

Automated Knowledge Discovery from Simulators

M.C. Burl*, D. DeCoste*, B.L. Enke⁺, D. Mazzoni*, W.J. Merline⁺, L. Scharenbroich*[†],

*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA

⁺Southwest Research Institute, Boulder, CO

[†]University of California, Irvine, CA

Abstract

In this paper, we explore one aspect of knowledge discovery from simulators, the *landscape characterization problem*, where the aim is to identify regions in the input/parameter/model space that lead to a particular output behavior. Large-scale numerical simulators are in widespread use by scientists and engineers across a range of government agencies, academia, and industry; in many cases, simulators provide the only means to examine processes that are infeasible or impossible to study otherwise. However, the cost of simulation studies can be quite high, both in terms of the time and computational resources required to conduct the trials and the manpower needed to sift through the resulting output. Thus, there is strong motivation to develop automated methods that enable more efficient knowledge extraction.

Unlike traditional data mining, knowledge discovery from simulators is not limited to a static, pre-determined dataset; instead, the simulator itself can be used as an oracle to generate new data of our own choosing. We exploit this opportunity by employing active learning and support vector machines (SVMs) to choose which are the most valuable simulation trials to run next. On two real-world scientific simulators, one for asteroid collisions and one for magnetospheric modeling, we demonstrate twofold and sixfold reductions, respectively, in the number of simulator trials required to achieve a particular level of fidelity in landscape characterization as compared with a standard grid-based sampling approach.

Keywords: simulator, knowledge discovery, landscape characterization, active learning, asteroid collisions, SVM.

1 Introduction

Large-scale numerical simulators are in widespread use for investigating a range of complex phenomena. Science applications include studies of the Earth’s core and mantle dynamics, climate prediction, atmospheric and ocean dynamics, fluid flows in micro-gravity environments, dynamics of the interior of stars, interaction of the solar wind with the Earth, formation of star-planet systems, and origins of life. Engineering applications include studies of aerodynamics, propulsion, multi-jointed actuated structures such as biped and quadruped robots, system reliability, and nuclear/chemical/biological processes.

The high-performance computing (HPC) community has expended considerable effort to produce big-

ger, faster, and more accurate simulations. However, the complementary problem of deciding which simulation trials to run and how to transform the output of a set of trials (gigabytes and terabytes of data¹) into knowledge has received little attention (for one exception, see [8]).

Although some traditional data mining techniques can be employed to analyze the output data, a unique aspect of knowledge discovery from simulators is that we are not limited to a static, pre-determined dataset; instead, the simulator itself can be used as an oracle to generate new data of our own choosing leading to rich opportunities for active exploration and experimentation. In this paper, we focus on a particular knowledge discovery problem, called *landscape characterization* [5], where the aim is to try to determine which initial conditions, parameters, or interaction equations (models), lead to a given output behavior from a dynamical system that is represented by a numerical simulator.

The approach we take relies on support vector machines (SVMs) [21, 7] and active learning techniques. SVMs are a powerful, supervised classification technique that have proven effective in a number of applications. The maximum margin property of SVMs, which often enables good generalization performance even with a limited number of training examples, and the representation of the decision boundary through a set of “support vectors²” and corresponding weights make SVMs particularly well-suited for the landscape characterization problem. Once the parameter landscape has started to emerge from initial trials (i.e., some positive and negative examples have been observed), we train an SVM to represent the decision boundary between the classes. Active learning can then be used to identify new points that if labeled (by a run of the simulator) would

¹Consider that a simple 3-D mesh model of a cubic meter volume with 1mm spatial resolution will generate at least one GB (gigabyte) of data for each measured attribute (e.g., temperature, pressure) for each time step in a simulation.

²Support vectors are instances from the training set that are generally near the decision boundary between classes.

maximize, in some sense, the amount of new information obtained. By *actively* choosing the next point to run in an intelligent fashion, we expect to obtain more information than just running a randomly-selected point or a point that was picked *a priori* without consideration of the results in previous trials. Experimental evidence on two large-scale scientific simulators supports this intuition.

The organization of the paper is as follows. In Section 2 we establish the basic framework for the landscape characterization problem. In Section 3 we describe the active learning techniques used in this work. In Section 4 we provide insight into how the appropriate SVM hyperparameters (kernel type, kernel-specific parameters, regularization parameter) can be determined in the active learning framework. We also introduce a novel twist to existing active learning paradigms by combining the “wishes” of multiple SVMs, each with different hyperparameters, into an appropriate choice for the next point to run. In Section 5, we lay out the evaluation methodology (how do we know whether active learning has been successful, etc.). In Section 6 we briefly describe the two real-world scientific simulators used in this work: (i) one models asteroid collisions using a smooth particle hydrodynamic (SPH) model of the energetic-collision physics involved in the impact, followed by an efficient N-body gravitational code that traces the dynamical evolution of the shattered asteroid fragments after the collision, and (ii) the other models the global dynamics of energetic plasma in the Earth’s inner magnetosphere. Section 7 presents the experimental results. Section 8 outlines directions for future work and Section 9 provides conclusions.

2 Landscape Characterization

2.1 Dynamical Systems Formalism. We formalize the concept of a simulator using language and notation from dynamical systems and control theory. We abstract a simulator as a discrete-time system in which a vector of internal state variables \mathbf{x} is updated at each time step.

$$(2.1) \quad \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k, \boldsymbol{\theta})$$

where $\mathbf{f}(\cdot)$ is a potentially nonlinear, vector-valued function. The variable \mathbf{x}_k represents the internal state at time step k , \mathbf{u}_k represents inputs to the system at time k , \mathbf{w}_k is a random vector representing a disturbance (or plant-noise) process, and $\boldsymbol{\theta}$ represents any fixed parameters that affect the behavior of the system. Note that the time variable k can be included in the state vector to allow various behaviors that depend on the absolute time. Also, the state can include binary- or discrete-valued indicator variables that cause switching in the

system dynamics. For example, in modeling quadruped walking, the system dynamics should switch depending upon which feet are in contact with the ground at a given moment. The parameter vector $\boldsymbol{\theta}$ is assumed to be fixed throughout a given run of the simulator (i.e., there are no dynamics within the parameters of $\boldsymbol{\theta}$). Note that $\boldsymbol{\theta}$ can be used to turn \mathbf{f} into a *class* of functions, by allowing some elements of \mathbf{f} to be activated or deactivated by specific parameters in $\boldsymbol{\theta}$.

