Active Learning for Sampling in Time-Series Experiments With Application to Gene Expression Analysis

Rohit Singh RSINGH@MIT.EDU
Nathan Palmer PALMER@MIT.EDU
David Gifford DKG@PSRG.LCS.MIT.EDU
Bonnie Berger[+] BAB@MIT.EDU
Computer Science and Artificial Intelligence Lab., Massachusetts Institute of Technology, Cambridge MA, USA

Ziv Bar-Joseph[*] ZIVBJ@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA, USA

Abstract

Many time-series experiments seek to estimate some signal as a continuous function of time. In this paper, we address the *sampling problem* for such experiments: determining which time-points ought to be sampled in order to minimize the cost of data collection. We restrict our attention to a growing class of experiments which measure multiple signals at each time-point and where raw materials/observations are archived initially, and selectively analyzed later, this analysis being the more expensive step. We present an active learning algorithm for iteratively choosing time-points to sample, using the uncertainty in the quality of the currently estimated time-dependent curve as the objective function. Using simulated data as well as gene expression data, we show that our algorithm performs well, and can significantly reduce experimental cost without loss of information.

Supplementary Webpage: `http://theory.csail.mit.edu/tsample`

1. Introduction

Time-series gene expression experiments are used to study a wide range of biological systems. They measure the relative expression levels of genes in a cell. Such data is used to identify the complete set of genes that participate in the system over time as well as infer causal relationships and interactions among these genes (Qian et al., 2001). A large fraction— over 30%— of gene expression experiments are time-series studies (Stanford Microarray Database, `http://genome-www5.stanford.edu`).

Expression profiles, i.e. continuous functions modeling variations in expression levels, for these systems are estimated by performing multiple microarray experiments over a period of time (Spellman et al., 1998). At a single time-point, each microarray experiment measures expression levels of many genes, often tens-of-thousands, simultaneously. However, each measurement is costly (in some cases as much as $1000), making it crucial to use as few experiments as possible to estimate the expression profile.

Although our primary focus in this paper is on gene expression experiments, the experiment design problem we discuss is general. There exist a wide variety of time-series experiments that measure hundreds, or even thousands, of signals at each time-point. In many cases, each additional sampling location significantly adds to cost or technical complications. Examples beyond gene expression analysis include atmospheric data— where hundreds of weather patterns are tracked over time (Chudova et al., 2003), and clinical data (e.g., blood measurements) that cannot be obtained more than a few times from each individual.

The high cost of data collection per experiment makes it important to pick the minimal set of sampling locations while still ensuring that an accurate representation of the underlying system may be reconstructed for each of the multiple signals being measured. Over-sampling is expensive and time-consuming, and diverts resources that could be used for performing complementary studies. On the other hand, under-sampling will lead to inaccurate estimation, and key features of the system's

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.
[+]Department of Mathematics, MIT
[*]Corresponding Author

time-dependent response may be missed.

A principled approach to sampling in such problems is essential. In the context of gene expression studies, the lack of such an approach has made it hard to compare the quality of datasets generated from related but independent studies. Indeed, multiple experiments from the same lab, reported in the same paper, and directed at the same biological system have utilized different sampling rates. Such inconsistencies can lead to incomplete results.

One of our key insights is that many of the datasets described above share a common property: that is, the ability to sample the system in an online fashion, particularly *between* two previously-sampled time-points. For example, in the case of microarray experiments, biological samples may be frozen and stored prior to hybridization. This allows researchers to extract and archive the biological samples at a very high rate, then make decisions about which samples to hybridize at a later time. We remark also that the expensive part of a microarray experiment is the hybridization step, rather than the act of extracting the sample. Similarly, when monitoring clinical trials, blood samples from each patient visit can be stored and (expensive) diagnostic tests performed later.

In this paper, we present an online algorithm that uses active learning to determine an appropriate sampling strategy for these experiment scenarios. One of our core contributions is the development of a principled, efficiently-computable objective function for measuring the uncertainty in the estimated signal. An important feature of this measure is that it enables us to sample from non-uniform functions effectively by using local cross validation (LCV) to focus in on local signal variations. In Sec 5 we briefly discuss how our approach can be easily transferred to the general case of sampling from and estimating any continuous function of one independent variable, x (i.e., $y = f(x)$ where x need not be time).

We demonstrate the efficacy of our method by applying it to both the problem of reconstructing a function from noisy simulated data, and that of selecting time-points for observation in a time-series gene expression experiment. In the latter case, by testing against an existing dataset, we show that our algorithm could have significantly reduced the data-collection cost without appreciable loss in the information extracted. Our method may be extended to do batch processing— sampling multiple time-points per iteration, instead of just one.

