Acceleration of Subspace Learning Machine via Particle Swarm Optimization and Parallel Processing

Hongyu Fu*, Yijing Yang*, Yuhuai Liu*, Joseph Lin*, Ethan Harrison†, Vinod K. Mishra† and C.-C. Jay Kuo *
* University of Southern California, Los Angeles, California, USA
† Army Research Laboratory, Aberdeen Proving Ground, Maryland, USA
‡ Mira Costa Highschool, Manhattan Beach, California, USA

Abstract—Built upon the decision tree (DT) classification and regression idea, the subspace learning machine (SLM) has been recently proposed to offer higher performance in general classification and regression tasks. Its performance improvement is reached at the expense of higher computational complexity. In this work, we investigate two ways to accelerate SLM. First, we adopt the particle swarm optimization (PSO) algorithm to speed up the search of a discriminant dimension that is expressed as a linear combination of current dimensions. The search of optimal weights in the linear combination is computationally heavy. It is accomplished by probabilistic search in original SLM. The acceleration of SLM by PSO requires 10-20 times fewer iterations. Second, we leverage parallel processing in the SLM implementation. Experimental results show that the accelerated SLM method achieves a speed up factor of 577 in training time while maintaining comparable classification/regression performance of original SLM.

I. INTRODUCTION

Feature-based classification and regression methods, where feature extraction and decision-making are two separate modules, have been well studied for general classification and regression tasks for decades in the field of machine learning. In recent years, deep learning (DL) methods handle feature extraction and decision-making jointly in training deep neural networks (DNNs) through back propagation (BP). In this work, we adopt the traditional machine learning pipeline and aim to improve the high-performance decision-making (i.e., classification and regression) module with features as the input.

Built upon the classical decision tree (DT) [1] idea, SLM has been recently proposed in [2]. SLM offers a high-performance machine learning solution to general classification and regression tasks. Both DT and SLM convert the high-dimensional feature space splitting problem into a 1D feature space splitting problem. DT compares each feature in the raw feature space and chooses the one that meets a certain criterion and searches for the optimal split point, which is the threshold value. In contrast, SLM conducts a linear combination of existing features to find a new discriminant 1D space and find the optimal split point accordingly. It demands to find optimal weights of the linear combination, which is computationally heavy, in the training stage.

It was shown in [2] that SLM achieves better performance than the support vector machine (SVM) [3], DT [1], multilayer perceptron (MLP) [4], [5] with fewer parameters on several benchmarking machine learning datasets. The performance improvement is reached at the expense of higher computational complexity. In this work, we investigate two ideas to accelerate SLM: 1) proposal of a more efficient weight learning strategy via particle swarm optimization (PSO) [6], and 2) software implementation optimization with parallel processing on CPU and GPU.

For the first idea, the search of optimal weights is accomplished by probabilistic search in original SLM. No optimization is involved. The probabilistic search demands a large number of iterations to achieve desired results. It is difficult to scale for a high-dimensional feature space. In contrast, PSO simulates the behavior of a flock of birds, where each bird is modeled by a particle. PSO finds the optimal solution in the search space by updating particles jointly and iteratively. The acceleration of SLM by PSO requires 10-20 fewer iterations than the original SLM.

For the second idea, SLM’s training time can be significantly reduced through software implementation optimization. In the original SLM development stage, its implementation primarily relies on public python packages such as Scikit-learn[7], NumPy, SciPy [8], etc. After its algorithmic detail is settled, we follow the development pathway of XGBoost [9] and LightGBM [10], implement a C++ package, and integrate it with Cython. Both multithreaded CPU and CUDA are supported.

By combining the above two ideas, accelerated SLM can achieve a speed up factor of 577 in training as compared to the original SLM while maintaining comparable (or even better) classification/regression performance. The rest of this paper is organized as follows. The SLM method is reviewed in Sec. II. The acceleration of SLM through PSO is introduced in Sec. III. The optimization of software implementation is presented in Sec. IV. Experimental results are shown in Sec. V. Finally, concluding remarks are given in Sec. VI.

II. SUBSPACE LEARNING MACHINE (SLM)

A. Methodology

Inspired by the feedforward multilayer perceptron (FF-MLP) [5], decision tree (DT) [1] and extreme learning machine (ELM) [11], a new machine learning model named SLM was proposed by Fu et al. in [2] recently. Its main idea is subspace partitioning, where an input feature space
is partitioned into multiple discriminant sets in a hierarchical manner. The whole input feature space is treated as a parent node and it is partitioned by a hyperplane into two subsets, where each subset has a lower entropy value. Each subset is denoted by a child node. This partitioning process is conducted recursively until samples in leaf nodes are pure enough or the sample number is small enough to avoid overfitting. Majority vote is adopted at each leaf node where all test samples in the node are predicted as the majority class. Different from traditional DT methods, SLM allows multiple splitting of a parent node at one level, which makes the SLM tree wider and shallower to prevent overfitting.

