IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 6, 2018, accepted August 7, 2018, date of publication August 20, 2018, date of current version September 21, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2866046

Proposing Logical Table Constructs for Enhanced Machine Learning Process

MUHAMMAD FAHIM UDDIN[1], (Student Member, IEEE), SYED RIZVI[2], (Member IEEE), AND ABDUL RAZAQUE[3], (Senior Member, IEEE)

[1]Department of Computer Science, School of Engineering, University of Bridgeport, Bridgeport, CT 06604, USA
[2]Department of Information Science and Technologies, Penn State University, Altoona, PA 16601, USA
[3]Department of Computer Science, New York Institute of Technology, New York, NY 11568-8000, USA

Corresponding author: Muhammad Fahim Uddin (muddin@bridgeport.edu)

This work was supported by the School of Engineering, University of Bridgeport, Bridgeport, CT, USA.

ABSTRACT Machine learning (ML) has shown enormous potential in various domains with the wide variations of underlying data types. Because of the miscellany in the data sets and the features, ML classifiers often suffer from challenges, such as feature miss-classification, unfit algorithms, low accuracy, overfitting, underfitting, extreme bias, and high predictive errors. Through the lens of related study and latest progress in the field, this paper presents a novel scheme to construct logical table (LT) unit with two internal sub-modules for algorithm blend and feature engineering. The LT unit works in the deepest layer of an enhanced ML engine engineering (eMLEE) process. eMLEE consists of several low-level modules to enhance the ML classifier progression. A unique engineering approach is adopted in eMLEE to blend various algorithms, enhance the feature engineering, construct a weighted performance metric, and augment the validation process. The LT is an in-memory logical component, that governs the progress of eMLEE, regulates the model metrics, improves the parallelism, and keep tracks of each module of eMLEE as the classifier learns. Optimum fitness of the model with parallel ''check, validate, insert, delete, and update'' mechanism in *3-D* logical space via structured schemas in the LT is obtained. The LT unit is developed in Python, C#, and R libraries and tested using miscellaneous data sets. Results are created using GraphPad Prism, SigmaPlot, Plotly, and MS Excel software. To support the built and implementation of the proposed scheme, complete mathematical models along with the algorithms, and necessary illustrations are provided in this paper. To show the practicality of the proposed scheme, several simulation results are presented with a comprehensive analysis of the outcomes for the metrics of the model that the LT regulates with improved outcomes.

INDEX TERMS Big data, predictive modeling, data mining, machine learning, algorithm, parallel processing of machine learning metrics reading, model tuning, algorithm blending, optimum fitting, feature engineering, overfitting, *eMLEE*, logical table.

I. INTRODUCTION

A. BACKGROUND

Machine learning *(ML)* unveils tremendous potential in the data science and predictive analytics. *ML* algorithms particularly in the supervised learning (*SL*) zones have advanced into improved modeling of the underlying data for decision making [1], predictive analytics [2], personality prediction [3] etc. Great surveys such as [4]–[6] including domains of the unstructured data [7] and social networking platforms have shown incredible importance of *ML* algorithms' tuning and improvements [8]. Useful techniques such as algorithm boosting [9], optimization [10], conditional densities, gradient descent, inference [11], parallel processing [12], and convex minimization have played progressive roles to improve the classifier learning of the existing techniques in *SL*.

The latest progress in the research of data mining and predictive modeling with the relevance of *ML* have promoted great opportunities and challenges for future works. Prior to developing the work presented in this article, we investigated the application of *ML* techniques reported in the literature. We found high relevance in the areas of healthcare domain applications to predict hospital admissions [13], [14], practical applications to deal with lethal diseases such as HIV [15], and biomedical device applications [16]. Other areas included security, facial recognition, engineering solutions,

and general modeling such as antenna design optimization [17], image classification [18], and real-world experiences such as driver safety [19], and algorithm optimization [20], [21].

Therefore, considering the progresses outlined above, and the literature study provided in Section II, we see the necessity of introducing a parallel processing unit in the *ML* underlying models. The diversity in the data and features have motivated us to improve the latest state of *ML* models by building enhanced *ML* engine engineering *(eMLEE)* process specially to address the challenges such as overfitting, underfitting, bias, low accuracy, poor generalization, and predictive errors. While the details of *ML* engine engineering *(i.e., eMLEE)* is beyond the scope of this article, *LT* constructs are presented. *LT* is a vital component of *eMLEE*. As shown in **Fig. 1**, *eMLEE* is composed of four modules. The triangular shape emphasizes on the idea of *3D* concept that the model operates on. The thick arrow between *eFES, eABT,* and *LT* module shows core integration than the other two modules of *eMLEE*. The major modules of *eMLEE* are enhanced Algorithm Blend and Tuning (*eABT*) and enhanced Feature Engineering and Tuning (*eFES*) that are regulated by *LT* internal unit.

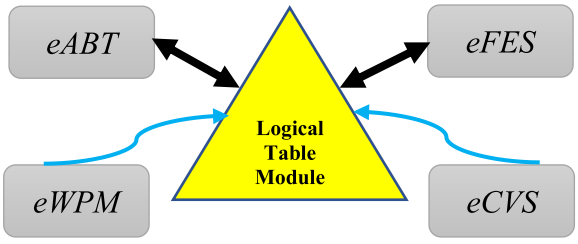


FIGURE 1. **This illustration shows the elevated system externals of *eMLEE*. *LT* interacts with *eFES* and *eABT* on the deeper level. It coordinates and regulates the metrics of the learning process in parallel mode.**

eMLEE model is based on parallel processing and learns from its mistakes (i.e., processing and storing the wrong predictions). Its modules are, *i*) enhanced algorithm blend and tuning (i.e., *eABT*) to optimize the classifier performance, *ii)* enhanced feature engineering and selection(i.e., *eFES*) to optimize the features handling, *iii)* enhanced weighted performance metric (*eWPM*) to validate the fitting of the model, and *iv)* enhanced cross validation and split (*eCVS*) to tune the validation process. Out of these, *eCVS* is at infancy of the research work. Existing research, as discussed in *Section II* has shown the limitations of general purpose algorithms in *SL* for predictive analytics, decision making, and data mining. Thus, *eMLEE* finds its place to fill the gaps that Section II discusses.

Finally, *LT* is built to coordinate the internal flow of *eMLEE* as introduced briefly in the above paragraph.

B. MOTIVATION AND NOVELTY

The motivation to develop this specialized unit comes from the uniquely thought, experimented, developed, and incorporated parallelism in an enhanced machine learning process with innovative blending and tuning as discussed earlier. *eMLEE* comes in to addressing ''No Free Lunch theorem'' problem, feature correlation, and selection improvement. It addresses challenges such as overfit, underfit, bias, predictive errors, and poor generalization. In our experimental tests during the evolution of this research, we felt the necessity of ''inline'' unit as a centralized part of this engine that governs, regulates, and keeps track of machine learning process on the underlying data. The challenge of trade-off between vital metrics such as complexity, accuracy, speed, etc. becomes also very important and that is where *LT* plays a significant role. *LT* creates parallel process for each element in each run governed by *3D* object co-ordinates (*x,y and z*) and then makes observations in the real time of classifier learning and updates its logical row in the table. This approach is novel to the best of our survey and knowledge.

C. CONTRIBUTIONS

Below are the contributions of the work presented in this article.

i. The in-memory processing unit is designed and governed by algorithms, that ensures the model internals are at maximum performance during blend.

ii. A blended model such as eMLEE will use LT unit to tune the performance metrics in the real-time. This feature is built using mathematical constructs.

iii. The 3D logical modeling is used to reserve x (underfitting), y (overfitting), and z (optimum-fitting). Algorithms are written to score each dimension during the learning process. 3D improves the visualization process during simulations.

iv. LT also is needed to teach the model to learn from its mistakes. However, this contribution is left for another article that elaborates deeply on eABT unit.

v. Improves the trade-off between various metrics such as complexity, speed, accuracy etc., using real-time evaluation and locating the optimized point for each element using 3D visualization and recording techniques.

vi. Finally, LT unit is structured as a centralized component of blended model, for keeping coordination between each component regulated with built-in parallel processing.

D. PARALLELISM

As stated earlier, *LT* regulates *eABT* and *eFES*. *LT* plays a key role to introduce an effective parallelism (i.e., parallel processing) in *eMLEE* engine. Here we summarize the parallel processing of the engine that incorporates the *LT* unit, as centralized and vital component of the system for the proposed enhanced machine learning process.

i) **Outer layer to *eABT***, where *eABT* unit communicates with other units of the *eMLEE* such as *eABT, eWPM, eCVS* and *LT*. Parallelism is done through real time metrics measurement with *LT* object. Based on

classifier learning, *eABT* reacts in the inner layer (defined next). Other units such as *eFES* and *eCVS* enhance the feature blend and test-training split in parallel, while *eABT* is being trained. In other words, all four units including *eABT* regulated by *LT* unit, are run in parallel to improve the speed of the learning process and validation for the blend as processed in *eABT* unit. However, *eABT* can also work without being related to the other units, if researchers and industrialists may however choose so.

ii) **Inner layer to *eABT***, where addition and removal of the algorithm are done in parallel. When the qualifying algorithm is added, the metrics are measured by the model to see if fitness improves, and then algorithms are added or removed one by one to see the effect on the fitness function. This may be done sequentially, but parallelism improves the insurance that each algorithm is evaluated at the same time, the classifier is incorporating metrics reading from *LT* object and speed of the process improves, especially when a huge dataset is being processed.

E. SIMULATION STRATEGY AND RESULTS PRODUCTION

We provided detailed information about our data sources and tools in Appendix. Because our research investigated the ensembling of algorithms (that learn based on different classifier curves), we considered a very miscellaneous set of training and testing data to ensure that our blend of algorithms stay in the optimum fitting range for the real-world experiments and analytics. Similarly, because of the feature engineering and tuning, it was authoritative to our work using data with assorted set of features involved.

We also uniquely adopted the following approaches to make our experiments more reliable, easy to interpret, reproduce, and analyze for the model's validation, integrity and evaluation.

1. We conducted 100's of experiments to cover wide range of datasets that helped achieved in-depth training of the model to study various ranges of metrics. We then re-evaluated our math constructs and algorithms to improve fitness. This way, our math constructs governed by our algorithms, ensured the integrity of the model through the lens of real-world data and testing.
2. We also sampled all these experiments and developed a novel approach of 10-experiemntal rule. This way, we could present our outcomes and analysis with improved visualization and interpretation, as presented in this article.

We used Python and R data analysis packages along with C# libraries to test our algorithms. We used Prism, SigmaPlot, and Excel to produce our simulated results. We uniquely adopted the approach of *3D* to have more observational value to our analysis for the proposed model. This article reports preliminary results in *3D* modeling and matured results in *2D* modeling of the simulations runs for the experiments we have conducted.

F. PAPER ANATOMY

The rest of the article is structured as follows. Section II discusses the related study that supports the contribution of *eMLEE* engine, the part of which is the *LT*, as proposed in this article. Section III & IV discuss the theory, algorithms, illustrations, and simulated results of the *LT* contructs for *eABT* and *eFES* respectively. Section V presents simulated and experimental results for the *LT* unit in depth. Section VI concludes with final remarks. Appendix and references are provided at the end.

Key Notations:

$A(x,y,z)$	*Algorithms Pool*
'x'	*over-fitness*
'y'	*under-fitness*
'z'	*optimum-fitness*
$\mathbb{R}$	*Ratio*
$\frac{1}{N_e}$	*Normalized error*
err	*Local error*
$Error$	*Global Error*
$k_{x,y,z}$	*Constant error function*
$\prod_{LT>0.5} \overline{\overline{A(z)}}$	*Regulating factor*
N	*Noise*
S	*Signal*
$\mathbb{B}_{A_n}$	*Blending Function for Algorithms*
$\mathbb{T}_{A_n}$	*Tuning Function*
C	*Cost Function*
∇d	*Euclidean distance*
γ	*Optimization Factor*
$Pr(err)$	*Probability on local error in even distribution space*
$+\mathbb{F}$	*Feature Adder handle*
$-\mathbb{F}$	*Feature Remover handle*
$u_x w_x, u_y w_y, u_z w_z$	*Weights optimization units in 3D space for each coordinate*
$Cor(x,y,z)$	*Correlation function*
$ss \subseteq \neg f_i$	*Markov blanket function for Probability of the fitness in each test*
$M(x,y,z)$	*Net Matrix Function for each fitness factor in x,y,z*

II. RELATED STUDY

Brief related study is provided in the areas of *ML*. This exploration of the literature helped and promoted our contribution of the *LT* for enhanced *ML* such as *eMLEE*.