Related work: Determining good sampling rates is a well-studied signal-processing problem (Orfanidis, 1995). However, most prior work in this area has focused on reconstructing individual signals. In our case thousands of signals are measured simultaneously. Because the signals are often correlated, clustering and profile estimation are interdependent, rendering traditional signal-processing methods ineffective.

Recently, several approaches have been proposed for reconstructing continuous representations for the types of datasets considered here (James & Hastie, 2001; Bar-Joseph et al., 2003; Chudova et al., 2003). However, these methods sidestep the issue of sampling strategies altogether, assuming that the given sampling rate is correct. Although there have been discussions about the importance of sampling rates for modeling biological systems (Bay et al., 2003) and on designing microarrays experiments (Baldi & Hatfield, 2002), we are not aware of computational approaches to the sampling problem for time-series expression studies.

A recent paper (Lizotte et al., 2003) discusses the use of active learning for minimizing data-collections costs while training Naive Bayes classifiers. A problem variant we consider here,CostThresh, is similar in spirit.

2. Problem Formulation

Assume that we have m signals per time-point. For example, m might be the number of genes in expression experiments, or the number of patients in clinical trials. Our goal is to identify the minimal set of experiments required in order to generate high-quality estimates of the m time-dependent function profiles. We describe two variants of this problem.

The first variant, ErrThresh, is suitable when the primary concern to meet a pre-specified error threshold for the estimated function profiles:

Problem ErrThresh
Input: A set of m observables per time-point; a sequence of N time-points $T_N = (t_1^n, \ldots, t_N^n)$ at which sampling can potentially be performed; and an associated error threshold C_e. Of these N time-points, an initial round of sampling has been done at S time-points $T_S = (t_1^s, \ldots, t_S^s)$ where $t_1^s = t_1^n$ and $t_S^s = t_N^n$. (Observations at the first and last time-points are necessary for "anchoring" the estimated function profile.)
Output: A set of L time-points $T_L = (t_1^l, \ldots, t_L^l)$ where $t_1^l = t_1^n$, $t_L^l = t_N^n$ and $T_S \subset T_L \subset T_N$ such that the function profiles estimated from the experiments done at these time-points have an expected error less than C_e.

The ideal error measure (for C_e) is almost always unavailable, since this measure would require comparing the estimated function to the true (continuous) func-

tion, the latter being unavailable in almost all cases. Instead, we describe an error function (Section 4.1) which performs reasonably well on common problems, though it is not theoretically ideal.

The second variant, CostThresh, is more pertinent when a fixed (and limited) set of resources (e.g., money) must be used in the best possible way.

Problem CostThresh
Input: Same as ErrThresh, except that C_e (the error threshold) is not supplied. Instead, it is specified that only a maximum of $K \leq N$ samples may be picked.
Output: A set of K time-points $T_K = (t_1^k, \ldots, t_K^k)$ where $t_1^k = t_1^n$, $t_K^k = t_N^n$ and $T_S \subset T_K \subset T_N$ at which these experiments should be performed to estimate the function profiles.

3. Algorithm

We first describe our continuous representation model, and then the statistical and computational techniques for measuring uncertainty in the estimated function profiles. Finally, we describe an online algorithm that ties these techniques together under an active learning framework and can be adapted to solve both ErrThresh and CostThresh.

3.1. Modeling Continuous Functions

Suppose that the true function is $f^*(t)$. Accounting for measurement errors, sampling at times $t_1, \ldots, t_n$ leads to the observations:

$$y_i = f^*(t_i) + \epsilon_i, \;\; \epsilon_i \sim N(0, \sigma^2), \;\; 1 \leq i \leq n \quad (1)$$

To obtain the optimal approximation $\hat{f}(t)$ to $f^*(t)$, we minimize the *penalized least squares error* (PLSE):

$$\hat{f} = arg\min_f \sum_{i=1}^{n} [y_i - f(t_i)]^2 + \lambda \int_{t_1}^{t_n} [f''(x)]^2 \mathrm{d}x \quad (2)$$

The first term measures the familiar least-squared error. The second term enforces a smoothness constraint on $f(t)$. The smoothing parameter λ controls the "bumpiness" of $f(t)$: $\lambda = 0$ leads to an interpolating curve; $\lambda \to \infty$ results in $\hat{f}$ being a straight line. An advantage of PLSE is that it can be proven that, for $\lambda > 0$, the optimal $\hat{f}(t)$ under PLSE is a cubic smoothing spline (De Boor et al., 2001). Such splines have good numerical properties; their computational manipulation is well-understood (e.g., in the graphics community) and can be done very efficiently (in $O(n)$ time).