Instead of partitioning a node based on the original input feature space, SLM searches for discriminant projections for each parent node and performs its partitioning in a projected 1D space. The projected 1D space from the input feature vector $x$ can be expressed as

$$ F_a = \{ f(a) \mid f(a) = a^T x \}, $$

where

$$ a = (a_1, \ldots, a_d, \ldots, a_D)^T, \quad ||a|| = 1, $$

is the projection vector on the unit hypersphere in $\mathbb{R}^D$ and $D$ is the dimension of the input feature space. The 1D subspace can be partitioned into two disjoint sets based on a threshold, $t_0$, selected by the discriminant feature test (DFT) in [12]. They are written as

$$ F_{a, t_0, +} = \{ f(a) \mid a^T x \geq t_0 \}, $$

$$ F_{a, t_0, -} = \{ f(a) \mid a^T x < t_0 \}. $$

We summarize the SLM tree construction as follows.

1) A discriminant subspace $S^0$ of dimension $D_0$ is selected from the original feature space.

2) $p$ projection vectors are generated to project feature space $S^0$ to $p$ 1D subspaces. Among them, the discriminability of each projected subspace is evaluated using the DFT loss in [12], to select the best $q$ discriminant 1D subspaces. Correlation among them are considered to ensure diversity.

3) The node is split based on each selected projected subspaces, denoted as $S^1$. This completes the node splitting in one level of the SLM tree.

4) Conduct node splitting recursively until the stopping criteria are met.

### B. Probabilistic Selection in SLM

The most important and challenging step in SLM is the selection of discriminant projection vectors that can separate samples into disjoint sets of lower entropy. Each projection vector is a linear combination of one-hot basis vectors, $e_d$, $d = 1, \ldots, D$, in form of

$$ a = a_1 e_1 + \cdots + a_d e_d + \cdots + a_D e_D, $$

where $a_d$ represents the coefficient of the corresponding basis vector $e_d$.

A probabilistic search scheme was proposed in [2] to achieve this task. It first orders $e_d$ based on its discriminant power evaluation using the DFT loss from the highest to the lowest. Then, Eq. (5) can be rewritten as

$$ a = a_1' e_1' + \cdots + a_d' e_d' + \cdots + a_D' e_D'. $$

where vectors $e_d'$ are ordered from the most to the least discriminant ones. Then, different sets of coefficients $\{a_1', a_2', \ldots, a_D'\}$ form different projection vectors. Although random selection can be performed to find optimal sets of coefficients, the search space is too large to be practical. Three hyper-parameters are proposed to guide the search in probabilistic search as described below.

- $P_d$: the probability of selecting coefficient $a_d'$
  Since basis vectors are rank ordered in decreasing discriminant power in Eq. (6), we let $P_d$ decrease exponentially as $d$ increases. In other words, the more discriminant a basis, the higher likelihood being selected. The resulting $a$ tends to be more discriminant as well.

- $A_d'$: the dynamic range of coefficient $a_d'$
  $A_d'$ is randomly selected among integer values expressed as

$$ a_d' = 0, \pm 1, \pm 2, \cdots, \pm |A_d|, $$

where

$$ A_d = a_0 \exp(-\alpha d). $$

The more discriminant a basis, the higher dynamic range in selection.

- $R$: the number of selected coefficients
  A subset of $R$ basis vectors out of $D$ are considered in the linear combination in Eq. (6). The remaining $(D - R)$ corresponding coefficients are set to 0. This further reduces the search space for a large $D$ value.

Given a set of selected projection vectors using the probabilistic search scheme as stated above, the cosine similarity between each pair is measured. To ensure the diversity of projected 1D spaces, we prune projection vectors of higher correlations from the final selected set.

### III. ACCELERATION VIA ADAPTIVE PARTICLE SWARM OPTIMIZATION (APSO)

As described in Sec. II, the probabilistic search is used in the original SLM to obtain good projection vectors. As the feature dimension increases, the search space grows exponentially and the probabilistic search scheme does not scale well. In this section, we adopt a metaheuristic algorithm called the particle swarm optimization (PSO) [13] to address the same problem with higher efficiency.