Tuia *et al.* [22] provided a survey of active learning algorithms in the field of remote sensing image classification. Mainly focused on *SVM* algorithm, they discussed the issue of efficient training set, having high impact on the expected outcome. Their findings, results, and discussion

showed that active learning algorithms are making great progress especially for image classification and the type of data it involves. However, their contribution was limited to active learning, especially for image classification and may not be suitable to apply for a diverse set of data and features. Garcia *et al*. [23] provided a survey on discretization techniques with empirical analysis in supervised learning. Discretization is an important approach specially to improve the underlying algorithm in terms of feature/attribute tuning and qualitative analysis. They provided in-depth analysis and guidelines of various methods with taxonomy table of their findings. Their findings also suggested an ideal selection of a method for given problem. Their findings and experiments showed accuracy of various *ML* techniques but did not provide other metrics that may be of special interest especially when blend is being engineered for a greater generalization. Wang *et al.* [24] discussed the process of purchase decision in subject minds using *MRI* scanning images through *ML* methods. Using recursive cluster elimination based *SVM* method, they obtained higher accuracy (71%) as compared to previous findings as per their research. They utilized Filter (*GML*) and wrapping methods (*RCE*) for feature selection. Their work though provided great foundation and motivation for feature processing but did not provide the in-depth experiments of application of the technique on neutral subjects where feature may mislead, and algorithm design must take this into account. Tandon *et al.* [25] discussed the importance of machine intelligence in big data domain towards natural language. Their work provided great motivation towards mining common sense that can extracted from words of people, but it did not provide in-depth analysis of algorithms or features that may impact such intelligence during learning process. Hernandez *et al.* [26] discussed the parallel processing optimization in big data applications. Their results showed improved recommendations score for resources and workload but did not address or consider the parallel processing of various algorithms to see if that could further improve their work. Dai and Song [27] work was focused on multiple classifier systems (*MCS*s) with their contribution of supervised competitive learning algorithm (*SCL*) to improve the accuracy of the classifiers. Though their work showed satisfactory progress for accuracy measurements, did not consider other metrics of the supervise learning classifier especially if algorithm blend is intended.

Some of the work in the areas of engineering domains such as antenna design, wireless communication, chip designs and other biomedical engineering are using advanced *ML* techniques with recent availability of digital data. Liu *et al.* [17] addressed the low efficiency of evolutionary algorithms in Electromagnetic(*EM*) design problems due to the cost, and thus proposed a new method called surrogate model differential evolution for antenna synthesis using *ML* techniques. Their work was very limited to *EM* applications and did not provide the wide applicability to other domains of similar challenges in *EM* or Electrical engineering domains. Yu *et al.* [28] focused on weaknesses of semi-supervised clustering algorithms and to address these challenges, they proposed closure based constraint approach and random bases semi-supervised framework. They used datasets from medical domains such as cancer patients. Their work lacks dealing with pairwise constraints and removal of redundant constraints. Such limitation may be addressed by the work in the feature optimization and engineering as we propose. Xiao-jian *et al*. [29] advanced the work in optimization extreme learning machine (*OELM*) for the error penalty parameter C. Their work extended the traditional *OELM* classifier with the regularized parameter v. Their work created useful foundation for classifier parameter optimization. However, they lacked to confirm the stability of optimization if different classifiers were used or tested. Lara and Labrador [30] provided a survey on *ML* application for wearable sensors, based on human activity recognition. They provided a taxonomy of learning approach and their related response time on their experiments. Their work also supported feature extraction as an important phase of *ML* process. Their work provides great motivation for feature engineering and further improvement in feature selection and optimization. Vergara and Estevez [31] reviewed feature selection methods. Authors presented updates on results in unifying framework to retrofit successful heuristic criteria. The goal was to justify the need of feature selection problem in-depth concepts of relevance and redundancy. However, their work lacks to address the issues of model fitting when a diverse set of features are involved in datasets. Mohsenzadeh *et al*. [32] utilized a sparse Bayesian learning approach for feature sample selection. Their proposed relevance sample feature machine (*RSFM*), is an extension of *RVM* algorithm, previously invented. Their results showed the improvement in removing irrelevant features and producing better accuracy in classification, better generalization, less system complexity, reduced overfitting and computational cost. Ma *et al*. [33] utilized Particle Swarm Optimization (*PSO*) algorithm to develop their proposed approach for detection of falling of elderly people and enhance the selection of variables (i.e., hidden neurons, input weights, etc.) Their experiments showed higher sensitivity, specificity, and accuracy readings. Their work lacked to consider various algorithms in comparison with *PSO* to see if it might impact the modeling of the various metrics.

The following points highlight the weaknesses/gaps outlined by the related study and our in-depth literature review.

a) Algorithm and Feature blending for various algorithms are at infancy state in the research work published and applied, and thus lack lots of improvements, such as incorporating each algorithm and feature for maximum accuracy possible.

b) Algorithms are not taught to learn from their mistakes and thus LT is needed to fill this gap.

c) Hidden features, that can be of great predictive value are often overlooked and thus LT can improve this gap. Similarly, research related to the removal of irrelevant and redundant features are rarely found.

d) A real-time optimization functions are rarely implemented in other models, when the model learns and may fail to fit. LT, however works in-parallel during training and testing process to fill this gap.

e) Finally, general-purpose algorithms, such as LT, are not implemented where new modules like eABT or eFES can be extended to the existing models, such eMLEE.

III. LT THEORY OF eABT MODULE

LT as previously discussed, is a vital central unit of *eMLEE*. *LT* is based on *3D* novel concept of optimization to regulate the metrics and learning progress of the blended model. In this section, we first discuss each definition in plain English and then provide the details of building the *LT* unit mathematical constructs in conjunction with algorithm definition.

A. DEFINITIONS IN PLAIN ENGLISH

Definition 1 covers the theory of Adder and Remover functions for the *3D* objects formulation based on the progress of blend of algorithms and features incorporation as the classifier learning continues. This way each element is cross-checked in parallel and metrics readings are recorded as a new logical row or updated as existing row.

Definition 2 covers the theory of specialized factors for *LT* unit based on fitness of the classifier learning. These are used to construct a vital function known as scoring function that quantifies each factor for *3D* evaluation and identifying signal and noise in the dataset for further optimization.

Definition 3 covers the theory of specialized function as *Error Bound* to support rule of optimum fitness. This introduces a novel concept for error regulation in the proposed *LT* model. Staying between 20 % and 80 % ensures that model never over or under learns the data. This has been proved to be an effective approach in the results being observed by our study and work.

Definition 4 covers the theory of two vital functions/ constructs: *i) Blending, and ii) Tuning function*. These constructs turned out to be very useful for classification goals. This definition also constructs the *Binary Decomposition* function that *LT* object uses to formulate and determine the cost function.

Definition 5 covers the detailed theory of *Cost* function based on **Def. 4.** This cost function plays another vital role to evaluate the comparing elements in the algorithms and features and compute the accurate quantification of blending and tuning functions.

B. MATHEMATICAL CONSTRUCTION OF THE UNIT' INTERNALS

LT operates in the memory and is dynamically updated. It keeps tracks of the algorithms $A(x, y, z) = \{A_1, A_2, \ldots \ldots, A_n, \}$ as the *ML* process evolves to accomplish the final optimum fitting after it has incorporated all the algorithms from the pool. This helps achieve the optimum blending and tuning. *LT* stores data based on three dimensions, where '*x*' = *over-fitness, 'y' = under-fitness and 'z' = optimum-fitness*. In our model, we will be using "*-ness*" to mathematically phrase the metric for modeling purposes. At this stage, we refer fitness to be the overall performance of the model, and our goal is to reduce '*x*' and '*y*' to as minimum as zero and improve '*z*' to the highest possible value. '$\mathbb{R}$' is the ratio between the single error from an algorithm and averaged error of all the algorithms in the blend.' $\frac{1}{N_e}$ 'is the normalization factor for the error *'err'*. *'Err'* indicates the Overall error determined for the blended model.

Let us define constant error function:

$$k_{x,y,z} = \frac{1}{\sqrt{2\pi\,\mu^3}} + \mathbb{R}_{eABT} \tag{1}$$

Where, μ computes all the values of *x,y, and z* components during learning.

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x, y, z)_i \tag{2}$$

$$\mathbb{R}_{eABT} = \frac{1}{N_e}\left(\sqrt{\frac{err}{err + Err}}\right)^2 \tag{3}$$

Definition 1: *Let there be a Adder Function as 'AddFunc(A(x, y, z))', that adds each algorithm in the blend being processed, with Scoring Function as 'ScoFunc(0:1)' for each dimension in 3D space. Let there be a Remover Function as 'RemFunc(*), that must hold at-least one element per each test. * indicates the computed dimension.*

LT structure uses the grouping and scoring module. Scoring is based on binary number weights as being illustrated in Fig. 5 and based on the following rule.

```
Rule 1
If (LTObject.ScoFunc(A(i)>0.5)
Then
     Assign ''1"
Else Assign ''0"
```

$$LT\,(x, y, z) = \begin{cases} 1 & if\ LT.ScoFunc > 0.5 \\ 0 & if\ LT.ScoFunc < 0.5 \\ & ?\ Undetermined \end{cases} \tag{4}$$

By combining Gauss-Markov and Chebyshev methods [34] we construct adder Function $\bigoplus$ as given by

$$AddFunc = (A_n \cup A_{n+1})\left[\frac{\prod_{LT>0.5}\overline{\overline{A(z)}}}{BIN(min(x, y)}\right] \tag{5}$$

Our rule of thumb was 0.5 or 50 % to see how the model learns. This way, we can separate the zone of over learning and under learning from a border line of 50 %. Once classifier learns the zoning limits, it will decide this number itself. $\prod_{LT>0.5}\overline{\overline{A(z)}}$ acts a regulating factor that provides the continuous product for each value of *z-dimension* for which the *BIN* function returns the least possible value of *x* and *y*.

The removing function is $\bigominus$ given by

$$RemFunc(*) = (A_n \cap A_{n+1})\left[\frac{\prod_{LT<0.5}\overline{\overline{A(x, y)}}}{BIN(max(x, y)}\right] \tag{6}$$

It is imperative to validate the Adder and Remover functions at this point, using well known technique of *Frobenius norm* [35] *form:*

$$\|M\|(x,y,z) = k_{x,y,z}\sqrt{\sum_{x=1}^{X}\sum_{y=1}^{Y}\sum_{z=1}^{Z}M_{x,y,z}^{2}(AddFunc, RemFunc)} \quad (7)$$

Where M shows the matrix and we will elobrate on it in the later section.

Definition 2: *Let Op.F, Un.F, Ov.F, and Bias.F be the factors for Optimum-fitting, Underfitting, Overfitting, and Bias respectively of the algorithm under test in 3D model. There must be an equal random distribution for each till LT regulates the scoring function (SF) for each metric.*

Each metric swings from *{0:1}* based on the correlation of algorithm during each test. *LT* object receives the score for each element while classifier learns in *3D* space. Dimentionality reduction and multivariate classification techniques [36] can be used to construct a function for LT, as shown in (8) and (9),

$$LT(z) = ltElement \times \lim_{z>0.5}\{SF(Op.F)\} \quad (8)$$

$$SF(x,y,z) = LT(z) - \sum_{i=1}^{X}\sum_{j=1}^{Y}LT(i,j) \quad (9)$$

ltElement shows the object for *LT* such as *eABT. LT* object creates an entry in the memory for tracking the metrics for each element such as a particular algorithm under test, and it assigns the weights (binary based) to each metric as per in definition 2, for which the following is constructed:

```
PROCEDURE 1
Execute LT.ScoreMetrics(Un.F, Ov.F)
Compute LT.Quantify(*LT)
Execute LT.Bias(Bias.F, *)
*_Shows the pointer to the LT object.
```

Clearly, the noise in the data (i.e., the irrelevant or redudant) does not have good predictive value, thus the metrics stated in the definition 2 are highly affected by it. We build a loss function in correspondent to Noise (N) and Signal (S) (i.e., data of predictive value). Thus, we construct our binary loss function, based on our rule of thumb from the signal and noise for the training.

$$L(DS(S(x,y,z),N(x,y,z)) = \begin{cases} 0, \&(N(x,y,z) \geq 0.5 \geq S(x,y,z)) \\ 1, \&(S(x,y,z) \geq 0.5 \geq N(x,y,z)) \end{cases} \quad (10)$$

Not likely but loss function tends to get very unstable in a blend or 2+ algorithms where classifier function has a very low variant and probability of distribution of fitness function across z-axis (i.e. fitness model) is very wide. Thus generalization ability of the net model becomes very significant. The 3-branch diagram in **Fig 2**. shows the spread of algorithm in each dimension, that corresponds to L and N component. An example is shown in rectangular shape for L and N co-variance, for algorithms blend. Each small circle represents the occurrence of a big circle on the left.

