Similarity-based Approaches to Learning from Demonstration

Brandon Packard

Computer Science
Drexel University
Philadelphia, PA (USA)

Abstract. Learning from Demonstration (LfD) is the process of learning to accomplish a task by observing a demonstrator performing that task. My research so far has focused in three problems related to LfD: similarity measures for structured data, feature selection/construction for structured data, and active learning. A description of each area, what I have explored so far, and a brief statement on results are provided. My plans for future research work are also stated and briefly discussed.

1 Introduction

Learning from Demonstration (LfD) is very common in humans [7, 1]. Among other strategies, humans commonly look to a demonstrator for information on how to perform a task. For example, a human might learn to move a paddle in certain ways by observing a demonstrator making the desired motions. The field of LfD has been intensively studied over the past 30-40 years with the goal of replicating humans' ability to learn complex tasks from observation. Work in this area has been reported under the labels of *Learning from Observation*, *Imitation Learning*, *Behavioral Cloning*, *Apprenticeship Learning*, or *Programming by Demonstration*, which are largely synonymous with LfD. Although some of them make special emphasis in some specific approach to LfD, the overall goal is to learn to replicate behavior via symbolic, structured data (via observations) gathered from the performance of an demonstrator.

LfD has many uses in a wide variety of areas. Inside video games, LfD has the potential to allow game designers to design artificial intelligences by simply demonstrating the desired behavior, instead of having to program it. This same idea can be extended to programming computers in general, allowing for people to create programs which exhibit desired behaviors by giving examples of those behaviors – potentially allowing even those with no programming knowledge to write programs. LfD could also have uses in medicals fields as well, where robots employing LfD could observe surgeons performing surgeries with high mortality rates, and learn to perform that surgery with a much lower mortality rate.

My research is focused on using Learning from Demonstration (LfD) to train learning agents to emulate demonstrator behavior. There are two specific problems that I have been trying to address in my work. The first of these is analyzing

case-based approaches to LfD. More specifically, examining similarity measures for structured, symbolic data and feature selection methods for these case-based approaches. The second problem, and the one that I am currently focusing on, is that of active-learning strategies for LfD. Since LfD violates the i.i.d. assumption of standard supervised learning, standard supervised approaches are not suitable (they lead to quadratic error growth over time [5]) Therefore, I am investigating dataset collection algorithms which will address this issue.

The rest of this paper is organized as follows: Section 2 discusses the use of structured similarity measures in LfD, and Section 3 discusses feature construction and selection in LfD. Next, Section 4 discusses my current work in active learning and some basic preliminary results. Finally, Section 5 wraps up the paper with conclusions and then a discussion of my intended future work.

2 Structured Similarity Measures for LfD

Structured representations of data work well for LfD problems, since in many domains of interest it is more natural to represent the world state using these representations rather than a more traditional propositional feature-vector approach. For example, in Super Mario, one of our evaluation domains, there can be an arbitrary number of enemies surrounding the player character. With a feature vector approach, it is tough to represent the information pertaining to this changing number of enemies. By using a representation such as horn clauses [2], however, the world state is easy to encode regardless of the number of enemies. Cases are built from information from the current world state and past actions. Also, we assume cases provided by the demonstrator are good quality.

In nearest neighbor (and many other CBR approaches), one or more similarity methods are used to determine what world state from the case-base most closely matches the currently encountered case. For propositional data, this often means directly calculating the distance between the numbers using some mathematical formula, but for symbolic data different methods need to be employed. The similarity measures that I have experimented with are:

- *Jaccard*: Given two clauses (represented as sets of terms), we define the Jaccard similarity as the size of their intersection over the size of their union, which can be characterized by the following equation: $J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$, where $X \cap Y$ is a clause that contains only those predicates present both in X and in Y, and $X \cup Y$ is a clause that contains the union of predicates in X and in Y. $|\cdot|$ represents the number of predicates in a clause. Notice that for a predicate to be in the intersection, it must be a perfect match.
- *Levenshtein*: Given two clauses (which are unordered sets of terms), the Levenshtein distance counts the number of edit operations that we need to perform to one of the clauses, in order to convert it into the other. The *edit* operations that we consider are adding a value, removing a value, and substituting a value by another. However, true edit distance has an exponential time complexity, so a matrix approximation which has a polynomial cost was employed instead.

– *Weighted Levenshtein*: This measure is simply Levenshtein distance with weights on the values (for example, given 3 items in Minecraft (a wooden sword, an iron sword, and a block of dirt), Levensthein would consider a wooden sword no closer to a golden sword than it is to a dirt block, while Weighted Levenshtein would recognize that it is certainly more similar to the former than the latter.

These methods, along with two baselines (a random agent and an agent who just picks the overall most likely action from the case-base regardless of the current world state) and an Euclidean-distance based method that ran on a propositional version of the data were empirically investigated in the domain of Minecraft [4]. It was found that Weighted Levensthein yielded the overall best results, followed by Levensthein and Propositional being about the same, and then Jaccard Distance. It is also interesting to note that although all four methods performed significantly better than both baselines, there was no method that performed statistically significantly better than any of the other three methods.

3 Feature Selection

Many times, when a domain is being encoded into a structured representation, irrelevant data tends to get encoded together with what the important data – primarily because it is not always clear what is needed for the machine learning algorithm to perform well. This confuses machine learning algorithms and can result in less effective learning. Therefore, feature selection and construction can help extract what data is the most important, and therefore bolster learning.

