Clustering-Based Active Learning

Garvit Juniwal†
UC Berkeley
garvitjuniwal@eecs.berkeley.edu

Sakshi Jain†
UC Berkeley
sakshi.jain@eecs.berkeley.edu

***Abstract*—In contexts where obtaining labels for data points is expensive, active learning is a widely used methodology for deciding which data points to get a label for. In this project, we propose and evaluate an active learning algorithm in context of an automatic grading and feedback generation tool for an online embedded systems course currently under development at UC Berkeley. This tool learns a fault classifier given few examples of student solutions that have and do not have the fault (postive and negative examples). The data points in our case are multi-dimensional time series obtained from physics simulations. We cluster our time series data points using dynamic time warping distance measure. Our active learning algorithm uses this clustering to select the data points to obtain labels for. We show that the on an average, the active learning technique not only provides better accuracy than random sampling, but also uses fewer number of training instances than the naive random case.**

I. Introduction

Massive open online courses (MOOCs) [1] and related technological advances promise to bring world-class education to anyone with Internet access. Due to large number of students taking these courses *automatic grading* has become a challenge. In automatic grading, a computer program verifies that a candidate solution provided by a student is "correct", i.e., that it meets certain instructor-specified criteria (the specification). In addition, and particularly when the solution is incorrect, the automatic grader (henceforth, *auto-grader*) should provide feedback to the student as to where he/she went wrong.

The course under consideration is *Introduction to Embedded Systems* at UC Berkeley [2]. In this course, students not only learn theoretical content on formal modeling, design, and analysis [3], but also perform lab assignments on programming an embedded platform interfaced to a mobile robot [4]. The lab assignment involves implementing a control algorithm on a robot so that it can perform certain navigation tasks involving obstacle avoidance and hill climbing. In the MOOC version of the course offered this summer EECS 149.1x, students perform the same laboratory exercises but they are provided with a physics based simulation software designed by NI which can be used for debugging and testing solutions by running the robot in some representative environements/arenas.

Juniwal et. al. [5] formalize the auto-grading problem for the virtual lab assignments. For each lab assignment, there is an *end-to-end correctness property*, hereafter referred to as the *goal* property. If the goal is satisfied, the student solution is deemed correct. Otherwise, it is incorrect, and more analysis must be performed to identify the mistake (fault) and provide feedback. This latter analysis is based on monitoring simulation traces of the student controller on a library of known faults, also formalized in STL. If any of these "fault properties" hold for a student controller, they are provided to the student as feedback. Both goal and fault properties are formalized in signal temporal logic (STL) [6]. From hereon, we will only talk about fault properties as goal properties can be treated as faults by negation. These properties involve some numerical parameters that need to be tuned. The paper also describes an algorithm to perform parameter synthesis taking as input a few refernce student solutions that have (and do not have) the fault. We refer to this as *training* phase. For each fault, the synthesized *test benches* can then be used to label new student solutions as being faulty or not. We refer to this as *classification* phase.

One trouble with this approach is the availability of "good" and "bad" reference solutions. An instructor has to manually look at the simulation video to decide whether a particular solution is good or bad and then it can be used for training the test bench. In essence, labeling of solutions is an expensive process. In this report, we describe the problem of getting labeled data as an active learning problem. We have a database of unlabeled student solutions that was collected during an on-campus offering of the course. After each classification phase, we use a clustering-based technique to select the examples that we need the instructor to label.

† These authors contributed equally to this work.

To elaborate on why we think clustering is helpful, we beleive that amongst many student solutions, the total solution strategies are still few. And most solutions which follow the same strategy will have the same label. Hence, if some clustering technique can identify each strategy as a separate cluster, then choosing one example from each cluster should account for a training set with good coverage and the obtained classifier will have high accuracy.

As described before, classfiying a solution w.r.t. a fault property involves first obtaining a simulation trace and then monitoring an STL formula on the trace. Hence, to cluster solutions w.r.t. a fault, we use the traces obtained by simulation in the corresponding environment. Traces are timed sequences of data like the position and orientation of the robot and values of the on-board sensors. To cluster timed series data, we compute *dynamic time warping* distance between every pair of traces and run density based clustering on the distance matrix (DBSCAN). We compare the accuracy of classifers obtained from training examples selected randomly and using clustering.

II. Technique

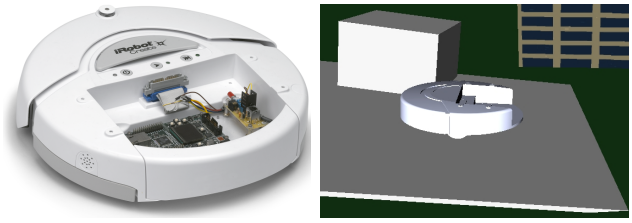
In this section we describe various aspects of an active learning algorithm that we use in our context.

