International Journal For Technological Research In Engineering
Volume 10, Issue 1, September-2022 *ISSN (Online): 2347 - 4718*

PERFORMANCE ANALYSIS OF MAPREDUCE BASED CONVOLUTIONAL NEURAL NETWORKS FOR EVENT DETECTION IN STREAMING DATA

[1]Puja Kumari, [2]Dr. Mukesh Kumar
[1]Research Scholar, [2]HOD & Assistant Professor
Department of Computer Science & Engineering
Rabindranath Tagore University, Raisen, Madhya Pradesh, India

Abstract- An event is a significant occurrence or large-scale activity that is unusual relative to normal patterns of behavior. May be associated with naturally occurring phenomena and manual system interactions Event detection is the process of analyzing event streams in order to discover sets of events matching patterns of events in an event context. Event detection is a subfield of computer vision that analyzes input videos with the goal of determining when a particular anomalous event has occurred.
In streaming data is an event is a fundamental unit of data. A time-correlated event pattern is a collection of occurrences. Human-computer interaction, surveillance, intelligent transportation systems, and space exploration are only some of the uses of event detection based on continuous media [1]. The process of studying event streams in order to uncover collections of events matching patterns of events in an event context is known as event detection. Event kinds are defined by event patterns and circumstances.

Keywords—Convolutional Neural Network (CNN) , hadoop,

1. INTRODUCTION

Subscribers to an event type should be notified if a set of events matching the pattern of the event type is detected during the analysis. Filtering and aggregation of events are usually part of the analysis. Machine learning has been heavily incorporated into state-of-the-art recognition systems [1,2]. The convolutional neural network (CNN), a multi-layer neural network, is a biologically inspired deep learning model. CNN can learn discriminative features automatically, unlike previous machine learning algorithms that rely on sophisticated handcrafted features. This network model may be applied directly to the original image and can extract classification features for image classification automatically, eliminating the complexity and blindness of traditional image classification. CNN, on the other hand, may be appropriate only in specific circumstances

We use a spatiotemporal convolutional neural network model (STCNN) to extract image features on the spatial dimension and motion information on the temporal dimension from successive continuous media frames to produce a sparse representation to analyse time serial dynamic continuous media images. The sparse auto-encoder (SA) is an unsupervised deep learning model that imposes sparse constraints on the training of each auto-encoder layer using the sparse coding concept [4-7, 13,14]. To improve STCNN, we use a sparse auto-combination technique inspired by the sparse auto-encoder algorithm to combine input feature maps to which a type of sparsity limitation is enforced at the convolution stage. As a result, the convolution layer can learn the best combination of feature maps.

Sensors, networks, and electronic technologies are rapidly developing, resulting in an explosion of continuous media data. This has a significant detrimental impact on the convolutional neural network's training speed for a certain task. As a result, parallel processing of large-scale data from image sensors has emerged as a critical issue to be addressed. The storage and processing of such enormous amounts of data could not be done on a regular PC or supercomputer. Some researchers have made numerous attempts to expand machine learning into large-scale Hadoop applications in order to mine possible information from massive amounts of data.

Sensors, Hadoop is a prominent cloud computing platform that is based on an open source implementation of the MapReduce methodology. The hardware requirements aren't excessive. Hadoop may be installed on multiple common PCs to create a strong distributed cluster. Combining distributed computing and machine learning will become a significant machine learning growth direction in the context of Big Data.

The AC-CNN is a sparse Auto-Combination spatio-temporal Convolutional Neural Network that can automatically learn advanced characteristics of continuous media data, according to this dissertation. Large-scale data mining is well-suited to it. We parallelize AC-CNN on a Hadoop cluster called AC-CNN-MR, which stands for AC-CNN with MapReduce, in order to increase its classification ability and mine the main elements of the action from large scale continuous media data [9,10,17].

2. CONVOLUTIONAL NEURAL NETWORK

The Convolution Neural Network is a Neural Network model that focuses on streaming data analysis for applications such as image categorization and object detection. CNNs, or Convolutional Neural Networks, have shown to be an effective method for streaming data processing under Deep Learning, alongside other Neural Network algorithms such as Standard Neural Network, Recurrent Neural Network, and Hybrid Neural Network. So far, the Convolutional Neural Network has been the most widely used method for stream

data analysis.

CNNs are most commonly employed for image analysis, but they may also be utilised for data analysis and classification challenges [28, 31,40-43]. The Convolutional Neural Network is based on the principle of filtering images before training a deep neural network. As previously said, a Convolutional Neural System is a type of deep learning strategy that has been popular in a variety of computer vision tasks and is gaining popularity in a variety of fields, including radiology. .