For the simulators discussed in this paper, there are no time-dependent inputs nor disturbances, so the state evolution is governed by the simpler equation:

$$(2.2) \quad \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \boldsymbol{\theta})$$

Recursively applying Equation 2.2 from time step $k = 0$ to time step $k = T$, we can write:

$$(2.3) \quad \mathbf{x}_T = \boldsymbol{\Phi}_T(\mathbf{x}_0, \boldsymbol{\theta})$$

where $\boldsymbol{\Phi}_T$ is defined recursively in terms of $\mathbf{f}(\cdot)$. We can go a step further and suppose that the initial value of the state, \mathbf{x}_0 , is recorded in the parameter vector, so the state at time T is totally determined by $\boldsymbol{\theta}$. We will refer to the set of possible parameter vectors as *input space* and designate this space Θ . Choosing $\boldsymbol{\theta}$ from Θ provides the input to the simulator and, for a deterministic simulator, determines the output. Note that input space may be a mixture of continuous-valued (real numbers) and/or discrete-valued (integers) parameters.

In control systems, it is common to introduce a *measurement process* in addition to the state update equation to specify what aspects of the system can be directly measured (the measurements are a function of the state vector and a random-valued measurement noise process). While not usually necessary with a simulator³, we will define a measurement process as follows:

$$(2.4) \quad \mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k, \boldsymbol{\theta})$$

where \mathbf{z} is the vector of observed values, \mathbf{h} is a potentially nonlinear, vector-valued function and \mathbf{v}_k is a random vector representing the measurement noise process.

2.2 Objective. As we have seen, the output of a deterministic simulator is totally governed by the input parameters $\boldsymbol{\theta}$. Frequently, the scientist or engineer using the simulator is interested to know which values of the

³We usually have full access to the state vector at every instant of time, although there could be situations where the simulator is a black box (or even a physical experiment) or the full state vector is not recorded in the simulation output.

input parameters lead to a desired output result. To be more concrete, in the asteroid collision simulations that we will introduce in Section 6, the input parameters include the size and composition of both asteroids, as well as the velocity and angle of impact. The scientist would like to understand what combinations of input values lead to a situation in which fragments of the two asteroids become gravitationally captured in a binary pair. This amounts to determining what region in Θ space leads to a specific output. If we attach binary labels to each simulation run (indicating whether desired output occurred), then we can state the landscape characterization problem as follows:

Landscape Characterization: *Determine the set of points $\theta \in \Theta$, such that $q(\theta) = 1$, where q is a binary-valued function that specifies whether a simulation is successful in producing the desired output or not*

Since the underlying parameter space Θ can be continuous-valued, the points in the solution region cannot be simply enumerated. What we really want is to produce a function, $\hat{q} : \Theta \rightarrow \{-1, 1\}$ such that \hat{q} agrees with q over most of the domain. In this sense, the landscape characterization problem is similar to the *concept learning* framework [14] of machine learning. The main difference is that in concept learning, we are given a fixed set of examples (instances with labels) drawn i.i.d. from an underlying distribution over instance space and from these examples we should try to derive $\hat{q}(\cdot)$. In the landscape characterization problem, we are merely given the simulator which can serve as an oracle to generate examples. The selection of examples is not i.i.d., but is conditionally-dependent on previously selected examples. Also, since running the simulator can be quite expensive (in terms of time and computer resources), we have the additional constraint that we want to make as few calls to the simulator as possible.

2.3 SVM Framework. A landscape characterization problem is considered to be “solved” when the agreement between $\hat{q}(\cdot)$ and $q(\cdot)$ is sufficient. We will use the Support Vector Machine (SVM) framework [21, 7] to represent the function \hat{q} :

$$(2.5) \quad \hat{q}(\theta) = \sum_{i=1}^{n_s} \beta_i K(\theta, \mathbf{s}_i)$$

where θ is a point in input space, n_s is the number of support vectors, \mathbf{s}_i is the i -th support vector, β_i is a scalar weight, and $K(\cdot, \cdot)$ is the kernel function.

As more data is acquired through ongoing runs of the simulator, our estimate for \hat{q} will change (hopefully improve!); hence, we will add a superscript (n)

to \hat{q} to designate the estimated landscape function after the n -th example from the simulator has been observed. Also, the SVM solution to a learning problem is dependent on a number of associated hyperparameters, such as the kernel type (polynomial, RBF, etc.), kernel-specific variables (polynomial degree, RBF bandwidth, etc.), and the choice of regularization constant (C). Different choices of hyperparameters lead to different characteristics for the resulting decision surface, e.g., “bumpy” or smooth. Hence, we consider multiple hyperparameters in order to find a good match to the true underlying function $q(\theta)$. At any stage, then, we will have multiple estimates of the landscape function where each estimate is based on a specific set of SVM hyperparameters, λ . We will use $\hat{q}_{\lambda}^{(n)}(\cdot)$ to designate the estimated landscape function after the n -th trial using SVM hyperparameters λ . For notational convenience in the next section, we will assume a single, fixed vector of SVM hyperparameters, but will return to considering multiple hyperparameters in Section 4.

3 Active Learning

Given the high cost of running the simulator, we want to solve the landscape characterization problem as efficiently as possible, i.e., with a minimal number of simulator trials. Active learning is a process that determines which new points in input space, if labeled, would be the most informative for refining the boundaries of the solution region. If active learning consistently chooses good points, then the most informative simulations given the current state of knowledge will be run and the solution region will be identified more quickly. If poor choices of points to run are made, then there is a steep penalty since days or weeks of simulator time may be used to generate a result that is not particularly valuable.

