High-Speed Function Approximation

Biswanath Panda, Mirek Riedewald, Johannes Gehrke
Dept. of Computer Science
Cornell University
{bpanda,mirek,johannes}@cs.cornell.edu

Stephen B. Pope
Dept. of Mechanical & Aerospace Eng.
Cornell University
pope@mae.cornell.edu

Abstract

We address a new learning problem where the goal is to build a predictive model that minimizes prediction time (the time taken to make a prediction) subject to a constraint on model accuracy. Our solution is a generic framework that leverages existing data mining algorithms without requiring any modifications to these algorithms. We show a first application of our framework to a combustion simulation problem. Our experimental evaluation shows significant improvements over existing methods; prediction time typically is improved by a factor between 2 and 6.

1. Introduction

Traditionally, learning algorithms for predictive models have focused on improving prediction quality, e.g., measured by accuracy, root mean squared error (RMSE), area under the ROC curve and other metrics [5]. Research in data mining also considered model building time, i.e., to improve the time it takes to learn predictive models for large data sets. However, there is another aspect of a predictive model, which is usually ignored by learning algorithms– *prediction time* – the time taken by the model to process an input and make a prediction. Let us describe an application where prediction time is important.

High-dimensional function approximation (HFA) for combustion simulations was recently introduced by [9]. Scientists study how the composition of gases in a combustion chamber changes over time due to chemical reactions. The composition of a gas particle is described by a high-dimensional vector. The simulation consists of a series of time steps. During each time step some particles in the chamber react, causing their compositions to change. This reaction is described by a complex high-dimensional function, which, given a particle's current composition vector and other simulation properties, produces a new composition vector. Combustion simulations usually require up to 10^8 to 10^{10} reaction function evaluations. For most experiments, a single evaluation of the reaction function costs tens of milliseconds of CPU time on a modern PC. This makes running large scale simulations computationally infeasible. Scientists address this problem by building computationally less expensive models that approximate the reaction function within a user defined error tolerance of ϵ [10]. Our work is motivated by these specialized solutions for building models with low prediction time.

Combustion represents one of many scientific applications that use approximate models to improve simulation runtime. Recently Bucila et al. [4] observed that ensemble models, while being the most accurate in many scenarios, are often too slow to be used in practice. Low prediction time is also important for online transactions, financial forecasting, fraud detection and numerous other applications.

We propose a meta-learning framework that leverages existing data mining models and algorithms to build models optimized for prediction time. The main idea is a local model approach, where we divide the domain of the learning problem into regions with associated data mining models. The search algorithms in our novel framework select appropriate regions and models across a large space of possible region/model configurations and reduce prediction time while maintaining high accuracy.

The rest of the paper is organized as follows. We define the problem of *Low Prediction Time Learning* in Section 2 and propose a solution framework in Section 3. Sections 4 and 5 discuss an instantiation of the framework and the improvements it provides on combustion simulation workloads. Section 6 discusses related work and Section 7 concludes the paper.

2. Problem Formulation

Assume we are given a probability distribution $\mathcal{D}$ on R^m and two functions $f : R^m \rightarrow R^n$ and M : $R^m \rightarrow R^n$. Let X be a random variable that takes on values from R^m according to distribution $\mathcal{D}$. We say that M is an (ϵ, δ)-approximation of f with respect to $\mathcal{D}$, if on expectation at least $1 - \delta$ fraction of points are within ϵ of the true function

