Lazy Meta-Learning: Creating Customized Model Ensembles on Demand

Piero P. Bonissone

GE Global Research Center, Niskayuna, NY 12309, USA
bonissone@ge.com

Abstract. In the not so distant future, we expect analytic models to become a commodity. We envision having access to a large number of data-driven models, obtained by a combination of crowdsourcing, crowdservicing, cloud-based evolutionary algorithms, outsourcing, in-house development, and legacy models. In this new context, the critical question will be model ensemble selection and fusion, rather than model generation. We address this issue by proposing *customized model ensembles on demand*, inspired by *Lazy Learning*. In our approach, referred to as *Lazy Meta-Learning*, for a given query we find the most relevant models from a DB of models, using their meta-information. After retrieving the relevant models, we select a subset of models with highly uncorrelated errors. With these models we create an ensemble and use their meta-information for dynamic bias compensation and relevance weighting. The output is a weighted interpolation or extrapolation of the outputs of the models ensemble. Furthermore, the confidence interval around the output is reduced as we increase the number of uncorrelated models in the ensemble. We have successfully tested this approach in a power plant management application.

Keywords: Machine learning, lazy learning, meta-learning, computational intelligence, fusion, ensemble, entropy, Pareto set, neural networks, coal-fired power plant management.

1 Analytic Model Building in the Near Future

Until recently, analytic model building has been a specialized craft. Usually, such models were handcrafted by specialized researchers and manually maintained. Typically, the model builder selected a data set for training, test, and validation, extracted a run-time model from the training data set using machine learning (ML) techniques as a compiler, validated the model using a validation set, and finally used the model to handle new queries. When a model deteriorated, the model builder created a new model by following a similar build cycle. As noted by the author in earlier papers [1-2], this lack of automation in model creation has led to bottlenecks in the models lifecycle, preventing their scalability and eventually leading to their obsolescence.

To address the lack of automation in model building, we have proposed the use of meta-heuristics, specifically the use of evolutionary algorithms, to generate runtime analytical models, while limiting the amount of human intervention in the process [3-5]. Although this proposal was a step in the right direction, it was still aimed at generating single models or, in the best case, static ensembles of models [6-9]. While the model builder was no longer the bottleneck in the loop defining and running the experiments needed to create and test the model, s/he was still an integral part of the process.

We believe that the advent of cloud computing has changed dramatically the process of developing analytic models. Cloud computing has lowered the barriers to entry, enabling a vast number of analytics-skilled people to access in a flexible manner large amounts of computing cycles for relatively low operational cost (and without any capital expenses). The literature on cloud computing is growing exponentially, making it difficult to provide the interested reader with a single reference. However, reference [10] provides a great explanation of the opportunities created by cloud computing for machine learning and crowdsourcing.

1.1 Multiple Sources of Analytic Models Enabled by Cloud Computing

To validate our assumption that analytic models are trending to become a commodity, let us explore some of the ways used to generate or obtain analytic models:

1. Crowdsourcing analytics
   (a) Using traditional crowdsourcing markets
   (b) Using competitions and prizes
   (c) Using games or puzzles
2. Evolving populations of models using evolutionary algorithms (GA's, GP's)
3. Outsourcing analytics
4. Traditional model development (including legacy models)

We will focus on the first two sources, which are enabled or accelerated by cloud computing. Crowdsourcing is a relatively new phenomenon that started about a decade ago [11]. It is becoming increasingly popular for outsourcing micro-tasks to a virtual crowd, creating new marketplaces - see for instance *Amazon's Mechanical Turk* (*mturk.com*). Recently, however these tasks have become more complex and knowledge intensive, requiring a more specialized crowd. Web portals such as *Kaggle* (*kaggle.com*), *TunedIT* (*TunedIT.com*), and *CrowdANALYTICX* (*crowdanalytix.com*) allow organizations to post problems, training data sets, performance metrics, and timelines for ML competitions. The portals register the potential competitors and manage the competition during its various stages. At the end, the models are scored against an unpublished validation set, the winners receive their prizes, and the sponsoring organizations have access to new analytic models [12]. Alternative ways to incentivize a crowd to solve a given problem is by transforming the problem into a game or a puzzle. A successful example of this approach can be found at *Foldit* (*foldit.it*), in which the gaming community was able to solve a molecular folding problem that had baffled biologists for over a decade.

A second possible source of models is the use of evolutionary algorithms to search the model space. The author has explored this approach over a decade ago [13-14] inspired by many efforts in this area. In the early days, researchers resorted to clusters of computers, such as the Beowulf [15] to distribute the computational load of GP-driven search. Recently, the MIT CSAIL Department developed a *Flexible Genetic Programming* (*groups.csail.mit.edu/EVO-DesignOpt/evo.php?n=Site.FlexGP*) framework, leveraging cloud computing to evolve a population of models, whose outputs are ultimately averaged to produce an answer [16].

The last two approaches are more traditional ways of generating analytical models by outsourcing them to universities, developing them internally, or using legacy models. Our goal is to create *a touch-free, domain agnostic* process that can use any subset of models, regardless of their sources, and determine at run-time the most suitable and diverse subset that should be used to construct the answer to a given query.

To ensure a more focused discussion, we will limit the scope of this paper to the use of analytics to support Prognostics and Health Management (PHM) capabilities. However, the approach illustrated in this paper is application domain agnostic.

1.2 Prognostics and Health Management (PHM) Motivation for Analytics

The main goal of Prognostics and Health Management (PHM) for assets such as locomotives, medical scanners, aircraft engines, and turbines, is to maintain these assets' performance over time, improving their utilization while minimizing their maintenance cost. This tradeoff is typical of contractual service agreements offered by OEMs to their valued customers.

PHM analytic models are used to provide *anomaly detection and identification* (leveraging unsupervised learning techniques, such as clustering), *diagnostic analysis* (leveraging supervised learning techniques, such as classification), *prognostics* (leveraging prediction techniques to produce estimates of remaining useful life), *fault accommodation* (leveraging intelligent control techniques), and *logistics and maintenance optimization* (leveraging optimization techniques). A more detailed description of PHM functionalities and how they can be addressed individually by Computational Intelligence techniques can be found in [17]. In this paper, we will take a more holistic view on how to address PHM needs, while at the same time we will remain agnostics on the specific technologies used to build each model.

Since analytics play such a critical role in the development of PHM services, it is necessary to ensure that the underlying analytic models are accurate, up-to-date, robust, and reliable. There are at least two PHM applications for which such accuracy is critical: (1) *anomaly detection* (1-class classification), in which high volume of *false positives* might decrease the usefulness of the PHM system; (2) *prognostics* (prediction), in which high prediction variance might prevent us from acting on the results.

We will focus on a PHM prediction application in which prediction accuracy is a stringent requirement for production optimization. We will show how to leverage computational intelligence and ML techniques, combined with the elasticity of cloud computing, to address these accuracy requirements.

1.3 The Novel Idea

In this paper we are shifting our focus from model creation to model ensemble assembly. Rather than creating and optimizing models based on expected queries, we want to build a vast library of robust, local or global models, and compile relevant meta-information about each model. At run-time, for a specific query, we will select an ensemble of the most appropriate models from the library and determine their weights in the model fusion schema, based on their local performance around the query. The model ensemble will be constructed dynamically, on the basis of the models' meta-information. The model fusion will use the meta-information to determine bias compensation and relevance weight for each model's output. Finally, the models run-time versions will be executed via a function call at the end of the fusion stage. This concept is illustrated in Figure 1.

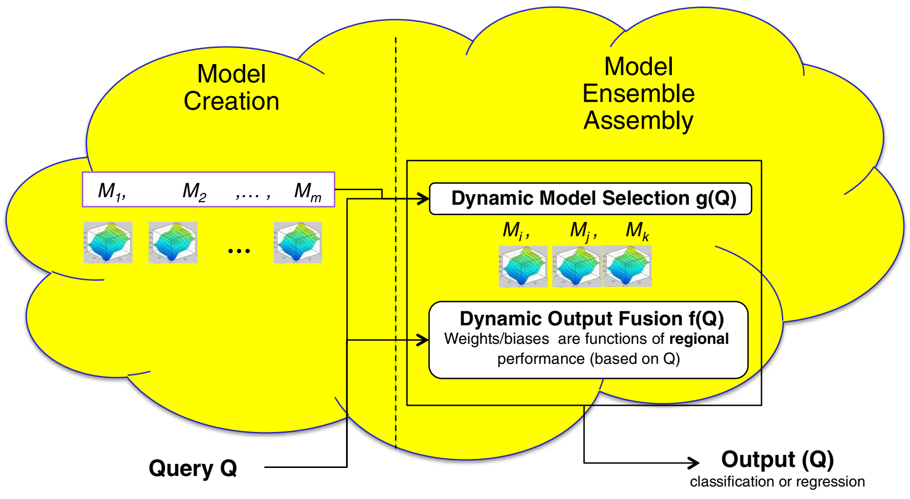

Model Creation
M_1, M_2 ,... , M_m
...
Model Ensemble Assembly
Dynamic Model Selection g(Q)
M_i, M_j, M_k
Dynamic Output Fusion f(Q)
Weights/biases are functions of **regional** performance (based on Q)
Query Q
Output (Q)
classification or regression

Fig. 1. The two modeling stages: Model Creation and Model Assembly

1.4 Paper Organization

In section 2, we will examine the relevant state of the art for this problem, while in section 3 we will describe a summary of our approach. In section 4, we will provide an in-depth analysis of the proposed approach, while in section 5 we will show some preliminary results using an ensemble of thirty neural networks to predict load, efficiency, and emissions in a power plant. In section 6, we will draw some conclusions from our experiments and highlight future work and extensions.