3.1 Initialization. To get the active learning process started, we first pick points randomly from input space (according to a uniform distribution, unless the scientists’ background knowledge of the problem can provide more guidance). These points are run through the simulator until we find at least one positive and one negative example⁴. Once the initialization is set, SVM can learn an initial decision boundary and let the active learning process take over choosing which points should be run next.

There are some opportunities to be more clever with the initialization. Using the scientists’ background knowledge to guide the process was mentioned above. Putting “nulls” in areas immediately around negative

⁴In most cases, examples from the positive class tend to be more rare than examples from the negative class.

examples might lead to a better initial exploration of the space. As the trials progress, the nulls could be contracted to permit sampling closer to known negative examples. Another idea is to use bootstrapping once a positive example is found to try to get a few more positives before starting the active learning. One bootstrapping approach would be to sample, say according to a multivariate Gaussian density, in the vicinity of a positive example. There is always a danger, though, of focusing too much effort on one part of the space. The active learning algorithms must carefully balance exploration (semi-randomly searching around the space) and exploitation (using the current state of knowledge to conduct a tightly-focused search). One idea for balancing these concerns would be to continue random exploration of the input space until some pre-determined number of positives was obtained. Thus, the initialization would insure at least some fixed number of positive seed points. However, in the few experiments where we tried this approach, it did not necessarily lead to quicker solutions.

Another tack for increasing the number of positive seed points would be to exploit additional domain knowledge in the form of invariances or monotonicity constraints. For example, the scientist may know that if an outcome is positive for a particular input energy level, then it will be positive for all higher energy levels, keeping the other parameters fixed. This information would allow a single initial positive example to imply a whole set of additional positive examples. (In fact, such information could be exploited at any time during the process by multiplying the effective number of examples similar to the “hints” framework developed in [1]).

3.2 Basic Ideas. Active learning is not restricted to any single inductive learning method, although much of the recent work in this area has used the SVM framework. Note that at any time in the process, the SVM decision boundary is only based on the examples seen so far and will not be perfect. Active learning will use this imperfect boundary as an indication of which cases are the contentious ones. Acquiring the true label for these cases (i.e., by running them through the simulator) provides valuable information for how the boundary should be modified.

For notational purposes, we will let $\mathcal{L}^{(n)}$ represent the set of examples that at the conclusion of trial n have been labeled. We will also think of a pool of unlabeled examples $\mathcal{U}^{(n)}$, which represents the set of unlabeled examples remaining at the conclusion of trial n . For the simulators problem, the input space of parameters is often continuous-valued, so it is not possible to enumerate all the unlabeled examples into such a set;

instead we will form $\mathcal{U}^{(n)}$ by randomly sampling from input space, choosing some target number of points N . The goal then is to choose an instance $\theta^{(n+1)}$ from $\mathcal{U}^{(n)}$, such that an SVM constructed from

$$(3.6) \quad \mathcal{L}^{(n+1)} = \mathcal{L}^{(n)} \cup \theta^{(n+1)}$$

will achieve the greatest increase in accuracy (bringing $\hat{q}(\cdot)$ into closer agreement with $q(\cdot)$). Of course, we do not know the true underlying function $q(\cdot)$, so the choice of $\theta^{(n+1)}$ must be based on heuristics.

3.3 Algorithms. A number of active learning heuristics have been developed in the literature. We briefly describe the main methods we considered.

- **Random:** This method selects $\theta \in \mathcal{U}$ randomly. It is equivalent to “passive” learning, since the choice of next point does not depend on what is “known” so far.
- **Simple [20]:** This method assumes that the unlabeled item closest⁵ to the current SVM decision boundary is likely to be the most helpful for updating the boundary. *Simple* ranks each example by its distance from the hyperplane and then chooses the minimum.

$$(3.7) \quad \theta^{(n+1)} = \arg \min_{\theta \in \mathcal{U}^{(n)}} \left| \hat{q}_{\lambda}^{(n)}(\theta) \right|$$

- **MaxMin Margin [20]:** Rather than guessing that the item closest to the decision boundary will yield the most information, this method chooses the unlabeled instance that guarantees the best (maximum) SVM margin assuming the worst-case about which label (+1 or -1) will be assigned to that instance.

Let m be the size of the separation between the positive and negative classes (the margin) for the current SVM. Then, for each $\theta \in \mathcal{U}^{(n)}$, this method trains two SVMs: one on $\mathcal{L}^{(n)} \cup \{(\theta, +1)\}$ (yielding m^+) and one on $\mathcal{L}^{(n)} \cup \{(\theta, -1)\}$ (yielding m^-). Finally, *MaxMin* selects θ such that $\min(m^-, m^+)$ is maximized. When *MaxMin* is working well, it tends to halve the “version space” of models that are consistent with the data after the new labeled example is obtained.

- **Diverse [6]:** This algorithm attempts to avoid choosing too many similar queries by choosing examples that are as diverse as possible with respect

⁵If there is not a unique closest point, we can choose randomly from the set of closest points.

to the version space. The diversity of $\mathcal{L}^{(n)} \cup \theta^{(n+1)}$ can be maximized by selecting $\theta^{(n+1)}$ whose maximum normalized kernel distance to all of the other labeled examples is minimized. The complete method uses a weighted sum of diversity and hyperplane distance, controlled by a parameter κ , where $\kappa = 0$ is the equivalent of focusing solely on diversity and $\kappa = 1$ is the same as *Simple*. We determined that $\kappa = 0.5$ worked well for our experiments.

