**Stochastic Optimization in Machine Learning Pipelines:**

**Selecting Features and Hyperparameters**

|  |  |
| --- | --- |
| **Robert Milletich\*** | **Anthony Asmar** |
| **Deloitte Consulting LLP** | **National Institutes of Health** |
| **Richmond, Virginia** | **Bethesda, Maryland** |
| **rmilletich@deloitte.com** | **anthony.asmar@nih.gov** |

**\*Corresponding Author: rmilletich@deloitte.com**

# ABSTRACT

Feature selection and hyperparameter tuning are important components of a machine learning pipeline. In practice, these components are often optimized independently of each other, for example, first optimizing feature selection and then optimizing hyperparameters. This results in the complex dependencies between the parameter spaces to be ignored. To account for such dependencies, Bayesian optimization routines have been developed that explicitly account for interactions and also uncertainty in the parameter spaces. Although Bayesian approaches are widely used, a major limitation is the difficulty in parallelizing the computation due to the serial nature of the parameter updates. To overcome limitations with dependencies in parameter spaces and parallel computing, we demonstrate the application of stochastic optimization algorithms in jointly optimizing feature selection and hyperparameter tuning on simulated and benchmark data sets. Specifically, we compare several stochastic algorithms (e.g., random search, genetic algorithms, particle swarm optimization, random search with dynamic updating) with non-stochastic and Bayesian algorithms to optimize feature selection and hyperparameter tuning of gradient boosting tree classifiers. Our results highlight the effectiveness of stochastic optimization for feature selection and hyperparameter tuning across a variety of data sets, including high-dimensional data and noisy data.

# ABOUT THE AUTHORS

**Dr. Robert Milletich** (Deloitte Consulting LLP) is a senior data scientist in the Analytics and Cognitive practice of Deloitte Consulting. His current research interests include ensemble learning and machine learning pipeline optimization. He has a M.S. in Experimental Psychology, M.S. in Applied Mathematics, and a Ph.D. in Psychological Sciences from Old Dominion University.

**Dr. Anthony Asmar** (National Institutes of Health) is a postdoctoral fellow in the Stem Cell Biochemistry Unit of the NIDCR. His current research interests include the intersection of computational biology and bench research for streamlined disease discovery. He has a M.S. in Biology and a Ph.D. in Biomedical Sciences from Old Dominion University.

**Stochastic Optimization in Machine Learning Pipelines:**

**Selecting Features and Hyperparameters**

|  |  |
| --- | --- |
| **Robert Milletich\*** | **Anthony Asmar** |
| **Deloitte Consulting LLP** | **National Institutes of Health** |
| **Richmond, Virginia** | **Bethesda, Maryland** |
| **rmilletich@deloitte.com** | **anthony.asmar@nih.gov** |

**\*Corresponding Author: rmilletich@deloitte.com**

# INTRODUCTION

Machine learning has become increasingly popular across many industries and remains one of the most in demand technologies for businesses. Despite this popularity, however, the process for developing, deploying, and continuously improving machine learning pipelines is becoming more challenging given complexities with models and the data they consume (Sculley et al. 2015). A substantive part of developing and maintaining accurate real-world machine learning systems involves optimizing components of the prediction pipeline. Among these, two important components are feature selection and hyperparameter tuning. Feature selection, also known as variable selection, is the process of identifying an optimal subset of relevant features as model inputs. Feature selection techniques are used for several reasons, including to improve the predictive accuracy and generalization of models by reducing overfitting and eliminating irrelevant or redundant features, reduce the computational cost and time for model training and inference, and simplify models to increase interpretability (Colaco et al., 2019; Liu et al., 2010). Hyperparameter tuning is the process of choosing a set of optimal hyperparameters (i.e., parameter whose value controls the learning process) for a learning algorithm (Hastie et al., 2009).

**Problem Statement**

Let the be a -dimensional vector of binary values for feature inclusion (i.e., 1=yes, 0=no) and be a -dimensional vector of continuous values for hyperparameters, then the joint optimization problem of selecting features and hyperparameters is the maximum for a function ,

|  |  |
| --- | --- |
|  | (1) |

where the function is often a cross-validation function that returns a performance metric such as classifier accuracy. In practice, techniques for solving Equation (1) can be categorized into three classes: (1) non-stochastic optimization, (2) Bayesian optimization, and (3) stochastic optimization. Non-stochastic optimization techniques decouple the joint optimization given by Equation (1) and focus on independently optimizing feature selection first (selecting ), then optimizing hyperparameters (selecting ), or vice-versa. A commonly used approach that is easy to implement is to first select relevant features using a random forest (Breiman, 2001) and then optimize a model’s hyperparameters using a grid search. The decoupling of optimization problems offers flexibility such that different feature selection techniques (e.g., filter methods, wrapper methods, embedded methods) can be combined with different hyperparameter tuning techniques (e.g., random search, Bayesian optimization). A notable limitation, however, is that the decoupling ignores complex dependencies between the feature and hyperparameter spaces, which may result in unstable solutions.