2 Related Work

We are proposing an approach for creating the best model ensemble on demand, based on the query information (as in Lazy-Learning), and performing the selection and dynamic fusion of the ensemble based on several performance criteria. In the literature we find vast amount of work covering model ensembles, meta-learning, lazy-learning, and multi-criteria decision making, but to the best of our knowledge there is no related work covering the intersection of these topics.

2.1 Model Ensembles

Individual models (classifiers or predictors) have a performance ceiling, which limits their performance, regardless of the amount of training or tuning. One way to raise this ceiling is by creating an ensemble of highly *diverse* models and performing a fusion of their outputs. There is currently an entire scientific community, *Multi-Classifier Systems* (MCS), devoted to this area.

The design of a successful classifier fusion system consists of three parts: design of the individual classifiers [18], selection of a set of classifiers [19-20], and design of the classifier fusion mechanism. The most critical factor for an effective model fusion, however, is the *diversity* of the individual classifiers, where model diversity is defined in terms of the orthogonality of their errors [21]. Strategies for boosting such diversity include: 1) using different types of classifiers; 2) training individual classifiers with different data set (bagging and boosting); and 3) using different subsets of features.

2.2 Meta- Learning and Lazy Learning

Meta-learning literally means *learning how to learn*, but in our context it means *learning how to use ML models*. Most meta-learning approaches deal with topics such as: Discovering meta-knowledge (e.g. rule induction of rules from data to create a rule-based system that will solve the object-level problem); Stacked generalization [22] (e.g. combining a number of different learning algorithms); Boosting (e.g. combining the same learning algorithm trained in different ways); Dynamic bias selection (e.g. modifying the ML algorithm bias to match a given problem); and Inductive transfer (e.g. trying to improve the learning process over time) [23].

An interesting approach is one proposed by Duch [24], in which he creates a framework of Similarity-Based Methods to represent many algorithms, such as k-NN, MLP, RBF, etc. A variant of the Best First Search is used to perform a local search for optimal parameters.

In [25] Schaul attributes to Utgoff [26] the development of the first meta-learning system that learns parameters and to Schmidhuber [27] the first learning algorithm to learn other ML algorithms using evolutionary search in the model space (using GP for improving GP). According to Schaul: *"…meta-learning can be used for automating human design decisions (e.g. parameter tuning) and then automatically revisiting and optimizing those decisions dynamically in the light of new experience obtained during learning. Another application is transfer of knowledge between multiple related tasks…"* [25].

We endorse this goal, but our proposal is not limited to parameter tuning. Our key idea is not to focus on the optimization and tuning of pre-computed models. Rather, we aim to create model ensembles *on demand*, guided by the location of the query in the feature space. This approach can be traced back to memory-based approaches [27-29], instance-based learning, and lazy-learning [30]. Figure 2 shows the Lazy Learning approach for locally weighted interpolation.

For a query Q, defined as a point $\bar{X}_Q$ in the state space **X**, and a set of data points defined in the cross-product **X** x **Y**, we find the points $u_j = (\bar{X}_j, y_j)$ that are close to

Q in their projection on **X**. We compute a matching score between the query and each of these data points, as a function of its proximity to the query. Such proximity is measured by a distance $d(\bar{X}_Q, \bar{X}_{Qj})$ that is interpreted as the dissimilarity between the two points in **X**. The distance is smoothed by a parameter h that defines the scale or range of the model. Usually h is computed by minimizing a cross-validation error. Each output y_j (corresponding to each of the points close to Q) is weighted by applying a kernel function K to this smoothed distance. A convex sum is used for the aggregation, which is identical to the Nadaraya-Watson estimator for non-parametric regressions using locally weighted averages. When dealing with extrapolation, weighted linear models are used instead of convex sums in the aggregation box, to provide for a better generalization [4].

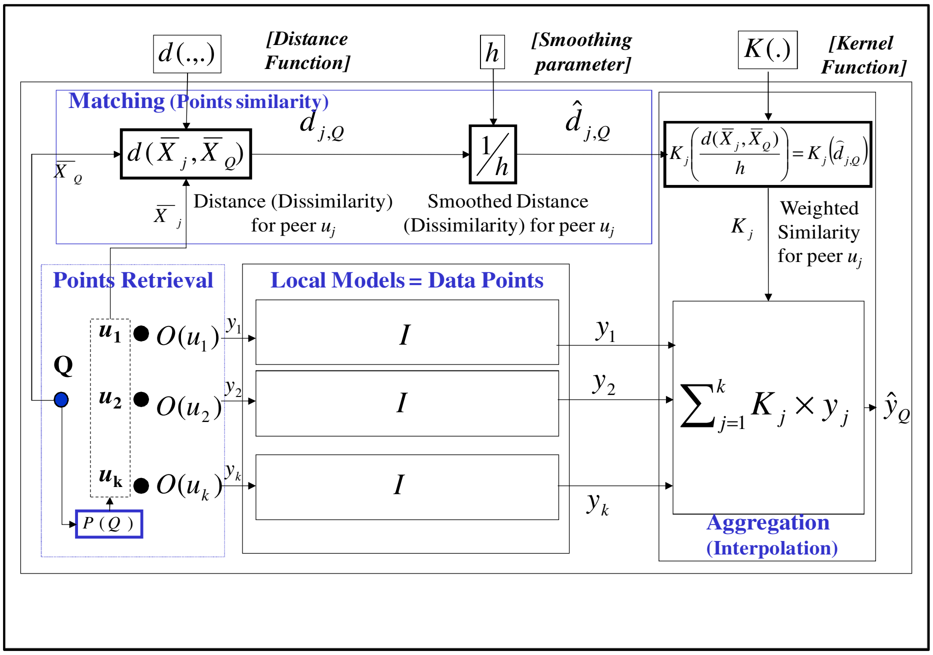

$d(.,.)$
[Distance Function]
h
[Smoothing parameter]
$K(.)$
[Kernel Function]
Matching (Points similarity)
$d(\bar{X}_j, \bar{X}_Q)$
$d_{j,Q}$
$1/h$
$\hat{d}_{j,Q}$
$K_j\left(\frac{d(\bar{X}_j,\bar{X}_Q)}{h}\right) = K_j(\hat{d}_{j,Q})$
$\bar{X}_Q$
$\bar{X}_j$
Distance (Dissimilarity) for peer u_j
Smoothed Distance (Dissimilarity) for peer u_j
K_j
Weighted Similarity for peer u_j
Points Retrieval
Local Models = Data Points
Q
u_1 $O(u_1)$ y_1
u_2 $O(u_2)$ y_2
u_k $O(u_k)$ y_k
$P(Q)$
I
I
I
y_1
y_2
y_k
$\sum_{j=1}^{k} K_j \times y_j$
$\hat{y}_Q$
Aggregation (Interpolation)

