Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Unlearning from Demonstration

Keith Sullivan and **Ahmed ElMolla** and **Bill Squires** and **Sean Luke**
Department of Computer Science, George Mason University, Fairfax, VA USA
ksulliv2@cs.gmu.edu, aelmolla@gmu.edu, wsquires@gmu.edu, sean@cs.gmu.edu

Abstract

When doing learning from demonstration, it is often the case that the demonstrator provides corrective examples to fix errant behavior by the agent or robot. We present a set of algorithms which use this corrective data to identify and remove noisy examples in datasets which caused errant classifications, and ultimately errant behavior. The objective is to actually modify the source datasets rather than solely rely on the noise-insensitivity of the classification algorithm. This is particularly useful in the sparse datasets often found in learning from demonstration experiments. Our approach tries to distinguish between noisy misclassification and mere undersampling of the learning space. If errors are a result of misclassification, we potentially remove the responsible points and update the classifier. We demonstrate our method on UCI Machine Learning datasets at different levels of sparsity and noise, using decision trees, K-Nearest-Neighbor, and support vector machines.

1 Introduction

Learning from Demonstration uses machine learning to teach an agent, often a robot, how to perform a behavior by following examples provided by a human demonstrator. Often this demonstrator is acting as a *trainer*, by which we mean that the demonstrator is iteratively observing the current robot or agent behavior and providing new corrective examples to fix perceived errors in the behavior. This continues until the trainer is satisfied with the current behavior.

Our work involves training robot and agent behaviors. For example, one might train a soccer robot to acquire the ball, approach it, pivot towards the goal, and kick the ball. Such behaviors are distinguished by an extreme paucity of examples, because the lion's share of meaningful examples are collected when the trainer wishes the robot to transition from some sub-behavior to another (e.g., to go from approaching to pivoting).

This paucity of examples poses a challenge for machine learning in a variety of ways, one of which is the profound impact of just a few incorrect or noisy examples. Ordinarily a machine learning experimenter would deal with this by adding more and more examples to the dataset until the learned model can correctly guess which examples are noise and which are not. The sample sparsity may make this difficult to do. But because we are training rather than simply applying machine learning, we have another option. We may assume that the corrective examples are largely expected to be accurate and use this assumption to directly remove incorrect examples from the dataset itself. Thus when iteratively training the agent, we can not only add new examples but also remove incorrect ones, thereby helping the agent learn an accurate model on a very limited set of data. We refer to this notion as *unlearning*.[1]

In our work, the machine learner is a classification algorithm, such as C4.5, K-Nearest-Neighbor (K-NN), or Support Vector Machines (SVMs). The procedure we envision is as follows. After the trainer has observed an errant behavior, he provides a set of correcting examples. For each corrective example, the agent queries the classifier to see if it is misclassifying this example. If it is, the agent identifies a set of responsible *misclassifying examples* which are candidates for possible eventual removal from the dataset. The corrective example is added to the dataset and the model is rebuilt.

Unfortunately, not all misclassifications are due to incorrect examples which must be punished or removed. Many misclassifications may come from an undersampled region, resulting in a (correct) example inappropriately responsible for classifying a large region. This is particularly problematic when, as is the case for us, examples are scarce. We need to be able to conservatively distinguish between noisy examples and correct examples in the undersampled region.

In this paper, we present two techniques for identifying noisy examples (versus correct but overgeneralizing ones) and unlearning them. We also present methods for extracting candidates for unlearning. Such methods rely on the specifics of the classifier being used: we show methods for decision trees, K-NN, and SVMs.

The paper is organized as follows: Section 2 reviews related work and Section 3 presents our two methods to detect errors within a constructed classifier. We present experiments in Section 4 and offer conclusions and future work in Section 5.

2 Related Work

The learning from demonstration or training literature may be divided roughly into two categories. First, there is a significant

[1] We know the same term is also used in the context of certain neural networks, and hope that our usage does not cause confusion.

literature in training policies for path- or trajectory-planning, such as [Bentivegna *et al.*, 2004; Dinerstein *et al.*, 2007; Kasper *et al.*, 2001; Nakanishi *et al.*, 2004]. In this literature there is often a wealth of data: for example, if one were training a helicopter trajectory, every slight modification of the joystick might be used as a data point. Additionally, this literature relies on accurate human demonstrations for the initial policy; [Grollman and Ballard, 2012] instead use only failed demonstrations to initialize the robot's policy. Researchers typically use reinforcement learning to develop an optimal policy starting from provided demonstrations. Convergence rates can be improved using structured prediction to extract knowledge from multiple demonstrations [Parker *et al.*, 2007].