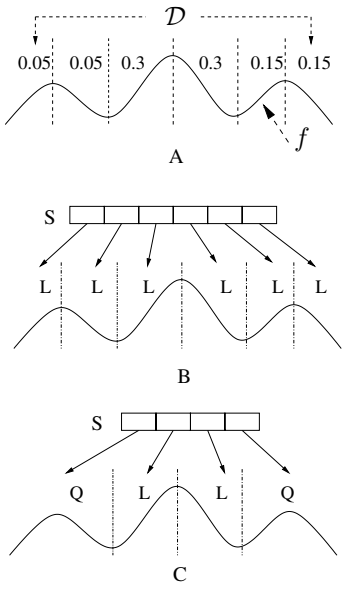

Figure 1. Example

value, i.e.,

$$\mathrm{E}[\mathrm{I}(X)] \geq 1 - \delta, \tag{1}$$

where $||$ is some metric and, I() is an indicator function such that for all $\mathbf{x} \in R^m$, $\mathrm{I}(\mathbf{x}) = 1$ if $||f(\mathbf{x}) - \mathrm{M}(\mathbf{x})|| \leq \epsilon$ and 0 otherwise. Also, let $c_{\mathrm{M}}(\mathbf{x})$ be the time taken by M to compute $\mathrm{M}(\mathbf{x})$.

We can now define the *Low Prediction Time Learning Problem* as follows. Given a set $\mathcal{I} = \{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \ldots, (\mathbf{x}_N, f(\mathbf{x}_N))\}$ find a function M (the *model*) such that M is an (ϵ, δ) approximation of f while minimizing the expected prediction cost

$$\text{ModelCost} = \mathrm{E}[c_{\mathrm{M}}(X)].$$

We now describe a simple example to illustrate why Low Prediction Time Learning is an interesting problem. The example will also provide insights into the general solution described in the next section. Suppose we want to approximate the one-dimensional function f shown in Figure 1(A) within a specified (ϵ, δ) error constraint for the distribution $\mathcal{D}$ shown in the figure. Further assume that we have a set of model types denoted by $\mathcal{M}$ that can be used to approximate the function. Let this set consist of polynomials up to degree 10, that is

$$\mathcal{M} = \{\sum_{i=0,n} a_i \cdot x^i | n = 0, 1, \ldots, 10\}$$

For simplicity, assume the cost of evaluating a polynomial of degree n is equal to the number of multiplication operations, i.e., it is $2n - 1$.

Suppose the true function f is a polynomial of degree 10. Then it is clearly possible to approximate f with a polynomial of degree 10 with (ϵ, δ) error. If we approximate f using a polynomial of degree 10, then the model will take 19 time units per prediction.

Observation 1: Assume f can also be approximated within (ϵ, δ) by a simpler and cheaper to evaluate 6th-degree polynomial. This reduces model cost to 11 time units per prediction. We call this the *Accuracy-Prediction Time Tradeoff*.

Observation 2: In part (B) of Figure 1 the function domain has been divided into 6 parts and a polynomial of degree 1(called L, for "linear", in the figure) is fit in each part. Assuming that for all points in a particular partition the linear model in that partition approximates the function within (ϵ, δ), this set of linear models defines another model that overall satisfies the (ϵ, δ) constraint. However, now the prediction time is not just an evaluation of a polynomial, but actually involves two steps. Given a query point, we first have to find the partition that contains the point (*search time*) and then evaluate the polynomial in the partition (*approximation time*).

In order to find a partition containing the query point, we need a search structure S on the partitions. In this example we use a simple linear list S as shown in part B of the figure. For a given query point, the list is scanned until the corresponding partition is found. For simplicity we assume that the search cost is equal to the number of list elements accessed. Hence for the overall prediction time we obtain on expectation $0.05 \cdot 1 + 0.05 \cdot 2 + 0.3 \cdot 3 + 0.3 \cdot 4 + 0.15 \cdot 5 + 0.15 \cdot 6 = 3.9$ units for search and 1 unit for evaluating the corresponding degree-1 polynomial, for a total cost of 4.9 units per prediction.

Fitting a polynomial of degree 6 resulted in a model with no search time but high approximation time. Partitioning the domain and using a linear model in each partition results in high search cost and low approximation cost. We call this the *Search-Approximation Time Tradeoff*.

