A Stitch in Time - Autonomous Model Management via Reinforcement Learning

Elad Liebman
The University of Texas at Austin
eladlieb@cs.utexas.edu

Eric Zavesky
AT&T Research
ezavesky@research.att.com

Peter Stone
The University of Texas at Austin
pstone@cs.utexas.edu

ABSTRACT

Concept drift - a change, either sudden or gradual, in the underlying properties of data - is one of the most prevalent challenges to maintaining high-performing learned models over time in autonomous systems. In the face of concept drift, one can hope that the old model is sufficiently representative of the new data despite the concept drift, one can discard the old data and retrain a new model with (often limited) new data, or one can use transfer learning methods to combine the old data with the new to create an updated model. Which of these three options is chosen affects not only near-term decisions, but also future needs to transfer or retrain. In this paper, we thus model response to concept drift as a sequential decision making problem and formally frame it as a Markov Decision Process. Our reinforcement learning approach to the problem shows promising results on one synthetic and two real-world datasets.

KEYWORDS

Reinforcement Learning; Model Retraining; Concept Drift

ACM Reference Format:
Elad Liebman, Eric Zavesky, and Peter Stone. 2018. A Stitch in Time - Autonomous Model Management via Reinforcement Learning. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

As automation grows, more and more industry control systems around us make decisions autonomously, ranging from image understanding [19] to movie recommendation systems [9] and network and service virtualization [3]. Underneath their hoods, many of these systems rely on models. These models can be descriptive (capturing the properties of data) or predictive (using known data to predict other, latent properties). However, such systems are often susceptible to the nonstationary, ever-changing dynamics of data in the real world. In recommendation systems, tastes and fashion change. In climate prediction, the properties of the environment change constantly. These changes, either gradual or abrupt, are commonly referred to as *concept drift*. Such shifts in the feature distribution and underlying label correspondence constitute a significant challenge to learning systems. In the face of concept drift, we consider the problem of how to generically and adaptively adjust models to mitigate the risks of drift. We refer to this challenge as *model retraining*, or model management. In this paper we propose a novel *reinforcement learning* (RL) approach, framing the model retraining problem as a sequential decision making task, and harnessing ideas from the RL literature to learn a robust policy for model update.

This paper makes two main contributions. First, we formally frame the autonomous model retraining problem as a reinforcement learning task. To our knowledge, we are the first to consider the benefits of this approach. Second, we propose a robust and general framework for learning an autonomous model adaptation policy that balances learning costs with overall performance and is completely decoupled from the underlying models. We show empirically that this approach is superior to other baseline meta-learning policies, including two competitive drift detection methods (DDM[7] and HDDM[6]). We also show how our approach can be scaled to more complex, multidimensional input, using deep neural nets.

2 RELATED WORK

The issue of concept drift has been the focus of many works in the past 20 years [8, 25–27]. Widmer and Kubat proposed a framework for continually deciding which samples to add and which to throw away (or "forget") as new data comes in [26]. Klinkenberg and Joachims presented a windowed support vector machine formulation meant to countermand drift [13], whereas Gamma et al. started a thread of detecting drift by considering the error rate over time [7]. More current examples include the work of Brzezinski and Stefanowski, who actively update an ensemble of classifiers weighted by their current accuracy, and combine them using Hoeffding trees [2], and HDDM, a method proposed by Frias-Blanco et al., that uses Hoeffding bounds to identify whether drift has occurred [6]. In another related work, Minku and Yao proposed a diversity based approach for adapting an ensemble to a drifting data stream[15]. These works are also connected to the large subfield of online learning [12]. The model retraining problem is also related to the notion of continual, or lifelong learning, both in the context of general machine learning [18], and in RL [17, 20]. However, lifelong learning is not the same as drift adaptation, since in the case of learning under drift, the learner is engaged in the same ongoing task, whereas in lifelong learning, the learner is expected to adapt to new tasks presented sequentially. Both lifelong learning, and our proposed task of autonomous model management can also be perceived as part of the transfer learning literature, a rich problem domain studied extensively both in the context of RL [23], and machine learning in general [16, 24]. A key difference between these methods and this paper is that we do not focus on the specifics of the underlying models or even the data itself. Instead, we propose an RL meta-learner that decides when and how the model should be updated, thus enabling the models themselves to be simple and generic. Previous methods for drift detection such as DDM and

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

HDDM (which we compare against) focus on the error rate, which contains less information than the core distributional properties of the data that our method uses. In addition, rather than trying to adaptively find the optimal decision threshold given current information, methods such as DDM employs hard-coded parameters, which are harder to tune appropriately.

3 MODEL RETRAINING AS A MARKOV DECISION PROCESS

Updating the current model of a given system affects not only the ability to act upon the data currently observed, but data observed in the future as well. Update a model too quickly and you may keep getting sidetracked by outlier occurrences, or waste resources on needlessly retraining too frequently. Wait too long to update, and the performance of your system might deteriorate drastically. Given this property, it makes sense to frame autonomous model management as a sequential decision-making task. As such, this problem is suitably formulated as a Markov Decision Process (MDP) [21]. In this formulation, the system is an agent, which, given the current model, observations of new data, and knowledge of past data, needs to routinely decide whether to update its model, and in what fashion.

Let us first consider a concrete example. Suppose you are managing a recommender system with a learned model that maps individual profiles to song recommendations. Song requests are received from multiple individuals over time. For quality assurance purposes, requests are grouped by hours. To decide whether the model is still useful, every k hours, the system aggregates all the songs played in a predetermined time window (say, a single hour), along with people's rating of the songs recommended to them (such that the aggregated information is supervised). Given this supervised sample of the data, how can the system decide best whether its model is up to date or not, and if not, how it should be updated?