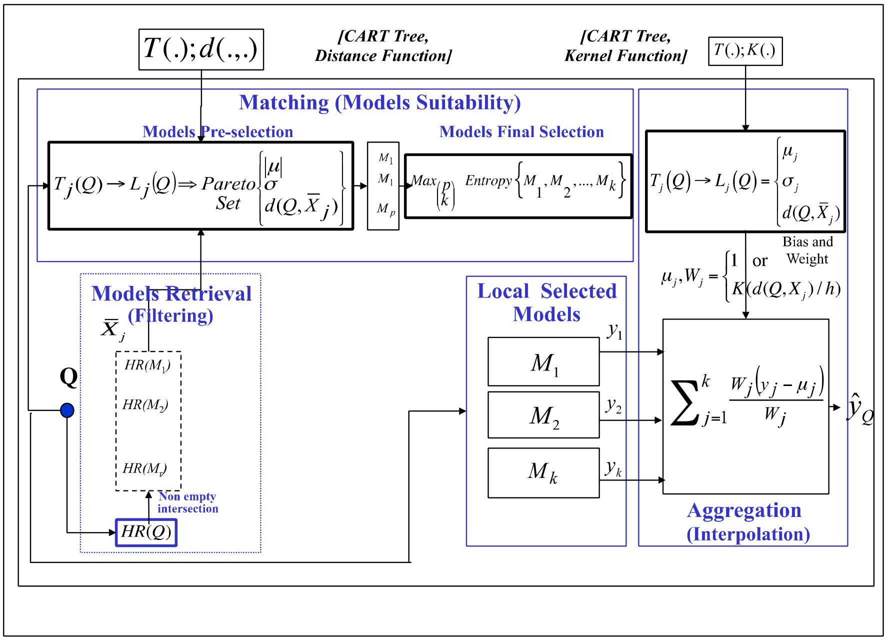
Fig. 2. Lazy Learning (or Locally Weighted Kernel-based) Interpolation - adapted from [4]

2.3 MCDM: Selection of Best Model Ensemble Based on Constrained Multi-Criteria

We plan to create a dynamic ensemble that best satisfies a set of performance metrics, leveraging techniques in Multi Criteria Decision Making (MCDM). There is a vast amount of literature on MCDM, ranging from pioneering books [31-32], to many conference proceedings and articles focused on improving search methods, preference aggregations, interactive solution visualization, etc.

The author's perspective on MCDM is described in [33]. It can be summarized as defining a proper problem representation followed by the intersection of search, preference aggregation, and (when needed) interactive visualization of their solutions.

- *Representation.* A typical MCDM process starts with a set of constraints over the solution space, defining the feasible solution set. Each point from this set is mapped into a performance space, whose dimensions are the criteria used to evaluate the MCDM solutions.
- *Multi Objective Search.* We search for the set of non-dominated solutions, forming the Pareto set. This step induces a partial ordering on the set of feasible solutions, and defines the concept of Multi-Objective Optimization (MOO).
- *Preferences.* MCDM requires an additional step over MOO, which is the selection of one or more non-dominated solutions, to maximize our aggregated preferences, thus creating a complete order over the feasible solutions sets.
- *Interactive Visualization.* In cases when the decision-maker is part of the solution refinement and selection loop, we need a process to enhance our cognitive view of the problem and enable us to perform interim decisions. We need to understand and present the impacts that intermediate tradeoffs in one sub-space could have in the other ones, while allowing him/her to retract or modify any intermediate decision steps to strike appropriate tradeoff balances. This last step will not be needed in our proposed MCDM approach.



$T(.);d(.,.)$
[CART Tree, Distance Function]
[CART Tree, Kernel Function]
$T(.);K(.)$
Matching (Models Suitability)
Models Pre-selection
Models Final Selection
$T_j(Q) \to L_j(Q) \Rightarrow$ Pareto Set $\{|\mu|, \sigma, d(Q,\bar{X}_j)\}$
M_1 M_1 M_p
$Max_{\binom{p}{k}}$ Entropy$\{M_1, M_2, ..., M_k\}$
$T_j(Q) \to L_j(Q) = \{\mu_j, \sigma_j, d(Q,\bar{X}_j)\}$
Bias and Weight
$\mu_j, W_j = \{1$ or $K(d(Q,X_j)/h)$
Models Retrieval (Filtering)
$\bar{X}_j$
Q
$HR(M_1)$
$HR(M_2)$
$HR(M_r)$
Non empty intersection
$HR(Q)$
Local Selected Models
M_1 M_2 M_k
y_1 y_2 y_k
$\sum_{j=1}^{k} \frac{w_j(y_j - \mu_j)}{w_j}$
$\hat{y}_Q$
Aggregation (Interpolation)

Fig. 3. Meta-Lazy Learning

In our case, model creation is equivalent to creating the solutions set, while model filtering generates the *feasible* solution set. The model pre-selection, based on the meta-information attached to each model, maps the solutions in the performance space and the extraction of the points in the Pareto set represents the MOO step. The model final selection, i.e. the selection of the k-tuple of local models with the least amount of error correlation (via entropy maximization) reflects our preferences. The aggregation is the final ensemble with the parameters defining dynamic fusion represents the outcome of the MCDM process, i.e. our customized ensemble on demand. Our approach is summarized in Fig. 3 and will be further explained in sections 3 and 4.

We have briefly covered the state of the art in multi classifier systems, lazy learning, meta-learning, and MCDM. Our goal is to address their intersection in a way that has never been investigated before.

3 Summary of the Approach

3.1 Cloud Computing, the Enabler

Although the concept of model ensemble has been proposed and analyzed since the early 2000 [18], the idea of creating dynamic ensembles at run-time has not been proposed yet. Such idea would not have been feasible or practical, had it not been for the advent of Grid- and Cloud-computing. After constructing an offline library of models with their associate meta-information, we can now leverage the cloud environment, with its parallel computation and automated provisioning of processors, to implement this approach and provide fast run-time responses. Our approach is predicated on a cloud-based Software as a Service (*SaaS*) paradigm.

3.2 Lazy Meta-Learning

In the ML literature, the concept of Lazy learning (LL) or memory-based learning departs radically from traditional ML approaches. Instead of training a model from the data to become a functional approximation of the underlying relationships, the LL approach states that the *model is in the data*. Upon receiving a query, at run-time LL creates a temporary model by finding the closest points to the query and performing an aggregation (usually a weighted interpolation or extrapolation) of the outputs of those points.

Our approach, labeled *Lazy Meta-Learning,* can be described by the analogy**:** *Meta-learning is Lazy Learning for models like learning is Lazy-Learning for data points.* This approach can be described as a multi-criteria decision-making (MCDM) process, whose structure follows the steps defined in [5]. The model design is performed by an offline meta-heuristics (the MCDM process), while the run-time model architecture is formed by an online meta-heuristics (the fusion module), and a collection of object models (the analytic models.)

Within the scope of this paper, a model could be a one-class classifier for anomaly detection, a multi-class classifier for diagnostics, or a predictor for prognostics. Each model will have associated meta-information, such as its region of competence and applicability (based on its training set statistics), a summary of its (local) performance during validation, an assessment of its remaining useful life (based on estimate of its obsolescence), etc.

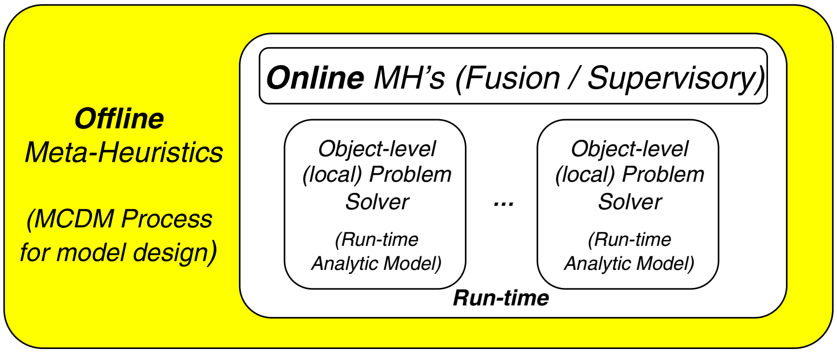

Offline
Meta-Heuristics
(MCDM Process for model design)
Online MH's (Fusion / Supervisory)
Object-level (local) Problem Solver
(Run-time Analytic Model)
...
Object-level (local) Problem Solver
(Run-time Analytic Model)
Run-time

Fig. 4. Model Design: Offline Meta-Heuristics to design, tune, optimize, adapt to changes, and maintain the runtime models over time. Model Architecture: (a) Online Meta-Heuristics to integrate or interpolate among multiple local object-models, manage their complexity, and improve their overall performance; (b) Multiple object-models, either in parallel configuration (ensemble) or sequential configuration (cascade, loop), to integrate functional approximation with optimization and reasoning with imperfect data (imprecise and uncertain).

3.3 MCDM Process for Model Creation and Dynamic Model Assembly

We decompose the MCDM process into two stages:

- *Model Creation*, an off-line stage in which we create the initial building blocks for the assembly and we compile their meta-information
- *Dynamic Model Assembly*, an on-line stage in which, for a given query we select the best subset of models

This process is followed by the execution stage, *Dynamic Model Fusion*, in which we evaluate the selected models and dynamically fuse them to solve the query. We will use different metrics to evaluate each stage, looking for coverage and diversity in the creation stage, while looking for accuracy and precision in the assembly and fusion stages.