3.4 Meta-strategies. Above the level of individual active learning algorithms, we can consider “meta-strategies” that use one or more of these individual algorithms to achieve better performance. Because the active learning algorithms rely on heuristics, it is not always best to accept the top-rated query returned by a particular active learning algorithm. Baram and colleagues [2, 3] focus on choosing among different active learning algorithms automatically. We use a meta-strategy in which the query to the simulator is chosen probabilistically based on the utilities in such a way that points with higher utilities have a better chance of being selected. In the experiments, we used a truncated Gaussian distribution to assign selection probabilities:

$$(3.8) \quad P(\theta) \propto \exp \left[-\frac{(u(\theta) - 1)^2}{2\sigma^2} \right]$$

where $u(\cdot)$ is the utility assigned by active learning normalized to the range $[0, 1]$. A value of σ around $1/3$ insures that a query with utility 1.0 is roughly 100 times more likely to be selected than a query with utility 0. Probabilistic active learning turns out to be more effective than simply selecting the top-ranked query every time, since the probabilistic behavior provides a trade-off between exploration and exploitation. In our experiments, we used a fixed value of σ , but incorporating a “cooling schedule” could prove more effective (i.e., using a larger σ value in early trials to insure more exploration, and using a smaller σ value in later trials to insure more exploitation).

4 Choosing SVM Hyperparameters

The SVM learning framework requires a set of hyperparameters, λ , including the kernel type (e.g., polynomial or RBF), kernel-specific parameters (e.g., degree for polynomial kernels, bandwidth for RBF kernels), and the regularization constant (C). One approach is to simply make a single choice of hyperparameters up front, $\lambda = \lambda_0$, and stick with it throughout all the active learning trials. Unfortunately, this method relies on prior experience or luck to get a good result.

4.1 Accuracy Estimation. A more robust alternative to making an up-front choice of hyperparameters is to consider a finite pool, Λ , of potential choices and try to get an accuracy estimate for each element in the pool.

Instantaneous x-val. This approach is the usual method used with SVMs; however, the situation is more difficult here, since the pool of labeled examples will not be large. After the n -th trial of the simulator, we have only n labeled points: $(\theta^{(i)}, q(\theta^{(i)}))$, for $i = 1, \dots, n$. This small set of points can be used to conduct leave-one-out cross-validations using each of the potential hyperparameter choices. From the cross-validations, we obtain an estimate $\hat{\rho}_{\lambda}^{(n)}$ of the true accuracy of $\hat{q}_{\lambda}^{(n)}$ for each $\lambda \in \Lambda$.

Bayesian approach. This approach is similar to *instantaneous x-val*, except that the instantaneous accuracy estimates are combined with a prior probability distribution to produce a posterior distribution over the accuracy. We take the initial prior distribution (before any trials have been conducted) to be a Beta-distribution:

$$(4.9) \quad \begin{aligned} p_0(\rho; \alpha, \beta) &= \beta(\rho; \alpha, \beta) \\ &\propto \rho^{\alpha-1}(1-\rho)^{\beta-1} \end{aligned}$$

where ρ is the accuracy. The prior can be interpreted as the equivalent of $\alpha - 1$ successes (correct classifications) in $\alpha + \beta - 2$ Bernoulli trials. The Beta-distribution has the *conjugacy* property so that the posterior distribution, which takes into account the new results and the prior, will remain a Beta-distribution. In fact, if the new instantaneous x-val results yield k correct classifications and $n - k$ incorrect classifications, the posterior distribution is $\beta(k + \alpha, n - k + \beta)$.

The mode of the Beta posterior is given by:

$$(4.10) \quad \rho^* = \frac{k + \alpha - 1}{n + \alpha + \beta - 2}$$

which serves as a point estimate (MAP) for the accuracy of the SVM trained with a particular choice of hyperparameters.

4.2 Choosing the Next Point. We now have two methods for estimating the accuracy of an SVM trained using a specific set of hyperparameters. The active learning framework could simply use the SVM that has the highest estimated accuracy (by one of the methods) to decide which input space point should be labeled next. While this winner-take-all approach to choosing the next point may work fine, we may be able to do better by taking the “wishes” of multiple SVMs (each

trained with a different choice of hyperparameters) into account when choosing the next point to label, similar in spirit to [13].

$$(4.11) \quad \theta^* = \arg \max_{\theta \in \mathcal{U}} \left(\sum_{\lambda \in \Lambda} w_{\lambda} \cdot u_{\lambda}(\theta) \right)$$

where $u_{\lambda}(\theta)$ is the utility assigned to the point θ by the active learning algorithm using $\hat{q}_{\lambda}(\cdot)$ as a classifier and w_{λ} is a weight assigned to the classifier, based on some assessment of the classifier’s accuracy. In the experiments conducted in Section 7.3, we used a measure based on the Beta posterior at the n -th step, $p_{\lambda}^{(n)}(\rho)$. Each weight w_{λ} is computed as follows:

$$(4.12) \quad w_{\lambda} = \frac{1}{Z} \cdot \prod_{\hat{\lambda} \in \Lambda, \hat{\lambda} \neq \lambda} E_{\hat{\lambda}}^{(n)}[\text{Pr}_{\lambda}^{(n)}(\rho)]$$

where Z is a normalization constant, $E_{\hat{\lambda}}^{(n)}[f(\cdot)]$ is the expectation of a function relative to the posterior distribution of the classifier parameterized by $\hat{\lambda}$, and $\text{Pr}_{\lambda}^{(n)}(\rho)$ is the cumulative probability distribution of the posterior parameterized by λ . Each weight, w_{λ} , is proportional to the probability that a given classifier is the most accurate relative to the others.

A different approach could follow the lines of [9] to take into account the full posterior distribution to sequentially build hypotheses and confidences regarding which hyperparameters dominate.