#### A. Introduction to PSO

PSO is an evolutionary algorithm proposed by Kennedy and Eberhart in 1995 [6]. It finds the optimal solution in the search space by simulating the behavior of flock using a swarm of particles. Each particle in the swarm has a position vector...
and a velocity vector. Mathematically, the $i$th particle has the following position and velocity vectors:

\begin{align}
\mathbf{x}_i &= [x_1, x_2, \ldots, x_D]^T, \\
\mathbf{v}_i &= [v_1, v_2, \ldots, v_D]^T
\end{align}

where $D$ is the dimension of the search space, respectively. The velocity and position vectors of the $i$th particle are updated in each iteration via

\begin{align}
\mathbf{v}_{i,t+1} &= \omega \mathbf{v}_{i,t} + c_1 r_1 \left[ \mathbf{p}_{B,t} - \mathbf{x}_{i,t} \right] + c_2 r_2 \left[ \mathbf{g} - \mathbf{x}_{i,t} \right], \\
\mathbf{x}_{i,t+1} &= \mathbf{x}_{i,t} + \mathbf{v}_{i,t}.
\end{align}

Eq. (11) has the following parameters:

- inertia weight $\omega$ used to control the influence of the current speed of the particle on the update.
- $\mathbf{gB}$: the position vector of the optimal particle in the population.
- $\mathbf{p}_{B,t}$: the best position of the $i$th particle in the history update process up to iteration $t$.
- $c_1$ and $c_2$: two learning factors.
- $r_1$ and $r_2$: two uniformly distributed random values between 0 and 1 used to simulate the stochastic behavior of particles.

The initial position and velocity vectors of all particles are randomly selected within a preset range. At each iteration, the velocity and position of each particle are updated according to Eqs. (11) and (12), respectively. Furthermore, we calculate the current loss and update $\mathbf{gB}$ and $\mathbf{pB}$ accordingly. If the algorithm converges, the position vector of the global optimal particle is the output. The pseudo-codes of the PSO are provided in Algorithm 1.

### Algorithm 1: Particle Swarm Optimization (PSO)

**Input:** Population, Dimension and MaxIteration  
**Output:** Best Particle’s position vector and loss

1. Initialize all particles  
2. for iteration < MaxIteration do  
3. 
4. for $i < \text{Population}$ do  
5. Update $V_i$ and $X_i$  
6. if loss$_i < \text{pbest}_i$ then  
7. Update $\text{pbest}_i$  
8. if loss$_i < \text{gbest}$ then  
9. Update $\text{gbest}$  
10. else  
11. pass  
12. $i = i + 1$ //next particle  
13. iteration = iteration + 1 //next iteration  
14. return $\text{gbest}$

The PSO algorithm has a couple of variants [14], [15], [16]. The search space of the projection vector in our problem is complex and there are quite a few local minima. A standard PSO can easily get stuck to the local minima. One variant of PSO, called the adaptive particle swarm optimization (APSO) [17], often yields better performance for multimodal problems so that it is adopted as an alternative choice in the SLM implementation.

Before an iteration, APSO calculates the distribution of particles in the space and identify it as one of the following four states: Exploration, Exploitation, Convergence, and Jumpout. APSO chooses parameters $c_1$ and $c_2$ in different states dynamically so as to make particles more effective under the current distribution. They are summarized below.

- In the exploration state, APSO increases the $c_1$ value and decreases the $c_2$ value. This allows particles to explore in the whole search space more freely and individually to achieve better individual optimal position.
- In the exploitation state, APSO increases the $c_1$ value slightly and decreases the $c_2$ value. As a result, particles can leverage the local information more to find the global optimum.
- In the convergence state, APSO increases both $c_1$ and $c_2$ slightly. An elite learning strategy is also introduced to promote the convergence of globally optimal particles, which makes the search more efficient.
- In the jumping-out state, APSO decreases the $c_1$ value and increases the $c_2$ value. This allows particles move away from their current cluster more easily. Thus, the particle swarm can jump out of the current local optimal solution.

For classification tasks in SLM, particle’s position vector is the projection vector and the DFT loss is utilized as the loss function. In original SLM, we split samples into several child nodes at each tree level. Since APSO can find the global optimum with the lowest loss function, only one globally optimal projection is selected per node in the accelerated version. As a result, the resulting SLM tree is a binary tree. Furthermore, in the accelerated SLM algorithm, the top $n$ discriminant features in each node are re-computed and they serve as the input dimension of the APSO algorithm. We will show in the experimental section that APSO demands fewer iterations to find the global optimum than probabilistic search. Thus, the overall search speed can be improved. The pseudo-codes of the SLM tree building based on APSO is given in Algorithm 2.