In an effort to explicitly account for model dependencies in parameter spaces, Bayesian optimization techniques can be used to jointly model the feature and hyperparameter spaces. These techniques efficiently trade off exploration and exploitation of the parameter space to identify a configuration that best optimizes some overall evaluation metric (Snoek et al., 2012). Among the several popular implementations, the most popular is the Tree of Parzen Estimators (TPE) as described in Bergstra et al. 2013. Although Bayesian techniques have been shown to have excellent performance in practice, a major limitation is the difficulty in parallelizing the search evaluation due to the serial nature of the parameter updates (Li et al., 2016b). As such, to overcome limitations with dependencies in parameter spaces and parallel search evaluation, stochastic optimization techniques can be leveraged.

Stochastic optimization methods generate and use random variables as candidate solutions to Equation (1). In the simplest implementation, random search (Bergstra & Bengio, 2012) repeatedly draws candidate solutions for a finite number of iterations and the best solution is kept as the optimal configuration. Random search has been shown to outperform all other methods, including TPE; however, at the expense of computing time (Li et al., 2016b). Other stochastic algorithms, such as genetic algorithms and particle swarm optimization, seek to improve on random search by using meta-heuristics to more efficiently search through the parameter space after evaluating an initial set of random candidates. The literature has shown the success of these algorithms in solving feature selection tasks and hyperparameter optimization tasks (Kabir et al., 2011; Lane et al., 2013; Li et al., 2009; Xue et al., 2013), yet little research exists on the effectiveness of such algorithms for tasks such as joint optimization in machine learning pipelines.

This paper evaluates the effectiveness of stochastic algorithms for joint optimization of feature selection and hyperparameter tuning. Given the expensive computing time, we introduce a modification to the random search algorithm using dynamic updating. In short, random search is run for a predefined number of iterations, then the feature selection and hyperparameter distributions are updated based on the past configuration results, and a new set of candidate solutions are sampled from their updated distributions.

**EXPERIMENTS**

In this section, we evaluate stochastic optimization methods for feature selection and hyperparameter tuning on both simulated and real-world data sets. Specifically, we compare stochastic optimization methods (i.e., genetic algorithm, particle swarm optimization, random search, random search with dynamic updating) with a non-stochastic optimization method (i.e., embedded random forest feature selector followed by a grid search) and a Bayesian optimization method (i.e., Tree of Parzen estimators). We use Scikit-Learn (Pedregosa et al., 2011) and HyperOpt (Bergstra et al., 2013) packages for implementing the non-stochastic and Bayesian methods, respectively. All stochastic algorithms are implemented by the authors and the code is publicly available at <https://github.com/rmill040/mlsopt>. Experiments are conducted on a M5A 16xlarge EC2 instance with 64 CPUs using Amazon Web Services.

**Data**

We simulate four binary classification data sets with varying numbers of features and relevant features: (1) 200 features with 20 relevant (sim200-20), (2) 200 features with 2 relevant (sim200-2), (3) 500 features with 50 relevant (sim500-50), and (4) 500 features with 5 relevant (sim500-5). All data sets are generated with 100 samples and approximately balanced class distributions. In addition, we use 11 standard benchmark data sets from the ASU feature selection website (Li et al., 2016a) and the UCI repository (Lichman, 2013). The data span both low-dimensional and high-dimensional regimes and binary and multiclass outcomes. See Table 1 for a summary of the experimental data.