Model Creation: The Building Blocks. We assume the availability of an initial training set that samples an underlying mapping from a feature space X to an output y. In the case of supervised learning, we also know the ground truth-value t for each record in the training set. We create a library of diverse, local or global models. We increase model diversity by using competing ML techniques trained on the same local regions. We assume that any of the sources mentioned in section 1.1 could be used to create these models.

Dynamic Model Assembly: Query Driven Model Selection and Ensemble. This stage is divided into three steps:

- *Model Filtering*, in which we retrieve the applicable models from the DB for the given query;
- *Model Pre-Selection*, in which we reduce the number of models based on their local performance characteristics (bias, variability, and distance from the query);
- *Model Final Selection*, in which we define the final model subset.

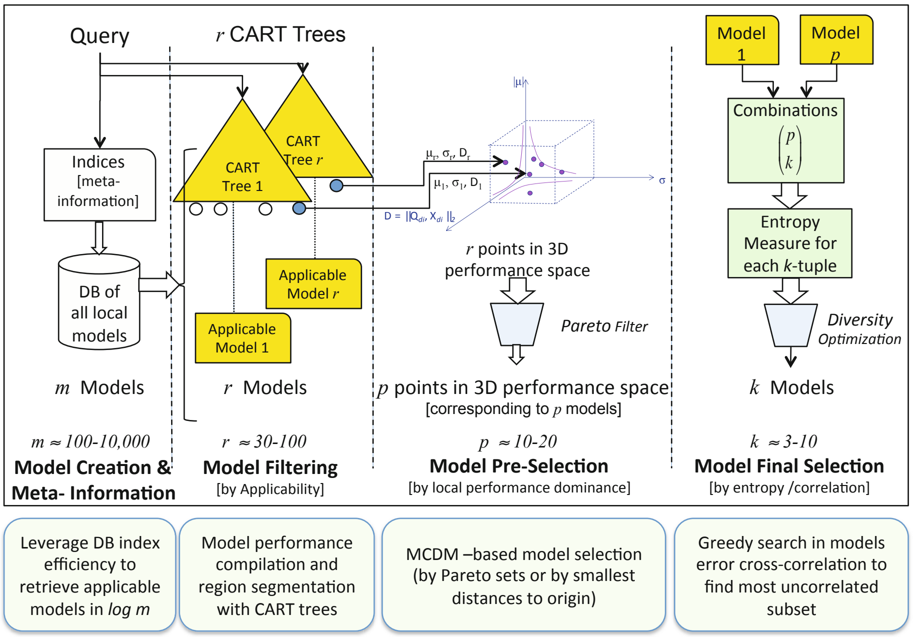

Query
r CART Trees
Indices [meta-information]
DB of all local models
CART Tree 1
CART Tree r
Applicable Model r
Applicable Model 1
μr, σr, Dr
μ1, σ1, D1
D = ||Qdi, Xdi ||2
|μ|
σ
r points in 3D performance space
Pareto Filter
Model 1
Model p
Combinations (p k)
Entropy Measure for each k-tuple
Diversity Optimization
m Models
r Models
p points in 3D performance space [corresponding to p models]
k Models
m ≈ 100-10,000
r ≈ 30-100
p ≈ 10-20
k ≈ 3-10
Model Creation & Meta- Information
Model Filtering [by Applicability]
Model Pre-Selection [by local performance dominance]
Model Final Selection [by entropy /correlation]
Leverage DB index efficiency to retrieve applicable models in log m
Model performance compilation and region segmentation with CART trees
MCDM –based model selection (by Pareto sets or by smallest distances to origin)
Greedy search in models error cross-correlation to find most uncorrelated subset

Fig. 5. Dynamic Model Assembly on Demand (Filtering, Selection)

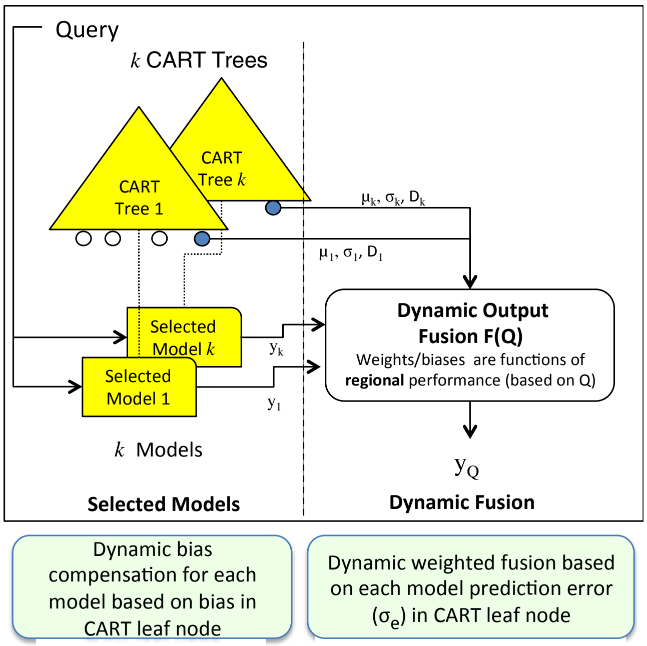

Query
k CART Trees
CART Tree 1
CART Tree k
μk, σk, Dk
μ1, σ1, D1
Selected Model k
Selected Model 1
yk
y1
Dynamic Output Fusion F(Q)
Weights/biases are functions of regional performance (based on Q)
yQ
k Models
Selected Models
Dynamic Fusion
Dynamic bias compensation for each model based on bias in CART leaf node
Dynamic weighted fusion based on each model prediction error (σe) in CART leaf node

Fig. 6. Dynamic Model Fusion on Demand

Dynamic Model Fusion: Generating the Answer. Finally, we evaluate the selected models and we aggregate their outputs after compensating for their biases and weighting them by their proximity to the query (or by their estimated error) to generate the solution to the query.

4 (The Devil Is in the) Details of Our Approach

4.1 Model Creation

This step is common to both regression and classification problems. The premise is that local models, trained on regions of the feature space, usually will have smaller variances than global models, trained on the entire space, providing that the local models are used *only* within their region of competence. Typically this requires a supervision mechanism (at the meta-level) to determine their degree of applicability and contribution to the final output. We rely upon the model meta-information to make this assessment. There are many ways to segment the original training set to define regions of competence for local models, but this topic is outside the scope of this paper.

Library of m Local Models. As stated above, for each prediction or classification problem we want to generate a large number of global and local models. We will assume that *m* is the total number of available models for each problem.

Prediction Problems. Each regression model M_i will define a mapping:

$$M_i\colon X \rightarrow Y, where\ i = 1, \dots, m;\ |X| = n; |Y| = 1;\ X\epsilon\mathbb{R}^n; Y\epsilon\mathbb{R}$$

In a more general case, we might want to predict multiple i.e., g variables, i.e.:

$$M_i\colon X \rightarrow Y, where\ i = 1, \dots, m;\ |X| = n; |Y| = 1;\ X\epsilon\mathbb{R}^n; Y\epsilon\mathbb{R}^g$$

Within the scope of this article we will limit ourselves to a single output.

Classification Problems. Each classification model M_i will define a mapping: $M_i\colon X \rightarrow Y, where\ i = 1, \dots, m; |X| = n; \lfloor Y \rfloor = (C+1)$, and C is the number of classes. Within the scope of this article, we will assume that the classifier output is a probability density function *(pdf)* over C classes. The first C components of the *pdf* are the probabilities of the corresponding classes. The (C+1)th element of the *pdf* allows the classifier to represent the choice "*none of the above*" (i.e., it permits to deal with the Open World Assumption). The (C+1)th element of the *pdf* is computed as the complement to 1 of the sum of the first C components. The final decision of classifier M_i is the *argmax* of the *pdf.*

Meta-information. Every time that we create a model, we need to capture its associated meta-information, i.e., information about the model itself, its training set, and its local/global performance in the validation set. Over time, we will define a standard API for this purpose. Table 1 summarizes our current preliminary thoughts on meta-information. The essence is to capture information that we can use later on to reason about the applicability and suitability of a model for a given situation (e.g., a set of queries).