### IV. Acceleration via Parallel Processing

The SLM algorithm in [2] was implemented in pure Python. While Python code is easy to write, the running speed is slow. Cython is a superset of Python that bridges the gap between Python and C. It allows static typing and compilation from Python to Cython. The bottleneck is the search of the
optimal projection vector. For a projection vector at a node, we calculate the loss at each split point so as to find the minimum loss in the 1D space. Since these computations do not depend on each other, they can be computed in parallel in principle. Python does not support multithreading due to the presence of the Global Interpreter Lock (GIL). To this end, multi-processing was deemed unsuitable due to higher overhead in the current context.

Instead, we implement multithreaded C++ extension for the optimal projection vector search, and integrate it into the remaining Cython code. In the C++ extension, we spawn multiple threads, where each thread covers the computation of a fixed number of splits per selected projection vector. The launched thread number is equal to the minimum of the physical core number to prevent excessive context switching. A task is defined as calculating the loss on a single split of a projected 1D subspace. These tasks are evenly allocated to each thread wherein each thread processes them sequentially.

Compared to CPUs, modern GPUs have thousands of cores and context switching occurs significantly less frequently. Thus, for GPU processing, each thread is responsible for a single task in our design. Once the task is completed, the GPU core moves onto the next available task, if any. Once the computation is completed, the loss values are returned to the main process where the minimum loss projection and split are computed.

V. EXPERIMENTS

Datasets. We follow the experimental setup in [2] and conduct performance benchmarking on 9 classification datasets (i.e., Circle-and-ring, 2-new-moons, 4-new-moons, Iris, Wine, B.C.W, Pima, Ionosphere, and Banknote) and 6 regression datasets (i.e., Make Friedman1, Makefriedman2, Makefriedman3, Boston, California housing and Diabetes). For details of these 15 datasets, we refer readers to [2].

Benchmarking Algorithms and Acceleration Settings. Ensemble methods are commonly used in the machine learning field to boost the performance. An ensemble model aims to obtain better performance than each constituent model alone. Inspired by the random forest (RF) [18] and XGBoost for DT, SLM Forest and SLM Boost are bagging and boosting methods for SLM, respectively. The counter part of SLM for the regression task is called the subspace learning regressor (SLR). Again, SLR has two ensemble methods; namely, SLR Forest and SLR Boost. They are detailed in [2].

We compare the run-time of six settings of SLM, SLM Forest and SLM Boost three methods in Table I and six settings of SLR, SLR Forest and SLR Boost in Table II. They are:

A Probabilistic search in pure Python,
B Probabilistic search in Cython and C++ singletreaded,
C Probabilistic search in Cython and C++ multithreaded,
D Probabilistic search in CUDA/GPU,
E APSO in pure Python,
F APSO in Cython and C++ multithreaded.

For a given dataset, settings [A]-[D] provide the same classification (or regression) performance in terms of classification accuracy (or regression MSE) while settings [E] and [F] offer the same classification accuracy (or regression MSE). Their main difference lies in the model training time. All settings are implemented on Intel(R) Xeon(R) CPU E5-2620 v3 @2.40GHz with 12 cores 24 threads and Nvidia Quadro M6000 GPU. All hyperparameters are the same. The number of trees for two SLM/SLR ensemble methods (i.e., SLM/SLR Forest and SLM/SLR Boost) is set to 30.

Acceleration by Parallel Processing. We compare the run-time of SLM, SLM Forest and SLM Boost for the 9 classification datasets in Table I and that of SLR, SLR Forest and SLR Boost for the 6 regression datasets in Table II. The shortest run-time is highlighted in bold face. We see from the tables that simply changing the implementation from Python to C++ yields a speed-up factor of x40 to x100 across all datasets. After that, the speed-up is incremental. Multithreading should lead to increased performance in theory. Yet, it is greatly influenced by the state of the machine in practice. For example, caching, task scheduling, and whether other processors are used by other processes. Moreover, additional overhead occurs in spawning, managing, and destroying threads. Typically, a thread of longer computation is more suitable for multithreading. Thus, the performance improvement in smaller datasets, where computations take less time, is less pronounced. Similarly, for a single SLM tree training, the model training time on GPU is notably worse due to the overhead of moving data to GPU memory. As the training scales (e.g., with ensembles of SLM/SLR trees such as SLM/SLR Forest and SLM/SLR Boost), CUDA/GPU generally outperforms multithreading.