For spline regression (i.e. fitting), we use Bar-Joseph *et al.*'s probabilistic smoothing splines model (Bar-Joseph et al., 2003). As we describe in detail later (Sec 3.5), this model can handle multiple correlated signals per time-point. For now, we just remark that the smoothing parameter λ (Eqn. 2) in our formulation directly translates to the number of control points p of the uniform knot vector used in the Bar-Joseph model. In what follows, we work with p instead of λ.

The smoothing characteristics of a curve can also be captured by a *smoothing matrix* (Cummins et al., 2001). Such a matrix A is defined by the relation $\hat{Y} = AY$ where $\hat{Y} = (\hat{f}(t_1), \ldots, \hat{f}(t_k))$ are the function estimates at $T_k = (t_1, \ldots, t_k)$, and $Y = (y_1, \ldots, y_k)$ are the actual observations. A_{ij} can then be thought of as the influence of the observation y_j on the estimate $\hat{f}(t_i)$. For example, for an interpolating curve, $A_{ii} = 1$ and $A_{ij} = 0$ for $i \neq j$. In our formulation, A is a simple function of p and T_k that can be derived from elementary Spline Theory (see References & Notes).

Optimal Smoothing Parameter: During spline regression using the Bar-Joseph model, the smoothing parameter p is a free parameter. For our purposes, however, choosing the *optimal* p is important for achieving a balance between underfitting (caused by an overly-smooth curve; small p) and overfitting (resulting in a far-too-bumpy curve; large p).

We use cross-validation (CV) to evaluate each value of p and select the optimal p^*. For each p and a set of observations at k time-points: perform k spline-regressions on a reduced $(k-1)$-dataset, leaving out point t_i (for $i = 1, \ldots, k$). Evaluate the error summed across the k curves and choose p^* with the lowest error.

$$p^* = arg \min_{p=4\ldots k+3} \sum_{i=1}^{k} [\hat{f}_{-i}^{(p)}(t_i) - y_i]^2 \quad (3)$$

where $f_{-i}(t_i)$ = leave-t_i-out cross-validation estimate of f, evaluated at the skipped timepoint, t_i.

3.2. Quantifying the Uncertainty in Estimates

There are two main considerations when choosing a time-point to sample: sampling in regions with too-few time-points, and denser coverage in areas where current observations indicate that more detail is needed. The latter are often marked by high local curvature in the estimated function. We now describe statistical techniques that capture both of these intuitions.

Confidence Intervals (CI): Given the optimal smoothing parameter, confidence intervals (CIs) for spline-based curves are given by:

$$CI(i) = z_{\alpha/2}\sqrt{\sigma^2 A_{ii}} \quad \text{(Wahba, 1983)} \quad (4)$$

where the desired confidence level is $1-\alpha$ (thus, for 95% confidence $z_{\alpha/2} = z(0.025) = 1.96$); σ^2 is the estimated variance and A is the smoothing matrix corresponding to the optimal smoothing parameter p^* and T_k. σ^2 captures the fitting error during spline regression (see Sec 3.1). The $\sqrt{A_{ii}}$ term formalizes the first intuition mentioned in the previous paragraph: if an estimate $\hat{y}_i$ is supported by multiple observations (i.e., $A_{ii} \ll 1$) then the corresponding CI should be smaller than when the estimate at t_i depends only on the observation at t_i (i.e., $A_{ii} = 1$).

However, the CIs estimated by the above formula do not sufficiently reflect local uncertainties in the estimate: once the optimal smoothing parameter (p^*) has been determined, confidence intervals do not depend on individual observations. In particular, two curves with the same p^*, T_k, and σ^2, but with different observed values, will have the same CIs (see supp. webpage for an illustration). In gene expression experiments that measure a process with non-uniform response rates (e.g., stress response), such global estimates do not adequately take into account the short, yet important, period in which most of the changes occur.

Local Cross Validation: To solve this problem, we use Local Cross Validation (LCV) (Cummins et al., 2001). It is a technique that allows us to account for local fluctuation in our estimates. Given the *global* optimal smoothing parameter p^* for the entire curve, we calculate local smoothing parameters, one per sampled time-point. The local smoothing parameter p^i at t_i is:

$$p^i = \underset{p=p^*,\ldots,k-1}{\operatorname{argmin}} \frac{1}{k} \frac{\sum_{j=1}^{k} (w_j (\hat{f}^{(p)}(t_j) - y_j)^2)}{(1 - \sum_{j=1}^{k} w_j A_{ii}^{(p)})^2} \quad (5)$$

where $A^{(p)}$, for any value p of the local smoothing parameter, is the smoothing matrix corresponding to p and T_k. $\hat{f}^{(p)}(t_1), \ldots, \hat{f}^{(p)}(t_k)$ are the function estimates derived using p and <u>all</u> k observations. Also, $w_j = A^*_{ij}$ where A^* is the smoothing matrix corresponding to T_k and the global smoothing parameter p^*. The intuition behind LCV is to compute p^i by "zooming in" on the curve in the vicinity of t_i, i.e., by disproportionately weighting local observations over others. If the zoomed-in view of the curve looks the same as the overall view, then $p^i = p^*$. However, if y_i is in an area of high curvature, then $p^i > p^*$.