Table 1. Meta-Information (Preliminary Version)

Model	Training Set	Validation Set
Label: M_i	Label: TS_i	Label: VS_i
Model Mapping: $M_i: X_i \rightarrow Y_i$ $X_i \in R^n$; $Y_i \in R$ $(regressions)$ $Y_i \in [0,1]^{(C+1)} (classif.)$	Model Applicability: Hyper-rectangle: HR_i $HR_i \in R^{2,n}$ is the range of values of each of the n features over TS_i	Local Model Performance: CART Tree: $T_i: X_i \rightarrow e_i$ $X_i \in R^n$; $e_i \in R$ $(regressions)$ $e_i \in [0,1]^{(C+1)} (classif.)$

Model Applicability: Hyper-rectangle HR$_i$. The Hyper-rectangle in the feature space defines each model's region of competence[1]. Each model M_i has a training set *TS$_i$*, which is a region of the feature space X. We define the Hyper-rectangle of each model M_i, *HR(M$_i$)*, to be the smallest hyper-rectangle that encloses all the training points in training set *TS$_i$*.. If a query point *q* is contained in *HR(M$_i$)*, we consider model M_i applicable for such query. For a set of query points *Q*, we consider the model applicable if *HR(Q)* is not disjoint with *HR(M$_i$)*.

Local Model Performance. We want to capture the local performance of the model by answering the question: "*For this type of query, how reliable is this model?*" For regression problems, we have used continuous case-based reasoning and fuzzy constraints [35], and lazy learning [36] to estimate the local prediction error. Within the same context of regression problems, the authors replaced the run-time use of lazy learning with the compilation of local performance via CART trees [34] for the purpose of correcting the prediction via bias compensation [7]. For classification problems, we find a similar lazy learning approach [37] to estimate the local classification error.

After many experiments to find the most efficient summary of the model performance, we opted for using a CART Tree T_i that maps the feature space to the ***signed error*** computed in the validation set, i.e. $T_i: X \rightarrow e_i$.

Each CART tree T_i will have a depth *d$_i$*, such that there will be up to 2^{di} paths from the roots to the leaf nodes (for a fully balanced tree). For each tree, we store each path from the root node to each leaf node. The path is as a conjunct of constraint rules that need to be satisfied to reach the leaf node. The leaf node is a pointer to a table containing the leaf node statistics:

- *N$_i$* – Number points in the leaf node (from the training/testing set)
- $\mu_i(e)$ – *Bias* (Average error computed over *N$_i$* points)
- $\sigma_i(e)$ – *Standard Deviation* of the error computed over *N$_i$* points
- *X$_{di}$* – Normalized centroid as percentage of its average (1,...,1) of the *N$_i$* points

In the future, we will extend the meta-information to capture temporal and usage information, such as model creation date, last usage date, and usage frequencies, which will be used by the model lifecycle management to select the models to maintain and update.

[1] By defining the applicability of a model as its hyper-rectangle in the feature space, we are limiting the use of the model to interpolation. Should we choose to use it for extrapolation, we would consider queries outside the hyper-rectangle (within some fuzzy tolerance).

4.2 Dynamic Model Assembly on Demand

Query Formulation: q or Q. To simplify the description of our approach we have used the case of the query being a single point query $\boldsymbol{q}$. However, our approach can be easily generalized to the case when the query is a data set (time-series or time independent). Let's refer to such query as $Q=[\boldsymbol{q}_1, \ldots, \boldsymbol{q}_z]$. In such case, the model filtering described in section 4.3.1 will be modified. Instead of retrieving for each point q those models M_i whose associated hyper-rectangles $HR(M_i)$ contain q, i.e., $q \in HR(M_i)$, we retrieve all models M_i whose associated hyper-rectangles $HR(M_i)$ are not disjoint with the Hyper-rectangle of Q, i.e. $HR(Q) \cap HR(M_i) \neq \emptyset$. This will avoid the overhead of multiple accesses to the DB[2]. The model assembly, composed by the *model pre-selection* and *model final selection* steps, will be performed iteratively for every point query $\boldsymbol{q}$ in Q. So these steps, as well as final model evaluation and fusion, will be linear in the number of query points z. The most efficient way to generate the set Q is to cluster similar query points using a *k-d tree* [38-39].

Model Filtering: From m to r Models. After creating m models, trained on their corresponding regions of the training space, we will organize the models in a database DB, whose indices will be derived from the models meta-information (see section 4.1.2). For a given query $\boldsymbol{q}$, the MCDM process starts with a set of constraints to define the feasibility set. In this case the constraints are:

- Model soundness, i.e., there are sufficient points in the training/testing set to develop a reliable model, competent in its region of applicability
- Model vitality, i.e., the model is up-to-date, not obsolete
- Model applicability to the query, i.e., the query is in the model's competence region

For each of the r retrieved models M_i, we will use a compiled summary of its performance, represented by a CART tree T_i, of depth d_i, trained on the model error vector obtained during the validation of the model.

Model Pre-selection: From r to p Models. We will classify the query using the same CART tree T_i, reaching leaf node $L_i(\boldsymbol{q})$. Each leaf will be defined by its path to the root of the tree and will contain d_i constraints over (at most) d_i features. Leaf $L_i(\boldsymbol{q})$ will provide estimates of model M_i performance in the region of the query. Following the MCDM process described in section 3, we map each feasible solution (the r models) into the performance space. This space is defined by the following criteria:

- Model bias: $|\mu_i(e)|$
- Model variability $\sigma_i(e)$
- Model suitability for the query: $\|\boldsymbol{q}_{di}, X_{di}\|_2$ (Distance of $\boldsymbol{q}_{di}$ to the normalized centroid (X_{di}), computed in a reduced, normalized feature space d

[2] This step might allow models that are irrelevant for a subset of points in Q. These models will be rejected in the model pre-selection (due to their high distances from the query point).

In this three dimensional space, we want to minimize all three objectives. We may further impose limits to the largest value that each dimension can have.

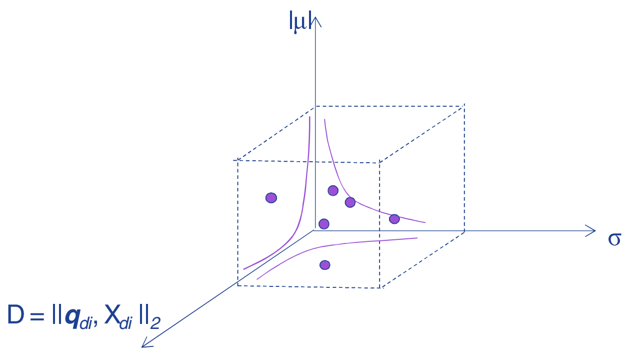


Fig. 7. Local Model Performance Space – r points represent r model performances

We can use a Pareto Filter to extract all the non-dominated points in this space. Should the result be too limited, we can resort to additional Pareto set (after removing the previous tier), until we have p models.

Model Final Selection: From p **to** k **Models.** During the previous steps, we relied on meta-information to reduce the number of models from the original m models available in the DB, to r applicable models, to p suitable models. Before performing the fusion, we need to *make sure that the models are diverse*, i.e. we need to explore the error correlation among smaller subsets of k models that we will use for generating the answer to the query. This step would require generating all possible k-tuples chosen from p models to evaluate their error correlations, i.e. $\binom{p}{k}$. Although, this is a much smaller number than $\binom{m}{k}$, it is still an onerous step to take for each point. So we will use a *greedy search* to further decrease this complexity.

Models in an ensemble should be different from each other for the ensemble's output to be better than the individual models outputs. The goal is to use an ensemble whose elements have the most uncorrelated errors. In reference [21], Kuncheva and Whitaker propose six different *non-pairwise diversity measures* to determine the models difference. We will use the *Entropy Measure E*, proposed in the same reference, as the way to find the k most diverse models to form the ensemble. Let's create an N by k matrix M, such that N is the number of records evaluated by k models.

Classification Problems. When the models are classifiers, cell *M[i,j]* contains a binary value *Z[i,j]* (1 if classifier j classified record i correctly, 0 otherwise). This metric assumes that we already obtained each classifier decision on the training/validation records, by applying the *argmax* function to the *pdf* generated by the classifier. Then, we compute diversity of the k classifiers by using the Entropy measure E (modified from reference [21]):

$$E = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{k-floor\left(\frac{k+1}{2}\right)} min\left(\sum_{j=1}^{k} M[i,j], k - \sum_{j-1}^{k} M[i,j]\right)\right] \quad (1)$$

E takes values in [0,1].