Accordingly, in our formulation, incoming data modeled by the system can be divided into batches of varying size (most typically, representing some time window e.g. days or hours). For each batch, the system needs to decide how to best process the data. In the process, the agent needs to balance performance and cost (as they are determined for a given domain). We assume that it is infeasible for an agent to observe the entire batch before making a decision on how to best adapt the model. For this reason, the agent relies on subsampling the data before making a decision. We assume this subsample is unbiased. After observing the subsample, the agent then decides how to best update the model prior to handling the entire set of samples in the batch. This process is repeated in the next timestep indefinitely.

In the next two subsections, we first formally recap what a Markov Decision Process is, then proceed to describe the model retraining problem as an MDP.

3.1 Markov Decision Processes

An MDP M is represented as $\langle S, A, P, R\rangle$ where S is the set of states an agent can be in, A is the set of actions that the agent can take at a state, P is the transition function that gives the transition probability of reaching a particular next state s' from state s after taking action a ($P : S \times A \times S \rightarrow \mathbb{R}$; $\sum_{s'} P(s, a, s') = 1$), and R is the reward received given a transition ($R : S \times A \times S \rightarrow \mathbb{R}$). In continuing (non-episodic) MDPs it is often customary to specify a discount factor $\gamma \in [0, 1]$, which specifies the uncertainty (and therefore diminishing importance) of future rewards compared to rewards observed right now. To perform optimally in a task that an MDP represents, an agent must find a policy $\pi^* : S \rightarrow A$ such that executing action $\pi^*(s)$ from any given state s would yield the highest expected sum of rewards over the sequence of states and actions $\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$. This is traditionally referred to as "solving" an MDP.

3.2 Formulation of the Model Retraining Problem

Formulated as a Markov Decision Process, the general structure of the model retraining problem is the same, regardless of domain or model purpose. In this subsection we describe how model retraining reduces to a Markov Decision Process.

The state is defined as a tuple $\langle M_t, obs_t\rangle$, with M_t the current model and obs_t the new observed data at time t. Data at time t contains multiple instances, and we refer to it as a *batch*. Batch sizes may vary from one timestep to the next. Furthermore, in this paper we assume not all batch data obs_t can be practically observed in real time to make a decision. Instead, some realistically observable and unbiased subsample of it, $\widehat{obs}_t$, is used by the agent, making the MDP partially observable. As $|\widehat{obs}_t| \rightarrow |obs_t|$, the properties of the sampled data converge to the exact values of the true world data at time t and the MDP becomes fully observable. Therefore, the state space S is defined as the Cartesian product of the model and observation spaces.

At each time step, the agent has the choice of either *retraining* a model from scratch based on the new data, *adapting* (or transfering) the current model to the new data, or leaving the current model as is. This defines an action space of three possible actions, $a \in \{RETRAIN, ADAPT, KEEP\}$. For convenience of notation, we define these actions as functions of the current state, $a : M \times OBS \rightarrow M, a_t(\langle M_t, obs_t\rangle) \rightarrow M_{t+1}$, where M is the model space and OBS is the observation space.

S and A induce a stochastic transition function P. Specifically, we can use the following shorthand notation:

$$P(\langle M_t, obs_t\rangle, a_t) = \langle a_t(\langle M_t, obs_t\rangle), obs_{t+1}\rangle$$

Note that stochasticity originates in the fact that we do not know what observations the agent will observe next, nor do we know how well the data sample corresponds to the actual data the model needs to process at time t (though we assume the sample is unbiased).

As a reward function, we factor in the utility of correctly handling observed data vs. the cost of each action. A predefined utility *acc_reward* is associated with every single observation $o \in obs$ that is accurately modeled at time t.

Furthermore, there is an inherent cost associated with each action. While keeping an existing model costs nothing, adapting and retraining are both associated with a computational penalty per processed sample. The resultant reward function takes the form of $r_t = |O_{modeled}| \cdot acc\_reward - action\_cost$, where $O_{modeled}$ is the set of all samples in the batch correctly handled by the model at time t, where the specific notion of "correctly handled" changes from one domain to the next. Note that this formulation satisfies the Markov property, as the reward and transition functions are entirely dependent on the current state and the action taken.

The cost of each action represents the computational cost of training a model, as well as the cost of collecting and annotating data. For instance, assume, in our music recommender system example previously discussed, that while the sample obtained every k hours is sufficient to determine whether the model is still useful, it is not enough to train a sufficiently reliable model for future timesteps, and therefore additional samples need to be obtained. In the case of transfer, fewer samples are needed, and the computational cost of updating the model is lower than in the case that a new model is trained from scratch. There is also the opportunity cost of potential system downtime while models are being updated. In such a case, it is less expensive to train a new model from scratch during a "slow" timestep when there aren't that many observations to handle, compared to a very busy timestep when the number of handled observations is very high. The specifics of such cost considerations are determined specifically for each environment, but the framework is sufficiently flexible to represent a wide range of such considerations.

To complete the MDP formulation, we also specify a discount factor $\gamma \in [0, 1]$, a constant which reflects how future costs and utilities should count towards present estimates. Given that the agent maximizes the sum of discounted rewards $\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$, setting γ to 0 only weights the immediate reward at each timestep, whereas setting γ to 1 gives equal weight to rewards obtained now or at any time in the future.

In the following section, we discuss how this formulation lends itself to a reinforcement-learning based approach to finding an optimal policy for model management that is domain agnostic in structure.

4 LEARNING A POLICY THROUGH APPROXIMATE VALUE ITERATION

Once the model retraining problem is formally defined as an MDP, a suitable policy needs to be learned for this MDP. As mentioned in Section 3, in certain cases the MDP can be explicitly "solved", resulting in the optimal policy. If the state and action spaces are finite and sufficiently small, this can be done through a dynamic programming procedure called *value iteration* [21]. However, in the case of continuous state spaces this option is no longer feasible. A variety of approximation strategies exist to mitigate this problem. Consider for instance the simplest approach of partitioning a continuous state space via a grid, then applying value iteration on the discretized state space (now rendered discrete and finite). As further discussed in Section 4.1, if the state space is Lipschitz-continuous [4], the error induced by this approximation can be bounded, and converges to 0 as the grid becomes tighter.