5 Evaluation Methodology

5.1 Synthetic Data. As an initial testbed for evaluating the active learning algorithms, we used synthetic data. To create synthetic datasets, we simply define a function $q(\theta) : \Theta \rightarrow \{-1, +1\}$, which serves as the oracle in place of the simulator. Because q can be evaluated quickly, the active learning algorithms can be easily tested and verified. More importantly, knowing what the underlying solution region really is, enables the performance of the active learning algorithm to be quantified, i.e., we can measure the agreement between $\hat{q}^{(n)}(\cdot)$ and $q(\cdot)$ as a function of n by discretizing Θ with an extremely fine grid. With the actual simulators, we usually do not have this luxury because it can take days or weeks to determine a single true value from $q(\cdot)$.

5.2 Grid Sampling. While the synthetic experiments are valuable, the real test is in the loop with a complex simulator. Because the underlying $q(\cdot)$ function is not known under such circumstances, we are limited to evaluating $q(\cdot)$ on as fine a grid (discretization of

Θ) as possible given the simulator speed. We consider the landscape characterization problem to be solved if $\hat{q}^{(n)}(\cdot)$ agrees with $q(\cdot)$ on a sufficient fraction of the grid points ($> 90\%$ in our experiments, but the results do not change much if a more stringent standard is applied; see for example Figure 3 in Section 7.2). We also use plots of the accuracy of $\hat{q}^{(n)}(\cdot)$ as a function of n to characterize learning speed.

5.3 Is Speedup the Whole Story? Suppose that obtaining a solution by direct evaluation of each point on the ground-truth grid takes n_g trials. Further, suppose that obtaining a solution that sufficiently agrees with the ground-truth solution via active learning takes n_a trials. Is the speedup $\frac{n_g}{n_a}$ the whole story?

While speedup is certainly an important metric, it does not capture the fact that the active learning approach likely achieves superior sub-grid accuracy (over say a nearest neighbor interpolation of the coarse grid). The reason is that active learning tends to concentrate its queries near the boundary of the solution region. Unfortunately, it is often computationally prohibitive to perform a complete set of runs at finer discretizations. A workaround that is informative, but not as comprehensive, is to determine on the finer grid any locations of disagreement between the $\hat{q}(\cdot)$ function estimated by active learning and the value obtained from nearest neighbor interpolation from the coarse grid. Points of disagreement could then be tested via the simulator to see which algorithm is right more often on points of contention (or a random subset of those points).

6 Scientific Applications

6.1 Asteroid Collisions. Asteroid collision simulations are a typical example of a complex numerical simulation. Collisions between objects are one of the prime geologic processes that have affected the surface of planets and asteroids over the age of the solar system. A simulator gives scientists the opportunity to study collisions on a scale that is not possible in Earth-based laboratories. Generally, only smaller-scale collisions can be recreated in labs, but from those, computer codes have been developed to simulate the physics of larger collisions and the subsequent gravitational evolution of collisional products.

One of the consequences of such collisions is the production of a gravitationally-bound binary, either a pair of objects of similar size (a double asteroid) or of dissimilar size (resulting in a small orbiting satellite, or moon around the original target of the impact). From sophisticated ground-based telescopes using adaptive optics, from radar observations, from asteroid lightcurve analysis, and from spacecraft imaging, scientists have

determined that such binaries do exist [12]. Simulations can help scientists understand how and under what conditions the observed binary systems were produced [10]. In addition, the simulations provide feedback about what types of masses and orbits might be expected. This aids in the allocation of scarce resources used to search for such pairs.

Our impact simulations take advantage of a state-of-the-art numerical model that combines results from a smooth-particle hydrodynamics (SPH) code [4], which accurately models the pressures, temperatures, and energies of asteroid-asteroid impacts, and an N-body code [11], which can efficiently track the subsequent trajectories of tens-of-thousands of individual post-collision fragments. Significant input parameters are the size and composition of the target and impactor, the impact angle and velocity, spin states, and the degree of fractures in the target body. Several characteristics of the problem are kept constant, such as the diameter of the target, composition, and density of the bodies. Once the relevant portions of the impact phase are complete (crater formation/ejecta flow fields established with no further fragmentation/damage), the outcomes of the SPH models are handed off as the initial conditions for N-body gravitational simulations, which follow the trajectories of the ejecta fragments for sufficient time (4 simulated days) to search for the formation of bound satellite systems. Each simulation trial takes over 10 days to run on a 1GHz CPU. Figure 1 shows the basic asteroid collision simulator.

The result of the asteroid impact simulations is a lengthy list of individual particles, each with various properties like mass, velocity, orbital parameters, etc. Post-processing can be used to calculate many useful output metrics such as ratios of masses and sizes between the most massive bound particles, average eccentricities of orbits, size-frequency distributions of ejecta fragments, and number of gravitationally bound systems. Only a small fraction of the input parameter space results in a satellite being formed.

6.2 Magnetospheric Modeling. Simulations of the Earth’s magnetosphere improve scientific understanding, but also play an important role in the prediction of “space weather” and avoidance of its consequences, which can include telecommunications interruptions, power grid problems (rapidly changing magnetic fields during magnetospheric storms induce currents in the cables of the power grid), loss of spaceborne assets, and hazards to astronauts.

Space physicists use a variety of magnetohydrodynamic and kinetic models to help understand the global dynamics of energetic plasma in the inner magneto-

sphere as a function of geomagnetic activity. The goal is accurate forward prediction driven by observational in-situ data. The plasma population is related to the energetic neutral atom (ENA) flux, which is observable through remote-sensing techniques, for example, using satellites such as IMAGE and POLAR. Since energetic neutrals are a “by-product” of the presence of plasma in space, the actual plasma distribution has to be inferred either through forward modeling or through mathematical inversion techniques. In its simplest form, the forward modeling simulator describes the plasma population via a 10-parameter model developed in a series of papers by Roelof et al [16, 17, 18]. (It also supports a more complex 35-parameter model.) Minimization techniques are used to reduce the difference between spacecraft-measured ENA images and simulated images generated from the Roelof model. The process takes about four hours of real-time on a single-CPU 1 GHz Intel-based processor.