The other category consists of training task plans or behaviors, such as [Nicolescu and Mataric, 2002; Veeraraghavan and Veloso, 2008; Goldberg and Mataric, 2002]. These models tend to yield sparse datasets because the most interesting data points lie on transitions between low-level actions or behaviors. For example, if we were training a robot to acquire the ball, then kick it, the crucial data describes when to transition from acquisition to kicking. We can repeatedly sample data points while in acquisition ("In this situation, keep on acquiring", "in this situation, keep on acquiring", and so on), but this data will far outnumber the transition data. Because such models often deal with a fixed set of possible low-level actions, classification is the usual learning paradigm.

Dealing with noisy data is a common issue in classification problems [Gamon, 2004; Kalapanidas *et al.*, 2003; Nettleton *et al.*, 2010]. However, such research has largely focused on determining the nature of the noise rather than correcting the data directly. More closely related to our work is the idea of correcting a sample bias where the training and testing data are drawn from different probability distributions [Huang *et al.*, 2006]. Majority voting among multiple classifiers has been used to identify errantly labeled data from physical experiments [Furey *et al.*, 2000], and classifiers have been used to fill in missing data values [Lakshminarayan *et al.*, 1996; Jereza *et al.*, 2010]. These techniques are closely related to the notion of *boosting* (most famously [Freund and Schapire, 1995]), where successive classifiers are trained on data weighted such that misclassified samples are given more consideration. Importantly however, unlike our work, these approaches do not assume the presence of a separate corrective dataset.

Another related area is *incremental learning*, where a classifier is built incrementally from currently available data or from a subset of large dataset. Incremental learning allows the construction of dynamic classifiers capable of adapting to changing data. Incremental learning addresses situations where not all data fits in memory, or where new data arrives as a stream. [Syed *et al.*, 1999; Fei-Fei *et al.*, 2007; Ross *et al.*, 2008]. Note that this new data is not meant to correct old data per se: it simply is newer information which may require an incremental restructuring of the model.

Researchers have implemented incremental decision trees [Gehrke *et al.*, 1999], Bayesian inductive learning [Michalski *et al.*, 1986], and neural networks [Polikar *et al.*, 2001]. Additionally, [Cauwenberghs and Poggio, 2001; Diehl and Cauwenberghs, 2003] developed an on-line incremental and decremental support vector machine. As training vectors are encountered, an exact solution is computed using all the previously seen data combined with the new training vector. This incremental approach continually partitions the data into three sets, one of which is the support vectors. Ma et al extended this approach to support vector regression [Ma *et al.*, 2003].

3 Method

We present two methods to do unlearning, one which assumes a distance metric in the feature space, and one which does not.

Let $\mathbf{C}$ be the current errant classifier model, and let $\mathcal{G}$ be the data set from which $\mathbf{C}$ was constructed. We wish to identify noisy examples in $\mathcal{G}$ and ultimately remove them. To help us, we have a set E of *corrective examples* supplied by the demonstrator. For each $e \in E$, if $\mathbf{C}$ is misclassifying e, both of our methods will identify a set M of *candidate examples* which were responsible for this misclassification. This set is computed by calling a *candidate-gathering algorithm* appropriate to the particular kind of classifier used. The methods then decide which misclassifying examples $m \in M$, if any, should be removed from $\mathcal{G}$ or should be downgraded for possible future removal. After possibly removing examples, the methods then add e to $\mathcal{G}$ and rebuild $\mathbf{C}$.

Each of these methods must distinguish between noisy examples and overgeneralizing but correct outliers. They both take different approaches based on the same two heuristics. First, a misclassifying candidate m is more likely to be noisy if it is highly similar to a corrective example e. On the other hand, if it is an overgeneralizing outlier example, it is likely to be misclassifying e from afar. This is our *similarity* heuristic. Second, if multiple misclassifying candidates are clustered near one another, they are less likely to be noise: rather e is probably a special case. This is our *strength in numbers* heuristic.

The first method, *metric unlearning*, assumes the presence of a metric distance measure and uses this to gauge measures of similarity, which it then applies to both heuristics. Of course, a reasonably scaled metric distance measure may not be available depending on the features involved. The second method, *non-metric unlearning*, makes no assumptions of distance: it simply lowers the "score" of misclassifying examples, ultimately removing examples whose scores have dropped below zero. Further, when using K-NN as our classifier, we have the additional opportunity to be even more conservative about not removing valid examples, by using a technique to reduce M to only those examples which are not likely to be on a "border" delimiting two different classes.