Let us assume our model can be parameterized as $M = \langle m_1, \ldots, m_k \rangle$. Given a reasonable model for observation distribution, it too can be parameterized, either by its sufficient statistics or using a non-parametric representation $obs_t = \langle d_1, \ldots, d_l \rangle$ (examples for such a representation include the weights of an artificial neural net or the coefficients of a regressor). This parameterization lends itself to a straightforward value iteration procedure based on a discretized representation of the parameter space.

For simplicity of notation, let us consider the following shorthand $[M - C, M + C]_x$ as representing a grid of width $2C$ around current model parameters M with stepsize x. For instance, for a simple two-dimensional model $M = \langle m_1, m_2 \rangle$, $[M - C, M + C]_x = M_1 \times M_2 = \{ (a, b) \mid a \in M_1 \text{ and } b \in M_2 \}$, where $C = \langle c_1, c_2 \rangle$, $M_1 = \{m_1 - c_1, m_1 - c_1 + x, m_1 - c_1 + 2x, \ldots, m_1 + c_1 - x, m_1 + c_1\}$ and $M_2 = \{m_2 - c_2, m_2 - c_2 + x, m_2 - c_2 + 2x, \ldots, m_2 + c_2 - x, m_2 + c_2\}$.

By discretizing both the model and observation spaces we have produced a discrete (and therefore finite) representation of the state space as the Cartesian product of $\mathcal{M} \times \text{OBS}$ where $\mathcal{M} = [M - C_m, M + C_m]_{x_m}$, and $\text{OBS} = [\text{obs} - C_{\text{obs}}, \text{obs} + C_{\text{obs}}]_{x_{\text{obs}}}$. This finite state representation enables us to perform the dynamic programming procedure of value iteration. Intuitively, the result of applying this procedure is the answer to the question of "if my current model parameters are M and the observation data is drawn from obs, should I RETRAIN, ADAPT, or KEEP?". Lastly we note that the expected reward is given by $\int P_s(x) \cdot M_s(x) dx$ where $M_s(x)$ specifies whether in state s the model M (which is part of the state) correctly services request x. An intuitive strategy for deciding on the discretization range is to set C values to be a factor of the estimated observation covariance Σ. For observation distribution parameters, under certain smoothness assumptions, modeling the deviation range as a multiplier of the variance manifested in the data makes it very likely that your learned policy will be well-defined for the next batch of observations.

Intuitively, the policy learned by the approximate value iteration (AVI) process, given a current model and an estimate of the data distribution, specifies deviation ranges in which keeping the current model is best, deviation ranges where it is best to do model transfer, and ranges where it is best to learn a new model entirely. The challenge (and power) of this approach lies in finding the near-optimal cutoffs for these decisions at each timestep in an adaptive fashion. As we later show, relying on a fixed rule of thumb to accomplish the same does worse than this value iteration approach. In the following subsection we briefly discuss how far off from optimal this policy can be.

4.1 Theoretical Intuition

If we were able to use the unapproximated value iteration procedure (for instance, in a case where the state and action spaces are sufficiently small), and to fully observe the data at each step, the resulting policy would be optimal as a guaranteed property of the value iteration algorithm. However, given that both the estimate and the approximation are imperfect, error in the resultant policy comes from two sources: (1) the sampling error in the estimate of the current distribution; and (2) the state-action approximation error. Item (1) is manageable due to the fact that as $|\widehat{obs}|$ increases, the statistical estimates converge to the true values of OBS. Furthermore, it is reasonable to assume that in many cases $\widehat{obs}$ can be an unbiased sample, meaning the expectation over the bias is 0 [5]. As for Item (2), it has been shown that if the approximated space is a non-expansion then the approximation error with respect to the optimal policy is bound as well [10]. Specifically, if the space is Lipschitz-continuous, which is a reasonable assumption for many data domains, including the ones we examine in this paper, then the error can be bound by $\frac{h}{1-\gamma}(K_1 + \gamma K_2 ||V^*||_Q)$ where the $||V^*||_Q$ quasi-norm is the span of the value function, $||V||_Q = sup_s V(s) - inf_s V(s)$, h is the grid width, γ is the discount factor, and K_1, K_2 are constants [4].

5 DISTRIBUTION MODEL RETRAINING

In the previous Section we discussed the abstract formulation of the model management problem as an MDP. In this Section, we discuss autonomous model management in the concrete case where the model needs to capture statistical properties of observed data.

One type of model maintenance problem that has multiple uses and serves as a good illustration for a framework is that of distribution tracking. Given a constant stream of multidimensional data, we need to generate a model that tracks the properties of the observed data. In this setting, we are interested in modeling the observed data as a multivariate Gaussian, with (μ, Σ) as the sufficient statistics. We consider observations in the current batch $o \in obs_t$ as modeling success cases, or "hits", if they are within a certain factor of our estimated μ. Similarly, we consider it a modeling failure, or a "miss", if an observation is outside that confidence range. At each timestep, the agent is presented with a set of observations obs_t, drawn from some unknown distribution. The agent is then allowed to sample some subset $\widehat{obs}_t$ of the observations, and decide whether it needs to update the model, and how. As described in the previous section, in this paper we consider three options at each timestep - keep the model unchanged, retrain a new model from scratch based on the new observations, or use transfer learning to update the old model with the new model. In the next Subsection we show how this setting is formally adapted as a Markov Decision Process.