7 Experimental Results

7.1 Asteroid Collision Experiments. Planetary science domain experts determined the number and ranges of input parameters that would be of greatest initial benefit. In particular, a 3-dimensional Θ space was chosen consisting of the following variables and ranges: impact velocity (3-7km/s), angle (10-80 degrees with respect to target-surface normal), and mass ratio (1 to 10000). Other potential input parameters, like the diameter of the target, composition, and bulk density, were held constant. A simulation was given a high grade if the impact was non-catastrophic (more than 50% of the target remained intact after the event) and resulted in an adequately-sized satellite orbiting the largest remnant. Runs where the grade exceeded a threshold (related to observability from Earth) were considered to be successes.

To establish ground truth for the domain, we first determined a reasonable grid resolution covering the ranges of the input parameters. These grid runs are not a normal part of the active learning approach, but are needed here to enable evaluation of performance. The simulator runtime is highly dependent upon the number of particles in the simulation. The best mix of science output and real-world applicability involved the use of 10^5 particles, which translates into an average simulation runtime of about 10 days on a single 1GHz processor. Finally, we chose a $(5 \times 5 \times 6)$ grid (150 runs) over input space and conducted the grid runs, which took many months to complete on a 32-node cluster. Of the 150 points in the uniform grid, 25 points led to the formation of a binary pair.

Next, we applied the active learning approach using

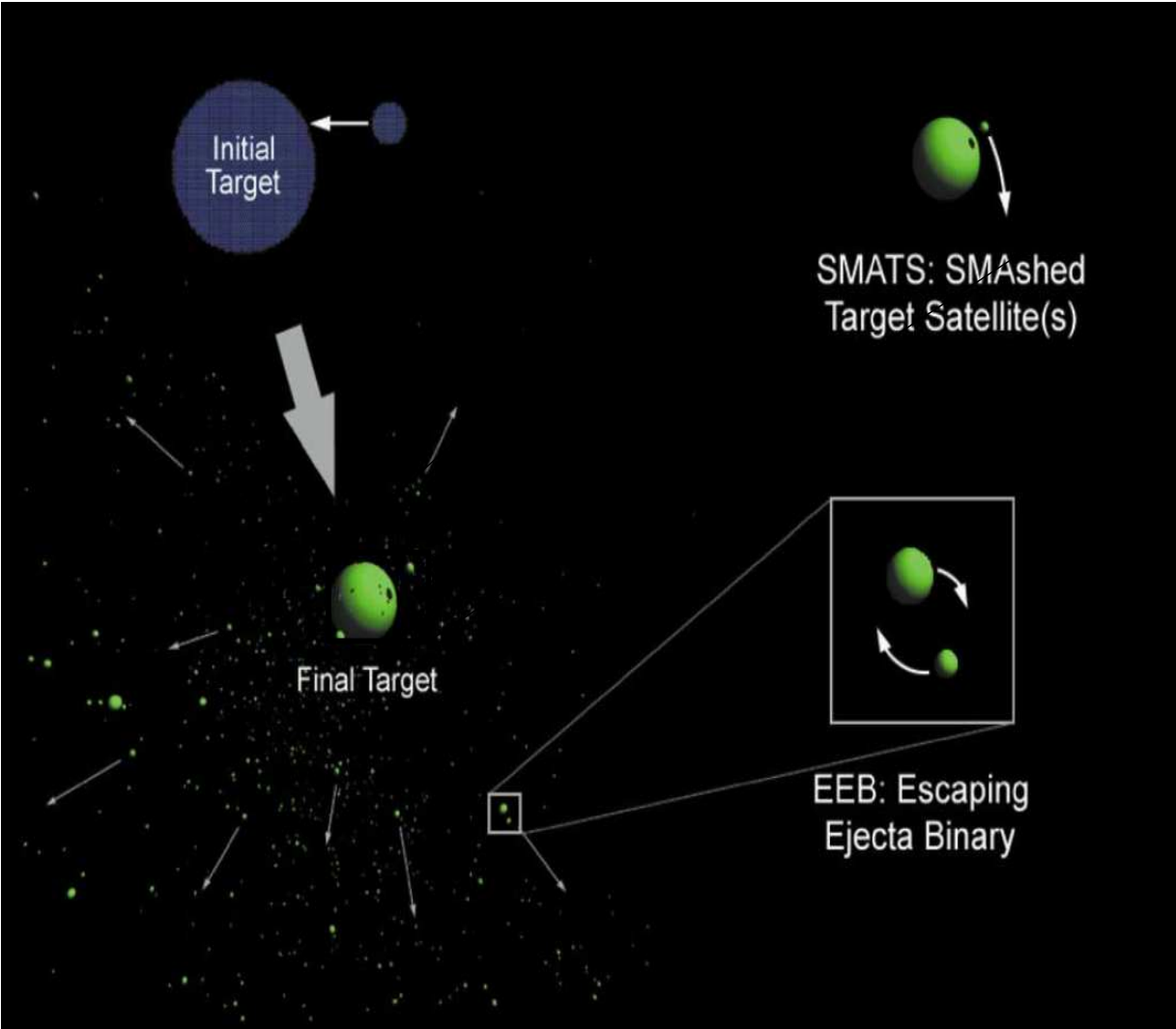


Figure 1: The asteroid collision simulator uses a smooth particle hydrodynamic code to model the pressure, temperature, and energies during the initial collision and an efficient N-body code to track the resulting collision fragments for four simulated days. Scientists are interested in two mechanisms that may lead to captured binary pairs: SMATS and EEBs. For SMATS, a collisional fragment is captured in orbit around the remnant of the original target body. For EEBs, a pair of ejected fragments form a gravitationally-bound system. Figure courtesy of Dan Durda [10].

the probabilistic version of *Simple*, which assigns higher utility to unlabeled points that are close to the current SVM decision boundary. In all, 137 queries to the simulator were made by active learning. Of these, 38 resulted in the formation of binaries. Figure 2a shows the 38 success points found by active learning. Figure 2b shows the approximate surface bounding the region of parameter space that produces successful results. The points shown as spheres include the 38 successes found by active learning as well as the 25 successes found in the grid-based trials. The yellow sheet enclosing the points is an iso-confidence surface from the learned SVM.