Finding the best set of region-model pairs is challenging as illustrated by Part (C) of Figure 1, where a different partitioning of the function and using polynomials of order 1 and order 2 (called Q, for "quadratic", in the figure) results in a lower prediction time model (4.6 units). Our technique reduces model prediction time by exploiting the tradeoffs discussed here.

3 Algorithmic Framework

In this section we formalize the ideas of the previous example and discuss how to explore the design space of possible regions and models in the function domain.

3.1 Model Definition

A region-model M for a function $f : R^m \rightarrow R^n$ consists of a set of convex regions $\mathrm{R} = \{r_i | r_i \subseteq R^m\}$, stored in some search structure S, and a mapping Q of regions to standard data mining models such that $\forall r_i \in \mathrm{R} : \mathrm{Q}[r_i] = m_i$. Here m_i is an instantiation of a model type in $\mathcal{M}$, where $\mathcal{M}$ is a set of types of data mining models (e.g. SVMs, neural networks or decision trees).

The search structure S supports a Lookup(S,$\mathbf{x}$) operation that returns a region $r \in \mathrm{R}$ containing $\mathbf{x}$. Given a query point $\mathbf{x}$ the prediction process consists of the following steps: (1) find $r = \mathrm{Lookup}(\mathrm{S}, \mathbf{x})$, (2) then select $m = \mathrm{Q}[r]$, and (3) compute prediction $m(\mathbf{x})$. We can now revisit the notion of an (ϵ, δ)-approximation of a function f with respect to a region-model by redefining I() in Eq. 1 as $\mathrm{I}(\mathbf{x}) = 1$ if $||f(\mathbf{x}) - \mathrm{Q}[\mathrm{Lookup}(\mathrm{S}, \mathbf{x})](\mathbf{x})|| \leq \epsilon$ and 0 otherwise.

As described earlier, the prediction time per query consists of two costs: search time and approximation time. Let $\mathrm{s_S}(\mathbf{x})$ be the time taken by Lookup to find a region r containing $\mathbf{x}$ using search structure S. Similarly, let $\mathrm{a}_m(\mathbf{x})$ be the time taken to compute an approximation using model $m = \mathrm{Q}[r]$. Then the expected total prediction time per query can be written as $\mathrm{ModelCost} = \mathrm{E}[\mathrm{s_S}(X) + \mathrm{a_{Q[r]}}(X)]$.

Notice that there might be (R,Q) configurations where some query points are not covered by any of the regions in R, i.e., Lookup(S,$\mathbf{x}$) returns no result. We assume that these query points are evaluated using an expensive ground truth model with high prediction time[1].

3.2. Algorithms

Let $\mathcal{I}$ denote a set of input points with known function values. We partition this set into a training set ($\mathcal{T}$) and a validation set ($\mathcal{V}$) for model building.

An exhaustive exploration of all possible combinations of region partitioning, model used for each region, and index for managing regions, is practically infeasible. To reduce the complexity, we divide the problem into subproblems. In particular, our algorithm has two major steps:

1. Generate a set of regions and find the best model for each region.
2. For each index structure under consideration, select the set of region-model pairs that minimizes expected prediction time for this index. Return the best solution.

These two steps that we call *Region-Model Candidate Set Selection* and *Region-Model Selection* are discussed briefly.

[1] In most applications there exists an expensive accurate model, e.g., the reaction function for the combustion problem.

Region-Model Candidate Set Selection: Any subset of points in $\mathcal{T}$ could be connected as a candidate region, resulting in a number of regions exponential in the training set size. We therefore have to resort to heuristics for generating "the most promising" candidate regions. To reduce the search space, without being overly restrictive, we propose the following general approach. Assume we are given a set of relatively small regions, which we refer to as *base regions*. These base regions could be obtained from a regular grid partitioning of the space, from the leaves in a regression tree [3], or based on ISAT's regions of accuracy [10]. The base regions do not need to be disjoint. We restrict region candidates to be either base regions or larger *derived* regions, which are the union of some base regions that are *near* each other. A concrete algorithm is discussed in Section 4.