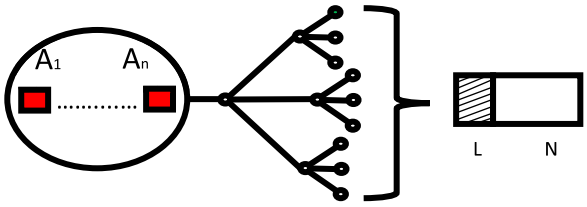


FIGURE 2. Illustration of Loss and Noise interoperation based on x, y, and z dimensions, for algorithms blend.

Definition 3: *Let lt.Err be the specialized error function that implements the rule of optimum fitness (RoOpFit) as 0.2 < lt.Err < 0.8. Every entry in LT must adhere to this rule.*

```
Rule 2
Except random errors, lt.Err must be
regulated to stay in between 20 to 80 %
to avoid over and under learning.
If lt.err < 0.2
    Then : Label it 'Overlearning'
Elseif lt.err > 0.8
    Then:Label it 'underlearning'
```

From the literature, we can implement the RMSE function for error determination, thus, we use our rule to build:

$$max(e:0.8) = \frac{1}{E}\sum_{i=1}^{Ne}\{(RMSE_i) - (100+0.2)/E\} \quad (11)$$

$$min(e:0.2) = \frac{1}{E}\sum_{i=1}^{Ne}\{(RMSE_i) - (100+0.8)/E\} \quad (12)$$

Using (11,12), we build the *RoOpFit* to lead towards determination of *lt.Err* function. *RoOpFit* regulates the error that *LT* object can trigger for each test. Using kernel density function [36] and margin limits in Lipschitzness [9], we build

$$RoOpFit = \max_{err} < 0.8\sum_{i,j}^{x,y}(A_{i,j}) - \min_{err} > 0.2\sum_{j,i}^{y,x}(A_{j,i}) \quad (13)$$

$$lt.Err(x,y,z) = \left\{\prod_{RoOpFit(z)} - \prod_{RoOpFit(z)} BIN((x,y,z)\right\}(A_n) \quad (14)$$

With this error function being constructed, we can easily see the divergence in the optimum zone of z-axis. As discussed before, the *LT* object reads the previous entry and then based on the data from the training blend classifier, it updates (i.e., writes or deletes) in its logical structure (i.e., new or existing row of records).

Definition 4: *Let $\mathbb{B}_{A_n}$ be a blending function and $\mathbb{T}_{A_n}$ be a tuning function that LT object must compute (detailed in algorithm definition). Let $BIN((x,y,z))$ be a binary decomposition function for each entry in LT. There must exist a Cost function as 'C'.*

In *SL*, the classifier function $Classifier(\Delta S(x,y,z)) = \frac{1}{N}$ as identified, where $\Delta S = \{(i_1,o_1),(i_2,o_2),\ldots\ldots(i_k,o_k)\} \in (\mathbb{I}(input)\times\mathbb{O}(output))^k$. Where: $\mathbb{I} \subset \mathcal{R}^d$,

and for regression: Errors $o_k \in \mathcal{R}$ For Classification: o_k is a discrete value. In Linear Classification, as generally done in *SVM* concept: we can use Lagrange multipliers [37] to present the problem in equivalent maximization on γ :

$$\gamma = argmax \sum_{k=1}^{N} \gamma_k - \frac{1}{2} \sum_{k,l=1}^{N} \gamma_k \gamma_l (o_k o_l < i_k, i_l >) \tag{15}$$

In **Fig 3**, triangle is lying on z-axis, with the direction of momentum as being engineered in the model. ∇d shows the Euclidean distance between two algorithms under test. The three Matrices shown are typical values for the sampling of the several hundred experiments. The encircled values show the optimized value of each axis for the desired optimization as LT object stores and reports.

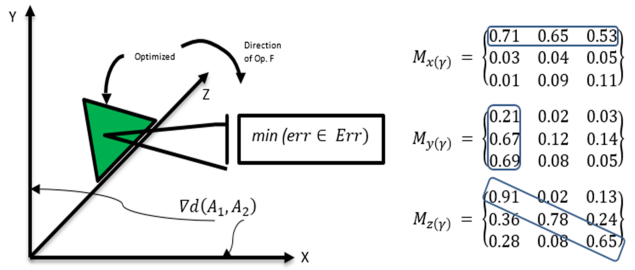


FIGURE 3. – Illustration of optimum fitness logical *(x, y, z)* triangle.

Equation (16) shows that Blend function is composed of three parts that work on *AddFunc*, *RemFunc* and score for each algorithm in each dimension as LT computes (See Algorithm 1 definition). Using Regularization in local minima where error is minimum but lipschitz loss [9] is unknown, we use vector product to keep the uniformity at minimum random distribution such that $z \neq 0$ *AND* $x, y < 0.5$, thus, we construct

$$\mathbb{B}_{A_n}(0:1) = \prod_{k=1}^{AddFunc(k)} A_k \times \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} + \prod_{m=1}^{RemFunc(m)} -A_k \times \begin{pmatrix} -x_m \\ -y_m \\ +z_m \end{pmatrix} \tag{16}$$

Tuning function is constructed using *Err* and *err* functions. As we stated earlier the *RoOpF* must be followed for blend to be tuned for improved optimization. *LT* object ensures by recording and manipulating the metrics, as per algorithm structure, discussed later. Thus, we can write:

$$\mathbb{T}_{A_n}(0:1) = \frac{1}{N} \sum_{i=1}^{N} -(\mathbb{B}_{A_n}) - \left\| ((Err - (Err + err)^2) \right\| \tag{17}$$

Definition 5: *There must exist a cost function as ltCost, that must adhere to the minimum distance required between two algorithm during test, in logical space for ltCost(x,y,z), for which the condition ltCost(00,z)* $\in \Delta$ *(Distance*$(x, y, z) > 0$*, always exist.*

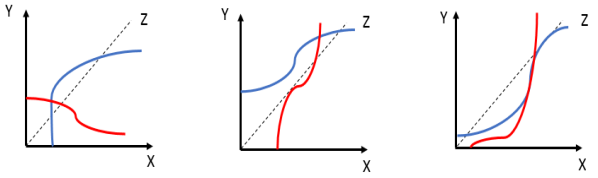


FIGURE 4. Illustration of $\mathbb{B}_{A_n}$ *(Blue)*, $\mathbb{T}_{A_n}$ *(Red)* as it theoretically spread in optimum space of x, y, z dimensions.

During recognition of hidden patterns or points in datasets, the loss or cost function (C):

$$C\left(f\left(i : input\right), o : output\right)) = \frac{1}{2} \left| f\left(i\right) - o \right|, \quad i_k \in \mathbb{I},\ o_k \in \mathbb{O} \tag{18}$$

LT is built on three constructs: *i)* to monitor and store the ratio $\mathbb{R}_{eABT}$, *ii)* to update the values of *x,y, and z* components of each algorithm classifier during training, and *iii)* to score the algorithm $A_n | \{0:1\}, n \in (N+1)$, using Blending Function $\mathbb{B}_{A_n}(0:1)$, and Tuning Function $\mathbb{T}_{A_n}(0:1)$.

$$LT.eABT = \mathbb{R}_{eABT} \times \sum_{n=1}^{N} A_n(f(x, y, z) \left| exp\left(\frac{\mathbb{B}_{A_n}}{\mathbb{B}_{A_n} + \mathbb{T}_{A_n}} \right) \right| \tag{19}$$

Fig. 4 shows three adjacent visual concepts. Our goal is to optimize the Blending and Tuning function with *LT* objects such that, it corresponds to high convergence in z-dimension. **Fig. 4** shows three examples of such cases, that may be expected throughout experimental observation with the real-world data. As shown in the **Fig. 5**, the values are updated based on the function that we built using simple linear regression, so when we fit a line on the given points, we can estimate the linearity of the classifier that is being built by the model as more algorithms are blended (governed by $\mathbb{B}_{A_n}(0:1)$) and then tuned (governed by $\mathbb{T}_{A_n}(0:1)$). We must incorporate the squared error as $[(mx_k + a) - y_k]^2$, which translates to difference between true value and predicted value, Thus:

$$T(m, a) = k_{x,y,z} \cdot \sum_{k=1}^{K} [(mx_k + a) - y_k]^2 \tag{20}$$

Fig. 5 shows the internal mechanics of the *eABT LT* working model. It is internally based on binary classification technique. As the logical table grows with the quantized output as explained above, it decides which algorithm is a good fit in the blended model. As we can observe, that *LT* governs the process at the lowest level of the model being proposed. It creates the entry for each dimension *(X, Y, and Z)* as shown. As a threshold, if the *LT* value is less than 0.5, it is assigned binary '0', and if it is > 0.5, it is assigned binary '1'. Based on this, the binary truth table is built, and is used in the algorithm 1.

Algorithm 1 Algorithm 1 $LT - eABT$ LT Governance

Goals: It governs *LT* structure in the memory to keep track of fitness of the model, for algorithm blend.
Input: $A.P = \{A_1, A_2, A_3, \ldots\ldots, A_n\}$ /* *the algorithms pool for improving generalization* */
Output: $NODES_{i\in N}$, $eABT^{\boxplus}$
Initiate: Create data libraries object as *ObjDS, ObjLT*
1: Set: $x, y, z \leftarrow ObjDS.RandomValues(0)$
2: ***While*** $(0.2 < err \in Err < 0.8)$ ***Do***
3: Compute: error constant using equation (1)
4: Set: $\mu \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x, y, z)_i$
5: Compute: $\mathbb{R}_{eABT}$ using equation (3)
6: ***For*** (each ObjLT.Evaluate(1) in *z*) ***Do***
7: Set: z ← 0
8: Compute: $T\ (m, a)$ using equation (20)
9: Set: $\gamma \leftarrow \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon}\right)$
10: Update: Probability of Hypothesis:
11: $H_t : I \rightarrow \{-1, +1\}$
12: ***If*** $(P_{t+1}\ (k) < 0.5)$ ***Then***
13: Set: $Err_t \leftarrow \sum_{k:H_t(i_k)\neq o_k} P_t\ (k)$
14: Read: ObjDS.Evaluate(Err_t, γ)
15: $P_{t+1}\ (k) \leftarrow \frac{P_t(k)}{Z_t} \times \begin{cases} e^{-\gamma t} & IfH_t\ (i_k) = o_k \\ e^{\gamma t} & IfH_t\ (i_k) \neq o_k \end{cases}$
16: Compute: ObjLT.Write($P_{t+1}\ (k)$)
17: ***Else***
18: Set: $Err_t \leftarrow ObjLT.Read(H_t, \gamma)$
19: Update: ObjLT.Update(err, Err)
20: ***End If***
21: Set: $A(x, y, z) \leftarrow (\Delta x + 1), (\Delta y + 1), (\Delta z + 1)$
22: Compute: $ERM(3D)$
23: Compute: Add/RemFunc based on eq (5,6)
24: Write: ObjLT.Write(*ERM(3D)*)
25: ***End For***
26: Compute: ObjLT.FitnessScore($A(i)$)
27: Update: ObjLT.Update(err(z), Err(z))
28: ***For*** (each node in $NODES_{i\in N}$) ***Do***
29: ***If*** $(Score(A_i \in A_{(i+1)}) > node(i)))$ ***Then***
30: Update: ObjLT.Zscore(ObjDs, A_z)
31: Read: ObjLT.Read(score(z))
32: ***End If***
33: Set: Next node
34: ***End for***
35: Compute: $\mathbb{B}_{A_n}$ /* *Using Equation (16)* */
36: Compute: $\mathbb{T}_{A_n}$ /* *Using Equation (17)* */
37: Update: ObjLT.$eABT^{\boxplus}(\mathbb{B}_{A_n}, \mathbb{T}_{A_n})$
38: ***End While***
39: Return: $eABT^{\boxplus}$

Based on illustration in **Fig. 5**, we can build our Blending and Tuning Functions for the *LT* using in-parallel binary weight distribution for each algorithm.