Prediction Problems. When the models are predictors, cell *M[i,j]* contains the error value *e[i,j]*, which is the prediction error made by model *i* on record *j*. In such case, we will follow the following process:

- *Histogram of Record Error:* Compute a histogram of the errors for each record *M[i,.]*. We need to define a reasonable bin size for the histogram, thus defining the total number of bins, *nmax*. Let *H(i,r)* be the histogram for record *i,* where *r* defines the bin number *(r=1, nmax).*
- *Normalized Histogram of Record Error:* Normalize histogram *H(i,r)*, so that its area is equal to one (becoming a *pdf*). Let $H_N(i,r)$ be the normalized histogram, i.e.:

$$H_N(i,r) = \frac{H(i,r)}{\sum_{r=1}^{nmax} H(i,r)} \quad (2)$$

- *Normalized Record Entropy:* Compute the normalized record entropy of the *pdf* (so that its value is in [0,1]), i.e.:

$$ent(i) = -\left(\frac{1}{\ln nmax}\right)\sum_{r=1}^{nmax} H_N(i,r) \times lnH_N(i,r) \quad (3)$$

where $\frac{1}{\ln nmax}$ is a normalizing factor so that *ent(i)* takes values in [0,1]

$$H_N(i,r) = \frac{H(i,r)}{\sum_{r=1}^{nmax} H(i,r)} \quad (4)$$

- *Overall Normalized Entropy:* Average the normalized entropy over all N records:

$$E = \frac{1}{N}\sum_{i}^{N} ent(i) \quad (5)$$

E takes values in [0,1].

For both classifiers and prediction problems, *higher overall normalized entropy values indicate higher models diversity***.**

Greedy Search in Combinatorial Space. In both cases we will use a greedy search, starting from *k*=2 and compute the normalized entropy for each 2-tuple, to find the one(s) with the highest entropy. We will then increase the value of *k* to explore all 3-tuples. If the maximum normalized entropy for the explored *3*-tuples is lower than the maximum value obtained for the 2-tuples, we will stop and use the 2-tuple with the highest entropy. Otherwise we will keep the 3-tuple with the highest entropy and explore the next level (*k*=4) and so on, until no further improvement can be found. With all the caveats of local search, this greedy approach will substantially reduce search complexity, as we will not need to explore all the combinations of ensembles.

4.3 Dynamic Model Fusion

The last step in our approach is to perform the dynamic fusion of the selected k models. When probed with the query q, each model M_i will produce and output $y_i(q)$. Each model also has a corresponding CART tree T_i, which will be used to classify the query q. In that case, the query will reach a leaf node $L_i(q)$ in the corresponding tree T_i. The leaf node will contain the local statistics for the points similar to the query, such as the bias (average of the error) $\mu_i(e|q)$, and the standard deviation of the error, $\sigma_i(e|q)$.

Fusion for Regression Problems. After many design experiments, which will be described in details in a follow-up paper, we concluded that the best fusion schema is accomplished by using dynamic bias compensation and by weighting the compensated output using a kernel function of the standard deviation of the error, i.e.:

$$\hat{y}(q) = \frac{\sum_{s=1}^{k} w_s(y_s(q) - \mu_s(e|q))}{\sum_{s=1}^{k} w_s} \tag{6}$$

where:

$$w_s = K\left(\frac{\sigma_s(e|q)}{h}\right) \tag{7}$$

and h is the usual smoothing factor for the kernel function *K(.)* obtained by minimizing the cross-validation error.

Fusion for Classification Problems. As shown in reference [37], the classification problem can be cast in a fashion similar to the regression problem. In reference [37] we used a local weighting fusion (similar to [7] but for classifications), and used dynamic bias compensation with equal weight contributions for the selected models. In section 6, we will discuss how to extend our approach, based on these preliminary results, to cover classification problems.

5 Customized Analytics Applied to Power Plant Management

5.1 Problem Definition

In references [40-41] we described the optimization problem for a power plant management system, in which a coal-fired boiler drives a steam turbine to generate electric power. For given environmental conditions, the problem was to determine the control variable set points that could generate the *load* (equality constraint), without exceeding *CO* and *SO* limits (inequality constraints), while minimizing both *Heat Rate* and *NOx* emissions. After using first-principles-based methods and domain-knowledge to identify the relevant model inputs, we built a nonlinear neural-network to map the inputs space (control variable set-points) and time variable, ambient uncontrollable variables, to each of the outputs of interest, which represented our objectives and constraints. As shown in the above references, we used an evolutionary multi-objective optimizer to evolve the set points and identify the Pareto-optimal set of input-output vector tuples that satisfy operational constraints.

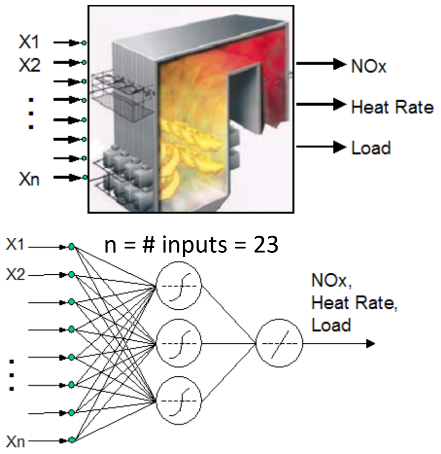


Fig. 8. Power plant input-output relationships. Each output (*NOx, Heat Rate, Load*) is modeled by a committee of predictive neural networks (adapted from reference [7]).

We noted the importance of reducing the neural networks uncertainty, as they generated the fitness function that drove the evolutionary search. In reference [41] we showed how we managed the model extrapolation error by using the equivalent of continuous case-based reasoning. We also need to address the intrinsic model uncertainty of the NNs. So, we performed preliminary experiments using the same data set, and created a *fixed committee* of neural networks, injecting diversity by bootstrapping their training set. We noted that model performance was significantly improved by fusing the outputs from the ensemble of models.

In references [36] we presented a method called *locally weighted fusion*, which aggregated the results of multiple predictive models based on local accuracy measures of these models in the neighborhood of the probe point for which we want to make a prediction. This neighborhood was computed on demand, following a Lazy Learning technique. In reference [7], we extended our experimentation by compiling the error information and avoiding the run-time search for the closest points. Figure 8, adapted from reference [7] shows the (23-3-1) structure of the NN's used in the ensemble.

This paper is an extension of the work initiated in reference [7]. We have proposed a complete architecture for selecting a *dynamic committee* of neural networks, and a *dynamic fusion*, based on the query and the meta- information. We extended the experiments with the same data set to compare our results with the ones previously obtained in references [7, 36].

5.2 Preliminary Experimental Results

The following tables show a sample of early experiments performed in [36] and [7], followed by the current experiments, in which we used part of the proposed architecture.

Table 2. Experimental Results (Baseline, Global, Local fusion without bias compensation)

Exp. #	Fusion Strategy	Heat Rate MAE [Btu/KwHr]	Heat Rate pg [%]	NOx MAE [lb/MBtu]	NOx pg [%]	Load MAE [MW]	Load pg [%]
1	Baseline: average of 30 predictors	91.79	0.00	0.0228	0.00	1.05	0.00
2	Best of 30 predictors	85.1	7.29	0.0213	6.58	0.987	6.00
3	Global Average	87.15	5.06	0.0214	6.14	1.042	0.78
4	Global GWF	86.91	5.32	0.0214	6.14	1.04	0.95
5	Global Least Square	83.05	9.52	0.02	**12.28**	0.984	**6.29**
6	H-Rect+W1 No bias	82.19	10.46	0.0202	11.40	1.024	2.48
7	H-Rect No bias	87.15	5.06	0.0214	6.14	1.042	0.78
8	H-Rect+W2 No bias	83.93	8.56	0.0208	8.77	1.03	1.93
9	1-nn No bias	81.19	**11.55**	0.0214	6.14	1.008	4.02
10	5-nn No bias	84.31	8.15	0.0206	9.65	1.029	1.97
11	CART EW No bias	87.15	5.06	0.0213	614	1.042	0.78
12	CART UW0 No bias	87.15	5.06	0.0213	614	1.042	0.78

The data set was comprised by 8,000+ records of daily operations, sampling the mapping between operational set points under given environmental conditions and power plant outputs, such as *Heat Rate* (*HR*), *NOx* emissions, and generated *load*. Roughly 25% of these records were used as the validation set.

In table 2, the baseline labeled as experiment 1, is the simple average of the outputs of the thirty neural networks. As such, the column labeled *pg* (*percentage gain*) is 0%. We will use *percentage gain* to show the percentage improvement over the baseline for each experiment. Experiment 2 reports the output of the best predictor (a posteriori). The remaining ten experiments in Table 2, show the results of various fusion combinations, *without performing bias compensation*, i.e., weighing the outputs of the models using different fusion schemes: variations of global fusion (experiments 3-5), local fusion based on Lazy learning (experiments 6-10), and CART trees to selected the local points (experiments 11-12). Most of these results were reported in reference [36]. The conclusion from this table is that *without bias compensation* the improvements are marginal: *HR pg:* 11%, *NOx pg:* 12%; *Load pg:* 6%.