A. Lab Assignment Details

The embedded systems laboratory course offered at the University of California, Berkeley employs a custom mobile robotic platform called the Cal Climber[7], [8]. This off-the-shelf platform is capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external controller. The problem statement is as follows:

> *Design a Statechart to drive the Cal Climber. On level ground, your robot should drive straight. When an obstacle is encountered, such as a cliff or an object, your robot should navigate around the object and continue in its original orientation. Use the bump and cliff sensors for this purpose. On an incline, your robot should navigate uphill, while still avoiding obstacles. Use the accelerometer to detect an incline and as input to a control algorithm that maintains uphill orientation.*

For debugging, students use the LabVIEW Robotics Environment simulator which is based on the Open Dynamics Engine [9] rigid body dynamics software that



(a) (b)

Fig. 1: (a) iRobot Climber laboratory platform. (b) iRobot Climber in the LabVIEW Robotics Environment Simulator

can simulate robots in a virtual environment, which, in our case, will be composed of obstacles and hills.

B. Environments and Traces

For the purposes of auto-grading, for each fault, we define a test bench Γ that consists of an environment E and an STL based classifer φ. A student solution C is simulated in the environement E to obtained a simluation trace $\mathsf{sim}(C, E)$, which is a multi-dimensional timed sequence consisting of variables such as postion and orienation co-ordiantes pos and angle, sensor signals such as bump and cliff and robot output such as the left and right wheel speeds lws and rws. The STL formula φ is checked on the simulation trace and C is labeled as faulty if the formula is satisfied.

C. Training and Classification

The test benches have some associated parameters (both in the environment and the STL formula). These parameters need to be tuned appropritely so that the test bench identifies the corresponding fault correctly. Juniwal et. al. [5] describe an algorithm for synthesis of parameter domains for test benches using positive and negatively labeled refernce solutions. We use this as a black box *training* module Train. A synthesized test bench then used by a *classification* module Classify that can label new solutions as being faulty or not.

Generating labeled data for the training module is expensive. An instructor would have to manually look at the simulation video to determine whether solution is faulty or not. This is the problem we tackle here. How can we make generation of training example easier and more efficient?

D. Active Learning

Active learning is a form of machine learning where the learning algorithm is able to interactively query the user to get the correct labels at new data points. Our problem fits well within this definition. We extend

the training module with another *selection* module that decides which new data point to get a correct label for. The overall active learning procedure is iterative. See Algorithm 1. The training module first generates a classifier based on some training data. Depending on the results of the classifier, the data points labeled as 0 and 1 are seprately clustered by a clustering module CLUSTER. Using the clusters formed, the selection module SELECT selects new data points to get correct labels for. All the selected data points that were incorrectly labeled by the classifier are now added to the training set and the classifier is trained again. This continues no more training data is added. Details of the clustering algorithm and the selection module are given in later sections.

Algorithm 1: ACTIVELEARNING (AL)

Input: A dataset of student solutions $\mathcal{D}$, a true labeling oracle $\mathcal{O}$, an environment E and an STL formula φ with untuned paramaters

Output: A classifier Γ for $\mathcal{D}$

$\text{TS} \leftarrow \emptyset$
while TS *keeps getting larger* **do**
  $\Gamma \leftarrow \textsc{Train}(E, \varphi, \text{TS})$
  $\mathcal{D}_0, \mathcal{D}_1 \leftarrow \textsc{Classify}(\Gamma, \mathcal{D})$
  $\theta_0, \theta_1 \leftarrow \textsc{Cluster}(\mathcal{D}_0), \textsc{Cluster}(\mathcal{D}_1)$
  $\mathcal{R}_0, \mathcal{R}_1 \leftarrow \textsc{Select}(\theta_0), \textsc{Select}(\theta_1)$
  $\text{TS} = \text{TS} \cup \{\mathsf{d} \text{ s.t. } (\mathsf{d} \in \mathcal{R}_0 \wedge \mathcal{O}(\mathsf{d}) = 1) \vee (\mathsf{d} \in \mathcal{R}_1 \wedge \mathcal{O}(\mathsf{d}) = 0)\}$
end
return Γ

E. Clustering with Precomputed Distances

Dynamic Time Warping Distance. For each test bench, we need to compute the pairwise distances between all solutions in the database that will be used by the clustering algorithm. Since the test benches are based on monitoring traces obtained by simulation in a certain environment, we also compute distance between two solutions by computing the distance between the corresponding traces. For each fault, we first identify the variables (amongst pos, angle, bump, cliff, lws, rws) that appear in the STL formula and project out the rest. Then, we compute the Dynamic Time Warping distance between the multidimensional time series [10]. Dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences (multidimensional in our cae) which may vary in time or speed. Similarities in student solutions could be detected using DTW, even if one bot was moving faster than the other, or if there were accelerations and decelerations during the trace. For our analysis, we use Euclidean distance as the pointwise distance measure.