5.1 MDP representation for the Distribution Tracking Problem

Following the formulation presented in subsection 3.2, we define the state as a tuple $\langle M_t = \langle \mu_t, \Sigma_t \rangle, obs_t \rangle$, with M_t and obs_t the model and the new observed data at time t. In each timestep the agent has the choice of either *retraining* a model from scratch based on the new data, *adapting* (or transfering) the current model to the new data, or leaving the current model as is. In the distribution tracking setting, updating the model from scratch simply means taking the maximum likelihood estimate of μ and Σ based on the sampled observations at time t, $\widehat{obs}_t$. In this paper, adapting the model means taking the average between the current model and the new model estimates $\mu_{t+1} = (\mu_t + \overline{\widehat{obs}_t})/2$, $\Sigma_{t+1} = (\Sigma_t + var(\widehat{obs}_t))/2$.

The transition function $P(\langle M_t, obs_t \rangle, a_t) = \langle a_t(\langle M_t, obs_t \rangle), obs_{t+1} \rangle$ is defined concretely as:

$$a_t(\langle M_t, obs_t \rangle) = \begin{cases} = \langle M_t, obs_{t+1} \rangle \text{ if } a_t = \text{KEEP} \\ = \langle \langle (\mu_t + \overline{\widehat{obs}_t})/2, \Sigma_t + \\ var(\widehat{obs}_t))/2 \rangle, obs_{t+1} \rangle \text{ if } a_t = \text{ADAPT} \\ = \langle \langle \overline{\widehat{obs}_t}), var(\widehat{obs}_t) \rangle, obs_{t+1} \rangle \\ \text{ if } a_t = \text{RETRAIN} \end{cases} \quad (1)$$

Finally, the reward function is formulated as $r_t = |O_{modeled}| \cdot acc\_reward - action\_cost$. An observation is successfully modeled if $o \in [(1-\delta)\mu, (1+\delta)\mu]$, where δ is the slack or accuracy factor. This concept is illustrated by example in Figure 1. We note that given a current model and an assumed distribution, the expectation of the reward can be computed analytically, a property we use in the value iteration process. The action cost is determined according to the concrete domain of application, and needs to reflect both the computational increased incurred by the sample size and the difficulty of obtaining samples.

5.2 AVI for Distribution Model Retraining

In this section we concretely adapt the abstract value iteration approach proposed in Section 4 to the distribution tracking problem, based on the MDP formulation described above (Section 5.1).

Recall that in the distribution tracking case, the model tracks the mean and the variance of an observed data stream as a multivariate Gaussian. For simplicity, let us assume that we are only interested in tracking the mean, that the data is one-dimensional, and that our current estimate of the mean is μ. Now we observe a sample set from the next data batch, $\widehat{obs}$. If we assume that new observations can only deviate by a certain range $C = C_{\text{obs}} \cdot \text{var}(\widehat{\text{obs}})$ from the current mean, we can assume a putative range for the expected mean of future observation batches, $\overline{\widehat{obs}} \in [\mu - C, \mu + C]$. Since the observations represent the same distribution we are trying to model, C may be used for both model and observation space discretization.

As described in Section 3, we discretize the range $[\mu - C, \mu + C]_x$ and perform value iteration on the grid values, with all the properties known. At every step of the dynamic programming process, we are essentially asking the following: if I observe k samples drawn from a distribution with mean a and variance b, and my current model is with mean c, what is the expected reward? Assuming $\mu_{t+1} \sim \mathbf{N}(\mu_t, var_t)$ and var_{t+1} remains the same, this

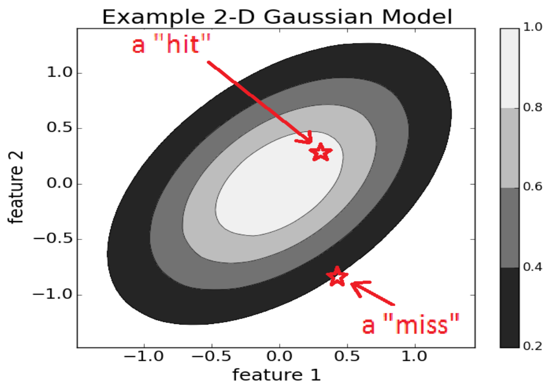

Figure 1: An example 2-dimensional model for a data distribution. A new observation that is sufficiently "close" to the mean estimate (as determined by a parameter) is considered a success (or a "hit"), whereas an observation that is not is a failure (or "miss"). X and Y axes are in 2-dimensional feature space, color represents estimated distribution frequency.

Algorithm 1 Distribution Tracking Approximate Value Iteration

```
1: Input: observation sample $\widehat{\text{obs}}$, current model mean $\mu$, current variance estimate $\Sigma$, variation fraction $C$, discretizing factor $k$, discount factor $\gamma$.
2: Set $C = C_{\text{obs}} \cdot \text{var}(\widehat{\text{obs}})$
3: Discretize range $\widehat{M} = [\mu - C, \mu + C]_k$, yielding an approximate state space $\mathcal{S} = \widehat{M} \times \widehat{M} = \{\langle m, \overline{obs} \rangle \mid m, \overline{obs} \in \widehat{M}\rangle$.
4: while not converged do
5:     $V_{i+1}(s) := \max_a \{\sum_{s'} P_a(s, s') (R_a(s, s') + \gamma V_i(s'))\}$
6: end while
7: return policy:
       $$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s') \left( R_a(s, s') + \gamma V(s') \right) \right\}$$
```

question can be answered analytically for each state, making the AVI process well-defined, as described in Algorithm 1.