3.1 Metric Unlearning

The metric unlearning method begins by scaling the corrective example e and the misclassifying candidates $m \in M$ so that their ranges go from 0 to 1 inclusive (see Algorithm 1). This makes the strong assumption that classifier features have been prescaled so that distance measures are appropriate. Next, the candidate $m^* \in M$ most *similar* to e is identified. We then construct a bounding hypersphere around the points in M where the centroid is the numerical average of the feature vector and the radius is the maximum distance between the centroid and all examples in M.

Algorithm 1 METRIC-UNLEARNING $(\mathbf{C}, E, \mathcal{G}, \gamma, \beta)$

for all $e \in E$ **do**
  $e' \leftarrow$ Scale e to $[0,1]$
  $M \leftarrow$ GATHER-EXAMPLES$(\mathbf{C}, e, ...)$
  $M' \leftarrow$ Scale all $m \in M$ to $[0,1]$
  $m^\star \leftarrow \text{argmax}_{m' \in M'}$ SIMILARITY(m', e')
  c $\leftarrow$ centroid of all points $m' \in M'$
  r $\leftarrow \max_{m' \in M'}$ DISTANCE(m', c)
  if $r = 0$ and SIMILARITY$(m^\star, e') > \gamma$ **then**
    Delete $m^\star$ from $\mathcal{G}$
  else if DISTANCE$(e', c) < r$ **then**
    $M'' \leftarrow M' - \{m^\star\}$
    c $\leftarrow$ centroid of all points $m'' \in M''$
    r $\leftarrow \max_{m'' \in M''}$ DISTANCE(m'', c)
    if $r = 0$ and SIMILARITY$(m'', e') < \beta$ **then**
      Delete m^* from $\mathcal{G}$
    else if DISTANCE$(e', c) > r$ **then**
      Delete m^* from $\mathcal{G}$
  Add e to $\mathcal{G}$
  Recompute $\mathbf{C}$ from $\mathcal{G}$

There are two possible situations. If there is only a single misclassifying candidate (and so the radius of the bounding hypersphere is zero), then the candidate m^* is deleted from $\mathcal{G}$ if it is sufficiently similar to e (using the threshold constant γ).

If there is more than one misclassifying candidate, we must test to see if strength in numbers is relevant. The goal is to determine if the nearby point m^* should be viewed as distinct from the other points M, or if all of M are best seen as acting as a group to misclassify e. Our (weak) test is as follows. We build a hypersphere around all points in M. If e is outside this hypersphere, we view all of M as a group. If e is within the hypersphere, we then build a second hypersphere consisting of all candidates except for m^*. If there are multiple such candidates besides m^* (i.e., the second hypersphere has non-zero radius), then we delete m^* if e is outside the second hypersphere, with the heuristic belief that m^* and e are significantly separated from the other points in M. If there is only one such candidate $m'' \in M$ other than m^* (and thus the second hypersphere radius is zero), we have less to go on. In this case, we delete m^* if e is *not* so similar to m'' that m'' and m^* may be considered clustered together. The threshold β determines this level of similarity. Finally we add e to $\mathcal{G}$ and rebuild the classifier $\mathbf{C}$.

The scaling is simply based on user-defined minimum and maximum values. Note that this algorithm uses two different measures of similarity. Because the scaling is user-defined, we try to use a conservative scale-free measure (cosine similarity) when possible: this is the function SIMILARITY. When computing hyperspheres, we cannot be scale-free, and so we fall back on the metric distance, DISTANCE, for comparison to hypersphere centers.

3.2 Non-metric Unlearning

Our non-metric approach simply punishes examples when they contribute to an incorrect classification (see Algorithm 2). Every time an example contributes to an incorrect classification, its score (initially 1.0) is reduced. If the score falls below 0.0,

Algorithm 2 NON-METRIC UNLEARNING $(\mathbf{C}, E, \mathcal{G}, \alpha)$

for all $e \in E$ **do**
  $M \leftarrow$ GATHER-EXAMPLES$(\mathbf{C}, e, ...)$
  for all $m \in M$ **do**
    $m_{score} \leftarrow m_{score} - \alpha/|M|$
    if $m_{score} < 0$ **then**
      Delete m from $\mathcal{G}$
  Add e to $\mathcal{G}$
  Recompute $\mathbf{C}$ from $\mathcal{G}$

then the example is removed. This approach allows examples to misclassify a few times before they are removed.