Table 1. Summary of Experimental Data Sets

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Samples | Features | Classes |
| AALMAL | 72 | 7,129 | 2 |
| cancer | 569 | 30 | 2 |
| CLL-SUB-111 | 111 | 11,340 | 3 |
| Madelon | 2,600 | 500 | 2 |
| orlraws10P | 100 | 10,304 | 10 |
| pixraw10P | 100 | 10,000 | 2 |
| spam | 4,601 | 57 | 2 |
| TOX-171 | 171 | 5,748 | 4 |
| warpAR10P | 130 | 2,400 | 10 |
| warpPIE10P | 210 | 2,420 | 10 |
| Yale | 165 | 1,024 | 15 |
| sim200-2\* | 100 | 200 | 2 |
| sim200-20\* | 100 | 200 | 2 |
| sim500-5\* | 100 | 500 | 2 |
| sim500-50\* | 100 | 500 | 2 |
| \* Simulated data. | | | |

**Model and Hyperparameter Distributions**

We use a gradient boosted tree model as the machine learning classifier. Gradient boosted models are supervised learning models that are built in a sequential manner using weak learners, such as shallow decision trees (Breiman et al., 1984), to form an ensemble of weak prediction models (Friedman, 2001). The XGBoost (Chen & Guestrin, 2016) package is used for the implementation of gradient boosted trees.

The statistical distributions of the 14 hyperparameters for optimization are presented in Table 2. In larger search spaces, we use log-based distributions to enable more efficient searching.

Table 2. Distribution of Hyperparameters for Gradient Boosted Tree Classifier

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameter | Distribution | Lower Bound | Upper Bound |
| n\_estimators | qUniform(q=50) | 50 | 1000 |
| max\_depth | qUniform(q=1) | 1 | 11 |
| min\_child\_weight | qUniform(q=1) | 1 | 20 |
| max\_delta\_step | qUniform(q=1) | 0 | 3 |
| learning\_rate | LogUniform | 0.001 | 0.5 |
| subsample | LogUniform | 0.50 | 1.0 |
| colsample\_bytree | LogUniform | 0.50 | 1.0 |
| colsample\_bylevel | LogUniform | 0.50 | 1.0 |
| colsample\_bynode | LogUniform | 0.50 | 1.0 |
| gamma | LogUniform | 0.0001 | 5.0 |
| reg\_alpha | LogUniform | 0.0001 | 1.0 |
| reg\_lambda | LogUniform | 1.0 | 4.0 |
| base\_score | LogUniform | 0.01 | 0.99 |
| scale\_pos\_weight | LogUniform | 0.1 | 10.0 |
| *Note*. qUniform is sampled as *round*(value/q)\*q, where value ~ Uniform(0, 1). The lower and upper bounds of LogUniform distributions are actually based on the log of the bound, but the log is omitted for readability. | | | |

**Feature Selection Distributions**

We use independent and identically distributed Bernoulli distributions with success probability of 0.5 to select each feature. In other words, each feature has a 50/50 chance of being selected as a feature for inclusion in the model.

**Experimental Algorithms and Settings**

We compare the performance and compute time for six algorithms on the simulated and real-world data sets. For all stochastic and Bayesian algorithms, a budget of 200 parameter configurations or iterations is set. For the feature selection with grid search algorithm, given that the number of configurations is based on the Cartesian product of a finite hyperparameter grid, the actual number of parameter configurations evaluated is 216, but relevant compute time metrics are adjusted to reflect the approximate time for running 200 candidates.

For random search with dynamic updating, genetic algorithms, and particle swarm algorithms, an initial 40 configurations are generated using the same random seed, and five rounds of updating are run for a total of 200 configurations evaluated. As such, given that these algorithms initialize with the same candidate solutions, they can be compared to determine the effectiveness of searching for solutions in high-dimensional parameter spaces. A description of each experimental algorithm is presented below.

A 5-fold stratified cross-validation procedure is used to evaluate the solutions and the accuracy score is the selected performance metric. The best parameter configuration id and total compute times are also collected to evaluate the effectiveness of each algorithm.

**Feature Selection with Grid Search (FSGS)**

We use an embedded feature selector (random forests) to first select features and then a grid search to select the optimal hyperparameters. Specifically, we use a random forest with 50 trees, a max depth of 7, and adjustments for balanced class weights. The hyperparameter grid for the six selected hyperparameters is presented in Table 3.