Given LCV-inferred smoothing parameters for each sampled time-point, Cummins *et al.* describe a <u>modified</u> formula for calculating CIs as $CI(i) = z_{\alpha/2}\sqrt{\sigma^2 A^i_{ii}}$ where A^i is the smoothing matrix corresponding to T_k and the local smoothing parameter p^i at t_i. The CIs calculated using LCV are more sensible than those calculated using CV. For more details, please see the supplementary webpage.

3.3. Choosing the Next Sampling Location

Our basic intuition is that the uncertainty in the current estimate, as captured by the CIs, can be used to inform future sampling decisions. We proposed a principled active learning approach that captures this intuition. One of our contributions is a predictive model, amenable to active learning, for the sampling problem.

The goal in *active learning* (Tong, 2001) is to build a predictive model $\mathcal{M}$ of the environment. The model is built by iteratively querying the environment, and using the information gained from the response to decide the next query to ask. We associate with the model a loss function $Loss(\mathcal{M})$ that indicates the inaccuracy in the model. Each query $\mathbf{q}_x$ provides an observation $\mathbf{o}_x$ about the model. At any time, one of many queries $\{\mathbf{q_1}, \ldots, \mathbf{q_r}\}$ can be asked. Given the current (imprecise) model $\mathcal{M}$, we can calculate for each query $\mathbf{q}_x$ its expected loss $\langle Loss(\mathbf{q}_x)\rangle = \mathrm{E}[Loss(\mathcal{M}^{\mathbf{o_x}})]$ where the expectation is taken over $\mathbf{o}_x$, the set of possible outcomes of $\mathbf{q}_x$. Of all available queries we then pick the one that minimizes the expected loss.

We now describe our model. $\mathcal{M}$ is the currently estimated function profile $\hat{f}(t)$. For each unsampled time-point t_r, the corresponding query $\mathbf{q}_{t_r}$ is: "what is the observed gene expression value at time t_r?" The observation $\mathbf{o}_{t_r}$ is the value y_r found after experiment.

We define the *Confidence Area* (CA) of a curve $y = f(x)$ as the area of the confidence band around the curve over the entire time-range, i.e., $[t_1, t_N]$. The confidence band, in turn, is the region bounded by the curves $f^+(x) = f(x) + \frac{CI(x)}{2}$ and $f^-(x) = f(x) - \frac{CI(x)}{2}$. Since we have CIs only at certain time-points along the curve, we approximate CA of the curve as the sum of the area of trapezoids marked by CIs at these points. Thus,

$$CA(T_k, \hat{f}) = \frac{1}{2} \sum_{i=1}^{k-1} ((t_{i+1} - t_i)(CI(i) + CI(i+1))) \quad (6)$$

We use the confidence area $CA(T_k, \hat{f})$ of the estimated function profile $\hat{f}$ as the loss function. Computing the expected loss $\langle Loss(\mathbf{q}_x)\rangle$ for a given query involves a complicated integration over spline coefficients using the Bar-Joseph *et al.*'s mixed effects model. It is further complicated by the LCV procedure that uses different spline curves for different points (depending on the selected smoothing parameter). Instead we follow Zhu *et al* (2003) and compute the loss of the expected observation $\langle \mathbf{o}_x \rangle$. This *expected* observation $\langle \mathbf{o}_{t_r} \rangle = \mathrm{E}[\mathbf{o}_{t_r} \mid \mathcal{M}, \mathbf{q}_{t_r}]$ under $\mathcal{M}$ can be calculated very easily: it is simply the prediction as per the current es-

timate, $\hat{f}(t_r)$. We update the data with this expected observation to infer the new (expected) model $\mathcal{M}^{\langle \mathbf{o}_x \rangle}$ and the corresponding loss $Loss(\mathbf{q}_x) = Loss(\mathcal{M}^{\langle \mathbf{o}_x \rangle})$. This can also be done efficiently: the new data-point y_r^{est} falls exactly on the existing estimate $\hat{f}$. Eqn (2) implies that no re-fitting is required; only CIs and CAs need to be recalculated.