Following the strength in numbers heuristic, we wish to punish misclassifying examples less when there are multiple other examples which are also misclassifying the corrective example. Accordingly, we modify the score reduction by dividing it by the number of misclassifying examples. The value $0 < \alpha \leq 1$ indicates the amount of score reduction when there is just a single misclassifying example: it should be set small enough that a misclassifying example is given a few chances before outright removal.

How do we apply the similarity heuristic? Only indirectly. If we have added a few corrective points E, in a certain region, and they did not reduce a misclassifying example m to the point of removal, is m still a threat? If m was distant from these points, it is not likely to affect regions around them any more (recall that the various $e \in E$ have been since added to $\mathcal{G}$). But if m very close to the points E, perhaps amongst them, it might have more of an impact.

3.3 Classifiers

Each of the previous two methods calls a function called GATHER-EXAMPLES($\mathbf{C}$, e, ...) whose purpose is to return all examples $m \in M$ responsible for causing the classifier $\mathbf{C}$ to misclassify the corrective example e. The implementation of this function varies from classifier to classifier. While the methods likely work with any classifier, we focus on three common classification algorithms: decision trees (C4.5), K-Nearest Neighbor (K-NN), and support vector machines (SVMs).

All three implementations work in largely the same way. First, the algorithm determines if e is indeed misclassified by $\mathbf{C}$. If it is, then the algorithm returns, as M, those examples originally used to build $\mathbf{C}$ which are responsible for the misclassification.

K-Nearest Neighbor Algorithm 3 presents our approach for K-Nearest-Neighbor. In K-NN, a point's class is determined by voting from among the K nearest examples to the point in the space. We determine D, the plurality of this vote, that is, those examples which had voted for the incorrect class.

If a point lies near the boundary between two classes, it may be misclassified simply because of the sparsity of the space, and the resulting plurality will all be valid examples properly lying on their side of the boundary. We wish to avoid returning these examples and potentially deleting them. Thus we go through an additional procedure to attempt to identify boundary examples. Specifically for each $d \in D$ we perform

Algorithm 3 GATHER-K-NN-EXAMPLES $(\mathbf{C}, e, \mu)$

$M \leftarrow \{\}$
$class \leftarrow$ K-NN-CLASSIFY$(\mathbf{C}, e)$
if *class* is incorrect **then**
  $D \leftarrow$ Plurality of K-NN-CLASSIFY$(\mathbf{C}, e)$
  for all $d \in D$ **do**
    $Q \leftarrow$ Plurality of (K+1)NN-CLASSIFY$(\mathbf{C}, d)$
    if $\leq \mu$ members of Q have class *class* **then**
      $M \leftarrow M \cup \{d\}$
return M

a K-Nearest-Neighbor classification on the point represented by d, using a value of K one larger than normally performed in the classifier. If a sufficient number μ of neighbors support the original class of d, then d is likely a border example rather than an isolated and potentially noisy example. We return all members of D which fail this test.

Decision Trees For decision trees, we modified the standard C4.5 decision tree algorithm to store the examples which were used to construct each leaf. The construction of M starts by identifying the leaf node L in which e was misclassified (see Algorithm 4). If the decision tree is not well-pruned, this node may have very few incorrect examples from which to form M, and so we have chosen to move up one level to the parent of L. All misclassifying examples found in leaf nodes rooted by the parent are added to M.

Algorithm 4 GATHER-DECISION-TREE-EXAMPLES $(\mathbf{C}, e)$

$M \leftarrow \{\}$
$class \leftarrow$ DECISION-TREE-CLASSIFY$(\mathbf{C}, e)$
if *class* is incorrect **then**
  $L \leftarrow$ leaf node in $\mathbf{C}$ used to classify e
  if L is the root node **then**
    $P \leftarrow L$
  else
    $P \leftarrow$ parent of L
  $D \leftarrow$ examples used to form subtree rooted by P
  for all $d \in D$ **do**
    if class of d matches *class* **then**
      $M \leftarrow M \cup \{d\}$
return M

Support Vector Machines The examples responsible for misclassification in a support vector machine are, of course, its support vectors. However, in a sparse problem there may not be many of them: and furthermore, there may be other points just outside the SVM's margin which would also misclassify the corrective point were certain support vectors removed. For this reason we have chosen to increase the margin of the SVM classifier and extract those misclassifying points falling within this larger margin (see Algorithm 5). This in some sense has a similar goal to choosing the parent of the leaf in the previous decision tree algorithm. The size of the margin increase is defined by the variable τ, which we have set to 2.

