0.88	0.89	0.70	0.80
10	0.96	0.77	0.82	0.80	0.79	0.65	0.65
20	0.94	0.93	0.79	0.75	0.67	0.65	0.70
dermatology: aggregate algorithm							
2	0.95	0.94	0.94	0.92	0.92	0.90	0.91
5	0.94	0.94	0.94	0.93	0.90	0.85	0.86
10	0.93	0.94	0.93	0.94	0.85	0.82	0.87
20	0.91	0.94	0.94	0.93	0.89	0.80	0.76
dermatology: control algorithm							
2	0.95	0.84	0.81	0.81	0.82	0.83	0.75
5	0.86	0.89	0.76	0.74	0.69	0.69	0.69
10	0.79	0.72	0.73	0.70	0.68	0.69	0.69
20	0.94	0.72	0.76	0.69	0.73	0.69	0.69
ionosphere: aggregate algorithm							
2	0.79	0.79	0.77	0.76	0.68	0.67	0.68
5	0.67	0.67	0.65	0.69	0.74	0.69	0.70
10	0.60	0.58	0.64	0.65	0.74	0.70	0.73
20	0.60	0.62	0.70	0.73	0.69	0.70	0.67
ionosphere: control algorithm							
2	0.67	0.67	0.63	0.73	0.69	0.67	0.69
5	0.67	0.68	0.62	0.70	0.69	0.70	0.71
10	0.62	0.68	0.61	0.62	0.64	0.64	0.64
20	0.68	0.68	0.66	0.66	0.64	0.68	0.64

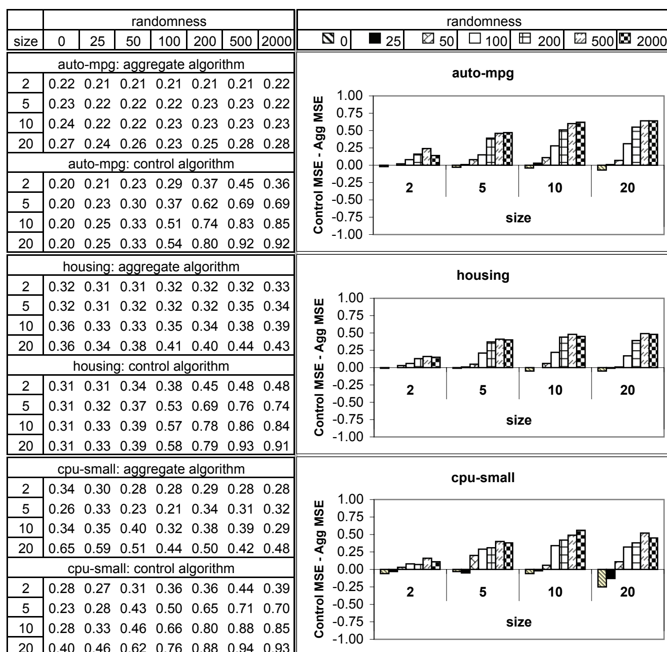
Fig. 4. Neural networks, classification accuracy.

the data is highly ordered, assuming that the output value for each point is the average of the output values for the aggregate set is a good approximation. For exceedingly random data (randomness value of 2000), our algorithm again performs similarly to, and occasionally worse than, the control algorithm. This is likely because with highly random aggregations, there is a considerable loss of information. In this case, the aggregate algorithm does not have enough data to draw better conclusions than the control algorithms. In general, though, the aggregate algorithm is a stronger approach.