Features can be roughly defined as the aspects of the data which are used to determine what case from the case-base is closest to the currently encountered case. In my research, I have examined various strategies for feature selection, most of which are based on the well known "wrapper" methods:

– Additive Wrappers: Wrappers which start with an empty subset of features and greedily select the best feature at each iteration.
– Subtractive Wrappers: Wrappers which start with a subset of features and greedily dispose of the features whose removal provides the best results
– Filter Methods: Methods which test each feature individually and gets rid of all but the top X of them (in my experiments, values of 10%, 20%, and 30% were used for X).
– Filtered Wrappers: Methods which first use a filter, and then perform an additive wrapper on the remaining set of features.
– Naive Sampling Feature Selection (NSFS): This is an alternative, wrapper inspired method which searches for the best feature subset based on Monte Carlo sampling, which uses Naïve Sampling [3].

All of these methods were empirically tested in the domains of Super Mario and Minecraft. It was found that all of these methods performed better than using all the features for both domains. The best (but not statistically significantly so) methods were subtractive wrappers, filtered wrappers, and NSFS.

These methods were also compared in terms of the number of calls to the similarity method which were required, wherein the filtered wrapper approach had by far the least number of calls, and the subtractive wrappers had by far the most, leading to the reasonable conclusion that filtered wrappers were overall the "best" method when accounting for both how well the feature selection performed and how long it took to execute. In terms of feature construction, tests performed in these domains with either type of feature construction increased results, but tests that included both types of feature construction performed significantly better than those which did not include any feature construction.

4 Active Learning for LfD

One of the major open problems in LfD is that when a learning agent moves out of the space for which it has demonstration data, the error starts to compound quadratically. One way to account for this is to attempt to give the learning agent demonstration data for situations in which it has none. Active learning is one technique for attempting to do so, by having the agent determine when it is stuck and then requesting more data from the demonstrator for that situation.

One of the cutting edge algorithms for this is DAgger [6]. DAgger is an iterative algorithm that works by having the demonstrator control 100% of the time for the first iteration, then exponentially less and less over future iterations, and stores the demonstrator's actions for each world state.

However, DAgger requires a lot of demonstrator interaction. In order to lessen how often the demonstrator needs to be consulted, I have come up with an alternative active LfD algorithm denoted as SALT. SALT is an iterative algorithm, like DAgger, but uses three different policies to control when and for how long the demonstrator is queried. Basically, SALT runs the demonstrator for one iteration to get a starting set of training data, and then lets the learner take over. The learner uses one of a set of policies to determine whether it has moved out of the state space for which is has demonstrations (that is to say, whether its current reward differs significantly from what the demonstrator's would have been) and calls the demonstrator for data when it is. This estimation of the reward the demonstrator would have obtained after a certain amount of time is estimated from analyzing past demonstrations. There is also a set of policies which determine how long the demonstrator should have control (for example, for a certain number of time steps or until the end of the trajectory), and another which determines how far the world state should be backed up (either a certain number of timesteps or until the start of the trajectory) before the demonstrator takes control. In addition, there are two flavors of SALT:

- $SALT_r$ (SALT with relabeling): This version of SALT has the demonstrator relabel the training data afterwards, so that at every time step we know what the demonstrator would have done (much like in DAgger).
- $SALT_n$ (SALT without relabeling): This version of SALT does not have the demonstrator relabel the training data afterwards, so only data from when the demonstrator is in control is added to the case-base.

So far, SALT has been compared to DAgger in the domain of Super Mario. Preliminary results seem to indicate that it trains agents which are not significantly worse than those DAgger trains, but requires much less demonstrator interaction than DAgger, making it more feasible for human demonstrators.

5 Conclusions and Future Work

In conclusion, my research work so far has focused around three key ideas. The first of these is analyzing various structural similarity measures and how well they perform on LfD tasks. The second key idea is that of feature selection and feature construction, or removing/adding features to the data in order to let the learner more effectively learn the demonstrator's behavior. Finally, active learning, or letting the learner decide when to ask for more training data from the demonstrator, has been my most recent area of focus.

In the future, I would like to keep exploring active learning and improving upon SALT to more effectively tackle the open LfD problem of leaving the state space for which the learner has demonstration data. In addition to this, I would like to extend my work to a second evaluation domain (a grid-based puzzle game known as "Thermometers") as well as a human demonstrator. Finally, I would like to explore the idea of weighting the data in the case-base based either on who was performing the task at the time or from what iteration of the algorithm they were added to the learner.

References

1. Heyes, C., Foster, C.: Motor learning by observation: Evidence from a serial reaction time task. The Quarterly Journal of Experimental Psychology: Section A 55(2), 593–607 (2002)
2. Nienhuys-Cheng, S.H., Wolf, R.d.: Foundations of Inductive Logic Programming. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)
3. Ontanón, S.: The combinatorial multi-armed bandit problem and its application to real-time strategy games. In: Ninth Artificial Intelligence and Interactive Digital Entertainment Conference (2013)
4. Packard, B., Ontañón, S.: Learning behavior from demonstration in minecraft via symbolic similarity measures. In: Proceedings of the Foundations of Digital Games conference (FDG) 2015 (2015)
5. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: International Conference on Artificial Intelligence and Statistics. pp. 661–668 (2010)
6. Ross, S., Gordon, G.J., Bagnell, J.A.: A reduction of imitation learning and structured prediction to no-regret online learning. arXiv preprint arXiv:1011.0686 (2010)
7. Schaal, S., et al.: Learning from demonstration. Advances in neural information processing systems pp. 1040–1046 (1997)