Next, we build our empirical risk minimization(*ERM*) function based on error-probability function, so we can then

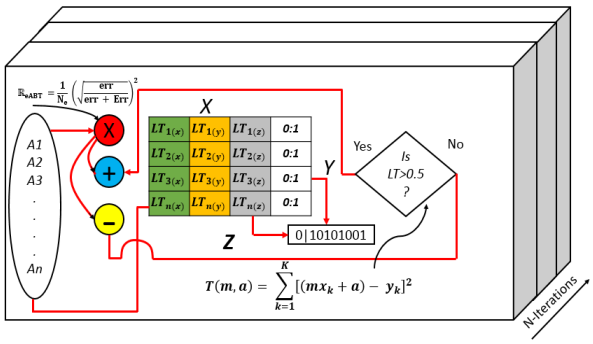


FIGURE 5. Illustration of *eABT* Logical Table Internals.

TABLE 1. Tuning and blending function typical observation.

Functions	Theoretical	Real/Experimental
$\mathbb{B}_{A_n}$	0.86	0.79
$\mathbb{T}_{A_n}$	0.93	0.87
$\mathbb{B}_{A_n} \cup \mathbb{T}_{A_n}$	+0.73	+0.78

TABLE 2. Observance of error functions in typical ratios.

Random	err (local minima)	Err (global minima)	$eABT^{\boxplus}$ Test Observance
x (0.19,0.27,0.38)	0.0013	0.0004	0.39
y (0.27,0.39,0.64)	0.0008	0.0032	0.51
z (0.49,0.65,0.69)	0.0082	0.0193	0.73

optimize the fitness space (*3D*) using *LT* to logically convert (i.e., move overfitting and underfitting to optimum fitting space) the invalid fitness score to which algorithm resists to learn. Error scores are used to measure the degree of success for an element (such as algorithm) to participate in group for the blend, especially in the interest of optimum fitting's. Thus, we define the following rules:

Rule 3

$$Pr(err) \sim x \rightarrow (y(i) \neq z(i))$$

Rule 4

$$Pr(err\ in\ z(i) \sim z \rightarrow (x(i+1) \neq (y+1)$$

On the most inner layers of the learning, the errors can be considered in two types: training errors and testing error, thus, to score the *ERM*, we can assume

$$ERM(err(train, test)) = \begin{cases} -erm, & err(z) < 0 \\ erm, & err(z) \geq 0 \end{cases} \quad (21)$$

By definition:

$$ERM(3D) = \frac{|\{i \in [n] : 3D(d_i) \neq (d+1)_i\}|}{n} \quad (22)$$

Where [n] = {1,2,3,4.....,N}

Inductive bias and hypothesis (h) class are used to rectify the problem of overfitting in *ERM* [35]. Inductive bias is

considered a set of restrictions where we bias the hypothesis class towards a predictor that will not overfit and thus, we can minimize its effect.

$$ERM(h) \in \underset{h \in H}{argmin}\, ERM(3D) \tag{23}$$

C. ALGORITHM DETAILS AND DEFINITION

In this section, we provide complete definition of *LT* Algorithm for *eABT* module. This algorithm uses two important libraries:

i) ObjDS, for general dataset sources in raw or formatted shape including what Python or R packaged offer, and

ii) ObjLT is an object of the *eMLEE API*, for *LT* module to access the function written for its working. The following definition is written in standard format for ease of implementation using standard languages and packages.

Algorithm pool (A.P) represents the *3D* array in the memory that stores the pointers for each dimension of *x,y, and z*. Based on the scoring mechanism explained earlier, it holds the computed values for each algorithm (*SL* algorithms) as they are brought into, for testing (Algorithm' ***eABT*** internal layers). Along with other functions, in conjunctions with Algorithms as defined next, finally the optimum *SL* algorithms are identified and weighted accordingly for classifier blending.

The main ***while*** loop at **step 2** makes sure that rule 2 is obeyed. **Steps 3-5** compute error functions and Ratio as constructed in the math model. The first ***For*** loop at **step 6** ensures the optimum fitness is regulated in the z-dimension. **Steps 7-12** build the probability distribution hypothesis so the cost function is decentralized for improved labelling of each element in each row as *LT* object receives it. ***If*** block at **step 13** checks and maintains the probability of the fitness to be greater than 50 % for more training to be continued and then we update the *LT* objects. **Step 26** sets the changes in each dimension for the algorithm element being incorporated and then updates the global object. **In steps 26-28**, we compute ERM function and use equation 5&6 to utilize adder and remover function. In **steps 31 to 39**, we update the *LT* objects for all the *ML* algorithms incorporated (added or removed) based on the desired fitness and error ranges as per rules defined in the module. **Steps 40-45** finally compute the blending and tuner functions as we constructed in the mathematical model and return the quantized data to the calling function of the algorithm object.

Fig. 6 shows that it is based on binary weighted classification scheme to identify the algorithm for blending and then assign a binary weight accordingly in *LT* logical blocks. The diamond shape shows the err distribution that is observed and recorded by *LT* module as new algorithm is added or existing is removed.

The illustrations shown in **Fig. 7,8,9** are the result of 20-experimental run for over 3000 data samples in our lab environment for evaluating *LT* module in *eMLEE* infrastructure. The simulation uses four colors. Blue, to indicate extreme overfitting in each dimension. Green and yellow

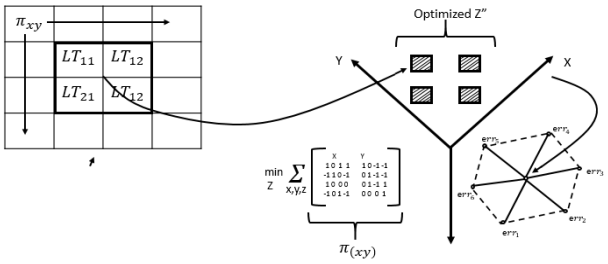


FIGURE 6. This illustration shows the concept of ***LT*** modular elements in ***3-D*** space as discussed earlier.

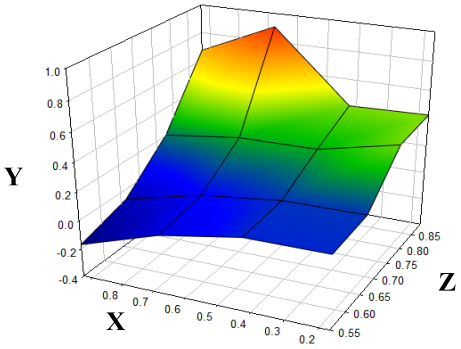


FIGURE 7. It shows the ***LT*** optimum fitness ability in each dimension. We noticed error at the negative value.

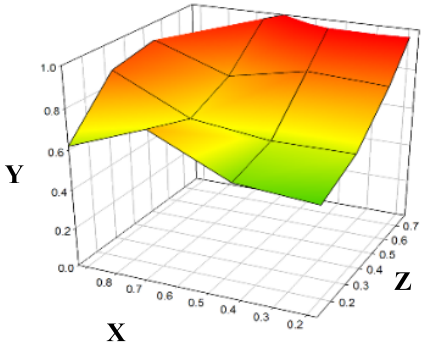


FIGURE 8. This shows the ideal behavior of the ***LT*** optimum fitness function. As we see, the blue section is virtually absent. And it further elaborates that z-dimension has the maximum convergence of the function, as ideally desired.

show that classifier was not able to distinguish between underfitting and overfitting, and orange color shows the optimum fitting.

IV. LT THEORY OF eFES MODULE

Very similar to constructs we built for *eABT LT* unit, this logical table also operates in the memory and is dynamically updated. It keeps tracks of the features $F(x, y, z) = \{F_1, F_2, \ldots\ldots, F_n, \}$ as *the ML* process evolves to accomplish the final optimum fitting after it has tried all the features from the pool. Similarly, it also stores data based on three dimensions, where '*x*' = *under-fitness,* '*y*' = *over-fitness and* '*z*' = *optimum-fitness*. Features in the given datasets are of

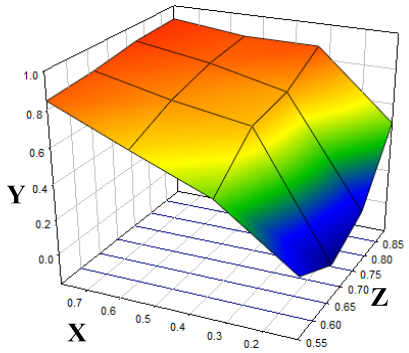


FIGURE 9. This is the real (experimental) behavior of Figure 8.

several types. They are also known as 'attributes' or 'variables'. Type includes: *i*) numeric, such as continuous values such as time, speed, height and weight or discrete such as age, counts, and ii) categorical such as Gender, Color, Race, and Ranks. Some of the categories of features are linguistic, structural, and contextual.

In this section, we first discuss each definition in plain English and then we provide the details of building the *LT* module mathematical constructs and provide algorithm definition.

A. DEFINITIONS IN PLAIN ENGLISH

Definition 6 covers the theory for Adder and Remover function like **Def. 1** but for feature engineering as the second layer of *LT* unit.

Definition 7 covers the theory of function that quantifies the score of each feature as it is introduced in the dataset so the correlation can be improved and decentralized.

Definition 8 as the last theoretical foundation builds the *Irrelevant and Redundant* functions so the scoring of each feature can be done row-wise for each test of the classifier learning. This way, the model learns to determine the optimized predictive value of each feature, as a part of feature engineering and optimization that *LT* object governs and handles. This way, the right numbers and type of features set is created.

B. MATHEMATICAL CONSTRUCTION OF THE UNIT INTERNALS

Definition 6: *Let there be two functions, Feature Adder as* $+\mathbb{F}$*, and Feature Remover as* $-\mathbb{F}$*, based on linearity of the classifier for each feature under test for which the RoOpF is valid (as described in Definition 3), and a feature is not repeated in the group.*

eFES LT module builds very important functions at initial layers for adding a good fit feature and removing a bad fit feature from the set of features available to it, especially when algorithm blend is being engineered. Clearly, as we discussed, not all features will have optimum predictive value and thus identifying them will count towards optimization. The feature adder function is built as:

$$+\mathbb{F}(x,y,z) = (F_n \cup F_{n+1}) \sum_{i=1}^{Z} (lt.score(i)) + \sum_{j,k=1}^{x,y} (lt.score(j,k) \quad (24)$$

The feature remover function is built as:

$$-\mathbb{F}(x,y,z) = (F_n \cap F_{n+1}) \sum_{j,k=1}^{x,y} (lt.score(j,k)) - \sum_{i=1}^{z} (lt.score(i)) \quad (25)$$

Very similar to k-means clustering [38] concept, that is highly used in unsupervised learning, *LT* implements feature weights mechanism(*FWM*) so it can report a feature with high relevancy score and non-redundant in a quantized form. Thus, we define:

$$FWM\ (X,Y,Z) = \sum_{x=1}^{X}\sum_{y=1}^{Y}\sum_{z=1}^{Z}(u_x w_x . u_y w_y . u_z w_z)(\Delta\ (x,y,z)) \quad (26)$$

$$\Delta\ (x,y,z) = \begin{cases} \prod_{l=1}^{L}(u_{lx} w_{lx}), & if\ z \neq 0,\ AND\ z > (0.5, y) \\ u_i \in \{0,1\}, & -1 \leq i \leq L \\ \prod_{l=1}^{L}(u_{ly} w_{ly}), & if\ z \neq 0,\ AND\ z > (0.5, x) \end{cases} \quad (27)$$

Definition 7: *Let there be a Feature Scoring Function as FScore in LT module for which the correlation between each feature as accepted is minimum. Let Cor(x,y,z) be a function to compute the score for the feature sets as grouped in the LT object.*

FScore(x,y,z) and *Cor(x,y,z)* are functions on the second layer that ensure each entry is recorded in the *LT* object as the process continues. We build,

$$FScore\ (x,y,z) = \int_{D_i} (F_i|X,Y,Z) p_i dV_{x,y,z} \quad (28)$$

$$Cor(x,y,z) = \begin{cases} H\ (f_i) - H(f_i|f_{i+1}) \\ H(f_{i+1} - H(f_{i+1}|f_i) \\ H\ (f_i) + H\ (f_{i+1}) - H(f_i, f_{i+1}) \end{cases} \quad (29)$$

PROCEDURE 2
Import Features: a finite number of features
Set F in n > 0, Integer T >0
Initialization: Define the categorical or numerical values, and set $F^{(n)}$ = Constant value
For F = $\{F_1, F_2, F_3, ..., F_n, \}$
  Select F(n) based on random function and define the distribution in space, $D[f(T)|F^{(n)}] \in \partial\ F(F^{(n)})$
  Update each $f \in F^{(n)}$, for which $f_n \geq F\{0.85, 0{:}1\}$ is valid
Return f_x, f_y, f_z

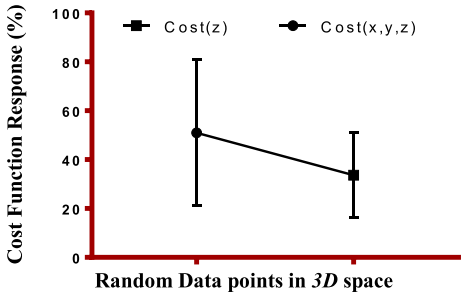


FIGURE 10. This test shows the variance of the LT module for the cost function for all three co-ordinates and then z (optimum-fitness). This is the ideal behavior.