For our classification neural networks, we used the same 1×10^{-5} learning rate and 0.001 convergence tolerance that we used for regression. We observed that it is quite possible to achieve greater accuracies on each dataset and even at particular aggregate collection size and randomness pairs by fine tuning the learning rate. This is a common issue with neural networks, especially in classification where there are two layers of weights to find simultaneously; local minimums are a challenge. There are a variety of approaches for dealing with this problem, but optimizing neural networks carefully was outside the scope of this paper. We therefore fixed the learning rate, and left it alone. Similarly, as above, we fixed the number of hidden nodes at 12. From the neural network classification results shown in Figure 4, it is clear that the aggregate and control algorithms with fixed learning rate and convergence value perform very differently on different datasets. Changes in aggregate collection size and randomness affect the aggregate and the control neural networks similarly. The aggregate algorithm outperforms the control most clearly for *dermatology*, where it is superior in almost every case. The aggregate algorithm also performs better on the ionosphere data than the control, although the trend is not nearly as clear as with the dermatology data. It appears that the aggregate does do consistently better with large collection size and high randomness than the control. The results for the breast cancer data are not as conclusive. There is no clear pattern in the results, and the aggregate is generally equal to or inferior to the control. Much of this may be attributed to the fact that the learning rate was not well adjusted to maximizing the accuracy of the aggregate network. We found that the quality of the results changed dramatically based on the setting of the learning rate parameter. This one set of experiments is clearly the weakest in this paper; all the other approaches we present show considerably better improvement over the control approaches.

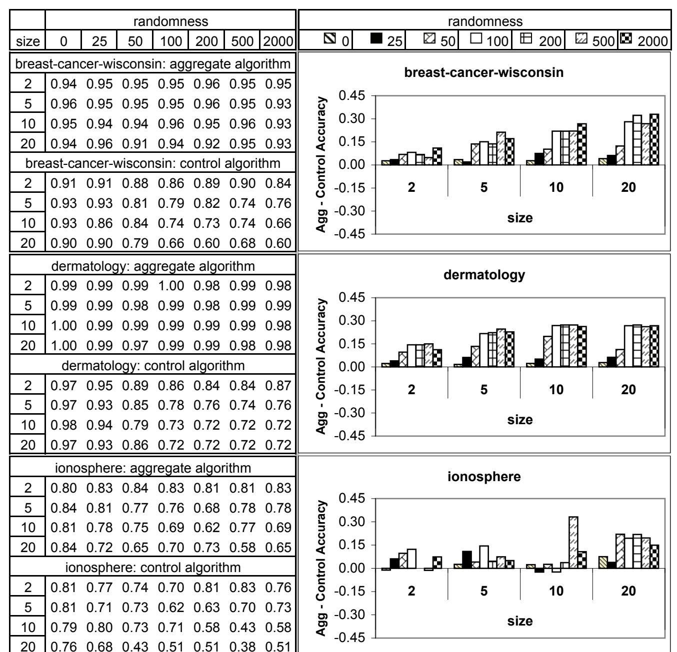
5.4 Support Vector Machines

For SVMs, we used the quadratic programming solver CPLEX [25] to handle the optimization. We varied the parameters C and D in order to achieve the right balance of objectives. In principle, this should have been done on a tuning set, pulled out of the training set, for each individual experiment. This opened up a complicated discussion as to how this should be done: in our scenario, the tuning set does not structurally mirror the test set. There are many approaches we might have tried. Conveniently, by varying parameters by orders of magnitude of 10, it was exceedingly clear for each dataset-algorithm pair that one particular set of parameters that optimized nearly all experimental values. Since we looked at the results for a large number of experiments simultaneously and picked a single set of parameters for all of them, it was clear that we were not somehow picking parameters to optimize a particular test set.