To summarize, for each unsampled time-point t_r, we compute the expected observation $y_r^{est} = \hat{f}(t_r)$. We then update the estimate $\hat{f}$ to include y_r^{est}, getting $\hat{f}_{+t_r}$. We add this expected observation to existing data and re-calculate $Loss(\mathcal{M}^{\langle \mathbf{o}_x \rangle}) = CA(T_k \cup \{t_r\}, \hat{f}_{+t_r})$. Finally, we pick the unsampled time-point corresponding to the minimum expected CA.

3.4. Summary: *Algorithm* Choose-Next-Point

Given data from j previously sampled time-points, use it to pick the $(j+1)$-th sampling location:

1. Generate a smoothing function for the j time-points:
   (a) Evaluate all possible smoothing parameters for the spline model using cross-validation.
2. Use the smoothing function to choose the next time-point that should be sampled:
   (a) Compute locally-sensitive confidence intervals over the continuous function at all of the sampled time-points
   (b) Use active learning to suggest the next time-point to sample, based on the confidence intervals

With each iteration, at step 1, the above algorithm computes an error estimate (see Section 4.1) In order to solve ErrThresh, we choose time-points until this error falls below C_e; to solve CostThresh, we continue until K time-points have been chosen.

3.5. Generalizing The Algorithm

Generalizing to Multiple Signals: Consider the general case where m observations (say, for m genes) are made per time-point so that m function profiles $f_1^*(t), \ldots, f_m^*(t)$ need to be estimated. Moreover, many of these function profiles are correlated, and can be clustered in, say, c clusters. We then use Bar-Joseph *et al.*'s (2003) probabilistic smoothing splines model for generating our estimates. In the gene expression context, this method simultaneously clusters the m genes into c clusters and computes the m smoothing splines by taking into account both the observations for the gene and the cluster to which it belongs to. Specifically, this method seeks to maximize the likelihood of the following probabilistic model

$$f_k^*(t) = s(t)(\mu_j + \gamma_k), \quad 1 \le k \le m, \;\; 1 \le j \le c \quad (7)$$

Here, signal k is supposed to be in class j; $s(t)$ is the set of spline basis functions evaluated at t; μ_j is a class-specific control-points vector and γ_k is a signal-specific control-points vector. In order to constrain signals in the same class, the set of signal-specific control points are required to follow a joint distribution such that $\gamma_k \sim N(0, \Gamma_j)$ where Γ_j is a class covariance matrix. The parameters of this model (including class membership) are learned using an EM algorithm.

The free parameters in the Bar-Joseph model are the smoothing parameter p and the number of clusters c. Earlier, we discussed how to pick p. Appropriate values of c are typically picked by prior knowledge. Our algorithm is robust to overestimates of c, because clusters with non-time-dependent genes will be down-weighted (see below). Lower-bounds for c, in turn, can be reliably estimated biologically.

We now discuss how to generalize the calculation of CIs/CAs in the case of experiments with m signals. We could weight all m signals equally, simply summing up the signal-specific CV/LCV scores. However, this fails to ignore the signals whose function estimates have already been accurately determined.

Instead, we extend our existing sampling strategy as follows: identify the signals in the dataset that will significantly benefit from a new set of observations and work with only this subset. Given the entire m-signal dataset, we cluster all m signals into c classes using the Bar-Joseph model. For each of these classes, we calculate cluster-specific CIs and CAs (confidence areas) by computing the average CA for signals in that cluster. We then remove all clusters whose CA is less than a predefined stopping criteria and equally weight all signals in the remaining clusters when choosing the next sample point.

Generalizing to a batch approach In some situations, performing one sampling at a time is inconvenient. Instead, our algorithm can be adapted for batch processing. Starting with a set of sampled points, we run our algorithm to select the next time-point to analyze. Next, we compute the expected values ($y_r^{est} = \hat{f}(t_r)$) for the selected time-point and treat them as if they were observed values. We now re-run the active learning algorithm to select the next time-point and so forth until z time-points have been selected. Sampling at these z locations may then be performed at the same time. The next iteration proceeds similarly. The stopping criteria of the algorithm remain unchanged.

4. Results

We evaluated our algorithm on simulated data as well as data from biological experiments. Due to space restrictions, only a representative subset of results are presented in the paper; the rest are on the supplementary webpage. We first discuss some implementation related optimizations.