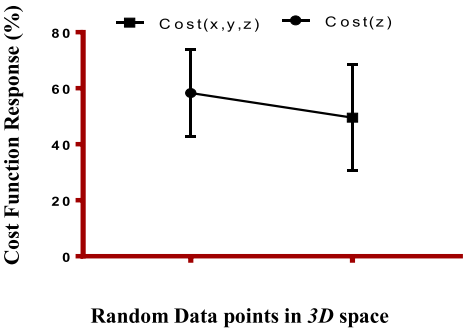


FIGURE 11. This test shows the real (experimental) behavior.

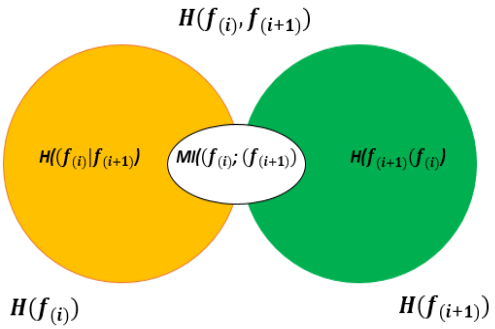


FIGURE 12. Demonstration to illustrate the entropy-based feature distribution in space based on binary system.

Fig. 12 shows two colors (yellow and green) for same function related to each feature (i) and ($i+1$). Thus, the entropy function is calculated in the inner layer of LT object as the features are added or removed based on matching scoring function.

Definition 8: *Let lt.IrrF and lt.RedF be two functions to store the irrelevancy and redundancy score of each feature for a given data set in LT object and then correlates it for each test in blend of algorithms using lt.BlendAlgo Function, such that each feature obeys the condition* $0.3 > lt.BlendAlgo\,(lt.IrrF, lt.RedF) > 0.7$*}.*

To construct *Irr.F* and *Red.F*, we implement Markov Blanket method in which we apply sequential filters to remove the feature one by one for higher *Red.F and Irr.F*. We alter the values between $\{-1$ to $+1)$ for theoretical consideration.

TABLE 3. Observance of error functions in typical ratios.

Random	err (local minima)	Err (global minima)	eFES⊞ Test Observance
x (0.11,0.21,0.32)	0.0023	0.0003	0.28
y (0.27,0.39,0.64)	0.0018	0.0044	0.61
z (0.49,0.65,0.69)	0.0087	0.0173	0.83

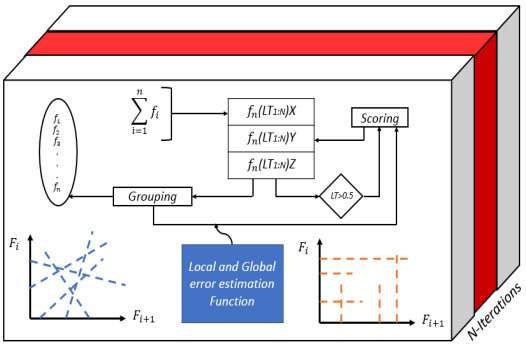


FIGURE 13. Illustrates of the N-experimental iteration of the conceptual flow.

It must be noted, that the values between {0 to 1} are realistic and mathematically possible. We build a mutual information (*MI*) function [31] so we can quantify the relevance of a feature upon other in the random set. This information is used to build the construct for *Irr.F*, as the classifier learns, it will mature the *Irr.F* learning module as defined in the algorithm *1*.

$$MI\,(Irr.F(x, y, z)|f_i, f_{i+1}) = \sum_{a=1}^{N}\sum_{b=1}^{N} p(f_i\,(a)\,,\, f_{i+1}\,(b)\,.log\left(\frac{f_i\,(a)\,,\, f_{i+1}\,(b)}{p(f_i\,(a)\,.f_{i+1}\,(b)}\right) \quad (30)$$

$$Irr.F = \sum_{i,j}^{K}\begin{Bmatrix} f_{ii}\, f_{ij} \\ f_{ji}\, f_{ji}\end{Bmatrix} = \begin{cases} MI\,(f_i; Irr.F) > 0.5 & Strong\ Relevant\ Feature \\ MI\,(f_i; Irr.F) < 0.5 & Weak\ Relevant\ Feature \\ MI\,(f_i; Irr.F) = 0.5 & Neutral\ Relevant\ Feature \end{cases} \quad (31)$$

We use the (31) to develop the relation of '*Irr.F*' and *MI* to show the irrelevancy factor and redundant factor based on binary correlation and conflict. Redundancy is another important quantity to compute for feature correlation, especially in classification problems. We use Markov Blanket [31], [39] to make the following assumptions, $ss \subseteq \neg f_i$ is Markov Blanket, if

$$p(\{F\{f_i, ss\}|\{f_i, (x, y, z)\}) = p(\{F\{f_i, ss\}|\{f_i\}) \quad (32)$$

Fig. 13 illustrates the N-experimental iteration of the conceptual flow shown. As we can observe, that *LT* governs the process at the lowest level. It creates the entry for each dimension *(X, Y, and Z)* .As a threshold, if the *LT* value is less than 0.5, it is assigned binary '0', and if it is > 0.5, it assigns '1'.

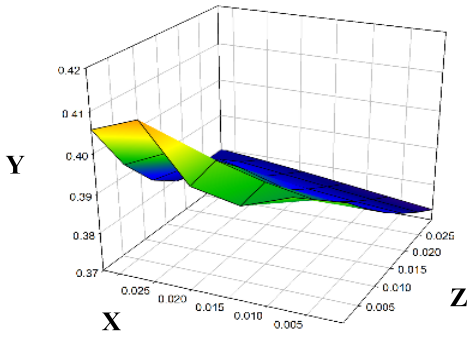


FIGURE 14. This illustrates the ideal outcome of eFES LT fitness function in 3D space. Notice the z-axis has the least blue color.

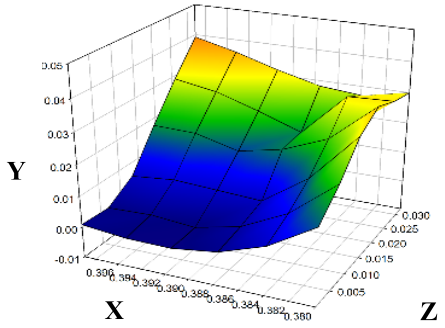


FIGURE 15. This illustrates the real(experimental) analysis of the test, we performed on validation *eFES* module.

This shows the mechanics of the logical design of the algorithm being proposed. It shows that it may take N number of iterations to tune the table function. As discussed earlier, *LT* keeps track of the feature engineering for optimum fitting and outlier detection for a model being trained. Threshold is set to 50 % for *LT* function return value. This shows that as features are added, the *LT* stays above 0.5, or features may need to be removed. The two-dimensional figures in Fig. 13(with blue and orange lines) demonstrate the underfitting and overfitting as the model encounters and reports back to the *LT* object.

Finally, we build our cost and matrix function as follows:

$$Cost\ (LT, Elt) = \frac{1}{T} \sum_{x=1}^{X(t\in T)} \sum_{y=1}^{YX(t\in T)} \times \sum_{z=1}^{ZX(t\in T)} (Elt)_{x,y,z} \times M_x, M_y, M_z, \quad (33)$$

$$M\ (x, y, z) = M\ (x) \times M\ (y) - M(z) \in \min_{z\to max} (M(x, y))$$

$$M\ (x) = \begin{Bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nn} \end{Bmatrix},$$

$$M\ (y) = \begin{Bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nn} \end{Bmatrix},$$

$$M\ (z) = \begin{Bmatrix} z_{11} & \cdots & z_{1n} \\ \vdots & \ddots & \vdots \\ z_{n1} & \cdots & z_{nn} \end{Bmatrix} \quad (34)$$

Equation (34) supports the functionality of **Fig. 7**. Next, we define *LT* algorithm for *eFES* module. The information related to accessing common libraries have been stated in the opening remarks of algorithm *LT-eABT* already in the earlier section.

C. ALGORITHM DETAILS AND DEFINITION

In this section, we provide detail of *eFES LT* algorithm based on the mathematical model and associated libraries.

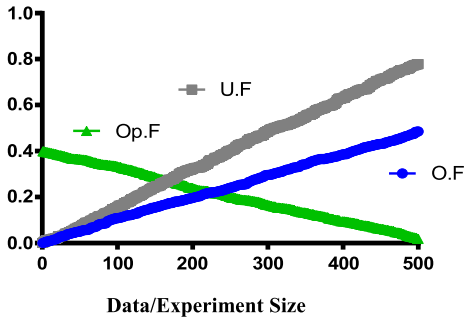


FIGURE 16. This test shows the three metrics (Underfitting *(UF)*, Overfitting (UF), and optimum-fitting (*OpF*) for 500 experimental-run. *LT* module successfully identified and process the features that contribute to each metric accordingly.

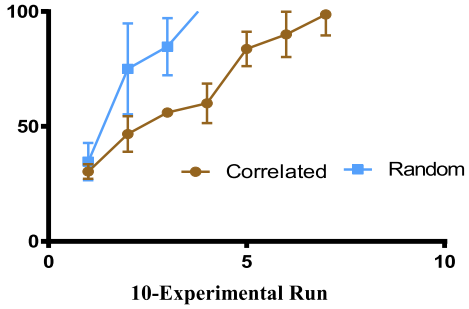


FIGURE 17. This test shows the matching function built in *eFES* algorithm for random vs correlated data points. As we observed, it shows the promising behavior (expected) when it is highly correlated than just the random test.

Step 1 initializes the optimum fitness factor. **Step 2** begins the ***while*** loop to check for Ratio that is governed by local and global errors correlation, so the function remains in-bounds of over-learning and under-learning logical *3D* space. **Steps 3-4** compute the global Error and hypothesis function for even probability distribution as discussed in the mathematical model. **Step 5** starts the ***For*** loop to evaluate each feature and quantifies *x,y, and z* as it spreads in space using *3D* logical elements, so the row can be updated accordingly. **Steps 20-23** compute the Ratio function so the local error can be regulated and then update the *LT* object in the library call. Then it resets each co-ordinate for next run in the loop. **Steps 24-28**

build the references for computing Feature Adder and Feature Remover function for feature grouping function using *LT* parallel evaluation technique as explained earlier in the model build of Section 4. In **Steps 29-33**, the ***If*** block checks for each algorithm entry so the Ratio can be re-calculated and this way, the *No Free Lunch Theorem* problem is also addressed. Finally steps **34-38** compute the main *eFES* function after updating the central probability function, so the bias can be minimized for each feature before adding to the group. Then in last step, the function reference is returned to the calling pointer of the algorithm.

```
PROCEDURE 3
For (each node in tree) Do
    Execute: Add Function for new element
    Update each node for maximum points
     If (t <= 0.5~in absolute T) Then
          Read next node
          Move to next node in tree and add
the previous node to the LT Object
     End If
     Set: x,y and z values from node to LT object
Update LT
     Execute the Algorithm~LT eABT
    While (there are more algorithms to test)
Do
           Compute the CF for each algorithm
           Run the optimization test as shown
in~Figure~18
           Update the results
           Find new~(t)
           t++ (increment)
          Update x,y, and z
    End While
   Execute Algorithm~LT eFES
   For (each feature in the set) Do
           Execute the Adder and Remover function
           Find new~(t)
           t-- (decrement)
          Update the CF Function
   End For
End For
```

With the results shown for e*ABT* and *eFES LT* modules in the lowest level of e*MLEE* infrastructure, we show our low-level framework for the *LT* mechanism in **Fig. 18**. We first develop the cost function for each node as shown in the illustration for each dimension in *3-D* logical space. As shown that z can vary from 1 to n values depending upon how many iterations will it take to achieve the optimum fitting of the model with *LT* object. $t \in T$ indicates all the values of tuning function in the unit terms during training of *LT* object. These values can vary both in negative and positive because *LT* object keeps track of both underfitting and overfitting of the model in its logical layers and rows.