size	0	25	50	100	200	500	2000
auto-mpg: aggregate algorithm							
2	0.22	0.21	0.21	0.21	0.21	0.21	0.22
5	0.23	0.22	0.22	0.22	0.23	0.23	0.22
10	0.24	0.22	0.22	0.23	0.23	0.23	0.23
20	0.27	0.24	0.26	0.23	0.25	0.28	0.28
auto-mpg: control algorithm							
2	0.20	0.21	0.23	0.29	0.37	0.45	0.36
5	0.20	0.23	0.30	0.37	0.62	0.69	0.69
10	0.20	0.25	0.33	0.51	0.74	0.83	0.85
20	0.20	0.25	0.33	0.54	0.80	0.92	0.92
housing: aggregate algorithm							
2	0.32	0.31	0.31	0.32	0.32	0.32	0.33
5	0.32	0.31	0.32	0.32	0.32	0.35	0.34
10	0.36	0.33	0.33	0.35	0.34	0.38	0.39
20	0.36	0.34	0.38	0.41	0.40	0.44	0.43
housing: control algorithm							
2	0.31	0.31	0.34	0.38	0.45	0.48	0.48
5	0.31	0.32	0.37	0.53	0.69	0.76	0.74
10	0.31	0.33	0.39	0.57	0.78	0.86	0.84
20	0.31	0.33	0.39	0.58	0.79	0.93	0.91
cpu-small: aggregate algorithm							
2	0.34	0.30	0.28	0.28	0.29	0.28	0.28
5	0.26	0.33	0.23	0.21	0.34	0.31	0.32
10	0.34	0.35	0.40	0.32	0.38	0.39	0.29
20	0.65	0.59	0.51	0.44	0.50	0.42	0.48
cpu-small: control algorithm							
2	0.28	0.27	0.31	0.36	0.36	0.44	0.39
5	0.23	0.28	0.43	0.50	0.65	0.71	0.70
10	0.28	0.33	0.46	0.66	0.80	0.88	0.85
20	0.40	0.46	0.62	0.76	0.88	0.94	0.93

Fig. 5. Support vector machines, regression mean square error.



size	0	25	50	100	200	500	2000
breast-cancer-wisconsin: aggregate algorithm							
2	0.94	0.95	0.95	0.95	0.96	0.95	0.95
5	0.96	0.95	0.95	0.95	0.96	0.95	0.93
10	0.95	0.94	0.94	0.96	0.95	0.96	0.93
20	0.94	0.96	0.91	0.94	0.92	0.95	0.93
breast-cancer-wisconsin: control algorithm							
2	0.91	0.91	0.88	0.86	0.89	0.90	0.84
5	0.93	0.93	0.81	0.79	0.82	0.74	0.76
10	0.93	0.86	0.84	0.74	0.73	0.74	0.66
20	0.90	0.90	0.79	0.66	0.60	0.68	0.60
dermatology: aggregate algorithm							
2	0.99	0.99	0.99	1.00	0.98	0.99	0.98
5	0.99	0.99	0.98	0.99	0.98	0.99	0.99
10	1.00	0.99	0.99	0.99	0.99	0.99	0.98
20	1.00	0.99	0.97	0.99	0.99	0.98	0.98
dermatology: control algorithm							
2	0.97	0.95	0.89	0.86	0.84	0.84	0.87
5	0.97	0.93	0.85	0.78	0.76	0.74	0.76
10	0.98	0.94	0.79	0.73	0.72	0.72	0.72
20	0.97	0.93	0.86	0.72	0.72	0.72	0.72
ionosphere: aggregate algorithm							
2	0.80	0.83	0.84	0.83	0.81	0.81	0.83
5	0.84	0.81	0.77	0.76	0.68	0.78	0.78
10	0.81	0.78	0.75	0.69	0.62	0.77	0.69
20	0.84	0.72	0.65	0.70	0.73	0.58	0.65
ionosphere: control algorithm							
2	0.81	0.77	0.74	0.70	0.81	0.83	0.76
5	0.81	0.71	0.73	0.62	0.63	0.70	0.73
10	0.79	0.80	0.73	0.71	0.58	0.43	0.58
20	0.76	0.68	0.43	0.51	0.51	0.38	0.51

Fig. 6. Support vector machines, classification accuracy.