Algorithm 5 GATHER-SVM-EXAMPLES $(\mathbf{C}, e, \tau)$

$M \leftarrow \{\}$
$class \leftarrow$ SVM-CLASSIFY$(\mathbf{C}, e)$
if *class* is incorrect **then**
  $w \leftarrow$ width of margin in $\mathbf{C}$
  $\mathbf{C}' \leftarrow \mathbf{C}$ with margin whose width is $w \times \tau$
  $D \leftarrow$ examples within margin of $\mathbf{C}'$
  for all $d \in D$ **do**
    if class of d matches *class* **then**
      $M \leftarrow M \cup \{d\}$
return M

4 Experiments

We conducted experiments to compare the metric and non-metric unlearning methods against simply leaving the noisy data in the data set and relying on the learning algorithm to compensate for it. We also compared against perfect noise reduction (by just eliminating all the noisy data). We expected the unlearning methods to do better than leaving the noise in, but worse than the perfect noise reduction.

Our experimental procedure was as follows. We randomly shuffled, then split the data into three pieces: a set of *unchanged* data (70%), a set of *corrective* data (20%), and a set of *testing* data (10%). From the corrective data we generated an equal sized set of *error* examples. Examples in the error set were the same as the corrective set but with the addition of Gaussian noise to the feature vector and random relabeling to an incorrect label. We trained a classifier using the unchanged data and error data, then performed unlearning using the corrective data, and finally tested for generality on the testing data. We repeated this splitting, correction, and generalization testing process 1000 times for each experiment.

Our goal was to compare methods using simple and small datasets and a variety of classification algorithms (similar in quality to those found in learning from demonstration).

- Dataset. We conduced experiments using smaller datasets drawn from the UCI Data Repository. We chose Iris, Glass, and Wine.
- Classification Algorithm. We did experiments using C4.5, K-NN, and SVMs. For K-Nearest Neighbor we chose K=1 and K=3. For the support vector machine we used a radial basis function kernel using an exponent of 0.5. For decision trees we examined both unpruned and pruned trees: the pruning algorithm we chose was Pessimistic Error Pruning. While PEP does not have as good performance as many other pruning algorithms in the literature, it has the distinct advantage of not requiring a separate "pruning set" of data. This was desirable because in real-world use we do not expect to have enough data examples to provide a separate pruning set.
- Degree of Noise. While the *amount* of noise (and corrective data) was fixed to 20%, we varied the *degree* of noise, in terms of variance of the Gaussian curve. The variance was $\eta(max - min)$, where *min* and *max* are the minimum and maximum legal values of the feature and η was set to 1/5, 1/20, and 1/100 (we also tested with other in-between η values, with similar expected results).

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9553	0.9131	**0.9307**	**0.9255**	0.9553	0.8002	**0.8901**	**0.8601**	0.9553	0.7519	**0.9461**	**0.8490**
Glass	0.6921	0.6707	0.6810	**0.6822**	0.6921	0.6441	**0.6816**	**0.6705**	0.6921	0.5653	**0.6887**	**0.6421**
Wine	0.9533	0.9370	**0.9464**	0.9442	0.9533	0.7998	**0.9506**	**0.8722**	0.9533	0.7566	**0.9520**	**0.8488**
						3-NN						
Iris	0.9537	0.9409	0.9468	0.9492	0.9537	0.8887	**0.9361**	**0.9295**	0.9537	0.8539	**0.9370**	**0.9331**
Glass	0.7008	0.6734	**0.6895**	**0.6980**	0.7008	0.6615	**0.6927**	**0.6971**	0.7008	0.6193	**0.6866**	**0.6828**
Wine	0.9615	0.9524	**0.9607**	0.9594	0.9615	0.8895	**0.9511**	**0.9472**	0.9615	0.8548	**0.9462**	**0.9408**
						Decision Tree (Unpruned)						
Iris	0.9459	0.8705	**0.8915**	**0.8877**	0.9459	0.8029	**0.8497**	**0.8535**	0.9459	0.8014	**0.8765**	**0.8616**
Glass	0.6701	0.6379	**0.6577**	**0.6572**	0.6701	0.6355	**0.6544**	**0.6514**	0.6701	0.6306	**0.6591**	**0.6492**
Wine	0.9332	0.8321	**0.8638**	**0.8636**	0.9332	0.7375	**0.8103**	**0.7956**	0.9332	0.7206	**0.8365**	**0.8079**
						Decision Tree (Pruned)						
Iris	0.9427	0.9135	0.9213	**0.9226**	0.9427	0.8761	**0.9081**	**0.9094**	0.9427	0.8799	**0.9250**	**0.9213**
Glass	0.6711	0.6330	**0.6520**	**0.6529**	0.6711	0.6274	**0.6460**	**0.6426**	0.6711	0.6301	**0.6501**	**0.6496**
Wine	0.9340	0.8591	**0.8811**	**0.8846**	0.9340	0.8185	**0.8749**	**0.8715**	0.9340	0.8093	**0.8892**	**0.8844**
						Support Vector Machine						
Iris	0.9102	0.3886	0.4280	**0.9070**	0.9102	0.7389	**0.8649**	**0.8705**	0.9102	0.7374	**0.8695**	**0.8668**
Glass	0.3346	0.3311	0.3163	0.3393	0.3346	0.3329	0.3313	0.3284	0.3346	0.3249	0.3259	0.3350
Wine	0.9329	0.3906	0.3991	**0.9350**	0.9329	0.6400	**0.8828**	**0.8861**	0.9329	0.6544	**0.8834**	**0.8867**