Algorithm 2 *LT*–eFES Logical Table Governance

Goals: It governs *LT* structure in the memory to keep track of fitness of the model.

Input: $A.P = \{A_1, A_2, A_3, \ldots\ldots, A_n\}$ /* *the algorithms pool for improving generalization* */ Raw Feature Set F(x, y, z) = $\{F_1 \in F_n\}$

Output: $NODES_{i\in N}$, $eFES^{\boxplus}$

1: Set: $\{Op.F \leftarrow 0, F\,(x.y, z) \leftarrow (0, 0, 0)\}$
2: ***While*** $((\mathbb{R}_{eABT}) < (r(e + E))$ ***Do***
3: Compute: $Err_t \leftarrow \sum_{k:H_t(i_k)\neq o_k} P_t\,(k)$
4: Compute: Hypothesis: $H_t : I \rightarrow \{-1, +1\}$
5: ***For***(# of F in Set) ***Do***
6: Set: $\gamma \leftarrow \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon}\right)$
7: Compute: x,y, and z for ObjLT.Random
8: Update: ObjLT.Update(x,y,z, γ)
9: ***If***($\gamma < (\gamma - 1)$) ***Then***
10: Set: $\gamma \leftarrow (\gamma + 1)$
11: Read: ObjLT.Read (x,y,z)
12: Compute: : $P_{t+1}\,(k) \leftarrow \frac{P_t(k)}{Z_t}$
13: ***Else***
14: Set: $\gamma \leftarrow (\gamma - 1)$
15: Update: ObjLT.Update(x,y,z, γ)
16: ***End If***
17: Compute: $\mathbb{R}_{eFES} \leftarrow \frac{1}{N_e}\left(\sqrt{\frac{e}{e+E}}\right)^2$
18: Update: ObjDS.Update($\mathbb{R}_{eFES}$, *ObjLT*)
19: Set: $x \leftarrow (x + 1)\,, y \leftarrow (y + 1)\,, z \leftarrow (z - 1)$
20: ***End For***
21: Write: ObjDS.Write(x,y,z, *ObjLT*)
22: Compute: $+\mathbb{F}$/* Using equation (24) */
23: Compute: $-\mathbb{F}$/* Using equation (25) */
24: Update: Scores for each algorithm, and
25: creates Nodes $NODES_{i\in N}$
26: ***For*** (each node in $NODES_{i\in N}$) ***Do***
27: ***If***($SCORE(A_i \in A_{(i+1)}) > node(i)$) ***Then***
28: Add: entry to LT
29: Re-compute: $\mathbb{R}_{eABT}$
30: Update: LT
31: ***End If***
32: Finally Update: $P_{t+1}\,(k)$
33: Compute: eFES(x, y, z) $\leftarrow$
34: $ObjLT.Optimize(\mathbb{R}_{eABT}, \mathbb{B}_{A_n}, \mathbb{T}_{A_n})$
35: ***End While***
36: Return: $NODES_{i\in N}$, eFES

V. RESULTS AND ANALYSIS

A. EXPERIMENTAL SETUP

The various datasets were used to improve the generalization of the model. The details of datasets are listed in Appendix. Datasets were divided in three sections, as a standard practice, *i) Train, ii) Test, and iii) Validation*. However, we also uniquely split the data (as defined in the algorithm) governed by the real-time metrics using *LT* object. In this process, the random slices of data were created and then they were flipped to elevate the predictive errors temporarily. This way, *LT* objects learn on maximum possible errors and then tune itself (i.e., algorithm) to improve the slice in the next run and so on. This is also supported in *LT* mathematical model (i.e., Definitions). This also chains the ideas of enhanced validation and parallelism as we stated in the Introduction section.

TABLE 4. *LT* metrics for *eFES* and *eABT*.

Experiments	$Accuracy_{LTeABT}$	CF_{LTeABT}	err_{LTeABT}	Err_{LTeABT}	err_{LTeFES}	Err_{LTeFES}	$Accuracy_{LTeFES}$	CF_{LTeFES}
1	0.102193	0.123183	0.897193	0.874731	0.737103	0.855492	0.225609	0.079028
2	0.114734	0.134734	0.737134	0.861543	0.626132	0.806411	0.275681	0.151088
3	0.143159	0.153159	0.743884	0.814910	0.603924	0.757090	0.378429	0.237126
4	0.181294	0.171794	0.527183	0.724807	0.577191	0.705442	0.481349	0.339088
5	0.306104	0.306804	0.332956	0.554672	0.528241	0.637021	0.474050	0.397934
6	0.340422	0.643059	0.371304	0.423340	0.511601	0.562172	0.547503	0.440004
7	0.537183	0.684847	0.316205	0.407649	0.446501	0.474173	0.637512	0.486617
8	0.757184	0.741634	0.304050	0.326716	0.334050	0.316478	0.662501	0.537134
9	0.839894	0.882090	0.231674	0.264370	0.321791	0.297144	0.696033	0.683194
10	0.941431	0.972132	0.218934	0.238912	0.201290	0.261134	0.732907	0.781943

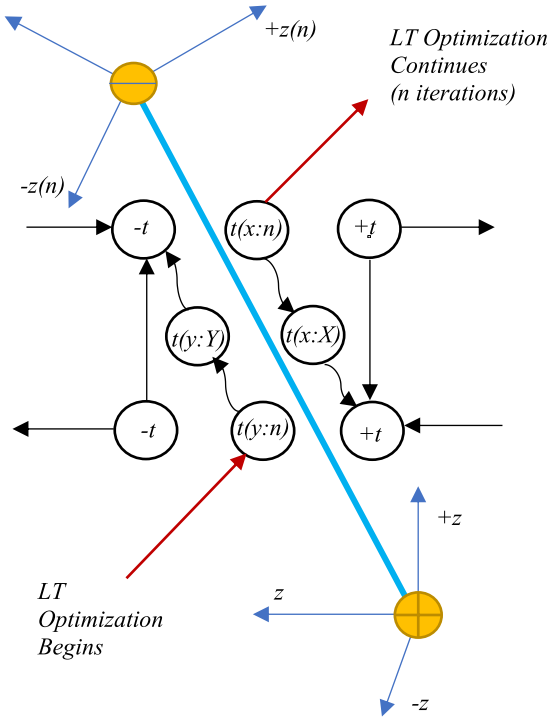


FIGURE 18. LT framework at low-level.

TABLE 5. X observations.

Dim	Theoretical	Experimental
CF	0.23	0.44
Err	0.90	0.81
Acc	0.83	+0.76

TABLE 6. Y observations.

Dim	Theoretical	Experimental
CF	0.17	0.53
Err	0.89	0.82
Acc	0.77	0.61

TABLE 7. Z observations.

Dim	Theoretical	Experimental
CF	0.98	0.87
Err	0.25	0.31
Acc	0.95	0.87

Validation datasets split tested how well the model was learning (i.e., learned skill) and testing datasets substantiated the bias of the model, as it learned. In main model of *eMLEE* several *ML* algorithms such as Support Vector Machines, Decision trees, Logistic Regression, Multiple Linear Regression, Bayes networks, etc., were used to test the model via the blending mechanism (i.e., *eMLEE* internals). However, the *eMLEE* underlying proposed algorithms, outside of the scope of this article, allow researchers to incorporate any supervised learning algorithm of their choice, to overcome the challenge of "No Free Lunch theory" as we discussed in the introduction section. That is the beauty and novelty of this model based on *LT*. We have used existing libraries of Python and R scientific packages on the exact datasets that we setup for our experiments so we could draw comparison charts and record tabular data. However, the comparison details of *eMLEE* are also outside of the scope of this paper.

Appendix lists the details of the libraries we have used to implement the mathematical model and algorithm as proposed in the article. However, end users are free to use the language of their choice to build it.

B. RESULTS PRESENTATION AND DISCUSSION

Table 4 lists the average outcome of 10-experimental process that was adopted as a part of the experimental validation of the constructs. Several experiments were performed on a diverse set of data to improve the generalization of the model

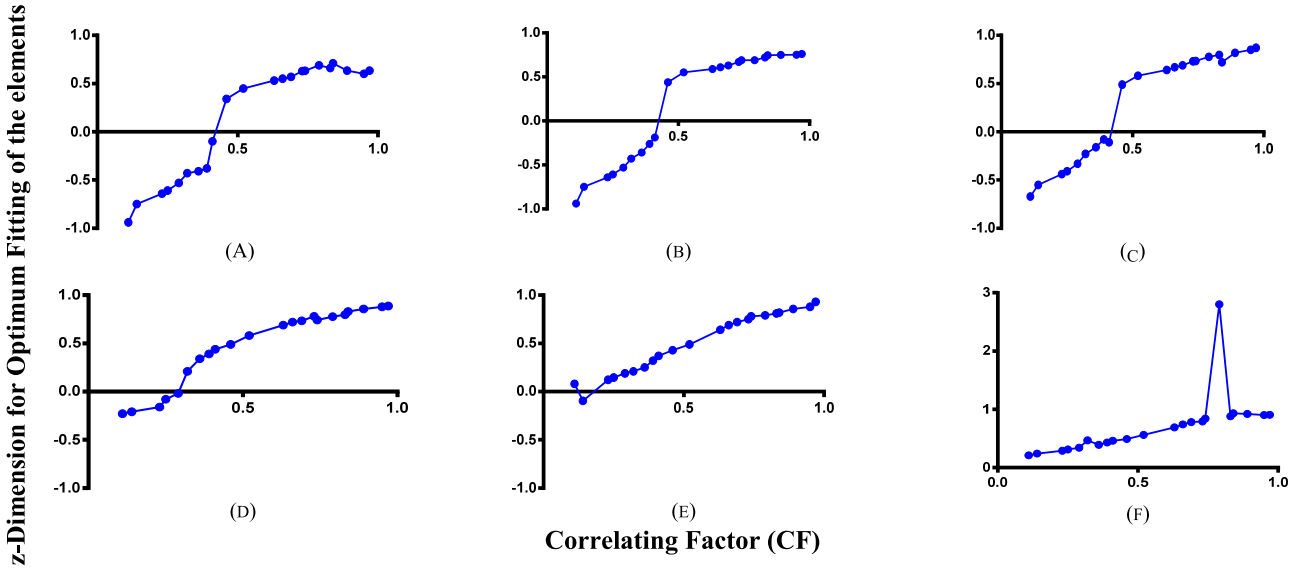


FIGURE 19. Experiments (A) to (C) shows the poor optimization for z with CF without using LT objects. However, we observe in (D) to (F) that model is learning to optimize itself for optimum CF for z-dimension using LT objects. The spike noticed in (F) is suspected to be error and will need future investigation. (A) through (C) were conducted using standard procedure where LT objects were not used.

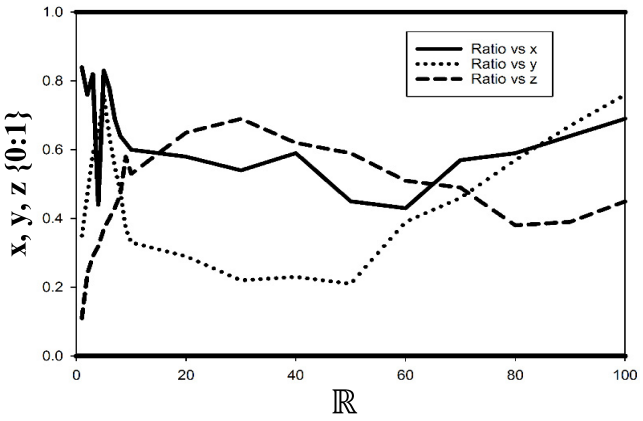


FIGURE 20. The outcome shown here exhibits the model behavior for ratio $\mathbb{R}$ in terms of each dimension of *x,y and z*. It is observed that regression is relatively higher for each dimension and is considered pre-mature learning of the model.