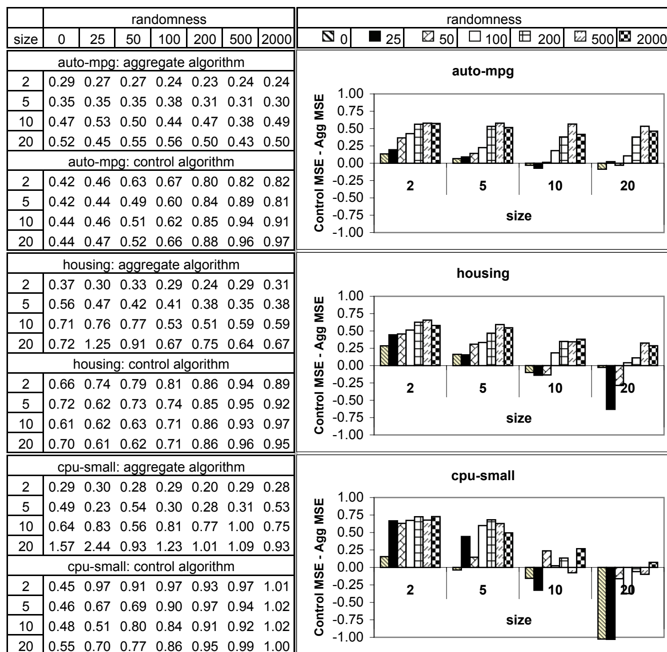
Figure 5 shows the results for the SVM aggregate regression algorithm compared with its control, where the loss insensitivity parameter ε was set to 0. The comparison between the aggregate case and the control case shows the increasing advantage of the aggregate approach as aggregate size and randomness increase. Unlike neural networks, the aggregate has the greatest advantage with high randomness. This is likely due to the fact that SVMs inherently avoid overfitting by regularizing the separating surface, which has a more dramatic impact with noisy data.

SVM parameters were chosen similarly to the regression case, with a different C and D pair chosen for each dataset. Once again, CPLEX was used for solving a quadratic program in order to optimize the SVM. Results are shown in Figure 6. The aggregate algorithm consistently outperformed the control across almost all randomness and aggregate values and each dataset. The aggregate's performance in relation to the control improves steadily as randomness and aggregate collection size increase, as we expected. It is particularly interesting to observe that aggregate performance was nearly constant across aggregate size and randomness (although less so with the ionosphere data). Control data showed similar trends to k-nearest neighbor with strong performance at low randomness, but quickly deteriorating accuracy as randomness and aggregate collection size increase.

5.5 Decision Trees

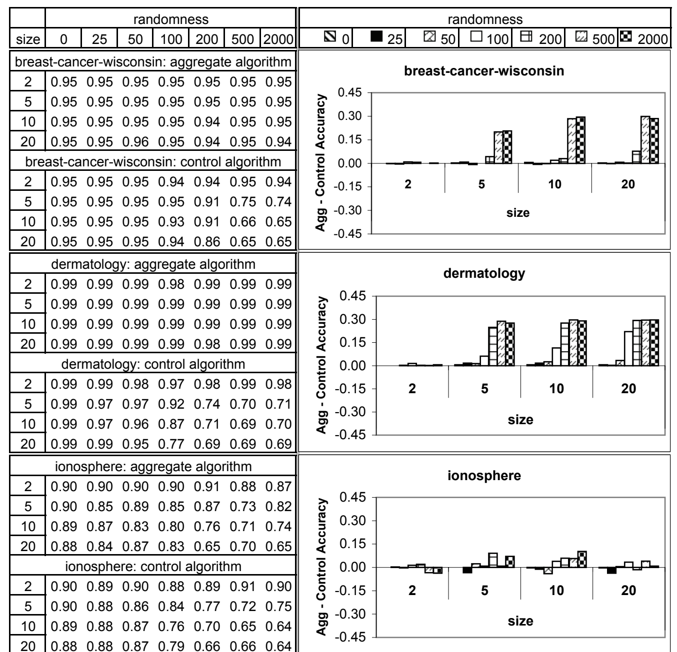
In designing our decision tree experiments, there was no traditional algorithm of which we were aware that made sense to use as a control. Using decision trees that proceed in a depth-first-fashion with local splitting criteria would seem to be incomparable to our global approach which expands nodes that have the largest number of points that map to them. For example, it was unclear which local splitting criterion to use for the control algorithm (entropy, gini index, etc.) should be used, when our aggregate algorithm uses no such local criterion at all. Therefore, for comparison purposes we chose to construct two new algorithms (one for regression, one for classification) that reflected the general structure of our new aggregate decision tree algorithms, yet still used an individual output value for each data point. As with all of our other control algorithms, we first assign each point in the dataset an individual output value. In the regression case, this is the average output value over all points in that collection. For a classification, each point is assigned the ratio of the number of positive classifications to the total number of points in the aggregate collection. The algorithms themselves then proceed as described below.