Table 1: Results for $\omega = 100\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

- Data Sparsity. While the Wine, Glass, and Iris data sets are already fairly small (between 100 and 250 data points), our learning from demonstration research tends to use even smaller sets. Thus we experimented with three data set sizes: the full ($\omega = 100\%$) data set, an $\omega = 50\%$ sized set, and a $\omega = 25\%$ sized set. For the last two, the set was reduced by removing random data points.

We were curious as to how sensitive our algorithms are to their parameters and so performed some informal parameter tuning. In the non-metric unlearning algorithms, we varied α from 0 to 1 in steps of 0.1, while in the metric unlearning algorithm, we varied γ over 0.5, 0.75, and 1.0 while β ranged from 0 to 1 in steps of 0.1. Additionally, we varied μ from 0 to 5 in steps of 1. In general, γ has little effect on classification accuracy. However, for unpruned decision trees on the Wine dataset, increasing γ results in decreased accuracy. Additionally, β appears to have minimal effect, but this is probably due to the infrequency of the second hypersphere containing a single point. In the non-metric algorithms, accuracy either stays constant or increases as α increases from 0 to 1. For K-NN, setting $\mu = 2$ results in the best performance with no statistically significant impact as μ increases to 5.

Based on these trends, for metric unlearning, we fixed $\gamma = 0.5$ and $\beta = 0.5$ while for non-metric unlearning we set $\alpha = 0.9$. For K-Nearest Neighbor, we set $\mu = 2$, and, as mentioned earlier, for Support Vector Machines we set $\tau = 2$. Table 1 shows the results. For each algorithm and dataset, we compared against simply adding a point (which we call the "naive" approach) and running our unlearning algorithms. Bold numbers indicate a statistically significant increases over the naive approach, while underlined numbers indicate the statistically higher performance between the metric and non-metric unlearning algorithms. All statistical tests were at the 95% confidence level with the appropriate Bonferroni correction.

In general, the unlearning algorithms perform better than the naive approach, with metric unlearning slightly outperforming non-metric algorithms. However, in all cases, the unlearning algorithms failed to completely remove the error points.

Next, we investigated how these trends hold up with smaller datasets by changing the experimental procedure: at the start of each iteration, we randomly shuffled the data and then used the top $\omega\%$. Table 2 shows the results for $\omega = 50\%$ and Table 3 shows the results for $\omega = 25\%$.

We see the same trends as before: unlearning performs better than the naive approach, with metric unlearning performing slightly better than non-metric unlearning. But as the dataset shrinks, unlearning algorithms start to perform closer to the naive approach due to the fewer available points and associated information for our unlearning algorithms.

5 Conclusions

We presented two algorithms to correct errant classifiers assuming a paucity of examples. This paucity is common in learning from demonstration environments. Our approaches use two heuristics, similarity and strength in numbers, to potentially remove noisy examples and protect correct examples which have overgeneralized the space. Our algorithms performed well compared to simply adding additional datapoints.