Implementation: While cross-validation (CV) is a powerful technique for estimating the smoothing parameter p as discussed above, it is also very time-consuming ($O(k^2m)$ curve-fittings) because we need to re-run our spline assignment and clustering algorithms once for each of the k measured time-points with each iteration of the algorithm. Instead, we use *Generalized Cross Validation* (GCV), introduced by Wahba (Wahba, 1983). GCV approximates the cross-validation score, avoiding the $O(k)$ increase in running time. Theoretically, the GCV score is a good approximation of the cross-validation score, and gives the same optimal p^*, asymptotically (Cummins et al., 2001). In practice too, we have found that the optimal smoothing parameter selected by GCV very often exactly agrees with the parameter selected by CV, even for the short time-series datasets seen in biological/medical contexts. For the simulated dataset discussed below, GCV was much faster (30 minutes for GCV vs. 191 minutes for CV). Furthermore, curves computed for GCV can be reused for LCV. See the supplementary webpage for more details.

4.1. Simulations

When working with simulated data, we know the actual function that generated the observations, and can therefore measure the accuracy of our derived model as the difference between the true function, $f^*(t)$, and the estimated function, $\hat{f}(t)$. In particular, we measure the *true error* e_f^T of $\hat{f}$ as the average difference between the two curves over the time-range of interest, i.e., $e_f^T = \frac{1}{(t_N^n - t_1^n)} \int_{t_1^n}^{t_N^n} |f^*(t) - \hat{f}(t)| \, \mathrm{d}t$.

Generation of Simulated Data: Simulated data was created by adding Gaussian noise to sinusoidal and linear functions. A set of 24 equally-spaced potential sampling locations, were available to each sampling strategy. Each dataset consisted of three clusters, each with 50 co-varying signals. By changing the frequency and positions of the sinusoids, datasets with varying hardness levels were created. In the following discussion, results are reported for three representative datasets, marked as "easy," "moderate," and "hard". See the supplementary webpage for more details and other datasets.

Benchmarking Strategies: We compared the function estimates derived from samples chosen by our algorithm to those obtained through random and uniform sampling strategies. These strategies, in particular uniform sampling, are quite common in biological papers. Also, note that uniform sampling cannot be implemented as an online algorithm, and hence can only be used to solve CostThresh.

Random sampling involves randomly adding an as-yet unselected time-point to the sample set, and can be used with either CostThresh or ErrThresh. In the results presented below, random sampling was repeated 10 times and the scores averaged.

CostThresh Our method performs better (in terms of e_f^T) than both random and uniform sampling, especially when using more difficult cost targets (12 or 15 of 24 total samples) and harder datasets. Table 1 shows a comparison of the true error for the functions derived from samples selected by our algorithm to the true error for the functions derived by random and uniform sampling. We performed the comparison for three different cost thresholds: 12, 15 and 18 samples. We accept the estimate to be a good fit with the original if $e_f^T < 0.15$.

ErrThresh We first need to define an error measure e_f over the estimated functions. Ideally, such an error measure will behave exactly like the true error, e_f^T. For simulated datasets, we *can* measure the true error, and use this to measure performance in ErrThresh problem instances. We show that our online algorithm performs significantly better than the random sampling approach, i.e., requiring fewer samples to achieve the same error (Fig 1). The difference is particularly apparent in the "hard" problem instances.

Designing An Error Measure: When considering real experimental data, the true error measure e_f^T is unknown, so we approximate it with the *GCV-error* measure, which we define as the confidence area of the estimated function calculated using GCV-based CIs. Simulations indicate that GCV-error is a reasonable error estimate: on average, a lower GCV-error corresponds to a lower true error and vice-versa (see Supp. Info.). Some discrepancy between GCV-error and e_f^T is unavoidable, since any error measure designed solely from noisy observations (i.e., without knowing a priori the underlying function) can not completely match the true error. To deal with these fluctuations, we have found the following stopping criterion to work well: keep sampling until, over three successive iterations GCV-error remains below C_e. Fig 2(a) shows how GCV-error decreases as a function of the number of sampled time-points.

We now describe how to set C_e, the error threshold.

TimePoints	12 of 24			15 of 24			18 of 24		
	AL*	Unif.	Rand.	AL*	Unif.	Rand.	AL*	Unif.	Rand.
Easy	**0.095**	0.168	0.228	**0.084**	**0.103**	**0.111**	**0.087**	**0.074**	**0.088**
Moderate	0.219	0.258	0.464	**0.110**	0.169	0.296	**0.085**	**0.086**	**0.115**
Hard	0.257	0.265	0.289	**0.140**	0.242	0.252	**0.111**	**0.106**	0.194

Table 1. Performance of Different Strategies on CostThresh, measured by *true error* (see Section 4.1): we compare the performance of our algorithm (*__AL=ActiveLearn__) against that of uniform sampling and random sampling in the CostThresh instance, i.e., when the number of final samples is fixed beforehand. An error of less than 0.15 indicates a good fit. As the table shows, ActiveLearn outperforms other methods, especially on harder problem instances and with fewer samples. Note that uniform sampling has limited usefulness because it can't be performed as an online algorithm.