The regression algorithm that we use for a control is very similar to the one that we describe in section 4.8. We still choose to split whichever node has the most points in it, and test every feature-value combination. The difference here is that we do not need to optimize over the entire tree in order to determine the optimal regression value for a new node. Since every training point has been assigned a specific regression value, we can locally determine the regression value for that node that minimizes mean square error. Specifically, it is easy to show that this value must be the average of all the output values for that node. Therefore, for each feature-value combination, we split a node and assign optimal values to its two children, and measure mean square error. The feature-value split that minimizes that error is



size	randomness 0	25	50	100	200	500	2000
auto-mpg: aggregate algorithm							
2	0.29	0.27	0.27	0.24	0.23	0.24	0.24
5	0.35	0.35	0.35	0.38	0.31	0.31	0.30
10	0.47	0.53	0.50	0.44	0.47	0.38	0.49
20	0.52	0.45	0.55	0.56	0.50	0.43	0.50
auto-mpg: control algorithm							
2	0.42	0.46	0.63	0.67	0.80	0.82	0.82
5	0.42	0.44	0.49	0.60	0.84	0.89	0.81
10	0.44	0.46	0.51	0.62	0.85	0.94	0.91
20	0.44	0.47	0.52	0.66	0.88	0.96	0.97
housing: aggregate algorithm							
2	0.37	0.30	0.33	0.29	0.24	0.29	0.31
5	0.56	0.47	0.42	0.41	0.38	0.35	0.38
10	0.71	0.76	0.77	0.53	0.51	0.59	0.59
20	0.72	1.25	0.91	0.67	0.75	0.64	0.67
housing: control algorithm							
2	0.66	0.74	0.79	0.81	0.86	0.94	0.89
5	0.72	0.62	0.73	0.74	0.85	0.95	0.92
10	0.61	0.62	0.63	0.71	0.86	0.93	0.97
20	0.70	0.61	0.62	0.71	0.86	0.96	0.95
cpu-small: aggregate algorithm							
2	0.29	0.30	0.28	0.29	0.20	0.29	0.28
5	0.49	0.23	0.54	0.30	0.28	0.31	0.53
10	0.64	0.83	0.56	0.81	0.77	1.00	0.75
20	1.57	2.44	0.93	1.23	1.01	1.09	0.93
cpu-small: control algorithm							
2	0.45	0.97	0.91	0.97	0.93	0.97	1.01
5	0.46	0.67	0.69	0.90	0.97	0.94	1.02
10	0.48	0.51	0.80	0.84	0.91	0.92	1.02
20	0.55	0.70	0.77	0.86	0.95	0.99	1.00

Fig. 7. Decision trees, regression mean square error.



size	randomness 0	25	50	100	200	500	2000
breast-cancer-wisconsin: aggregate algorithm							
2	0.95	0.95	0.95	0.95	0.95	0.95	0.95
5	0.95	0.95	0.95	0.95	0.95	0.95	0.95
10	0.95	0.95	0.95	0.95	0.94	0.95	0.95
20	0.95	0.95	0.96	0.95	0.94	0.95	0.94
breast-cancer-wisconsin: control algorithm							
2	0.95	0.95	0.95	0.94	0.94	0.95	0.94
5	0.95	0.95	0.95	0.95	0.91	0.75	0.74
10	0.95	0.95	0.95	0.93	0.91	0.66	0.65
20	0.95	0.95	0.95	0.94	0.86	0.65	0.65
dermatology: aggregate algorithm							
2	0.99	0.99	0.99	0.98	0.99	0.99	0.99
5	0.99	0.99	0.99	0.99	0.99	0.99	0.99
10	0.99	0.99	0.99	0.99	0.99	0.99	0.99
20	0.99	0.99	0.99	0.99	0.98	0.99	0.99
dermatology: control algorithm							
2	0.99	0.99	0.98	0.97	0.98	0.99	0.98
5	0.99	0.97	0.97	0.92	0.74	0.70	0.71
10	0.99	0.97	0.96	0.87	0.71	0.69	0.70
20	0.99	0.99	0.95	0.77	0.69	0.69	0.69
ionosphere: aggregate algorithm							
2	0.90	0.90	0.90	0.90	0.91	0.88	0.87
5	0.90	0.85	0.89	0.85	0.87	0.73	0.82
10	0.89	0.87	0.83	0.80	0.76	0.71	0.74
20	0.88	0.84	0.87	0.83	0.65	0.70	0.65
ionosphere: control algorithm							
2	0.90	0.89	0.90	0.88	0.89	0.91	0.90
5	0.90	0.88	0.86	0.84	0.77	0.72	0.75
10	0.89	0.88	0.87	0.76	0.70	0.65	0.64
20	0.88	0.88	0.87	0.79	0.66	0.66	0.64