Acceleration by APSO. To demonstrate the effectiveness of APSO, we compare the classification and regression performance of probabilistic search and APSO in Tables III and IV, respectively. Their performance in terms of classification accuracy and regression MSE is comparable. As shown in the last column of both tables, the iteration number of APSO is about 10% (for SLM) and 5% (for SLR) of that of probabilistic search.

Joint Acceleration by Parallel Processing and APSO. We can see the effect of joint acceleration of parallel processing and APSO from Tables I and II method F. We observe
that the single-threaded C++ implementation of APSO has a comparable speed of the GPU version of probabilistic search. We have not yet done the GPU version of APSO acceleration but expect that it will provide another level of boosting in shortening the training run time.

In summary, APSO acceleration reduces the time complexity of SLM/SLR by reducing the number of iterations to 5-10%. Its C++ implementation and parallel process reduces the training time of each iteration by a factor of x40 to x100. With the combination of both, APSO SLM/SLR with C++ implementation gives the best overall performance in training time, classification accuracy and regression error. As compared to the original SLM/SLR, its training time can be accelerated by a factor up to 577.

VI. CONCLUSION AND FUTURE WORK

Two ways were proposed to accelerate SLM in this work. First, we applied the particle swarm optimization (PSO) algorithm to speed up the search of a discriminant dimension, which was accomplished by probabilistic search in original SLM. The acceleration of SLM by PSO requires 10-20 times fewer iterations. Second, we leveraged parallel processing in the implementation. It was shown by experimental results that, as compared to original SLM/SLR, accelerated SLM/SLR can achieve a speed up factor of 577 in training time while maintaining comparable classification and regression performance.

The datasets considered in [2] and this work are of lower dimensions. Recently, research on unsupervised representation of images and videos has been intensively studied, e.g., [19], [20], [21], [22], [23], [24], [25], learned representations and their labels can be fed into a classifier/regressor for supervised classification and regression. This leads to

---

**TABLE I**

| Settings | circle-and- | 4-new- | Iris | Wine | B.C.W | Pima | Ionosphere | Banknote |
|----------|-----------|--------|------|------|-------|------|------------|---------|
|          | 2-new-moons | 4-new-moons |       |      |       |      |            |         |
| SLM      | 7.026   | 4.015  | 6.019 | 82.939 | 65.652 | 234.597 | 339.077 | 248.410 | 118.000 |
|          | 0.287   | 0.188  | 0.200 | 0.465 | 0.800 | 2.931 | 3.920 | 5.217 | 0.697 |
|          | 0.346   | 0.0943 | 0.236 | 0.295 | 0.700 | 2.657 | 3.263 | 3.838 | 0.439 |
|          | 4.071   | 3.159  | 3.245 | 3.205 | 3.572 | 3.956 | 4.660 | 5.021 | 4.370 |
|          | 7.597   | 3.146  | 7.200 | 4.188 | 3.106 | 11.605 | 13.892 | 8.123 | 8.588 |
|          | 0.270   | 0.0887 | 0.278 | 0.192 | 0.831 | 2.837 | 2.738 | 1.371 | 0.619 |
| SLM Forest | 160.264 | 107.695 | 157.307 | 233.514 | 265.288 | 506.735 | 10024.238 | 340.787 | 752.104 | 230.587 |
|          | 8.507   | 5.272  | 8.471 | 17.142 | 26.254 | 134.251 | 101.631 | 152.625 | 40.365 |
|          | 7.708   | 4.954  | 7.678 | 5.946 | 15.066 | 98.969 | 48.987 | 137.390 | 16.341 |
|          | 220.021 | 96.712 | 219.002 | 105.768 | 94.788 | 360.089 | 396.989 | 247.069 | 259.000 |
|          | 8.75    | 4.058  | 9.02  | 5.822 | 20.081 | 98.490 | 31.684 | 75.964 | 14.054 |
| SLM Boost | 143.332 | 116.156 | 151.904 | 234.704 | 2591.919 | 718.338 | 987.947 | 6395.744 | 3412.654 |
|          | 7.459   | 5.185  | 7.533 | 16.087 | 21.873 | 117.01 | 104.47 | 148.586 | 23.443 |
|          | 7.987   | 3.434  | 8.293 | 12.144 | 15.624 | 115.226 | 80.273 | 131.712 | 14.207 |
|          | 7.741   | 4.894  | 8.182 | 6.093 | 15.573 | 97.959 | 46.057 | 76.589 | 15.588 |
|          | 233.071 | 97.541 | 233.347 | 107.709 | 95.534 | 362.295 | 396.989 | 247.069 | 262.546 |
|          | 6.837   | 3.581  | 8.425 | 5.915 | 20.159 | 94.413 | 40.434 | 79.012 | 15.516 |