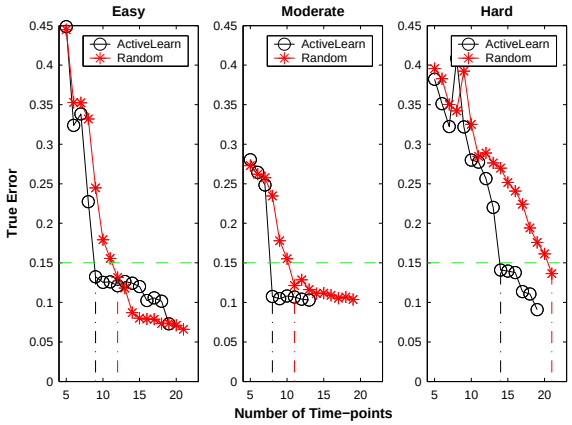


Figure 1. Performance of different sampling strategies on ErrThresh with the *true error* measure e_f^T and a threshold of 0.15. For both ActiveLearn (our method) and Random sampling, the true error with k time-points is shown. The vertical dashed support lines indicate the number of samples at which each strategy crossed the threshold. Our method requires fewer samples to cross the threshold. The difference is especially significant for the hard dataset.

The variance, σ^2, can be estimated from repeat data. Given a time-range $\Delta t = t_N^n - t_1^n$, we then set $C_e = \Delta t(z_{\alpha/2}\sqrt{\sigma^2 a})$ where a represents an estimate for the desired A_{ii}. This formula is inspired by the formula for confidence area (Eqn. 6) as C_e is a threshold for it. Our experiments indicate $a = 0.8$ provides good-quality fits for many datasets.

4.2. Biological Results

In order to test our algorithm on real experimental data, we used the Cdc15 cell cycle gene expression data (Spellman et al., 1998). Expression experiments were carried out at 24 time-points (every 20 minutes between 10 and 70, every 10 minutes between 80 and 240 and every 20 minutes between 250 and 290), making it one of the few publicly-available microarray datasets thought to be over-sampled. The main goal of this experiment was to identify cycling yeast genes. Cycling genes are genes with periodic fluctuations in expression level, likely because of their involvement in cell cycle.

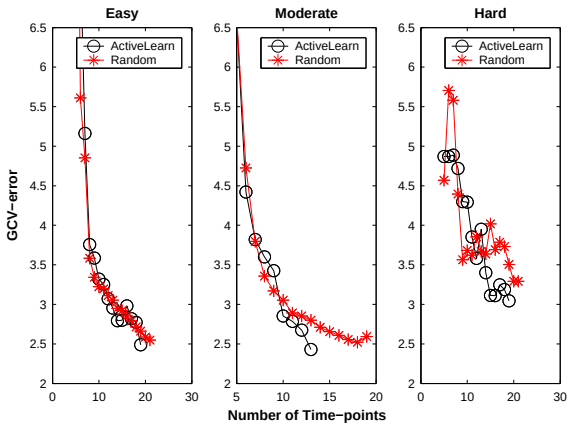
This dataset is an appropriate testbed for our method since identification of cycling genes is a question that lends itself well to the comparison of sampling strategies. Moreover, with the large number of samples collected, our results can be easily validated.

Our goal was to check if our method could save significant cost for very little information loss– a key concern to biologists. In particular, with the Cdc15 dataset we wanted to check if our method could have extracted the full set of cycling genes with fewer samples. To do so, we ran CostThresh on it, once asking for expression profiles based on 18 samples, once for 20 samples, and once allowing all 24 samples to be used. We then used the periodogram method (Wichert et al., 2004), which performs Fourier analysis on the derived function estimates, to identify the top 500 cycling genes from the resultant time-dependent functions. We compared these genes to those which have previously been identified for the Cdc15, Alpha, and Cdc28 datasets. Using 18 time-points (25% savings), $93-94\%$ of the genes reported by our method to be cycling are in agreement with those previously identified in the Cdc15, Alpha, and Cdc28 studies. When using 20 time-points (17% savings), this number increases to $96-97\%$. Furthermore, functions for cycling genes with clear signals were identically reconstructed when we asked CostThresh for its pick of 20 time-points as when all 24 points were used.