In Table 3, we *perform bias compensation for all experiments*. In a separate set of experiments (not shown for sake of brevity) we concluded that the best CART trees are the one mapping the feature space to the **signed error** (as opposite to the unsigned error or the output of the model). Thus, with the exception of experiments 13-17, which show the results for lazy leaning type of local fusion (i.e., without using CART tree as part of the meta-information), all other experiments (18-33) use the signed error derived CART tree to compile their local performance in the validation set.

Experiments 18-27 show the result of using all thirty models with different weighting schemas: from equal weights (exp. 18), to unequal weights derived from linear or exponential kernel functions using as arguments various elements of the performance space in the pre-selection stage, e.g., distance of the query from the centroid, standard deviation of the error, etc. Overall the results are comparable (with very similar standard deviations of the error). The best result in this set of the experiments is from Unequal Weights-7 (UW7), in which we used a linear kernel function and the standard deviation of the error, i.e.: $K\left(\frac{\sigma_S(e|q)}{h}\right)$.

Table 3. Experimental Results (All with bias compensation, CART trees)

Exp. #	Fusion Strategy	Heat Rate MAE [Btu/KwHr]	Heat Rate pg [%]	NOx MAE [lb/MBtu]	NOx pg [%]	Load MAE [MW]	Load pg [%]
13	H-Rect+W1 bias	69.20	24.61	0.0140	38.60	0.855	18.57
14	H-Rect bias	69.23	24.58	0.0140	38.60	0.855	18.56
15	H-Rect+W2 bias	69.16	24.66	0.0140	38.60	0.854	18.63
16	1-nn bias	72.99	20.48	0.0143	37.28	0.861	17.98
17	5-nn bias	76.34	16.83	0.0169	25.88	0.903	14.04
18	CART EW bias	60.62	33.96	0.0117	48.68	0.718	31.62
19	CART UW0 bias (nf)	68.56	25.31	0.0148	35.09	0.817	22.19
20	CART UW1 bias (nf)	60.57	34.01	0.0111	**51.32**	0.725	30.92
21	CART UW2 bias (nf)	60.45	34.14	0.0116	49.12	0.719	31.50
22	CART UW3 bias (nf)	64.62	29.60	0.0130	42.98	0.733	30.20
23	CART UW4 bias (nf)	62.31	32.12	0.0125	45.18	0.721	31.35
24	CART UW5 bias (nf)	60.10	34.53	0.0145	36.40	0.792	24.59
25	CART UW6 bias (nf)	59.75	34.90	0.0129	43.42	0.745	29.04
26	CART UW7 bias (nf)	59.77	34.88	0.0115	49.56	0.713	**32.10**
27	CART UW8 bias (nf)	61.87	32.60	0.0122	46.49	0.743	29.25
28	CART UW7 bias (f1)	59.75	34.91	0.0117	48.68	0.721	31.35
29	CART UW7 bias (f2)	65.04	29.14	0.0135	40.79	0.776	26.10
30	CART UW7 bias (f3)	64.82	29.38	0.0130	42.98	0.772	26.46
31	CART UW7 bias (f4)	62.18	32.26	0.0122	46.49	0.744	29.12
32	CART UW7 bias (f5)	60.54	34.04	0.0117	48.68	0.726	30.86
33	CART UW7 bias (f6)	59.69	**34.98**	0.0116	49.12	0.716	31.86

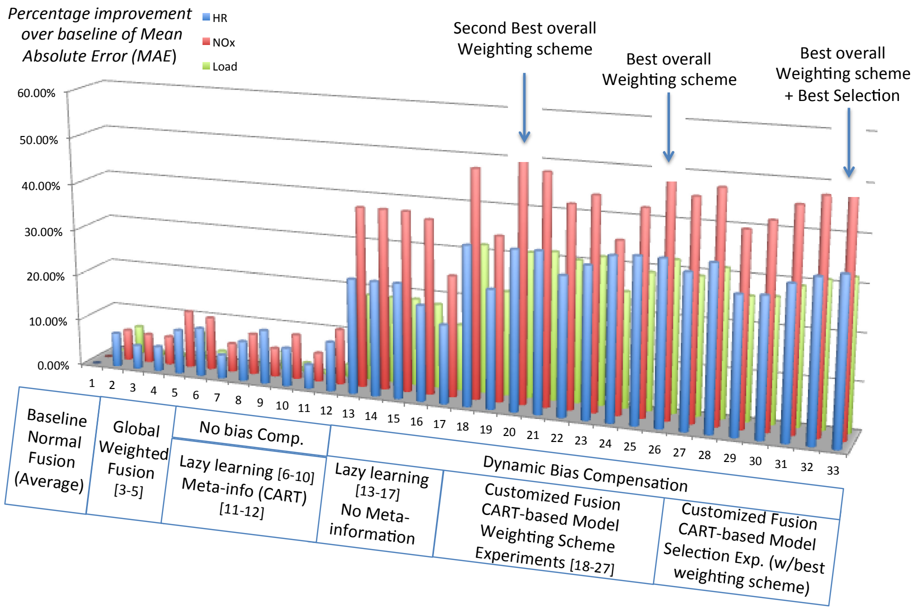


Fig. 9. Summary of experimental results

Finally, in experiments 28-33, we fixed this weighting scheme and searched for the best value of *p*, the number of models to use in the ensemble. The results, only valid for this specific application and for the initial DB of models, indicate that ~80% of the models provide the best results. All experiments results are illustrated in figure 9.

6 Conclusions

6.1 Analysis for Regression Problems

The experiments presented in this paper are preliminary in nature and have not completely exercised our approach. By using thirty neural networks trained on the entire feature space, we did not compare global and local models. As a result, we did not need to perform the filtering stage, i.e., $r = m = 30$. The selection process indicated that the best performance was obtained with about 80% of our models (i.e., $p = 25$). This was the result of performing a local search in the number of models. This process should be repeated for different applications and for a different DB of models. From the same experiments, we learned that there are statistically significant differences in performing dynamic bias compensation, and in using the local bias compiled in the CART trees. After experimenting with unsigned error, model output, and signed error, we concluded that the latter was the best way to generate the CART trees as part of the models meta-information. Finally, given that we controlled the original model generation and used bootstrapped training sets to inject diversity in the models, we did not need to perform an entropy-based final selection stage (i.e., $p = k = 25$). However, we will need to execute this step for larger DB of models in which we are agnostics about the model generation process. Our next step will be to test this hypothesis, by creating a DB with hundreds of models, generated by *FlexGP* [16], and exercising the remaining stages of our approach. The following table summarizes our current findings.

Table 4. Regression Problems: Design Parameters and Processes (Preliminary Version)

Design Phase	Design Choices	Generalization / Process
Meta-Information	CART Trees based on signed error; limited pruning	OK for all regressions
Model Filtering	N/A: All global models applicable by construction	**Process**: Apply Hyper-rectangles for local models
Model Pre-Selection	80% of applicable models (for this application)	**Process**: Greedy search in error space to find number of pre-selected models
Final Model Selection	N/A: model diversity built-in with bootstrapping	**Process**: Greedy search in entropy space to find error-uncorrelated models
Dynamic Fusion	Dynamic bias compensation (μ_e from leaf node)	OK for all regressions
	Unequal weights based from kernel function based on model prediction error (σ_e)	OK for all regressions

6.2 Future work for Classification Problems

In reference [37], we used lazy learning to perform local fusion for classification problems. We want to extend our approach to these types of problems and design a set of experiments to validate it. Specifically, let us assume that a classifier maps a *n*-dimensional feature space *X* into a *C*-dimensional space *Y*, where *C* is the number of classes. Then, the output of each model *k* for the training record *j*, is a normalized probability density $\varpi_k(j)$, where $\varpi_k(j)$ is a (*C*+1) dimensional vector. The error of each model *k* for the training record j is computed as $= e_k(j) = \varpi_k(j) - t_k(j)$, where $t_k(j)$ is a binary *C+1* dimensional vector in which only one element is 1, indicating the correct classification for training record *j*. Since $\varpi_k(j)$ is a normalized probability density, then the sum of all the elements of $e_k(j)$ equals 0.

By following a process similar to the one for the regression problems, we will train a CART Tree T_k for *each* class value. The tree will map the feature space *X* to the error vector *e* (for that specific class value*)*, such that in each leaf node we will cluster the subset of the training records that have similar classification errors. We will compute the mean error of the *k*th classifier over the points in each leaf node. In a fashion similar to the regression, we will refer to the average error of the leaf node $L_s(q)$, in which query q was classified, as $\mu_k(e|q)$. For a given query we will use the same model assembly steps (filtering, preference, and final selection based on entropy maximization). For the selected *k* models, we will perform a similar *bias compensation*. For the case when all *k* models are equally weighted, we have:

$$\hat{y}(q) = argmax\left\{\frac{1}{k}\sum_{s=1}^{k}(y_s(e) - \mu_s(e|q))\right\} \quad (8)$$

We also plan to perform a design of experiments to determine the most appropriate weighting scheme for each model in a manner similar to our regression experiments.