The active learning runs were stopped after 137 iterations because the predictions of the current SVM and the grid-based ground truth had reached a high level of agreement. However, the end users were visually satisfied with the convergence of results after only 75 active learning runs. Therefore, compared to the grid approach, active learning required half as many runs to achieve end-user satisfaction.

From Figure 2, it is clear that the solution space is of sufficiently complex shape that a finer grid would have been useful to more accurately assess the performance. Unfortunately, we did not have sufficient computer time to run say a $(9 \times 9 \times 11)$ grid (891 runs). As a compromise, we did run additional tests on a subgrid with the area of greatest interest.

One of the grid-cubes was split into much smaller cubes, in particular it was split into $(4 \times 4 \times 4)$ subgrid. The results over this finer resolution grid showed that there was a quite-clear boundary plane intersecting our initial grid-cube. Also, the fine grid results suggested that a $(17 \times 17 \times 21)$ grid would have been desirable for evaluating the results, but this leads to an infeasible 6069 runs of the simulator.

7.2 Magnetospheric Modeling Experiments.

The input space of the magnetospheric modeling simulator was also 3-dimensional. As in the asteroid collision experiments, we first ran a grid of 729 points $(9 \times 9 \times 9)$, which were regularly-spread over the input space, to establish ground truth. The magnetospheric simulator offered few options for reducing the runtime — each run required about 4 hours on a single 1GHz Beowulf cluster node.

The output of the magnetospheric simulator earned a high grade if the weighted difference of each pixel value between a generated image and observation was low and the final image pixel distribution was smooth. Results were normalized against a range of expected values (derived from some earlier grid runs), and runs that yielded values below a threshold of 0.5 or 0.6 were deemed successful. (Note that the threshold affects how

small and irregularly shaped the region of success will be).

We used three different input data sets to effectively create three different inversion problems. For the E2b magnetospheric inversions, the active learning tools resolved the boundaries of the primary solution region very well. Only a few points (5%) out of the 729 labeled through grid runs were misdiagnosed by the SVM that was learned after 145 calls to the simulator. In fact, as shown in Figure 3a, the SVM that was learned after just 24 calls to the simulator achieved 90% agreement with the ground truth (grid) results. For the E2 dataset, the active learning results were a bit weaker. As shown in Figure 3b, it took nearly 41 calls to the simulator to find the first success point and approximately 250 calls to reach the 90% agreement level, potentially because the solution region had a more complex “stringy” shape.

7.3 Hyperparameter Experiments.

Figure 4a shows the result of an experiment on choosing the SVM hyperparameters for the asteroid collision simulator. The curves prefixed by *rbf* and *poly* in the legend show the *true accuracy*⁶ $\rho_{\lambda}^{(n)}$ versus n for a particular choice of hyperparameters (used throughout all the trials). If you were fortunate and guessed “poly 9 1” or “poly 13 1” at the start (the latter means using a polynomial kernel with degree 13 and constant 1), then you would get the best performance. However, if you picked “poly 17 1”, “poly 5 1”, or “poly 2 1”, you would get poor performance, with no increase in the (low) predictive accuracy even after 75 simulator trials. The curve labeled *multi-kernel instantaneous x-val* uses the instantaneous x-val method to estimate the accuracies of the SVMs based on various hyperparameter combinations and then combines the wishes of these SVMs to select the next point to label as described in Section 4. The curve labeled *multi-kernel historical x-val* is similar, but uses the Bayesian approach to estimate the accuracies.

It is clear that making a good choice for the hyperparameters affects the accuracy and learning rate. Choosing a kernel up front can yield results that are very good or very bad; unfortunately, there is not an easy way to know in advance what you will get. The cross-validation approaches, while not quite as strong as picking the best hyperparameters by luck, both yield solid results. The Bayesian approach (historical x-val) appears to work a bit better than the instantaneous x-val approach (not only is the accuracy better, but

⁶Normally, the true accuracy could not be computed during the course of an experiment, but here, we went back and computed it in post-processing using the ground truth labels that were acquired via the experiment.

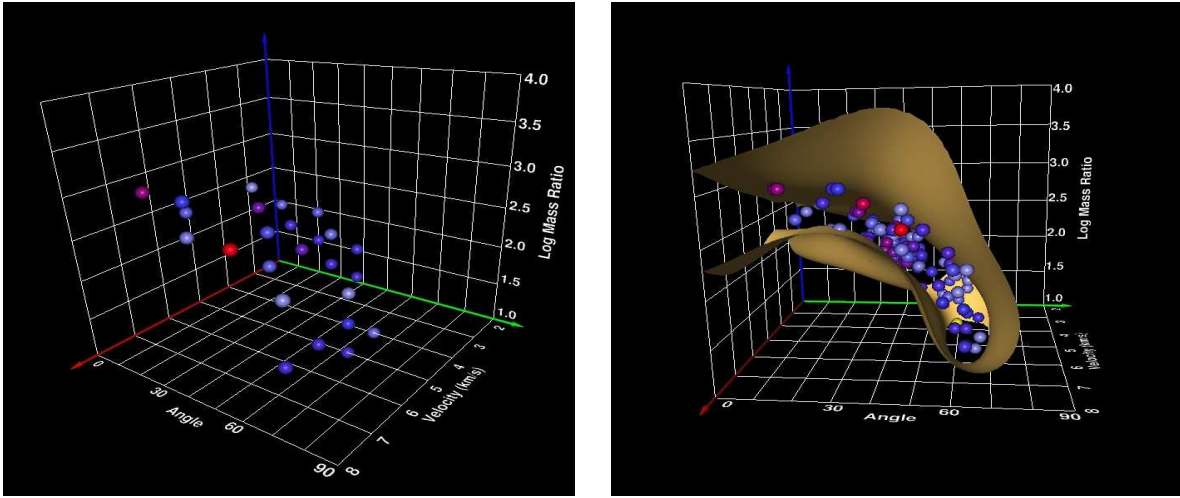


Figure 2: (a) For the asteroid collision application, active learning found 38 success points (shown here as spheres) out of 137 simulations. Based on these 38 points, active learning is able to learn an SVM that accurately predicts the labels at the 150 grid points. (b) Approximate surface bounding the region of parameter space that produces successful results. The points shown as spheres include the 38 successes found by active learning plus the 25 successes found by the grid-based approach. The yellow sheet enclosing the points is an iso-confidence surface from the learned SVM.