For each region under consideration, base region or derived, the next step is to find the lowest prediction time model of type in $\mathcal{M}$ for the region that satisfies the (ϵ, δ) error constraint (measured on validation set $\mathcal{V}$).

Region-Model Selection: The region-model generation algorithm produces a set with elements of the form $(r_i, m_i, \mathrm{t}_{m_i})$, where r_i is a region, m_i a model type, and t_{m_i} the approximation time of the m_i. We call this set of region model pairs RM. Notice that each of the models in RM has to satisfy the (ϵ, δ) error constraint for its region. Region-model selection involves selecting a subset of RM and initializing a model M (as defined in Section 3.1) that has lowest prediction time. Therefore, selection finds a model that minimizes $\sum_{\mathbf{x} \in \mathcal{V}} (\mathrm{s_S}(\mathbf{x}) + \mathrm{a_{Q[r]}}(\mathbf{x}))$.

Several factors make the region-model selection problem difficult. First, lookup cost in a search structure depends on the properties of the regions it stores like their dimensionality, overlap, and orientation. If multiple regions in search structure S contain a given query point, then approximation cost depends on the region-model pair that will be finally used in the prediction. These issues aside, we can show that even if we make very restrictive assumptions about search time and approximation time, the region model selection problem is NP-hard.

Due to the complexity of the selection problem, we have to resort to heuristics for solving it. We propose to use a greedy heuristic, shown in Algorithm 1. The algorithm starts out with an initial solution of base regions that is biased toward high search cost and low approximation cost. In each step the algorithm replaces a set of regions in the current solution with a larger region from the set of candidate regions, such that the larger region covers all the removed regions. This is done greedily by selecting the region that brings about the largest reduction in prediction time. The algorithm stops when no more improvement is possible.

Notice that Algorithm 1 assumes the existence of a cost

Algorithm 1 : Greedy Region Selection

Require: RM, Validation Set $\mathcal{V}$, Cost function $\mathcal{C}$

1: Sol ($\subseteq$ RM) = $\{(r_i, m_i, \mathrm{t}_{m_i}) | r_i$ is a base region$\}$
2: Cost = $\mathcal{C}$(Sol)
3: **while** Cost improves **do**
4: TempSol={}
5: **for all** $(r, m, \mathrm{t}_m) \in S \wedge (r, m, \mathrm{t}_m) \notin$ Sol **do**
6: Rem = $\{(r_i, m_i, \mathrm{t}_{m_i}) | (r_i, m_i, \mathrm{t}_{m_i}) \in \text{Sol} \wedge r_i \subseteq r\}$
7: tSol$_r$ = Sol + (r, m, t_m) - Rem
8: tCost$_r$ = $\mathcal{C}$(tSol$_r$)
9: TempSol = TempSol $\cup$(tSol$_r$, tCost$_r$)
10: **if** $\exists$(tSol$_r$, tCost$_r$) $\in$ TempSol s.t. tCost$_r$ $<$ Cost **then**
11: (Sol,Cost)=(tSol$_r$, tCost$_r$)
12: S= Regions in Sol, Q= Region-Model map for Sol
13: **return** S,Q

function ($\mathcal{C}$), which, given a set of region-model pairs and a validation set $\mathcal{V}$, returns the prediction time of the best model that can be created using the given region-model pairs. We will discuss this in more detail in the next section.

4. Instantiations

There are many ways to instantiate the above framework, differing in how base regions are generated and merged and the search structure used to store the regions. Due to the lack of space we only discuss a general instantiation where each individual point in $\mathcal{I}$ is a base region and define a merge that creates regions enclosing the 1, 2,..., n nearest neighbors of a point. This creates fairly general regions in terms of shape, size, and overlap; some multidimensional index is used as the search structure (S). We discuss a variation of this idea for the combustion application where scientists build models with flexible region definitions. The solution easily generalizes to many other supervised learning problems.