6 DISTRIBUTION TRACKING - EMPIRICAL EVALUATION

In this section we study our model retraining agent architecture in three different domains: one synthetic and two real world datasets. In all the experiments, $\gamma = 0.95$. While adjusting for different discount factors is outside the scope of this paper, our experimentation with other γ values in the range $[0.9, 0.99]$ indicates that results are qualitatively robust in that respect. In our experiments, we assume a utilty of 10 for each serviced request and a cost of 5 for using samples in training. In the case of transfer, a smaller model is trained, using only 50% of the amount of samples needed to retrain from scratch. Throughout the paper we use a sampling rate of 10% per batch. These parameters were chosen for simplicity, as they seem to reflect reasonable tradeoffs. Based on informal experimentation, the results do not seem qualitatively sensitive to the specific parameter configuration.

6.1 Proof of Concept - Synthetic Data

As a first step, we concretely illustrate our approach using a synthetic domain. This step is useful since controlling the process which generates the data gives us the freedom to both test the validity of our approach and test its limitations.

Our synthetic domain is simple - data is drawn from a 2-dimensional Gaussian distribution with unknown $\langle \mu, \Sigma \rangle$ parameters. At each timestep, a number of observations drawn from a Poisson distribution $|obs_t| \sim \textbf{Poi}(\lambda)$. Additionally, the distribution shifts in a random walk process at each timestep - μ and Σ drift by factors drawn from a different unknown distribution is added. To make the distribution meaningful, an upper bound is placed on the value of the true underlying variance.

We compare our value iteration approach to seven baseline policies: (1) a "do nothing" policy which always stays with its current model; (2) a "retrain always" policy, which retrains a model, paying the cost associated with this action, at each timestep; (3) an "adapt always" policy, which always updates the existing model using new data; (4) a random policy, which chooses actions uniformly irrespective of the current state at each timestep; (5) a fixed policy, which adapts the model if the observed data has deviated by more than 25% of the current estimate, and retrains if it has deviated by more than 50%. (6) DDM [7], a drift detection method that identifies potential drift based on the distributional properties of the error rate. DDM identifies two levels of drift risk - "warning" level and "drift" level, so these two levels correspond naturally to "transfer" and "retrain" actions, respectively (default parameters were used). (7) HDDM [6], a more recent approach to drift detection that relies on Hoeffding bounds. HDDM also identifies "warning" and "drift" modes for the data, which again correspond with "transfer" and "retrain" (default parameters were used).

We analyze the performance of each model reuse strategy over 30 timesteps with randomly drawn batches in this synthetic domain. The experiment is repeated over 10 iterations at each step to provide a measure of statistical significance to the reported results. The results are provided in Figure 2. As can be observed, our system significantly outperforms the other baselines. Figure 2(a) shows the overall average reward per step, whereas (b) and (c) show the accuracy and the costs, respectively, illustrating how the AVI approach carefully balances these two considerations in model retraining.

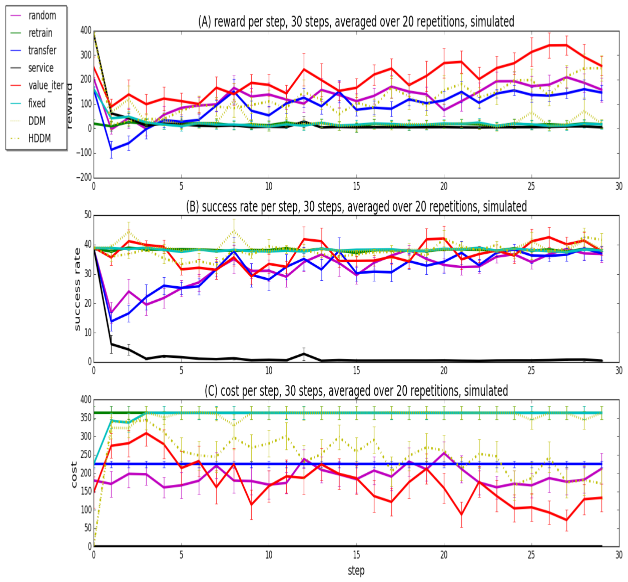

Figure 2: Reward per step over 30 time steps for our approximate value iteration (AVI) system compared to the other model retraining policies. Results are averaged over 20 simulations per step and are statistically significant (using a paired t-test[a]). Results are in the synthetic domain. (a) avg. reward (b) average success rate (c) avg. cost. Figure is best viewed in color.

[a] **In all places where mentioned, statistical significance was determined for cumulative reward at the end of the 30 steps for all agents.**

6.2 Real World Domain I - ThisIsMyJam Dataset

It is important to ascertain whether our approach holds merit with real world data that may not (and is indeed unlikely to) behave like a constantly shifting Gaussian distribution. For this purpose we look at an interesting real world domain: the ThisIsMyJam archive, which curates people‘s reported song preferences (AKA “jams”) [11]. This Is My Jam (2011-2015) was an online social music network where users could post one song at a time, their current “jam”. A jam lasted for up to a week, emphasizing the ephemeral nature of jams as favorite songs at that particular point in time. By cross-referencing the attested “jams” with the Million Song Dataset [1], one can extract many auditory features of songs liked by multiple people over varying time spans between the years 2011-2015. As a proof of concept, we set out to track the distribution of tempo and loudness of preferred songs on a weekly basis, using the same assumptions used in the synthetic domain. This task makes a good real-world test case for handling drift, since aggregated song preferences can vary wildly over short periods of time, and being able to track distributional properties of musical preferences is a valuable aspect of any music recommendation system that operates over extended periods. In the following experiments, the grid granularity we used was 50, and $C = 3$. Figure 3(a) illustrates the results in this domain. The results indicate that despite the domain being extremely noisy over time (as reflected by the correlated fluctuations in performance of all the algorithms examined), the value-iteration-based approach we propose consistently outperforms the baselines in terms of balancing performance and cost.