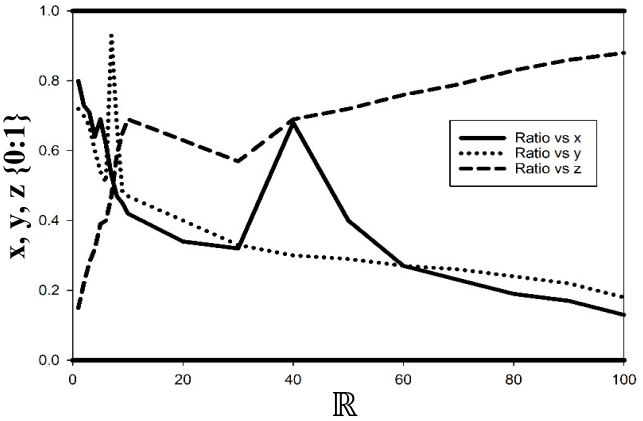


FIGURE 21. The observation here shows improvement and considered mature classifier learning of the *LT* process. As we noticed that z-dimension (as hoped in the design of the model) is depreciating with respect of the error ratio $\mathbb{R}$.

and then 10 experiments were sampled to show the learning process of the model. As we observe (as expected) that the metrics are improved as we conducted more experiments. The experiment number indicates the sampling range of the total statistics. ***Err*** corresponds to overall error for a single experiment once the learner process stabilizes and ***err*** corresponds to local err **or** per experiment during a single training period. It is noted that Correlating Factor (*CF*) improves as the process learns with more experiments and data.

Fig 20 to 25 provide the experimental outcomes of the vital functions such as Ratio (R), Adder, Remover, and various metrics to further validate the stability and optimum fitness of the proposed model. Some cases as shown also compare the ideal and real behavior of the model so it can be seen very clearly how close the model stays with the real-world test, specially when it is exhausted by the diverse set of data.

Table 5, 6 and 7 show the typical results of the *3-D* measures. The snapshot of the results shows the difference of model behavior for theoretical (what we thought it will be) and experimental (what it turned out to be). These results are reported to support the model's stability in real the world data in line with testing data.

Fig 19 shows six experimental studies of the proposed model. As discussed earlier, the model aims to achieve the fitness space optimization in z-dimension while it swings via its logical learning process between two extremes for underfit

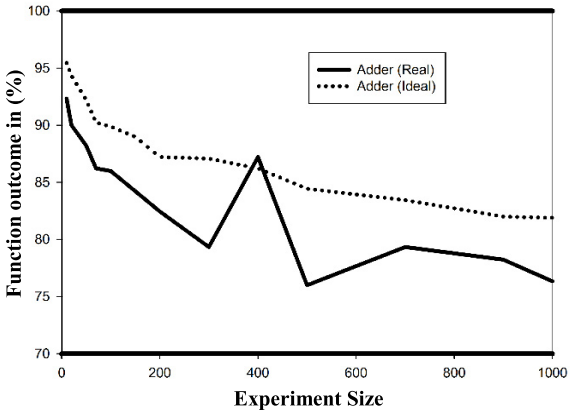


FIGURE 22. Adder real and ideal function is shown here. We observe that when experiment size is at lower end, it shows higher % and as the experiment size increases, the function outcomes drop and this behavior is in line with model internals as expected. The triangular spike is a training error.

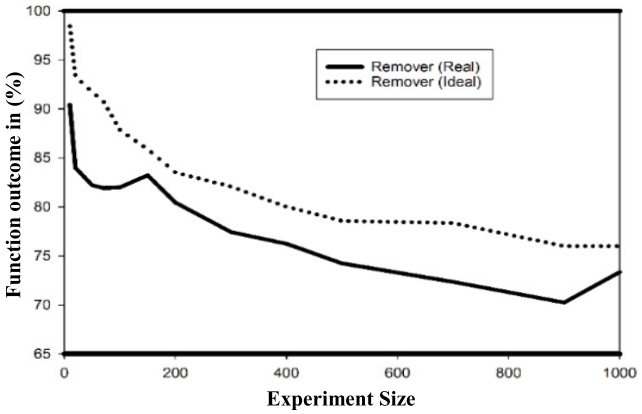


FIGURE 23. Remover real and ideal function is shown here. It shows that model exhibited theoretical stability of its internals.

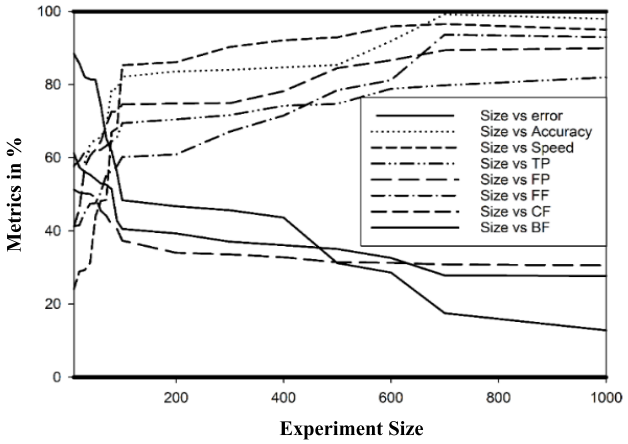


FIGURE 24. This test shows the matching function built in eFES algorithm for random vs correlated data points. As we observed, it shows the promising behavior (expected) when it is highly correlated than just the random test without using LT objects.

and overfit (x,y). It must be noted, as we stated earlier that we are providing the sub-set of our diverse set of experiments for this article. We attempted to validate our model so that the

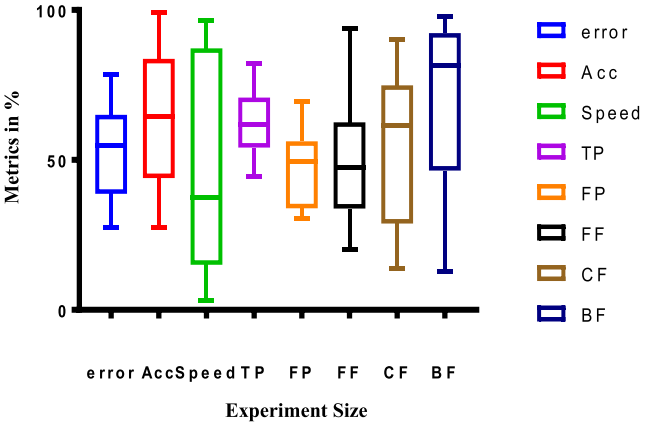


FIGURE 25. This test shows the candle stick analysis commonly used for stocks predictions. True Positive (TP), False Positive (FP), Fitness Factor (FF), Correlating Factor (FF), Bias Factor (BF) shows the move between 0 to 100 % for 20 experimental-run tests. This analysis helps particularly in understanding the direction of the move of the metric when classifier learns. For example, if we can consider the speed of the process, we can see the move in green candle from almost 0 to 98 % throughout the process.

reported simulations and experimental outcomes represent the model behavior, integrity, and stability in the real world.

VI. CLOSING REMARKS

A. CONCLUSION

This article presented vital component as a Logical Table (*LT*) unit and its constructs in the internals of enhanced Machine Learning Engine Engineering (*eMLEE*) Model. *LT* worked at the lowest level of this engine and regulated the entire processing when model was blending and tuning various good fit algorithms. It enhanced the feature selection and optimization for optimum-fitting of the Machine Learning (*ML)* process for predictive analytics. *LT* constructs introduced the novel parallelism for the enhanced algorithm blend classifier learning. *LT* constructs provided a logical way of recording the metrics of the classifier learning for *3D* objects focused on overfit, underfit, and optimum fit for each algorithm' and feature' incorporation during real time training. This approach uniquely supported the enhancement towards improved accuracy, reduced errors, bias and overfitting.

Experimental datasets were split into test, train and validation sets so the model learning and bias can be evaluated in a parallel fashion.

Overfitting, poor generalization, higher errors, low accuracy, and bias became significant when blend was being engineered such as *eMLEE*. Thus, *LT* structure was invented as a part of *eMLEE* development to provide in-parallel regulation and governance of the metrics that needed to be recorded during the learning process. This approach also aided to the solution of addressing 'No Free Lunch Theorem' problem which generally does not ensure that a good fit algorithm is not left untested.

Related study was discussed to highlight the importance of *ML* latest progress in the research and to justify the invention of *LT* for *eMLEE*.

This article was accompanied with simulated results (produced in *Graphpad, SigmaPlot, MS Excel, and Plotly* software's), mathematical equations/constructs, underlying algorithms, and necessary illustrations to elaborate the conceptual details of the proposed constructs and the promising outcomes.

LT model addressed the challenge of trade-off between vital metrics such as complexity, accuracy, speed, etc.*LT* created parallel processes for each element in each run governed by *3D* object co-ordinates (*x, y and z*) and then made observations in the real-time of classifier learning and updated its logical rows in the table. This approach was found to be novel to the best of our survey and knowledge.

LT model presented in this article was also accompanied by eight mathematical definitions, three sub-procedures, and two by-design algorithms. This article also presented several visuals and frameworks to clarify the mechanics of the proposed unit along with promising results in conjunction with tabular data to draw useful observations.

Finally, it was concluded that *LT* worked efficiently as a centralized module/unit of *eMLEE* to improve the metrics coordination and learning process for *ML*. The experimental results were included in this article to support the *LT* progress at this stage, and it showed that metric correlation and observations were found improved as compared to the learner process that was not trained using *LT*.

B. FUTURE WORKS

A progress and further development is in progress for *eMLEE* modules and algorithms. More data is planned to be tested for improving the generalization and stability of the model. Once results are ready with improved outcomes, articles will be prepared and submitted to the journals in line with the latest works of ours and others. We will be developing/testing more algorithms, especially in the domains of unsupervised learning. We are improving/developing a model known as ''Predicting Educational Relevance For an Efficient Classification of Talent (*PERFECT*) algorithm Engine (*PAE*). *PAE* is based on *eMLEE* and incorporates three algorithms known as Noise Removal and Structured Data Detection (*NR-SDD*), Good Fit Student (*GFS*), and Good Fit job Candidate (*GFC*). We have published the preliminary results [40] and are working to apply *LT* model in its latest form to study, explore, and validate further enhancements.

C. COMPARISON MEMO

Because of the novelty of the concept and unit's mechanics introduced and built in this article, we could not find exact relevant model or technique in the literature we surveyed. However, as we stated earlier in **Section I** that *LT* unit worked as a centralized module in *eMLEE*. It controlled and regulated the parallelism of the entire *ML* process, and it showed promising improvements in several experiments and validation process of the classifier learning specially in the ensembling and boosted approach (i.e., *eMLEE*). We have recently published an article [41] about enhanced Feature Engineering and Selection (*eFES*) that consumed *LT*. In that article, we have provided several evaluation tables with relevant experimental quantified data for *eFES* comparison with other existing techniques.

APPENDIX

A. DATA SET

We have utilized the data from the following domains listed below. Some datasets were raw, CSV, and SQL lite format with parameters and field definitions. We transformed all our input data into the SQL Server data warehouse. Some of datasets are found to be ideal for doing healthcare preventive medicine, stock market, epidemic, and crime control prediction.

1. http://www.Kaggle.com - Credit Card Fraud Detection, Iris species, Human Resource Analytics, 2015 Flight Delays and Cancellations, Daily news for Stock Market Prediction, 1.88 Million US Wildfires, SMS Spam Collection Dataset, Twitter User Gender Classification, Brest Cancer Wisconsin Data Set, Retail Data Analytics, US Dept. of Education: College Scoreboard, Death in the United States, US Mass Shootings, Adult Census income, Fatal Police Shootings, Exercise Pattern Prediction, Netflix Prize Data, Pima Indians Diabetes Database, WUZZUF Job Posts, Student Survey, FiveThirtyEight, S&P 500 stock Data, Zika Virus epidemic, Student Alcohol Consumption, Education Statistics, Storm Prediction center.
2. http://snap.standford.edu – Facebook, Twitter, Wiki and bitcoin data set.
3. https://docs.google.com/forms/d/1l57Un32YH6S kltntirUeLVpgfn33BfJuFLcYupg43oE/viewform? edit_requested=true - online questionnaire from students across 12 campuses in the world
4. http://archive.ics.uci.edu/ml/index.php - Iris, Car Evaluation, Heart disease data set, Bank Marketing Data
5. https://aws.amazon.com/datasets/ - Enron Email Data, Japan Census data, 1000 Genomics Project,
6. https://cloud.google.com/bigquery/public-data/ - We are experimenting it using BigQuery in our Sandbox environment and will publish results in the future.
7. https://www.reddit.com/r/bigquery/wiki/das -
8. https://docs.microsoft.com/en-us/azure/sql-database/ sql-database-public-data-sets

B. TOOLS

Due to the years of background in databases and data architecture, we selected the Microsoft SQL Server [42] (Business Intelligence, SQL Server Analysis Services, and Data mining) as our data warehouse. Preliminary work was conducted in Microsoft Azure *ML* tools. We used Microsoft Excel data mining tools [43]. Due to our programing background, we used Microsoft C# (mostly for learning in the beginning) and Python and R language for the primary building of this

model, and algorithms. There are various popular and useful Python data analysis and scientific libraries[1] such as Pandas, Numpy, SciPy[2], Matplotlib, scikit-learn, Statsmodels, ScientificPython, Fuel, SKdata, Fuel, MILK, etc. For R language[3], there are various libraries such as gbm, KlaR, tree, RWeka, ipred, CORELearn, MICE Package, rpart, PARTY, CARET, randomForest. We used some of them as were relevant to our work and we are in the process of learning, experimenting and using them for the future work. We also used GraphPad Prism[4] to produce simulated results.