Table 3. Grid of Hyperparameters for Gradient Boosted Tree Classifier

|  |  |
| --- | --- |
| Hyperparameter | Grid |
| n\_estimators | [100, 500, 1000] |
| max\_depth | [1, 6, 11] |
| learning\_rate | [0.1, 0.01] |
| subsample | [0.80, 1.0] |
| colsample\_bytree | [0.80, 1.0] |
| base\_score | [0.2, 0.5, 0.8] |

**Tree of Parzen Estimators (TPE)**

We use the HyperOpt implementation of the TPE algorithm. The TPE algorithm is a sequential model-based optimization (SMBO) using TPE to select optimal configurations. SMBO methods sequentially construct models to approximate the performance of untested parameter configurations based on the performance of already evaluated configurations, and then subsequently choose new configurations to test based on this model. More specifically, the TPE approach models and , where represents the parameter configurations (i.e., selected features and hyperparameters) and the associated accuracy score. is modeled by transforming the generative process of hyperparameters, replacing the distributions of the configuration prior with non-parametric densities.

**Random Search (RS)**

We use a vanilla implementation of random search for joint optimization of feature selection and hyperparameter tuning. The algorithm simply generates a parameter configuration using the predefined statistical distributions and evaluates the performance.

**Random Search with Dynamic Updating (RSDU)**

As described above, we develop a modification to the random search algorithm by using dynamic distribution updating. The algorithm is implemented as follows:

1. Initialize 40 parameter configurations
2. Evaluate parameter configurations
3. Select top 50% of best solutions and update statistical distributions
   1. For feature distributions, selection probabilities are updated as the proportion of times a feature appears in the top 50% of solutions. We also set a muting threshold where features with an updated selection probability less than 0.25 are excluded for future evaluations.
   2. For hyperparameter distributions, the lower and upper bounds are updated based on the lowest and highest sampled values for each hyperparameter. This allows the originally wide statistical distributions to become tighter as the search progresses.
4. Sample 40 new parameter configurations based on updated statistical distributions
5. Repeat steps 2-4 five times for a total of 200 tested configurations.

Figure 1 presents an example run comparing RS with RSDU with 10 iterations and 10 candidates at each iteration. Note, the bottom plot demonstrates the effect of distribution updating as the search progresses.

A screenshot of a social media post

Description automatically generated

Figure 1. Hyperparameter Optimization Comparing Random Search

with Dynamic Updating and without Dynamic Updating

**Genetic Algorithm (GA)**

We use a common variant of the GA for joint optimization of feature selection and hyperparameter tuning. The algorithm is implemented as follows:

1. Initialize 40 chromosomes’ genes
2. Evaluate chromosomes’ fitness
3. Perform tournament selection to obtain 40 new parent chromosomes
4. Perform uniform crossover to obtain new population of 40 chromosomes
5. Perform mutation on new population
6. Repeat steps 2-5 for five iterations for a total of 200 tested configurations

**Particle Swarm Optimization (PSO)**

We use a basic variant of the PSO algorithm for joint optimization of feature selection and hyperparameter tuning. The algorithm is implemented as follows:

1. Initialize 40 particles’ positions and velocities
2. Initialize 40 particles’ best positions as starting positions and update the swarm’s best position
3. Update particles’ velocities
4. Update particles’ best positions and swarm’s best position
5. Repeat steps 3-4 for five iterations for a total of 200 tested configurations

**Results**

Table #. Classifier Accuracy by Experimental Data Set and Algorithm

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Algorithm | | | | | |
| Name | FSGS | TPE | RS | RSDU | GA | PSO |
| AALMAL | .9457 | **1.0000** | .9867 | .9733 | .9724 | .9867 |
| cancer | .9543 | **.9719** | .9614 | **.9719** | .9684 | .9684 |
| CLL-SUB-111 | .8198 | **.8656** | .8379 | .8557 | .8557 | .8289 |
| Madelon | **.8542** | .8246 | .8096 | .8131 | .7788 | .6912 |
| orlraws10P | .9300 | **.9900** | **.9900** | **.9900** | **.9900** | **.9900** |
| pixraw10P | .9400 | .9900 | .9900 | .9900 | .9900 | **1.0000** |
| spam | .9472 | **.9500** | .9315 | .9404 | .9409 | .9233 |
| TOX-171 | .8190 | .8832 | .8714 | .8714 | .8657 | **.8894** |
| warpAR10P | .8769 | .8846 | .8615 | .8615 | .8615 | **.9077** |
| warpPIE10P | .9667 | **1.0000** | .9952 | .9952 | .9905 | **1.0000** |
| Yale | .7273 | .7758 | .7455 | .7455 | .7697 | **.8061** |
| sim200-2\* | **.9100** | **.9100** | .8900 | .8900 | **.9100** | .8700 |
| sim200-20\* | .7400 | .7700 | .7600 | .7300 | **.7800** | .7200 |
| sim500-5\* | .9300 | .9400 | **.9500** | **.9500** | .8900 | .9100 |
| sim500-50\* | .6100 | **.6300** | .5800 | .5600 | .5900 | .6200 |
| Best Algorithm | 2 | 8 | 2 | 2 | 3 | 6 |
| *Note*. ADD HERE.  \* Simulated data. | | | | | | |