For our clustering module, we use Density-based spatial clustering of applications with noise or DBSCAN [11]. This clustering technique clusters the samples based on provided estimation of density of the cluster nodes. It can take as input pre-computed distances between samples (an important factor while choosing this method) and does not require the number of clusters to be specified. We optimize the two input parameters ϵ and *min_points*[1] for each of the fault analysis presented in the report.

F. Selection Module

The selection module implements the policy used for selecting data points to be added to the training set. This is done bearing in mind that the training algorithm works well if the training data is balanced in terms of number of positive and negative examples. Training data balancing is a standard technique in machine learning [12].

For initialization of the training set during the first iteration, we cluster all the samples using the clustering module. We then select a randomly chosen sample from each cluster, look up its label and add to the training set. If the number of samples for positive and negative training is skewed, we continue picking more training instances either until a threshold upper bound is reached or we are unable to reduce the skew any further. To reduce the skew, we randomly pick a cluster from which a minority instance was obtained (positive or negative) and sample with again hoping to obtain another instance of the minority class thereby reducing the skew.

Once the initialization step is complete, we move on running the classifier and obtaining predicted labels on the test set. If the accuracy on the test set is not 100%, we try and improve our training set by adding examples of samples which were marked wrongly. In order to achieve that, we re-cluster all the samples (test and training) in each class separately, randomly pick a cluster which has not been already represented in the training set (i.e. the cluster and the training set has no sample in common) and pick a random sample from the same. We do this for both class and add the respective sample to the training set if the predicted label was not same as the actual label. This step is performed in a loop until we are unable to

[1]please refer to [11] for further details on these parameters

increase the size of the training set or 100% accuracy has been achieved.

III. Evaluation

To evaluate the active learning technique developed in this report, we compare our technique *AL* against the technique *Random* where we randomly choose our training set. We evaluate the two techniques based on overall accuracy achieved, the size of training set used and the balance of training labels. To simply the comparison, we set the upper bound on the number of training instances used in *AL* and total number of randomly chosen samples in *Random* as 30.

The measure *M* we use for measuring accuracy for each fault is, product of the true positive and true negative rate obtained by the technique under consideration.[2]. Analysis results for the 7 distinct faults are shown in Table I. From the table, it can be seen that *M* is individually higher in case of *AL* than *Random* for all the faults except *avoid_left*. While the aggregated value of *M* for *Random* is ~ 0.48 it is higher and hence better for *AL*, ~ 0.77.

Next, we compare the total number of training examples for the two cases, *Random* and *AL*. *AL* on an average used approximately 22 training examples while *Random* used exactly 30 examples for each case, i.e. 26% drop in the size of the training set for better accuracy. In order to evaluate how well balanced are the training sets obtained using the two techniques, we use *balancing ratio* i.e the ratio of number of negative training examples and number of positive training examples. The closer this value is to 1, the better balanced are the two training sets. Table I gives a clear break down of the number of positive and negative training examples used for each fault per technique. In our evaluation, we find that this ratio was *∼4.3* for *Random* while it was *∼1.2* for *AL*, *AL* clearly outperforming *Random* in this case.

Since *AL* on an average performs better than *Random* on both accuracy measure and balancing measure, we believe that *AL* is a better choice for creating smaller yet more effective training set than random sampling.

IV. Conclusions

In this project, we propose and evaluate an active learning technique to minimize the cost of labeling data points for automatic grading of student solutions and feedback generation for an online embedded systems course. We evaluate our technique with the currently used random sampling technique based on accuracy and the size of training set used. We show that on an average, one can obtain better fault identification accuracy with active learning while using approximately 26% less number of training samples. Thus, we believe that our technique provides a more effective way and cost efficient way to identify faults and hence is a better choice for a training module.