The ISAT algorithm used by the domain scientists [9] approximates the combustion reaction function by a set of (possibly overlapping) high-dimensional ellipsoids with linear models inside these ellipsoids. These regions are obtained based on selective evaluations of the reaction function, which is the ground truth model for this application.

To ensure that the regions satisfy the model definition in Section 3.1, we use a slightly modified version of the ISAT algorithm. The main modification is a stricter error control mechanism that periodically checks existing regions in the model and updates region boundaries to not include parts of the space where the model is producing poor approximations. Studies also indicated that hyper-rectangular regions work at least as well as ellipsoids, we will therefore use hyper-rectangular base regions. In the rest of the paper, this modified algorithm is referred to as ISAT.

Domain scientists also observed that their long-running simulations ($> 10^9$ queries) almost always have the following two properties. First, the future query distribution of the simulation can be fairly accurately estimated after a few million queries. Second, simulation time is dominated by model prediction time, i.e., model construction and maintenance time are negligible. We describe the instantiation of our framework for such simulations.

Without loss of generality we model the simulation as a 2-phase process. During the first phase (a few million queries) the ISAT algorithm is run. This algorithm produces a set of base regions in the function domain with a similar model in each region. In order to create this set of region-model pairs, the ISAT algorithm has to evaluate the reaction function for some query points. These points will be used as the training and validation data for our technique ($\mathcal{I}$). Then in the second phase we apply our framework using $\mathcal{I}$ as the input data set and build a new model optimized for prediction time that is used for the rest of the simulation.

Our instantiation for the combustion problem starts with the set of regions created by the ISAT algorithm during phase one as the base regions. Larger regions are created by merging a base region with its nearest neighbors. Specifically, for each base region r, we add the following derived regions: r merged with its first nearest base region, r merged with its two nearest base regions, and so on until some upper limit n of neighbors. Duplicate derived regions are eliminated. We define a derived region as the smallest bounding hyper-rectangle of the merged base regions. Conceptually, we could use individual points in $\mathcal{I}$ as base regions but if cardinality of $\mathcal{I}$ is large this would make nearest neighbor search costly.

Having defined the region creation process, we can use the algorithms introduced in the previous section, provided we have defined a cost function $\mathcal{C}$.

Cost Function ($\mathcal{C}$): For high dimensional indexes, it is difficult to accurately estimate the search cost of a query just based on the set of regions to be stored, without actually building the index. Unfortunately, building the index for each iteration of the region selection algorithm (step 8 in Algorithm 1) is very expensive. We illustrate this using a selected index and discuss cheaper alternatives.

Random List stores regions in a simple list. The lookup operation scans the list from the beginning until a region containing the query point is found. Lists are not sophisticated index structures, but are known to perform well for disk-based accesses in high dimensions [13] and also as in-memory data structures for combustion simulations [9].

Different orders of regions in the list will result in different prediction costs. Given a set of regions, it is infeasible to try all possible orders to find the best one. The idea behind

the random list approach is to compute and minimize the expected cost assuming all region orders are equally likely, and then to pick the best order for the set of regions with the lowest expected cost.

Given a selected set of region model pairs of size |S|, the cost function computes $\sum_{\mathbf{x}_i \in \mathcal{V}}(\frac{|S|}{f_i+1} + Avg(\mathrm{t}_{m_1} \ldots \mathrm{t}_{m_{f_i}}))$. The intuition for the formula is as follows. For a set of regions, if a query point lies in multiple regions, then in any random order of the list it is very likely that a region containing the query point is found early. Therefore, the search time for a query point is approximated as $\frac{|S|}{f_i+1}$ where f_i is the number of regions that query point $\mathbf{x}_i$ lies in. The approximation time for a query point is simply the average of the cost of the models in the regions that the query point lies in (each one is equally likely to be found first in the list). After the selection algorithm finds a set of regions with the lowest expected cost, we try a few different sort orders of these regions and pick one with the lowest cost. Cost functions for other index structures can be similarly developed but we do not discuss them here [8].