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9471	0.8982	**0.9176**	**0.9194**	0.9471	0.7839	**0.8969**	**0.8534**	0.9471	0.7458	**0.9445**	**0.8430**
Glass	0.6532	0.6257	0.6437	0.6502	0.6532	0.5911	**0.6345**	**0.6346**	0.6532	0.5383	**0.6543**	**0.6004**
Wine	0.9471	0.9190	**0.9351**	0.9343	0.9471	0.7871	**0.9482**	**0.8724**	0.9471	0.7556	**0.9462**	**0.8546**
						3-NN						
Iris	0.9533	0.9356	0.9470	0.9474	0.9533	0.8744	**0.9323**	**0.9304**	0.9533	0.8435	**0.9313**	**0.9280**
Glass	0.6418	0.6177	0.6374	0.6412	0.6418	0.6052	**0.6412**	**0.6338**	0.6418	0.5721	**0.6358**	**0.6236**
Wine	0.9514	0.9437	0.9509	0.9465	0.9514	0.8814	**0.9377**	**0.9343**	0.9514	0.8530	**0.9373**	**0.9305**
						Decision Tree (Unpruned)						
Iris	0.9396	0.8531	**0.8776**	**0.8765**	0.9396	0.7714	**0.8404**	**0.8337**	0.9396	0.7681	**0.8591**	**0.8370**
Glass	0.6241	0.5875	0.6075	0.6003	0.6241	0.5756	0.5966	**0.5988**	0.6241	0.5695	**0.6034**	**0.5938**
Wine	0.8896	0.8036	**0.8420**	**0.8348**	0.8896	0.7275	**0.7987**	**0.7769**	0.8896	0.7045	**0.8191**	**0.7810**
						Decision Tree (Pruned)						
Iris	0.9421	0.9024	0.9175	**0.9145**	0.9421	0.8634	**0.9096**	**0.9058**	0.9421	0.8512	**0.9190**	**0.9204**
Glass	0.6311	0.5905	0.6094	**0.6093**	0.6311	0.5777	**0.6009**	**0.6040**	0.6311	0.5634	**0.5987**	**0.6006**
Wine	0.8873	0.8324	**0.8621**	**0.8543**	0.8873	0.7906	**0.8480**	**0.8446**	0.8873	0.7722	**0.8483**	**0.8514**
						Support Vector Machine						
Iris	0.6733	0.5751	**0.6700**	**0.6733**	0.6733	0.5235	**0.6350**	**0.6401**	0.6733	0.5321	**0.6424**	**0.6304**
Glass	0.3411	0.3000	0.2833	0.3431	0.3411	0.3056	0.3611	0.3333	0.3411	0.3372	0.3330	0.3280
Wine	0.4557	0.3865	**0.4505**	**0.4973**	0.4557	0.3923	0.4462	**0.4209**	0.4557	0.3873	0.4273	**0.4106**

Table 2: Results for $\omega = 50\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9336	0.8792	**0.9134**	0.9052	0.9336	0.7882	**0.9072**	**0.8438**	0.9336	0.7256	**0.9318**	**0.8294**
Glass	0.6038	0.5892	0.6097	0.5977	0.6038	0.5442	**0.5945**	**0.5997**	0.6038	0.5040	**0.6092**	**0.5650**
Wine	0.9345	0.9045	**0.9227**	0.9222	0.9345	0.7735	**0.9407**	**0.8548**	0.9345	0.7382	**0.9293**	**0.8393**
						3-NN						
Iris	0.9378	0.9202	0.9354	0.9314	0.9378	0.8474	**0.9174**	**0.9238**	0.9378	0.8328	**0.9182**	**0.9232**
Glass	0.5897	0.5755	0.5950	0.5927	0.5897	0.5577	**0.5953**	**0.5953**	0.5897	0.5172	**0.5772**	**0.5768**
Wine	0.9465	0.9273	0.9330	0.9417	0.9465	0.8560	**0.9240**	**0.9270**	0.9465	0.8462	**0.9228**	**0.9203**
						Decision Tree (Unpruned)						
Iris	0.9350	0.8410	**0.8758**	0.8602	0.9350	0.7552	**0.8354**	**0.8180**	0.9350	0.7296	**0.8392**	**0.8250**
Glass	0.5947	0.5177	0.5400	0.5662	0.5947	0.5083	**0.5483**	0.5302	0.5947	0.4865	**0.5360**	0.5297
Wine	0.8503	0.7687	**0.8068**	**0.8025**	0.8503	0.6875	**0.7755**	**0.7565**	0.8503	0.6845	**0.7900**	**0.7578**
						Decision Tree (Pruned)						
Iris	0.9326	0.8700	**0.9016**	**0.8958**	0.9326	0.8064	**0.8790**	**0.8838**	0.9326	0.8144	**0.9040**	**0.8972**
Glass	0.5637	0.5268	0.5552	0.5548	0.5637	0.5182	0.5488	**0.5488**	0.5637	0.5013	**0.5438**	0.5440
Wine	0.8415	0.7880	**0.8215**	**0.8248**	0.8415	0.7292	**0.8043**	**0.7958**	0.8415	0.7362	**0.8140**	**0.8087**
						Support Vector Machine						
Iris	0.4682	0.3853	**0.4389**	**0.4556**	0.4682	0.3772	0.4124	**0.4220**	0.4682	0.3750	**0.4164**	**0.4226**
Glass	0.3422	0.3375	0.3403	0.3447	0.3422	0.4500	0.4000	0.3278	0.3422	0.3366	0.3399	0.3397
Wine	0.3905	0.3618	0.3776	0.3968	0.3905	0.3945	0.3923	0.3888	0.3905	0.3832	0.3904	0.3823

Table 3: Results for $\omega = 25\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

References

[Bentivegna *et al.*, 2004] Darrin C. Bentivegna, Christopher G. Atkeson, and Gordon Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[Cauwenberghs and Poggio, 2001] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *NIPS*, 2001.