Table #. Compute Time in Minutes by Experimental Data Set and Algorithm

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Algorithm | | | | | |
| Name | FSGSa | TPE | RS | RSDU | GA | PSO |
| AALMAL | **0.19** | 55.33 | 0.91 | 0.99 | 0.91 | 0.22 |
| cancer | 0.16 | 3.88 | **0.13** | 0.17 | **0.13** | 0.15 |
| CLL-SUB-111 | 1.19 | 121.87 | 8.00 | 9.28 | 8.67 | **0.93** |
| Madelon | 5.73 | 16.77 | 3.99 | 3.93 | 1.45 | **0.40** |
| orlraws10P | **2.47** | 137.09 | 16.12 | 15.75 | 11.69 | 3.27 |
| pixraw10P | 2.03 | 145.91 | 15.23 | 15.37 | 8.44 | **1.80** |
| spam | 1.49 | 9.70 | 1.03 | 0.93 | 0.66 | **0.25** |
| TOX-171 | 4.43 | 78.07 | 8.44 | 9.57 | 3.73 | **1.13** |
| warpAR10P | 3.67 | 38.52 | 4.56 | 5.24 | 4.25 | **1.47** |
| warpPIE10P | 5.01 | 49.74 | 7.50 | 8.99 | 3.53 | **0.87** |
| Yale | 5.26 | 35.29 | 3.61 | 3.51 | 2.62 | **1.31** |
| sim200-2\* | 0.14 | 4.08 | 0.14 | 0.16 | **0.13** | 0.14 |
| sim200-20\* | 0.13 | 4.15 | **0.12** | 0.16 | **0.12** | 0.14 |
| sim500-5\* | 0.24 | 6.62 | 0.21 | 0.22 | 0.19 | **0.15** |
| sim500-50\* | 0.29 | 5.85 | 0.21 | 0.21 | 0.16 | **0.14** |
| Best Algorithm | 2 | 0 | 2 | 0 | 3 | 10 |
| *Note*. ADD HERE.  \* Simulated data.  a | | | | | | |

Experimental results are presented in Table 4. With regards to average classification accuracy, the TPE obtained the

Table 4. Summary of Algorithm Performance on Experimental Data Sets

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Classification Accuracy | | Compute Time in Minutes | | Percent Best Iterationa | |
| Algorithm | Mean | SD | Mean | SD | Mean | SD |
| FSGS | .8647 | .1104 | 2.16c | 2.11c | - | - |
| TPE | .8924 | .1060 | 47.52 | 50.65 | 54.50 | 29.97 |
| RS | .8774 | .1173 | 4.68 | 5.39 | 16.50 | 26.40 |
| RSDU | .8759 | .1129 | 4.97 | 5.54 | 24.50 | 33.97 |
| GA | .8769 | .1118 | 3.11 | 3.71 | 42.10 | 28.42 |
| PSO | .8741 | .1196 | 0.82 | 0.88 | 49.83 | 31.05 |

highest classification accuracy of 89.24% and the FSGS obtained the lowest classification accuracy of 86.47%. The PSO, GA, RS, and RSDU performed similarly with classification accuracies of approximately 87-88%. Overall, an analysis of variance (ANOVA) model reveals that these classification accuracies are not statistically significant, *F*(5, 84) = 0.0916, *p* = .993.

For average compute times, the PSO obtained the lowest compute time of 0.82 minutes, whereas, the TPE obtained the highest compute time of 47.52 minutes. The longer compute times for TPE was expected, however, these times are also based on using 64 threads in the XGBoost classifier model. For computing environments with less resources, these compute times would noticeably increase. Overall, an ANOVA model reveals that these compute times are significantly different, *F*(5, 84) = 11.25, *p* < .0001. Tukey’s pairwise comparisons reveal that among the 15 algorithm comparisons, each non-Bayesian algorithm has significantly lower compute times than the TPE algorithm (*p* = .001). These differences highlight the efficiency that parallel computing can have on searching large parameter spaces.