6.3 Real World Domain II - ECOMP

Lastly, we study the effectiveness of the proposed framework for tracking distributions on another, fundamentally different real world dataset, based on AT&T’s ECOMP Framework. The *chats* dataset looks at real world anonymized data derived from a customer care scenario, where a model must determine if a customer’s needs were fully satisfied after a digital chat with a representative. In this study, digital chats are pooled from 9/1/15 to 4/1/16 (214 days, over 1.8M samples) and transformed into term frequency (1363 features) with annotations as either true (the issue was resolved) or false (a subsequent chat was received) - this last annotation serves as the label we wish to predict. In this experiment, however, we are only interested in tracking the request distribution, reflecting the frequencies of certain terms. At each experiment we build a joint model on the top two principal components of the word count features. The results are presented in Figure 3. As before, we compare to the other baseline policies described above, and show our RL approach significantly outperforms the other ones. The grid granularity used was 50, and $C = 5$. Since data is provided in day ranges, in our experiments we treat all requests on a given day as a single batch. Figure 3(b) illustrates the results in this domain. This domain is of interest because success in tracking the profile of customer requests over time suggests overall applicability of the proposed drift management approach for many real world online services. Our results indicate that in this environment our proposed method does particularly well, outperforming the other baselines by a large margin.

7 PREDICTION MODEL RETRAINING

In this Section we discuss another instantiation of the autonomous model management problem, that of maintaining prediction models.

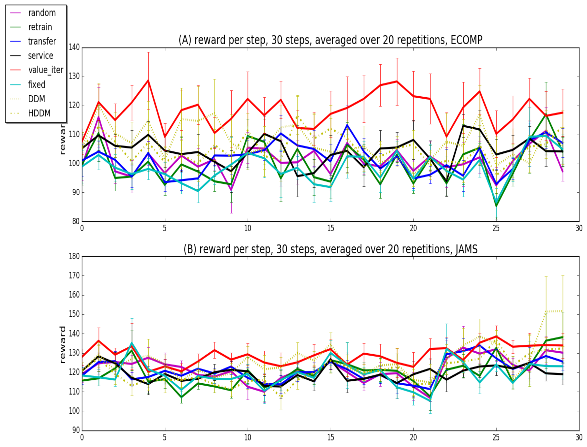

Figure 3: Reward per step over 30 time steps for our approximate value iteration (AVI) system compared to the other model retraining policies. Results are averaged over 20 simulations per step. (a) Results are in the AT&T ECOMP domain. (b) Results are in the song preference domain. Figure is best viewed in color. In both cases results are statistically significant (using a paired t-test).

While tracking feature distributions is a valuable task, in many real world scenarios the purpose of a trained model is to use the incoming data to predict other, hidden properties. In this section we study this problem, focusing on binary label classification. Given a constant stream of multidimensional data, we generate models that are meant to track the distributional properties of this data, *as well as* its correspondence to a predicted variable (or label).

7.1 MDP representation for the Prediction Problem

The case of tracking a *function* of the data may seem inherently different, but conceptually, it is in fact a generalization of the previous scenario. From the state space perspective, recall that in the distribution tracking setting described in Section 5 the states are defined as the tuple $\langle M_t, obs_t \rangle$, with M_t and obs_t the model and the new observed data at time t. In the prediction model retraining case, the formulation is exactly the same, but the internals of M are different - instead of storing sufficient statistics about the distributions, the model now contains a parameterization of the mapping from the observed data to the predicted label. However, in general structure, this task can also be seen as a concrete instantiation of the general problem described in Section 3. If in the distribution tracking case we were only interested in having new observations be sufficiently close to our model estimation, we now wish to explicitly predict labels for observations. In this case, a success (or a “hit”) is a case when we predict the label of a new observation correctly, and a failure (or a “miss”) is the case we didn’t. It is easy to see now why this setting strictly generalizes the distribution tracking case - one can construct a prediction task where the predicted label is “1” if the observation is within a certain ellipse around the distribution mean, and “0” otherwise (where the hidden decision boundary needs to be learned by the classifier). This model prediction formulation exactly mimics the distribution tracking case.

In cases where the model can be explicitly parameterized, almost the exact approach as that taken in Algorithm 1 can be taken, with only a few modifications, namely to the grid partitioning of the state space, which is now defined as $\mathcal{S} = \{\langle m, \overline{\text{obs}} \rangle \mid m \in \widehat{M}, \overline{\text{obs}} \in \text{OBS}\}$.

While not all modeling approaches are equally amenable to this type of model parameterization, many useful families of classifiers, such as artificial neural networks and logistic regression (which we use in this paper), expose their internal parameters. This property makes such models particularly suitable to be used in conjunction with the model management framework proposed in this paper.

7.2 Real World Domain - ECOMP

As a proof of concept for the predictive model update case, we once again use the AT&T's ECOMP data. In our experiment, we use the two features most correlated with the predicted label. As an underlying predictive model we use logistic regression. As described above, this model is straightforwardly usable within the context of our framework since it explicitly exposes its parameters. Recall that in the logistic regression setting we fit coefficients β_0, β_1 such that they capture the log odds ratio $ln(\frac{Pr(label=1|x)}{1-Pr(label=1|x)}) = \beta_0 + \beta_1 \cdot x$. Given that this is the case, an arithmetic mean of the coefficients is a reasonable analogue to the notion of transfer defined in the distribution tracking case. We also note that the expected utility of servicing requests adhering to one decision rule based on another decision rule can again be computed analytically. In this case, we used a grid granularity of 10 per dimension, $C = 5$, and days as timesteps. Using the ECOMP dataset to monitor a predictive model's ability to classify user requests as solved vs. unsolved is a real world use case for the proposed framework. We show that despite this setting being significantly harder, the AVI approach still does a better job of balancing cost and accuracy over time compared to the seven baselines we tested.