Fig. 8. Decision trees, classification accuracy.

the one that we choose.

The classification algorithm that we use for control purposes is a similar adaptation of the aggregate algorithm that we describe in section 4.7 and the regression control algorithm above. As in the aggregate classification algorithm, we start with the node with the most points in it, and try every feature-value split. For each split, we assign all four combinations of positive and negative classifications to the two children. If we consider a positive value to be a 1 and a negative value to be a 0, we can then calculate a mean square error with the training data that has ended up at each point, as each training point is labeled with the ratio of positive classifications to total for its aggregate collection. The feature-value-classification split that minimizes mean square error is the one that we choose.

In running the experiments for our aggregate regression approach, we experimented briefly with C and chose to fix it at .05. As with other algorithms in this paper, better experimentation would yield better results, and a tuning set should be used to ensure proper experimental procedure. We simply chose to find a single C that worked reasonably over all datasets and variations thereof to keep the experiments limited. In short, we made it harder for our own algorithm to succeed by not conducting more fine-tuned experiments on C. We also chose to limit the tree depth to 5 in order to pick a simple pruning approach that would serve easily for comparison purposes.Results are shown in Figure 7. The aggregate algorithm largely outperformed the control across all datasets with varying degrees of success. Increases in randomness had no clear effect on the aggregate results, while increased aggregate collection size did increase error. For the control, on the other hand, both increased randomness and aggregate size increased error. It is worth noting that the control outperformed the aggregate at large aggregate collection size and low randomness. This is because of the criterion that we used for evaluating splits; our algorithm assigns inaccurate regression values to leaf nodes in these situations. An appropriate choice for C mitigates this problem, but the value .05 is too small to correct it in this particular case. In general, the choice of C greatly affects results; better experiments with a tuning set would have improved the quality of our results further.

We show the results of our classification experiments in Figure 8. We again set a depth limit of 5. For the dermatology and breast cancer data, the aggregate algorithm returned nearly constant accuracies, regardless of randomness and aggregation size. The control algorithm, on the other hand, resembled much more closely the sort of trend we had observed with k-nearest neighbor, where aggregate size 2 is fairly constant, while accuracy for the other groups decreases quickly with increased randomness and larger group size. Overall, our algorithm showed similar results to the control with no randomness, but as randomness increased, it quickly showed itself to be superior. The ionosphere results do not demonstrate as clearly the superiority of our aggregate algorithm over the control, although our algorithm still performed better overall.

6 Conclusions and Future Work

We have proposed a new machine learning problem, known as the aggregate output learning problem, that does not seem to have been previously examined in the literature. This problem, though inspired from atmospheric data analysis, could have broad ramifications

in working with data masked for privacy purposes. We present a formal framework for this problem for both regression and classification, and provide adaptations of k-nearest neighbor, neural networks, support vector machines, and decision trees (classification and regression for each) to handle the aggregate problem. We summarize a series of experiments for both frameworks that show our approach to be highly effective. For SVM classification, we have shown a new connection between aggregate output learning and semi-supervised learning. The aggregate output learning problem may thus illustrate new insights into semi-supervised learning.

There is clearly room for fine-tuning and improvement with each of the algorithms, and particularly with decision tree regression and neural network classification. Determining the particular kinds of data that are most appropriate for each approach would be beneficial when applying the algorithm to data sets. Optimizing our linear SVMs using techniques that the SVM community has developed would increase the efficiency and portability of our algorithm. The support vector machine approaches that we have proposed are all linear, and we would like to develop nonlinear versions as well.