The percent best iteration metric is calculated as the 100 \* best iteration index / 200, where lower scores indicate the optimal solution is found earlier in the search algorithm. For average percent best iteration, the RS obtained the lowest percent of 16.50%, whereas, the TPE obtained the highest percent of 54.50%. Overall, an ANOVA reveals that the these metrics are statistically different, *F*(4, 70) = 4.447, *p* = .0029. Tukey’s pairwise comparisons reveal that the among the 10 algorithm comparisons, the RS has significantly lower percent best iteration metrics compared to both PSO (*p* = .027) and TPE (*p* = .008).

With regards to the count of times an algorithm was selected as best, both the TPE and PSO are selected among 5 out of 16 data sets as the top performing algorithms. The GA and RSDU are selected 3 times and 1 time, respectively, out of the 16 data sets as the top performing algorithms. Random search was never selected as a top performing algorithm across any data set. Interestingly, these results demonstrate that stochastic optimization algorithms are able to obtain comparable results to their Bayesian counterparts.

**CONCLUSION**

In this paper, we evaluated approaches to feature selection and hyperparameter optimization using stochastic-based algorithms to jointly optimize the search space. The idea is to solve both feature selection and hyperparameter tuning in the same algorithm. We demonstrate the effectiveness of the stochastic-based algorithms across a series of simulated and benchmark data sets and show comparable predictive performance in a fraction of the time as commonly used state-of-the-art methods such as Bayesian optimization using TPE.

# REFERENCES

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine*

*Learning Research, 13*(1), 281-305.

Bergstra, J., Yamins, D., & Cox, D. D. (2013). Making a science of model search: Hyperparameter pptimization in

hundreds of dimensions for vision architectures. *To appear in Proceedings of the 30th International Conference on Machine Learning (ICML 2013).*

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5-32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont,

CA: Wadsworth.

Chen, T., & Guestrin, C. (preprint 2016). XGBoost: A scalable tree boosting system *arXiv:1603.02754*.

Colaco, S., Kumar, S., Tamang, A., & Biju, V. G. (2019). A review on feature selection algorithms. I*n Shetty N.,*

*Patnaik L., Nagaraj H., Hamsavath P., Nalini N. (eds) Emerging Research in Computing, Information, Communication and Applications. Advances in Intelligent Systems and Computing, Volume 906* (pp. 133-153). Singapore: Springer.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine*. Annals of Statistics, 29*(5),

1189-1232.

Gheyas., I. & Smith, L. S. (2010). Feature subset selection in large dimensionality domains. *Pattern Recognition,*

*43*(1), 5-13.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *Elements of statistical learning* (2nd ed.). New York, NY:

Springer.

Kabir, M. M., Islam, M. M., & Murase, K. (2011). A new local search based hybrid genetic algorithm for feature

selection. *Neurocomputing, 74*, 2194-2928.

Lane, M. C., Xue, B., Liu, I., & Zhang, M. (2013). Particle swarm optimization and statistical clustering for feature

selection. In *Advances in artificial intelligence* (pp. 214-220). Cham, Switzerland: Springer.

Lichman, M. (2013). UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>.

Li, J., Cheng, K., Wang, S., Morstatter, F., Robert, T., Tang, J., & Liu, H. (2016a). Feature selection: A data

perspective. *arXiv:1601.07996*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016b). Hyperband: A novel bandit-based

approach to hyperparameter optimization. *arXiv:1603.06560*.

Liu, H., & Motoda, H. (2007)*. Computational methods of feature selection*. Chapman and Hall/CRC Press*.*

Liu, H., Motoda, H., Setiono, R., & Zhao, Z. (2010). Feature selection: An ever-evolving frontier in data mining.

*Journal of Machine. Learning. Research Proceeding Track,* *10,* 4-13.

Li, Y., Zhang, S., & Zeng, X. (2009). Research of multi-population agent genetic algorithm for feature

selection. *Expert Systems with Applications,* *36*(9), 11570-11581.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., …, Duchesnay, E. (2011).

Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Philips, T., Ebner, D., … & Dennison, D. (2015). Hidden technical

debt in machine learning systems. *In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (pp. 2503-2511). Cambridge, MA: MIT Press.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *In*

*Advances in neural information processing systems* (pp. 2951–2959).

Xue, B., Zhang, M. J., & Browne, W. N. (2013). Particle swarm optimization for feature selection in classification: A

multi-objective approach. *IEEE Transactions Cybernetics, 43*(6), 1656-1671.