[Diehl and Cauwenberghs, 2003] C. P. Diehl and Gert Cauwenberghs. SVM incremental learning, adaptation and optimization. In *NIPS*, volume 4, pages 2685–2690, 2003.

[Dinerstein *et al.*, 2007] Jonathan Dinerstein, Parris K. Egbert, and Dan Ventura. Learning policies for embodied virtual agents through demonstration. In *IJCAI*, pages 1257–1252, 2007.

[Fei-Fei *et al.*, 2007] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental Bayesian approach testing on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[Freund and Schapire, 1995] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Second European Conference on Computational Learning Theory*, pages 23–37, 1995.

[Furey *et al.*, 2000] Terrence S. Furey, Nello Cristianini, Nigel Duffy, David W. Bednarski, Michél Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[Gamon, 2004] Micheal Gamon. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In *International Conference on Computational Linguistics*, 2004.

[Gehrke *et al.*, 1999] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. BOAT — optimistic decision tree construction. In *International Conference on Management of Data*, volume 28, pages 169–180, 1999.

[Goldberg and Mataric, 2002] Dani Goldberg and Maja J Mataric. Maximizing reward in a non-stationary mobile robot environment. *Autonomous Agents and Multi-Agent Systems*, 6, 2002.

[Grollman and Ballard, 2012] Daniel H. Grollman and Aude G. Ballard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331 – 342, Nov 2012.

[Huang *et al.*, 2006] Jiayuan Huang, Alexander J. Smola, Arthur Gretton, Karsten M. Borgwardt, and Bernhard Schölkopf. Correcting sample bias by unlabeled data. In *NIPS*, 2006.

[Jereza *et al.*, 2010] José M. Jereza, Ignacio Molinab, Pedro J. García-Laencinac, Emilio Albad, Nuria Ribellesd, Miguel Martíne, and Leonardo Francoa. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial Intelligence in Medicine*, 50(2):105–115, 2010.

[Kalapanidas *et al.*, 2003] Elias Kalapanidas, Nikolaos Avouris, Marian Craciun, and Daniel Neagu. Machine learning algorithms: A study on noise sensitivity. In *First Balkan Conference in Informatics*, 2003.

[Kasper *et al.*, 2001] Michael Kasper, Gernot Fricke, Katja Steuernagel, and Ewald von Puttkamer. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems*, 34(2-3):153–164, 2001.

[Lakshminarayan *et al.*, 1996] Kamakshi Lakshminarayan, Steven A. Harp, Robert Goldman, and Tariq Samad. Imputation of missing data using machine learning techniques. In *KDD*, 1996.

[Ma *et al.*, 2003] Junshui Ma, James Theiler, and Simon Perkins. Accurate on-line support vector regression. *Neural Computation*, 15:2683–2703, 2003.

[Michalski *et al.*, 1986] Ryszard S Michalski, Igor Mozetic, Jiarong Hong, and Nada Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *AAAI*, 1986.

[Nakanishi *et al.*, 2004] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91, 2004.

[Nettleton *et al.*, 2010] David F Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33:275–306, 2010.

[Nicolescu and Mataric, 2002] Monica N. Nicolescu and Maja J. Mataric. A hierarchical architecture for behavior-based robots. In *AAMAS*, pages 227–233. ACM, 2002.

[Parker *et al.*, 2007] Charles Parker, Prasad Tadepalli, Weng-Keen Wong, Thomas Dietterich, and Alan Fern. Learning from demonstrations via structured prediction. In *AAAI*, 2007.

[Polikar *et al.*, 2001] Robi Polikar, Lalita Upda, Satish S. Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE SMC, Part C*, 31(4):497–508, Nov 2001.

[Ross *et al.*, 2008] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1–3):125–141, 2008.

[Syed *et al.*, 1999] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling concept drifts in incremental learning with support vector machines. In *KDD*, 1999.

[Veeraraghavan and Veloso, 2008] Harini Veeraraghavan and Manuela M. Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604. IEEE, 2008.