[2]This measure is specifically insightful for our current application auto-grading, because the classifier is inherently lenient and a better classifier would be the one that can identify atleast a few cases on existence of faults in the solutions

Appendix

A. Clustering based on mixtures of HMMs

We present the following alternative clustering strategy based on mixtures of HMMs for theoretical interest. We did not evaluate this approach because the rudimentary implementation of Baum-Welsch in MATLAB takes too long to train on each trace of 1000 elements (10 mins with 5 clusters and 5 states per cluster) and we need to cluster 300 traces on an average for each fault.

Each timed trace can be thought of as being generated from a a mixture of HMMs. Each HMM in the mixture can be thought of as corresponding to a high level strategy in of the control algorithm implemented on the robot that generates the trace. From the of timed traces, we could train this mixture of HMMs.

Assume that the mixture consistes of k components with i-th component having initial state probabilities λ_i, emmision probabilities B_i and transition probabilities A_i. The mixing proportions are given by $p = (p_1, \cdots, p_k)$. The naive approach that comes to mind to train this model is EM based. We could infer p and (λ_i, A_i, B_i) in two phases and iterate. But we make the following observation which leads to a simpler approach.

Note that a mixture of HMMs can be viewed as a single HMM with a state space of size $n = n_1 + n_2 + \cdots + n_k$, where n_i is the size of the state space of the i-th model. The transition probability matrix of this composite HMM is A = diag$(A_1, \cdots, A_k)$, the observation matrix is B = diag$(B_1, \cdots, B_k)$, whilst the initial state distribution π is given by $(p_1\pi_1^T, \cdots, p_k\pi_k^T)^T$. Hence, in theory, the Baum-Welch algorithm can be used to find parameters of the mixture models if the models if the model parameters are initialized to respect the structure of this composite HMM model, i.e., by fixing a_{ij} to zero whenever i and j do not belong to the same submodel.

References

[1] *The Year of the MOOC*, New York Times, November 2012. [Online]. Available:

Fault Name	Training size		True Pos		True Neg		*M*	
	Random	*AL*	Random	*AL*	Random	*AL*	Random	*AL*
1. avoid_front	23/133 + 7/74	15/133 + 15/74	133/133	133/133	67/74	70/74	0.91	0.95
2. avoid_left	23/164 + 7/45	15/164 + 15/45	164/164	164/164	42/45	39/45	0.93	0.87
3. circle_old	1/7 + 29/200	3/7 + 11/200	0/7	6/7	200/200	193/200	0.00	0.83
4. default_env	30/486 + 0/5	4/486 + 4/5	486/486	486/486	0/5	4/5	0.00	0.80
5. hill_climb_back	26/427 + 4/63	14/427 + 16/63	427/427	427/427	55/63	60/63	0.88	0.95
6. hill_climb_right	28/442 + 2/48	29/442 + 1/48	442/442	442/442	11/48	18/48	0.23	0.38
7. avoid_right	24/169 + 6/40	10/169 + 3/40	70/169	169/169	40/40	26/40	0.41	0.65

TABLE I: Comparison of total number of training examples, true positive rate, true negative rate and accuracy measure *M* for the two techniques, Random and *AL*

http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html

[2] E. A. Lee and S. A. Seshia. EECS 149 course website. University of California, Berkeley. [Online]. Available: http://chess.eecs.berkeley.edu/eecs149

[3] ——, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Berkeley, CA: LeeSeshia.org, 2011. [Online]. Available: http://LeeSeshia.org

[4] J. C. Jensen, E. A. Lee, and S. A. Seshia, *An Introductory Lab in Embedded and Cyber-Physical Systems*. Berkeley, CA: LeeSeshia.org, 2012. [Online]. Available: http://LeeSeshia.org/lab

[5] G. Juniwal, A. Donzé, J. C. Jensen, and S. A. Seshia, "Cps-grader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory," in *Proceedings of the 14th International Conference on Embedded Software (EMSOFT)*, October 2014.

[6] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *FORMATS/FTRTFT*, 2004, pp. 152–166.

[7] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *First IEEE Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy)*, Istanbul, Turkey, 2011. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/837.html

[8] J. C. Jensen, "Elements of model-based design," Master's thesis, University of California, Berkeley, February 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-19.html

[9] R. Smith. Open dynamics engine. [Online]. Available: http://ode.org

[10] T. Giorgino, "Computing and visualizing dynamic time warping alignments in r: The dtw package," *Journal of Statistical Software*, vol. 31, no. 7, pp. 1–24, 8 2009. [Online]. Available: http://www.jstatsoft.org/v31/i07

[11] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998. [Online]. Available: http://dx.doi.org/10.1023/A%3A1009745219419

[12] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explorations*, vol. 6, no. 1, pp. 20–29, 2004.