Acknowledgements. I would like to thank my colleagues Ya Xue, Fred Xue, Dustin Garvey, Steve Gustafson, Dongrui Wu, Scott Evans, Jianhui Chen, Weizhong Yan, and many others for their feedback and constructive criticism that helped shaping this idea.

References

1. Bonissone, P.: The life cycle of a fuzzy knowledge-based classifier. In: Proc. North American Fuzzy Information Processing Society (NAFIPS 2003), Chicago, IL, pp. 488–494 (2003)
2. Patterson, A., Bonissone, P., Pavese, M.: Six Sigma Quality Applied Throughout the Lifecycle of and Automated Decision System. Journal of Quality and Reliability Engineering International 21(3), 275–292 (2005)
3. Bonissone, P., Varma, A., Aggour, K.: An Evolutionary Process for Designing and Maintaining a Fuzzy Instance-based Model (FIM). In: Proc. First Workshop of Genetic Fuzzy Systems (GFS 2005), Granada, Spain (2005)
4. Bonissone, P., Varma, A., Aggour, K., Xue, F.: Design of local fuzzy models using evolutionary algorithms. Computational Statistics and Data Analysis 51, 398–416 (2006)

5. Bonissone, P.: Soft Computing: A Continuously Evolving Concept. Int. J. Computational Intelligence Systems 3(2), 237–248 (2010)
6. Bonissone, P., Cadenas, J.M., Garrido, M.C., Diaz, R.A.: A Fuzzy Random Forest. International Journal of Approximate Reasoning 51(7), 729–747 (2010), doi:10.1016/j.ijar.2010.02.003
7. Bonissone, P., Xue, F., Subbu, R.: Fast Meta-models for Local Fusion of Multiple Predictive Models. Applied Soft Computing Journal 11(2), 1529–1539 (2008), doi:10.1016/j.asoc.2008.03.006
8. Bonissone, P., Eklund, N., Goebel, K.: Using an Ensemble of Classifiers to Audit a Production Classifier. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) MCS 2005. LNCS, vol. 3541, pp. 376–386. Springer, Heidelberg (2005)
9. Evangelista, P., Embrechts, M., Bonissone, P., Szymanski, B.: Fuzzy ROC Curves for Unsupervised Nonparametric Ensemble Techniques. In: IJCNN 2005, Montreal, Canada, pp. 3040–3045 (2005)
10. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report EECS-2009-28, EECS Department, University of California, Berkeley (2009)
11. Howe, J.: Crowdsourcing- Why the Power of the Crowd Is Driving the Future of Business. Random House, New York (2008)
12. Vance, A.: Kaggle's Contests: Crunching Numbers for Fame and Glory. Businessweek, January 04 (2012)
13. Bonissone, P., Subbu, R., Aggour, K.: Evolutionary Optimization of Fuzzy Decision Systems for Automated Insurance Underwriting. In: Proc. FUZZ-IEEE 2002, Honolulu, HI, pp. 1003–1008 (2002)
14. Aggour, K., Bonissone, P., Cheetham, W., Messmer, R.: Automating the Underwriting of Insurance Applications. AI Magazine 27(3), 36–50 (2006)
15. Bennett III, F.H., Koza, J.R., Shipman, J., Stiffelman, O.: Building a parallel computer system for $18,000 that performs a half peta-flop per day. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, pp. 1484–1490. Morgan Kaufmann, San Francisco (1999)
16. Sherry, D., Veeramachaneni, K., McDermott, J., O'Reilly, U.-M.: FlexGP: Genetic Programming on the Cloud. To Appear in Parallel Implementation of Evolutionary Algorithms, EvoStar 2012, Malaga, Spain (2012)
17. Bonissone, P.P., Iyer, N.: Soft Computing Applications to Prognostics and Health Management (PHM): Leveraging Field Data and Domain Knowledge. In: Sandoval, F., Prieto, A.G., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 928–939. Springer, Heidelberg (2007)
18. Roli, F., Giacinto, G., Vernazza, G.: Methods for Designing Multiple Classifier Systems. In: Kittler, J., Roli, F. (eds.) MCS 2001. LNCS, vol. 2096, pp. 78–87. Springer, Heidelberg (2001)
19. Kuncheva, L.: Switching between selection and fusion in combining classifiers: An experiment. IEEE Transactions on Systems, Man, and Cybernetics, Part B 32(2), 146–156 (2002)
20. Tumer, K., Ghosh, J.: Error correlation and error reduction in ensemble classifiers. Connection Science 8, 385–404 (1996)

21. Kuncheva, L., Whitaker, C.: Ten measures of diversity in classifier ensembles: Limits for two classifiers. In: Proceedings of IEE Workshop on Intelligent Sensor Processing, Birmingham, p. 10/1-6 (2001)
22. Wolpert, D.H.: Stacked generalization. Neural Networks 5, 241–259 (1992)
23. http://en.wikipedia.org/wiki/Meta_learning_(computer_science)
24. Duch, W., Grudzinski, K.: Meta-learning: searching in the model space. In: Proc. of the Int. Conf. on Neural Information Processing (ICONIP), Shanghai, China (2001)
25. Schaul, T., Schmidhuber, J.: Meta-learning. Scholarpedia 5(6), 4650 (2010)
26. Utgoff, P.: Shift of bias for inductive concept learning. In: Michalski, R., Carbonell, J., Mitchell, T. (eds.) Machine Learning, pp. 163–190 (1986)
27. Schmidhuber, J.: Evolutionary principles in self-referential learning. Diploma thesis, Institut für Informatik, Technische Universität München (1987)
28. Atkeson, C.G.: Memory-based approaches to approximating continuous functions. In: Casdagli, M., Eubank, S. (eds.) Nonlinear Modeling and Forecasting, pp. 503–521. Addison Wesley, Harlow (1992)
29. Atkeson, C.G., Moore, A., Schaal, S.: Locally Weighted Learning. Artificial Intelligence Review 11(1-5), 11–73 (1997)
30. Bersini, H., Bontempi, G., Birattari, M.: Is readability compatible with accuracy? From neuro-fuzzy to lazy learning. In: Freksa, C. (ed.) Proceedings in Artificial Intelligence 7, pp. 10–25. Infix/Aka, Berlin (1998)
31. Deb, K.: Multi-objective optimization using evolutionary algorithms. J. Wiley (2001)
32. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithm MOP Approaches, Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic (2002)
33. Bonissone, P., Subbu, R., Lizzi, J.: Multi Criteria Decision Making (MCDM): A Framework for Research and Applications. IEEE Computational Intelligence Magazine 4(3), 48–61 (2009)
34. Breiman, L., Friedman, J., Olshen, R.A., Stone, C.J.: Classification and regression trees. Wadsworth (1984)
35. Bonissone, P., Cheetham, W.: Fuzzy Case-Based Reasoning for Decision Making. In: Proc. FUZZ-IEEE 2001, Melbourne, Australia, vol. 3, pp. 995–998 (2001)
36. Xue, F., Subbu, R., Bonissone, P.: Locally Weighted Fusion of Multiple Predictive Models. In: IEEE International Joint Conference on Neural Networks (IJCNN 2006), Vancouver, BC, Canada, pp. 2137–2143 (2006), doi:10.1109/IJCNN.2006.246985
37. Yan, W., Xue, F.: Jet Engine Gas Path Fault Diagnosis Using Dynamic Fusion of Multiple Classifiers. In: IJCNN 2008, Hong Kong, pp. 1585–1591 (2008)
38. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
39. Gray, A., Moore, A.: N-Body Problems in Statistical Learning. In: Proc. Advances in Neural Information Processing Systems, NIPS (2001)
40. Subbu, R., Bonissone, P., Eklund, N., Yan, W., Iyer, N., Xue, F., Shah, R.: Management of Complex Dynamic Systems based on Model-Predictive Multi-objective Optimization. In: CIMSA 2006, La Coruña, Spain, pp. 64–69 (2006)
41. Subbu, R., Bonissone, P., Bollapragada, S., Chalermkraivuth, K., Eklund, N., Iyer, N., Shah, R., Xue, F., Yan, W.: A review of two industrial deployments of multi-criteria decision-making systems at General Electric. In: First IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM 2007), Honolulu, Hawaii (2007), doi:10.1109/MCDM.2007.369428