The results in this experiment are presented in Figure 4. Even in this considerably more complicated case, the RL approach we present in this paper is significantly superior to the other model update policies examined.

8 SCALING UP MODEL RETRAINING WITH FITTED VALUE ITERATION AND DEEP NEURAL NETS

As encouraging as the results in sections 6 and 7 are, the concrete implementation used in those sections is not without limitations. Perhaps most important is the issue of scale - any reinforcement learning paradigm reliant on grid value iteration is heavily subject to the curse of dimensionality. Fortunately, alternative function approximators, more amenable to high-dimensional input, exist. In this section, we adapt a different approximate value iteration approach, Fitted Value Iteration [10], to the model retraining problem, using a Convolutional Neural Network [14] as the approximator. We subsequently demonstrate that how with this modification our framework scales to much higher dimensional input.

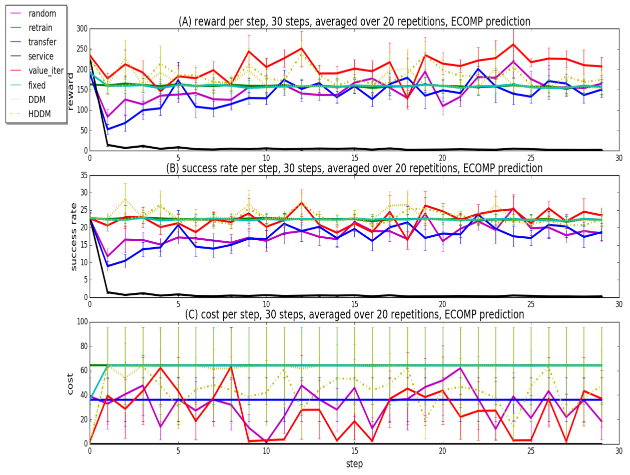

Figure 4: Reward, success rate and cost per step over 30 time steps for our approximate value iteration (AVI) system compared to the other model retraining policies in the AT&T ECOMP domain. In this experiment, the models do not simply track a distribution but rather attempt to make predictions over the data. Results are averaged over 20 simulations per step and are statistically significant (using a paired t-test). Figure is best viewed in color.

8.1 Fitted Value Iteration

Sampling-based Fitted Value Iteration, or FVI [10], is an off-policy approximate dynamic programming algorithm that computes an approximation of the optimal value function, $\hat{V}(s)$ by repeatedly sampling a finite subset of the state space, $S_{FVI} = \{s^1, s^2, \ldots, s^m\}$, and using it to perform value iteration, while also refining a function approximator that predicts $\hat{V}(s)$ for $s \in S$ that haven't been observed yet. More concretely, in our general case:

$$\forall i \in \{1 \ldots m\}$$
$$y^i = R(s^i) + \gamma \cdot max_a(E_{(s'|s^i,a)}[\hat{V}(s')])$$
$$\hat{V}(s) := SL(\{(s^1, y^1, (s^2, y^2), \ldots, (s^m, y^m)\})$$

$\hat{V}(s)$ is initialized arbitrarily (in our case, most conveniently, to all zeros). d after each update scan, a supervised learning algorithm SL is used as a function approximator that approximates the value function over the complete state-space, based on the "labeled" examples generated with by the previous version of the approximator, $\{(s^1, y^1, (s^2, y^2), \ldots, (s^m, y^m)\}$. While FVI is not guaranteed to converge to the optimal policy, it often performs well in practice, and is theoretically well-behaved [22].

8.2 Generalizing the Model Retraining Framework with Neural Fitted Value Iteration

In this subsection we consider how to concretely extend the distribution tracking approach described in Section 5 to an arbitrary number of dimensions using an FVI approach. Instead of sampling a grid representation of the entire state space, we wish to iteratively sample from the state space $\langle M^i, obs^i \rangle$. For this purpose, we can use the sampled batches of observations to sample <model, true

Algorithm 2 Distribution Tracking Fitted Value Iteration

```
1:  Input: observation sample $\widehat{obs}$ which is a union of observation samples
    from the last $k$ timesteps, max number of iterations $q$, initial approxi-
    mate value function $\hat{V}_0$, discount factor $\gamma$.
2:  $i = 0$
3:  while $\hat{V}$ not converged and $i < q$ do
4:      draw states $S_{i+1} = \{\langle M^i, obs^i \rangle\}$
5:      compute approximate reward for each state $\{r_i\}$
6:      $y_{i+1}(s) := r_i(s) + \max_a \{\sum_{s'} P_a(s, s')\gamma\,(V_i(s'))\}$
7:      Train new approximator $\hat{V}_{i+1} = CNN(S_{i+1}, y_{i+1})$
8:      $i = i + 1$
9:  end while
10: The resultant policy:
        $$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s') \left( R_a(s, s') + \gamma \hat{V}(s') \right) \right\}$$
```

observation distribution> configurations. Because in the multivariate case computing the expected utility of a model given a true distribution analytically may be expensive, we opt to approximate this utility using Monte Carlo sampling. Specifically, we draw from the "real distribution" and estimate how many of these requests are adequately handled by the model, giving us an immediate reward signal for the state which fits the analytical estimate in expectation.

For the purpose of function approximation we use a 5-layer convolutional neural net regressor. The training input is a tuple of both the model and true distribution coefficients, and the supervised signal is $R(\langle M^i, obs^i \rangle) + \gamma \cdot max_a(E_{(s'|s^i,a)}[\hat{V}(a(\langle M^i, obs^i \rangle))])$. We iteratively draw a new subset and update the function approximator using FVI until either convergence or a predetermined maximal number of iterations is reached. The full procedure is described in Algorithm 2.