5. Experiments

As a proof of concept, we implemented and tested our approach for the combustion application using data from a Hydrogen+Air simulation (reaction function $f : R^{10} \rightarrow R^{11}$). The dataset comprises 5 million simulation query points, each with a 10 dimensional composition vector.

The overall setup is as follows. We run the ISAT algorithm on the first 3 million query points to generate the base regions and training/validation data set $\mathcal{I}$, which are used by our algorithm as discussed in Section 4. A random sample of $\approx 2 \times 10^5$ query points from the last 2 million queries is used as an independent test set to compare total simulation time of the original ISAT model with our proposed method (Opt).

All experiments used a $70-30$ split of $\mathcal{I}$ into training ($\mathcal{T}$) and validation ($\mathcal{V}$) set and $\delta = 0.1$. For each base region, 8 derived regions are created by merging the base region with its nearest neighbor base regions. All experiments were run on a Windows XP PC with a 2.79GHz processor and 8GB RAM.

5.1. Results

Here we discuss results only with the Random List index. Table 1 summarizes the results. In the table k is the average false positive rate of the index[2] and Obs δ is the observed δ on the test set.

[2]The false positive rate of an index, for a query is measured as the number of regions examined before finding a region containing the query point.

Experiment 1 is for $\epsilon = 5 \times 10^{-3}$. ISAT built regions with linear models (L)[3] and our framework used both linear and quadratic (Q) models. ISAT created 63 regions. Since index size and search cost are small in this case, our method (Opt) does not merge many of the linear regions into quadratic ones (only 9). Nevertheless a significant reduction in prediction cost by $\approx 30\%$ is achieved. The increase in approximation cost (some query points are approximated using quadratic models) is offset by the decrease in search cost. Recall that our algorithm for a random list tries a few random orders and returns the best as the solution. For ISAT there is no optimization algorithm for selecting the best list order, therefore we report average cost across 30 different random sort orders and standard deviation.

To show that both approximation and search cost must be considered for prediction time optimization, we repeated Experiment 1 using a simpler optimization goal—only minimize search cost ("Only S"). In this case the selection algorithm aggressively merges regions to cover validation points with the smallest number of derived regions containing quadratic models. As the results show, the additional decrease in search cost is not significant enough to offset the higher approximation cost. A surprising observation in this experiment is that the number of regions created by "Only S" is greater than for ISAT, even though the selection algorithm usually *replaces* a set of regions with a larger region. This happens because it is possible to select a candidate region that does not completely contain any regions in the current solution, but significantly overlaps with a lot of them (i.e., Rem={} in Line 6 and 7 of Algorithm 1). Adding such a region increases list size but may still reduce expected search cost per query as some query points now are covered by multiple regions (recall that search cost=$\frac{|S|}{f_i} + 1$).

Experiment 2 uses the same setup as Experiment 1 but with $\epsilon = 5 \times 10^{-5}$. As ϵ is stricter, it is not surprising that ISAT creates a larger number of regions and hence search cost dominates prediction time. Opt in this case more aggressively selects regions with quadratic models, causing the approximation time to increase significantly. An even larger decrease in search time results in $\approx 70\%$ improvement in total time.

Discussion. Our experiments show that different indexes and model types work well in different simulation settings. Our proposed method (Opt) correctly and automatically captures the tradeoffs in the problem and effectively adapts the model to the index and simulation parameters. Our method does not improve runtime at the cost of degrading prediction quality (see δ values in Table 1). Finally, in all our experiments the cost of the optimization algorithm was negligible compared to the total cost of a long-running simulation as used by the domain scientists. Note that we

[3]ISAT always uses the same model in every region; it must be specified when the simulation starts.