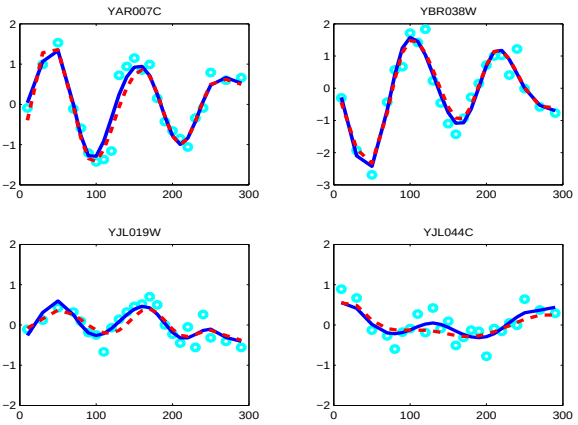
5. Conclusions and Future Work

In this paper we have presented an online algorithm that uses active learning to determine an effective sampling strategy for time-series experiments where the cost of data collection is high. We described an efficiently-computable objective function for measuring the uncertainty in the estimated smoothing splines, and showed how this function may be used with active learning to suggest the next sample point.

Observe that our algorithm can be applied, without modification, to sample from and estimate any continuous function with one independent variable, i.e., $(y_1, \ldots, y_d) = g(x)$ where x need not be time. For example, in a large sensor network, obtaining continuous



(a)



(b)

Figure 2. **(a):** GCV-error as a function of number of time-points. On average, ActiveLearn's GCV-error is lower than Random sampling's, especially in moderate and hard cases. **(b):** Profiles recovered using 20 time-points chosen by ActiveLearn (dashed lines), compared with profiles using all 24 time-points (solid lines). Dots are expression values. Top row: Two cycling genes correctly identified using both sampling strategies. Bottom row: Two genes identified as cycling using 24 points but not when using 20 points. Even in such cases the recovered expression pattern is very similar. However, the low signal-to-noise ratios for these genes makes it hard to unambiguously determine cyclicity.

readings from all sensors has prohibitive communication/power costs (Deshpande, 2004). At the same time, several of these readings maybe redundant. Given some k, our algorithm can help identify the optimal subset of k sensors whose readings, taken together, result in the smallest overall uncertainty in the global observation.

In the future, we would like to develop a variant of our algorithm that can be used to estimate a principled quality metric for time-series data. Then, by applying the objective function presented in this paper to two separate datasets that study the same system, we can compare their relative sampling quality.

References & Notes

Baldi, P. & Hatfield, G. (2002). *DNA Microarrays and Gene Expression.* Cambridge University Press.

Bar-Joseph, Z. *et al.*(2003). Continuous representations of time series gene expression data *J of Comp Bio, 3-4*, 341–356.

Bay, S. D. *et al.* (2003). Temporal aggregation bias and inference of causal regulatory networks *Proc. of the IJCAI Workshop on Learning Graphical Models for Comp. Genomics.*

Chudova, D. *et al.* (2003). Translation-invariant mixture models for curve-clustering. *Proc of the 9^{th} ACM SIGKDD Int'l Conf on Knowledge Disc and Data Mining.*

Cummins, D., Filloon, T. & Nychka, D. (2001). Confidence intervals for nonparametric curve estimates. *J Am Stat Assoc, 96:453*, 233–246.

DeBoor C. *et al.* (2001) A Practical Guide to Splines.

Deshpande, A. *et al.* (2004). Model Driven Data Acquisition in Sensor Networks. *Proceedings of VLDB 2004.*

James, G., & Hastie, T. (2001). Functional linear discriminant analysis for irregularly sampled curves. *Journal of the Royal Statistical Society, to appear.*

Lizotte, D., Madani, O., & Greiner, R. (2003). Budgeted Learning of Naive-Bayes Classifiers. *Proc. of UAI 2003.*

Orfanidis, S. (1995). *Introduction to signal processing.* Prentice Hall.

Qian, J. *et al.* (2001). Beyond synexpression relationships: local clustering of time-shifted and inverted gene expression profiles identifies new, biologically relevant interactions. *J Mol Biol, 314(5)*, 1053–66.

Spellman, P. T. *et al.* (1998). Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Mol. Biol. of the Cell, 9*, 3273–3297.

Tong, S. (2001). *Active learning: Theory and applications.* Doctoral dissertation, Stanford University.

Wahba, G. (1983). Bayesian confidence intervals for the cross-validated smoothing spline. *J Royal Stat Soc, Ser B, 45*, 133–150.

Wichert, S. *et al.* (2004). Identifying periodically expressed transcripts in microarray time series data. *Bioinformatics, 20*, 5–20.

Zhu, X., Lafferty, J., & Ghahramani, Z. (2003). Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. *ICML 2003 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining.*

Computing the Smoothing Matrix A: given smoothing parameter p, a set of time-points $T_k = (t_1, \ldots, t_k)$ and the set of spline basis functions (see $s(t)$ in Eqn. 7), we compute S, a $k \times p$ matrix whose i-th row represents the coordinates of t_i in terms of the spline basis functions. A is then given by $A = S(S^T S)^{-1} S^T$