8.3 Empirical Evaluation

We illustrate the effectiveness of the FVI approach using a synthetic dataset of a 20-dimensional multivariate Gaussian that shifts randomly in each dimension in each timestep. As before, using a synthetic dataset for testing is useful since controlling the process which generates the data gives us the freedom to fully test the validity and robustness of our approach. Data is drawn from a 100-dimensional Gaussian distribution with unknown $\langle \mu, \Sigma \rangle$ parameters. As before, at each timestep, a number of observations (or requests) is drawn from this distribution (the number of observations drawn is itself a random variable drawn from a Poisson distribution $|obs_t| \sim \mathbf{Poi}(\lambda)$). Additionally, the distribution shifts in a random walk process at each timestep - μ and Σ drift by factors drawn from a different unknown distribution. We use the same cost parameters as in Section 5, with the same baselines, all generalizable to the 20-dimensional case. The results are presented in Figure 5. As one might observe from the results, despite the overall challenge, the value-iteration approach still does best in terms of balancing retraining costs and performance.

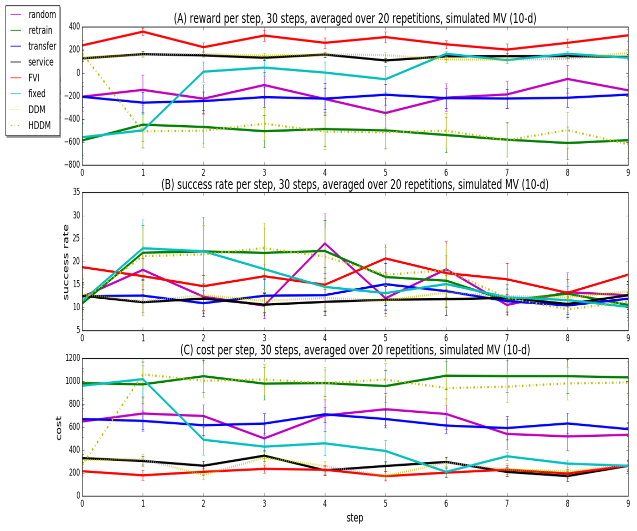

Figure 5: Reward, success rate and cost per step over 30 time steps for our fitted value iteration (FVI) system compared to the other model retraining policies on simulated 10-dimensional data. Results are averaged over 20 simulations and are statistically significant (using a paired t-test). Figure is best viewed in color.

9 SUMMARY & DISCUSSION

The risk of concept drift has a potentially devastating effect on many real world systems that involve modeling. To this end, in this paper we present a reinforcement learning approach for continual model updating. Rather than building concept drift resistance into the learned model, we frame the model update problem as a sequential decision making task, and adapt an approximate value iteration approach to learn a policy for when to update the model, and how. This framework is generic, has theoretical grounding, and can be easily applied to many different real world systems. We empirically evaluate our approach on three different datasets, and show it outperforms other baseline update policies.

Our approach is just the first step toward more robust systems that can adapt to changing environments. Certain issues, such as the appropriate partitioning of streaming data into batches, and identifying performance under different types of drift, should be explored in future work. In addition, other reinforcement learning algorithms and more refined transfer strategies could greatly benefit the system, and would make for excellent follow-up research.

10 ACKNOWLEDGMENTS

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset.. In *ISMIR*, Vol. 2. 10.

[2] Dariusz Brzezinski and Jerzy Stefanowski. 2014. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (2014), 81–94.

[3] Margaret Chiosi and Brian Freeman. 2015. AT&T's SDN Controller Implementation Based on OpenDaylight. (27-31 7 2015). http://doi.acm.org/10.1145/2843948 Open Daylight Summit.

[4] Chee-Seng Chow, John N Tsitsiklis, et al. 1989. An optimal multigrid algorithm for discrete-time stochastic control. (1989).

[5] William G Cochran. 1977. Sampling techniques. 1977. *New York: John Wiley and Sons* (1977).

[6] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. 2015. Online and non-parametric drift detection methods based on HoeffdingâĂŹs bounds. *IEEE Transactions on Knowledge and Data Engineering* 27, 3 (2015), 810–823.

[7] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*. Springer, 286–295.

[8] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 44.

[9] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4 (Dec. 2015).

[10] Geoffrey J Gordon. 1995. Stable function approximation in dynamic programming. In *Proceedings of the twelfth international conference on machine learning*. 261–268.

[11] Andreas Jansson, Colin Raffel, and Tillman Weyde. [n. d.]. THIS IS MY JAMâĂŤ-DATA DUMP. ([n. d.]). https://archive.org/details/thisismyjam-datadump

[12] Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. 2004. Online learning with kernels. *IEEE transactions on signal processing* 52, 8 (2004), 2165–2176.

[13] Ralf Klinkenberg and Thorsten Joachims. 2000. Detecting Concept Drift with Support Vector Machines.. In *ICML*. 487–494.

[14] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8, 1 (1997), 98–113.

[15] Leandro L Minku and Xin Yao. 2012. DDD: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering* 24, 4 (2012), 619–633.

[16] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[17] Mark Bishop Ring. 1994. *Continual Learning in Reinforcement Environments*. Ph.D. Dissertation. University of Texas at Austin.

[18] Paul Ruvolo and Eric Eaton. 2013. ELLA: An Efficient Lifelong Learning Algorithm. *ICML (1)* 28 (2013), 507–515.

[19] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 815–823.

[20] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. 2010. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development* 2, 2 (2010), 70–82.

[21] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[22] Csaba Szepesvári and Rémi Munos. 2005. Finite time bounds for sampling based fitted value iteration. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 880–887.

[23] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[24] Lisa Torrey and Jude Shavlik. 2009. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques* 1 (2009), 242.

[25] Alexey Tsymbal. 2004. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106 (2004).

[26] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine learning* 23, 1 (1996), 69–101.

[27] Indrė Žliobaitė. 2010. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784* (2010).