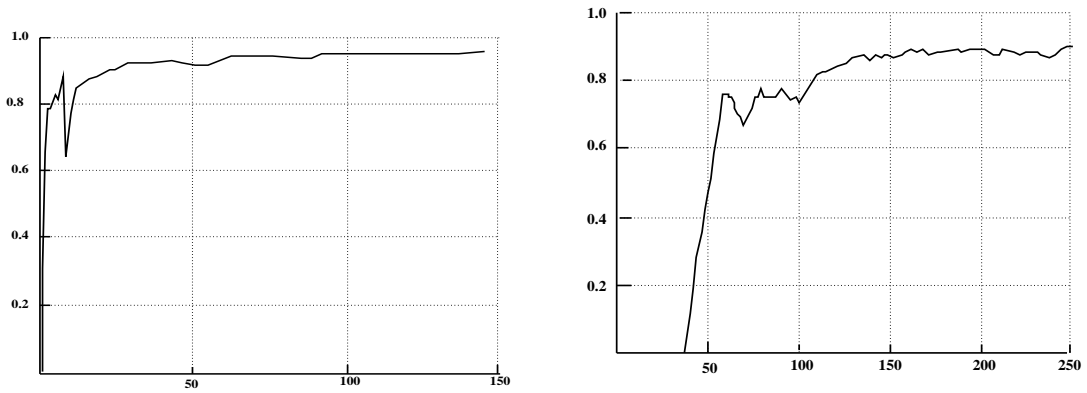


Figure 3: (a) Smoothed percentage of correct predictions by active learning for the E2b magnetospheric simulations. (b) Result for E2 magnetospheric simulations.

the variance is smaller, which is not apparent from the graph).

Figure 4b shows the corresponding results for the magnetospheric modeling simulator (E2 data). Qualitatively these results are similar to those from the asteroid simulator: using the multi-kernel approach with cross-validation for kernel selection is not as good as being psychic and choosing the best kernel up front, however it is definitely better than choosing a poor kernel.

8 Future Work

8.1 Label Cost. Current active learning algorithms often assume that the cost of acquiring a label is constant and independent of the instance to be labeled. For simulations, the label cost tends not only to be high, but highly variable depending upon the instance to be labeled. It would be valuable to develop a proxy for the cost associated with each label by constructing a function that can estimate the cost given the costs of nearby labels. The active learning process could then consider the estimated cost as an internal bias when selecting new points to query, choosing lower cost over higher cost, all other selection criteria being equal. Hence, the goal for selecting new parameter combinations is no longer just to maximize information about the SVM decision surface, but to perform this maximization at minimum cost.

Likewise, current active learning algorithms assume that all points in the parameter landscape are equally probable. Depending upon what a simulation is modeling, this flat probability surface is rarely true. In the real world, some input parameters or combinations are far more likely to occur than others. For example, for asteroid impact simulations, the probability distribution of the “impact angle” parameter peaks at 45 degrees and drops to nearly 0 at 0 and 90 degrees. The set of impacts near 45 degrees are more common and potentially more valuable to study. The active learning process should also consider the prior probability of a point in the landscape as an internal bias when deciding which points to label next.

8.2 When to Stop Active Learning? Another important consideration, for an end-user, is knowing when to stop the Active Learning iterations. Without a grid of runs to assess the results, how does the user know when the landscape has been adequately characterized? We do not have a principled answer to this question, although seemingly we would want to check the predictive power on future samples from the input space that were not used in forming the SVM (generalization error).

8.3 Introspection and Early Stopping of Simulations A simulator is inherently different than an oracle in that the internal state of the simulator is usually directly accessible. Rather than paying the full cost for running a trial to completion, it may be possible for the simulator to use introspection, looking at its internal state, to decide to prematurely terminate (or checkpoint) simulations that appear to be less interesting in favor of simulations that appear more promising.

Of course, an algorithm that looks at early, incomplete simulation results might sometimes make an incorrect guess, for example by guessing that an asteroid collision simulation will not result in a binary pair, when in fact one will eventually occur. Therefore, the overall strategy must allow for a balance between the nominal behavior (run all simulations to completion) and the more aggressive behavior (stop some simulations early if the results don’t look like they will be interesting). We could also view the simulator as offering a probabilistic label with an associated confidence.

9 Conclusion

Knowledge discovery from simulators is a rich, emerging application area. Some aspects of this problem, such as post-analysis of output data, can be largely addressed with existing data mining techniques; however, other aspects, such as the landscape characterization problem, require entirely new approaches. In this paper, we utilized several variants of active learning to reduce the number of simulation trials required to identify regions of input space that yield desired output space behaviors. In experiments with actual, large-scale scientific simulators, our experience to date shows that the active learning approach can increase the efficiency of landscape characterization over a standard gridding approach by 2x to 6x. In addition, the active learning approach more accurately captures the boundaries of the region of interest than a grid-based approach.

Acknowledgement

This work was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration with funding provided by the Intelligent Systems and Applied Information Systems Research Programs. The authors wish to thank Dan Durda, Bill Bottke, and Joerg-Micha Jahn of SwRI for providing expertise and assistance with the two science simulators, as well as Rebecca Castaño of JPL for keeping the project moving forward under unusual circumstances.

References