**TABLE II**

| Settings | Make Friedman1 | Make Friedman2 | Make Friedman3 | Boston | California_housing | Diabetes |
|----------|---------------|---------------|---------------|--------|--------------------|---------|
|          | 152.291 | 60.232 | 186.535 | 153.173 | 756.868 | 147.933 |
|          | 2.136   | 0.653  | 2.137 | 1.255 | 26.417 | 1.136 |
|          | 0.937   | 0.331  | 2.100 | 0.780 | 5.625 | 0.608 |
|          | 1.577   | 0.448  | 3.832 | 1.31 | 5.908 | 0.722 |
|          | 29.742  | 28.294 | 44.511 | 30.608 | 96.319 | 27.162 |
|          | 0.639   | 0.332  | 0.755 | 0.675 | 5.891 | 0.322 |
| SLM Forest | 5406.271 | 3346.395 | 7459.572 | 6454.979 | 21092.217 | 4413.089 |
|          | 62.527  | 19.244 | 57.117 | 55.204 | 1049.859 | 46.622 |
|          | 35.576  | 13.533 | 65.511 | 30.943 | 254.415 | 29.606 |
|          | 19.469  | 6.48 | 42.182 | 18.04 | 125.104 | 38.379 |
|          | 1025.801 | 940.359 | 1051.498 | 792.33 | 3190.336 | 77.153 |
|          | 17.993  | 13.718 | 19.898 | 11.178 | 193.741 | 10.879 |
| SLM Boost | 4678.86  | 1955.317 | 67/0.380 | 4097.66 | 21612.672 | 4208.828 |
|          | 62.156  | 19.724 | 59.435 | 38.154 | 817.868 | 34.767 |
|          | 26.704  | 10.546 | 70.716 | 29.149 | 159.434 | 20.528 |
|          | 19.293  | 6.551 | 51.222 | 19.817 | 124.707 | 38.379 |
|          | 919.874 | 920.619 | 1834.926 | 910.271 | 3336.018 | 901.092 |
|          | 14.170  | 13.114 | 24.759 | 12.186 | 195.459 | 10.594 |
the traditional pattern recognition paradigm, where feature extraction and classification modules are cascaded for various tasks. The feature dimensions for these problems are in the order of hundreds. It is important to develop SLM for high dimensional image and video data to explore its full potential.

REFERENCES

[1] L. Breiman, J. Friedman, C. Stone, and R. Olshen, “Classification and regression trees (crc, boca raton, fl),” 1984.
[2] H. Fu, Y. Yang, V. K. Mishra, and C.-C. J. Kuo, “Subspace learning machine (slm): Methodology and performance,” arXiv preprint arXiv:2205.01299, 2022.
[3] C. Cortes and V. Vapnik, “Support-vector networks,” Machine learning, vol. 20, no. 3, pp. 273–297, 1995.
[4] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” Psychological review, vol. 65, no. 6, p. 386, 1958.
[5] R. Lin, Z. Zhou, S. You, R. Rao, and C.-C. J. Kuo, “From two-class linear discriminant analysis to interpretable multilayer perceptron design,” arXiv preprint arXiv:2009.04442, 2020.
[6] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proceedings of ICNN’95–international conference on neural networks, vol. 4, pp. 1942–1948, IEEE, 1995.
[7] F. Pedregosa, G. Varoquaux, A. Gramfort, M. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., “Scikit-learn: Machine learning in python,” the Journal of machine Learning research, vol. 12, pp. 2825–2830, 2011.
[8] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., “Scipy 1.0: fundamental algorithms for scientific computing in python,” Nature methods, vol. 17, no. 3, pp. 261–272, 2020.
[9] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794, 2016.
[10] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” Advances in neural information processing systems, vol. 30, 2017.
[11] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: theory and applications,” Neurocomputing, vol. 70, no. 1-3, pp. 489–501, 2006.
[12] Y. Yang, W. Wang, H. Fu, and C.-C. J. Kuo, “On supervised feature selection from high dimensional feature spaces,” arXiv preprint arXiv:2203.11924, 2022.