ExptNo:(S, ϵ)	Method	$\mathcal{M}$	$\lvert\mathrm{S}\rvert$	k	Obs δ	Search Time(ms)	Approximation Time(ms)	Total Time(ms)	StdDev (ms)
1: $(\mathrm{RL}, 5 \times 10^{-3})$	ISAT	L	L:63	26	0.01	623	337	960	68
	Opt	L,Q	L:28,Q:9	6	0.005	114	434	548	-
	Only S	L,Q	L:26,Q:41	1	0.0002	84	1750	1834	-
2: $(\mathrm{RL}, 5 \times 10^{-5})$	ISAT	L	L:2263	977	0.05	20477	383	20860	2983
	Opt	L,Q	L:1430,Q:332	122	0.01	2071	1620	3691	-

Table 1. Results Summary

have illustrated improvements using only two simple model types. By adding more models to the framework, even better results can be obtained, though at a higher cost of optimization.

6. Related Work

Closest to our approach is recent work on model compression [4] where an ensemble model is approximated with a neural network to improve prediction time. Robot motion planning algorithms reduce prediction time in local regression models [11]. None of this prior work formalizes the learning problem and examines the fundamental tradeoffs we discuss.

There is lot of work on local models. Instance based learning [7] is a special class of local models where rather than explicitly defining regions, function values at unknown points are interpolated from neighboring training samples. A regression tree [3] creates regions in the function domain. Regression trees are often pruned for accuracy [3], and our framework when applied to a regression tree could use pruning for improving prediction time. Optimizing regression trees for prediction time has not been studied. Existing techniques in the combustion community use region-models that differ in the types of regions and models used [10, 1, 6, 9, 12]. We address the more general, and much harder, problem of optimizing search and approximation costs together. Numerous methods have been proposed for finding cost models for high dimensional index structures with different focus from our work [2].

7. Conclusions and Future Work

We introduced and formalized the low prediction time learning problem. We proposed a general framework that leverages existing data mining models to minimize model prediction time and used it to significantly speed up a scientific application. A longer version of this paper with implementation details, cost functions for other index structures and additional results is available [8]. Understanding how existing data mining models can be optimized for prediction time is an interesting direction for future research.

Acknowledgments. This work has been supported by the National Science Foundation through grants CBET-0426787, EF-0427914 and IIS-0612031. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] J. B. Bell, N. J. Brown, M. S. Day, M. Frenklach, J. F. Grcar, R. M. Propp, and S. R. Tonse. Scaling and efficiency of PRISM in adaptive simulations of turbulent premixed flames. In *28th International Combustion Symposium*, 2000.

[2] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[3] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and regression trees*. McGraw-Hill, 2000.

[4] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. In *ACM SIGKDD*, pages 535–541, 2006.

[5] R. Caruana and A. Niculescu-Mizil. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In *ACM SIGKDD*, pages 69–78, 2004.

[6] J. Y. Chen, W. Kollmann, and R. W. Dibble. Pdf modeling of turbulent nonpremixed methane jet flames. *Combustion Science and Technology*, pages 315–346, 1989.

[7] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[8] B. Panda, M. Riedewald, J. Gehrke, and S. B. Pope. High-speed function approximation. Technical Report TR2007-2092, Cornell University, 2007. http://techreports.library.cornell.edu.

[9] B. Panda, M. Riedewald, S. B. Pope, J. Gehrke, and L. P. Chew. Indexing for function approximation. In *VLDB*, 2006.

[10] S. B. Pope. Computationally efficient implementation of combustion chemistry using *in situ* adaptive tabulation. *Combustion Theory Modelling*, (1):41–63, 1997.

[11] S. Schaal, C. Atkeson, and S. Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *IEEE Int'l Conf. Robotics and Automation*, pages 288–293, 2000.

[12] I. Veljkovic, P. Plassmann, and D. C. Haworth. A scientific on-line database for efficient function approximation. In *ICCSA*, pages 643–653, 2003.

[13] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.