REFERENCES

[1] The Economist Intelligence Unit, "The deciding factor: Big data & decision making," Capgemini, Paris, France, Tech. Rep., 2012, pp. 1–24. [Online]. Available: https://www.capgemini.com/wp-content/uploads/2017/07/The_Deciding_Factor_Big_Data_Decision_Making.pdf

[2] V. Kotu and B. Deshpande, *Predictive Analytics and Data Mining*. Amsterdam, The Netherlands: Elsevier, 2015.

[3] F. Pianesi, "Searching for personality," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 146–158, Jan. 2013.

[4] A. Vinciarelli and G. Mohammadi, "A survey of personality computing," *IEEE Trans. Affect. Comput.*, vol. 5, no. 3, pp. 273–291, Jul. 2014.

[5] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Comput. Surv.*, vol. 49, no. 2, pp. 1–50, 2016.

[6] K. V. Kanimozhi and M. Venkatesan, "Unstructured data analysis—A survey," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 3, pp. 223–225, 2015.

[7] S. Feldman, J. Hanover, C. Burghard, and D. Schubmehl, "Unlocking the power of unstructured data," IDC Health Insights, Framingham, MA, USA, White Paper #HI235064, Jun. 2012. [Online]. Available: http://public.dhe.ibm.com/software/data/sw-library/ecm-programs/IDC_UnlockingThePower.pdf and http://uhcjsc.com/pdf/Unlocking%20the%20Power%20of%20Unstructured%20Data.pdf

[8] M. Russell, *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2014.

[9] S. Ben-David and S. Shalev-Shwartz, *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2014.

[10] S. J. Wright, S. Nowozin, and S. Sra, "Optimization for machine learning," in *Optimization*, vol. 1. Cambridge, MA, USA: MIT Press, 2011, p. 509.

[11] J. Grus, *Data Science From Scratch*. Sebastopol, CA, USA: O'Reilly Media, Inc. 2015.

[12] S. Vijaylakshmi and J. Priyadarshini, "Big data analysis based on mathematical model: A comprehensive survey," *J. Eng. Appl. Sci.*, vol. 10, no. 5, pp. 2103–2107, 2015.

[13] B. Graham, R. Bond, M. Quinn, and M. Mulvenna, "Using data mining to predict hospital admissions from the emergency department," *IEEE Access*, vol. 6, pp. 10458–10469, 2018.

[14] K. R. Bisaso, G. T. Anguzu, S. A. Karungi, A. Kiragga, and B. Castelnuovo, "A survey of machine learning applications in HIV clinical research and care," *Comput. Biol. Med.*, vol. 91, pp. 366–371, Feb. 2017.

[15] E. M. F. El Houby, "A survey on applying machine learning techniques for management of diseases," *J. Appl. Biomed.*, vol. 16, no. 3, pp. 165–174, 2018.

[16] A. Petrou, M. Kanakis, S. Boes, P. Pergantis, M. Meboldt, and M. S. Daners, "Viscosity prediction in a physiologically controlled ventricular assist device," *IEEE Trans. Biomed. Eng.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8268082, doi: 10.1109/TBME.2018.2797424.

[1]https://wiki.python.org/moin/NumericAndScientific

[2]https://www.scipy.org/

[3]https://cran.r-project.org/

[4]https://www.graphpad.com/scientific-software/prism/

[17] B. Liu, H. Aliakbarian, Z. Ma, G. A. E. Vandenbosch, G. Gielen, and P. Excell, "An efficient method for antenna design optimization based on evolutionary computation and machine learning techniques," *IEEE Trans. Antennas Propag.*, vol. 62, no. 1, pp. 7–18, Jan. 2014.

[18] A. Samat, P. Du, S. Liu, J. Li, and L. Cheng, "E^2LMs: Ensemble extreme learning machines for hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 4, pp. 1060–1069, Apr. 2014.

[19] T. Liu, Y. Yang, G. B. Huang, Y. K. Yeo, and Z. Lin, "Driver distraction detection using semi-supervised machine learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1108–1120, Apr. 2016.

[20] O. F. Reyes-Galaviz, W. Pedrycz, Z. He, and N. J. Pizzi, "A supervised gradient-based learning algorithm for optimized entity resolution," *Data Knowl. Eng.*, vol. 112, pp. 106–129, Aug. 2017.

[21] S. M. Salaken, A. Khosravi, T. Nguyen, and S. Nahavandi, "Extreme learning machine based transfer learning algorithms: A survey," *Neurocomputing*, vol. 267, pp. 516–524, Dec. 2017.

[22] D. Tuia, M. Volpi, L. Copa, M. Kanevski, and J. Munoz-Mari, "A survey of active learning algorithms for supervised remote sensing image classification," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 3, pp. 606–617, Jun. 2011.

[23] S. Garcia, J. Luengo, J. A. Sáez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 734–750, Apr. 2013.

[24] Y. Wang, V. Chattaraman, H. Kim, and G. Deshpande, "Predicting purchase decisions based on spatio-temporal functional MRI features using machine learning," *IEEE Trans. Auton. Mental Develop.*, vol. 7, no. 3, pp. 248–255, Sep. 2015.

[25] N. Tandon, A. S. Varde, and G. de Melo, "Commonsense knowledge in machine intelligence," *ACM SIGMOD Rec.*, vol. 46, no. 4, pp. 49–52, 2018.

[26] B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero, "Using machine learning to optimize parallelism in big data applications," *Future Gener. Comput. Syst.*, vol. 86, pp. 1076–1092, Sep. 2017.

[27] Q. Dai and G. Song, "A novel supervised competitive learning algorithm," *Neurocomputing*, vol. 191, pp. 356–362, May 2016.

[28] Z. Yu *et al.*, "Adaptive ensembling of semi-supervised clustering solutions," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1577–1590, Aug. 2017.

[29] D. Xiao-Jian, L. Yuan, Z. Zhi-Feng, and X. Xin, "Optimization extreme learning machine with ν regularization," *Neurocomputing*, vol. 261, pp. 11–19, Oct. 2017.

[30] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1192–1209, 3rd Quart., 2013.

[31] J. R. Vergara and P. A. Estévez, "A review of feature selection methods based on mutual information," *Neural Comput. Appl.*, vol. 24, no. 1, pp. 175–186, 2015.

[32] Y. Mohsenzadeh, H. Sheikhzadeh, A. M. Reza, N. Bathaee, and M. M. Kalayeh, "The relevance sample-feature machine: A sparse Bayesian learning approach to joint feature-sample selection," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 2241–2254, Dec. 2013.

[33] X. Ma, H. Wang, B. Xue, M. Zhou, B. Ji, and Y. Li, "Depth-based human fall detection via shape features and improved extreme learning machine," *IEEE J. Biomed. Health Inform.*, vol. 18, no. 6, pp. 1915–1922, Nov. 2014.

[34] M. Kubat, *An Introduction to Machine Learning*. Cham, Switzerland: Springer, 2015, doi: 10.1007/978-3-319-20010-1_10.

[35] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning*. New York, NY, USA: Cambridge Univ. Press, 2014.

[36] E. Alpaydın, *Introduction to Machine Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2010.

[37] A. Smola and S. V. N. Vishwanathan, *Introduction to Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[38] G. Forman, "Feature Selection for Text Classification," *Comput. Methods Featur. Sel.*, vol. 16, pp. 257–274, Oct. 2007.

[39] M. Eirinaki and M. Vazirgiannis, "Web site personalization based on link analysis and navigational patterns," *ACM Trans. Internet Technol.*, vol. 7, no. 4, p. 21, 2007.

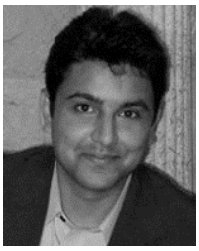
[40] M. F. Uddin and J. Lee, "Proposing stochastic probability-based math model and algorithms utilizing social networking and academic data for good fit students prediction," *Social Netw. Anal. Mining*, vol. 7, no. 1, p. 29, Jul. 2017.

[41] M. F. Uddin, J. Lee, S. Rizvi, and S. Hamada, "Proposing enhanced feature engineering and a selection model for machine learning processes," *Appl. Sci.*, vol. 8, no. 4, p. 646, 2018.

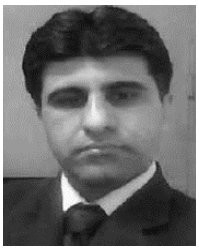
[42] B. Woody, D. Dean, D. Guhathakurta, G. Bansal, M. Conners, and W.-H. Tok, *Data Science With Microsoft SQL Server*. Redmond, WA, USA: Microsoft Press, 2016.

[43] G. Fouché and L. Langit, "Data mining with excel," in *Foundations of SQL Server 2008 R2 Business Intelligence*. Berkeley, CA, USA: Apress, 2011, pp. 301–328.

MUHAMMAD FAHIM UDDIN received the bachelor's and the master's degrees in electrical engineering in 1999 and 2004, respectively. He is currently pursuing the Ph.D. degree with the University of Bridgeport, Bridgeport, CT, USA. He was a .NET Developer and SQL Admin from 2005 to 2007. He has over 15 years of experience in IT including, but not limited to, as a .NET Developer and an SQL SERVER Admin/Developer and Business Intelligence Developer. He has been a DBA/Data Architect in the Information Technology Services, with the City Government, since 2007. He is also an Adjunct Faculty at the University of Bridgeport. He has over 12 years of teaching experience. His research interests includes big data, machine learning, social data and personality prediction, student prediction, predictive modeling, artificial intelligence and artificial mind, parallel processing, cognitive computing and intelligence, and big data in healthcare, preventive, personalized and preventive healthcare through big data (7 Ps), social networking data, data science.



SYED RIZVI received the Ph.D. degree in computer science and engineering from the University of Bridgeport in 2010. He is currently with Penn State University, PA, USA, where he is an Assistant Professor of information science and technologies. He is also an Affiliate Member of the Wireless and Mobile Communications Lab, University of Bridgeport. He has authored and co-authored over 70 technical refereed and non-refereed papers in various conferences, international journal articles, and book chapters in research and pedagogical techniques. His research interests lie at the intersection of computer networking, network security, and modeling and simulation. Recently, he has been working on security issues in rogue access point detection, cognitive radios for wireless communications, and modeling and simulation of nano-networks. In the past, he has done research on the design, analysis, implementation, and comparisons of algorithms in the areas of wireless/multiuser communications, wireless sensor networks, information security, and parallel/distributed systems.



ABDUL RAZAQUE received the Ph.D. degree in computer science and engineering from the University of Bridgeport, USA. He served as an IT Project Director funded by UNESCO and World Bank in rural areas. He is currently an Assistant Professor with the New York Institute of technology. He has authored over 120 international academic publications including journals, conferences, and book chapters. His research interests include the wireless sensor networks, cloud computing security, design and development of mobile learning environments, multimedia applications, and ambient intelligence. He has chaired over 12 highly reputed international conferences and delivered lectures as a Keynote Speaker. He was the Chair for the Strategic and Planning Committee IEEE SAC Region-1, USA, and a Relational officer IEEE SAC R-1 for Europe, Africa, Middle East, and USA. He was the Editor-in-Chief of the *International Journal for Engineering and Technology*, Singapore. In addition, he is an Editor, an Associate Editor, and a Member of Editorial Board for several international journals.