ACKNOWLEDGMENTS

Much of the programming for our experiments, as well as significant assistance with the literature search, was done by a collection of undergraduates as part of a data mining course at Carleton College. Research assistants also made a major contribution. We acknowledge and thank David Barbella, Sami Benzaid, Deborah Chasman, Mark Dyson, Reid Gilman, Thomas Hagman, Bret Jackson, Daniel Lew, Brandy McMenamy, Peter Nelson, Charles Noneman, Jerad Phelps, Kevin Reschke, Ed Williams, Paul Wilmes, and Kelson Zawack. The work done by these students was critical to the success of this project. Other portions of this work were funded by NSF ITR grant IIS-0326328.

REFERENCES

[1] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[2] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. Cambridge, MA: MIT Press, 2001.

[3] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[4] K. Bennett and A. Demiriz, "Semi-supervised support vector machines," in *Advances in Neural Information Processing Systems*, vol. 12. MIT Press, 1998, pp. 368–374.

[5] A. M. Bensaid, L. O. Hall, J. C. Bezdek, and L. P. Clarke, "Partially supervised clustering for image segmentation," *Pattern Recognition*, vol. 29, pp. 859–871, 1996.

[6] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez, "Solving the multiple-instance problem with axis-parallel rectangles," *Artificial Intelligence Journal*, vol. 89, 1997.

[7] G. Arminger, N. Lijphart, and W. Müller, "Die verwendung loglinearer modelle zur disaggregierung aggregierter daten," *Allgemeines Statistisches Archiv*, vol. 3, pp. 273–291, 1981.

[8] N. Quadrianto, A. J. Smola, T. S. Caetano, and Q. V. Le, "Estimating labels from label proportions," in *ICML*, ser. ACM International Conference Proceeding Series, W. W. Cohen, A. McCallum, and S. T. Roweis, Eds., vol. 307. ACM, 2008, pp. 776–783.

[9] N. McGrogan, C. M. Bishop, and L. Tarassenko, "Neural network training using multi-channel data with aggregate labelling," in *Proceedings of the Ninth International Conference on Artificial Neural Networks*, vol. 2. IEE, 1999, pp. 862–867.

[10] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving classification of customer data without loss of accuracy," in *Proceedings of the Fifth SIAM Int'l Conference on Data Mining*, 2005, pp. 92–102.

[11] B.-C. Chen, L. Chen, R. Ramakrishnan, and D. R. Musicant, "Learning from aggregate views," in *Proceedings of the 22nd International Conference on Data Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, 2006.

[12] R. Jiroušek and S. Přeučil, "On the effective implementation of the iterative proportional fitting procedure," *Computational Statistics & Data Analysis*, 1995.

[13] D. R. Musicant, J. M. Christensen, and J. F. Olson, "Supervised learning by training on aggregate outputs," in *Proc. of the Seventh Int'l Conference on Data Mining*. IEEE Press, 2007, pp. 252–261.

[14] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. New Jersey: Pearson Education, Inc., 2003.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall, 1999.

[16] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge: Cambridge University Press, 2000.

[17] G. Fung and O. Mangasarian, "Semi-supervised support vector machines for unlabeled data classification," *Optimization Methods and Software*, vol. 15, pp. 29–44, 2001.

[18] R. Collobert and S. Bengio, "SVMTorch: Support vector machines for large-scale regression problems," *Journal of Machine Learning Research*, vol. 1, no. 1, pp. 143–160, February 2001.

[19] S. Rüping, "mySVM," September 2001, http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM.

[20] M. M. Dunham, *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2003.

[21] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Ttrees*. Chapman and Hall/CRC, 1984.

[22] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[23] P. M. Murphy and D. W. Aha, "UCI repository of machine learning databases," 1992, http://www.ics.uci.edu/~mlearn/MLRepository.html.

[24] Delve, "Data for evaluating learning in valid experiments," http://www.cs.utoronto.ca/~delve.

[25] *ILOG CPLEX 10.1*, ILOG CPLEX Division, Sunnyvale, CA, 2007.