3. AC-CNN FOR EVENT DETECTION

We take 6 successive 3230 size frames as the input of the AC-CNN in the aforesaid architecture to capture motion information encoded in multiple contiguous continuous media frames, taking the current frame as the centre. Assume the input frames are all grayscale images of 32x32 pixels. If the sizes differ, the scaling procedure must be used to convert them to 32-bit sizes.

Using a 756 convolution kernel, the C1 layer can extract 36 feature maps from seven consecutive input frames, which means using 26 distinct learnable convolution kernels to extract 26 different features. Despite the fact that event classification is based on a large number of complex characteristics, 26 feature maps produced from the input frames are perfectly capable of classifying the simple action.

A subsampling layer is the S1. Its goal is to scale the C1 layer's acquired feature maps, which will improve the ER-resistance CNN's to scale changes and minor deformation. The sub-sampling layer's scaling factor can't be too large. We wouldn't be able to extract useful features from the raw image data if this wasn't the case. The C1 layer's thirty-six 3030 size feature maps are uniformly scaled, resulting in thirty-six 1010 feature maps as the output of the S1 layer.

We take 6 consecutive 32x32 size frames as the input of the AC-CNN in information encoded in multiple contiguous continuous media frames, with the current frame as the centre. Assume that all of the input frames are grayscale images of 32x32 pixels. If the sizes differ, the scaling procedure must be used to convert them to 3232 sizes.

The C1 layer can generate 36 feature maps from seven successive input frames using the 756 convolution kernel, meaning that 28 different learnable convolution kernels were employed to extract 28 different features. Despite the fact that event classification is reliant on a large number of complicated attributes, the 28 feature maps generated from the input frames are more than adequate of categorising a simple action. Because of its magnitude, the convolution is quite enormous.

The S2 layer is a sub-sampling layer, comparable to the S1 layer. We may get two sets of feature maps, each with thirty-four 101 size feature maps, by multiplying the scaling factor by two.

Convolutional layers are also present in the C3 layer. It convolutions two groups of feature maps from the S2 layer with a 25.5 size convolution kernel, resulting in two sets of feature figures, each with thirty-two 88.5 size feature maps. This convolution kernel is 606 times larger than the preceding one, which was 55. The most important argument is that a sub-sampling layer is simple to understand. The size of the resulting feature map is 99.2 percentage, if we utilise a 32% size convolution kernel. We can't use the scaling factor 4 for sub-sampling because nine isn't even. The result of the C3 layer can then be used to blend 32 feature maps into a single group.

The S3 layer is followed by the F layer, which is a fully connected layer. The completely connected state denotes that each S3 layer neuron is coupled to each F layer neuron, resulting in a generic neural network. Actually, we can join all of the S3 layer's neurons as a network layer with 1800 neurons, and then we can connect all of the S3 layer's neurons to the F layer, which has sixty-four neurons. As a result, the S3 layer and the F layer have a total of 122032 connections.

After the F layer, the ultimate output layer is a fully connected layer. The number of its neurons represents the number of events that can be recognised.

MapReduce Implementation of Matrix Multiplication

The AC-CNN training process is a continuous iteration procedure, with each iteration heavily reliant on the findings of the prior iteration. It is not suited for parallelization on the framework utilising MapReduce. However, each repetition is actually a matrix multiplication. The Algorithm 1 is a MapReduce-based global AC-CNN training system.

Algorithm 1: AC-CNN-MR training Algorithm

```
1:train AC-CNN (sample data)
2:init a global AC-CNN
3:for sample in trials
4:      output=feed-forward-MapReduce(AC-CNN, sample)
5: error=Calculate-error(output, label)
6:     Error-backpropagation-MapReduce(AC-CNN, error)
7:end for
8:save AC-CNN
```

Each sample is utilised to keep it up to date. We use MapReduce parallelization for forward propagation and error back propagation when updating. We propose a matrix parallel computing approach based on MapReduce to expedite the training of SATCNN using MapReduce. Assume you have two matrices, Amt and Btn. The usual algorithm for calculating Cmn=AB is as follows:

Algorithm 2: Matrix Multiplication Algorithm

```
Input: A_{m×t}, B_{t×n}
Output: C_{m×n}
C_{m×n}:=0;
for i:m
  for j=1:n
    for k=1:t
      C_{i,j} = C_{i,j} + A_{i,k} B_{k,j};
    end for
  end for
end for
```

The above algorithm has a temporal complexity of T=O. (mnk). It's the same as the cubic level calculation amount. This is a time-consuming procedure. As a result, we can submit various Ci,j calculations to separate machines for parallel processing. As a result, we create a MapReduce-based parallel matrix multiplication technique.

We solve Cmn=AB given the matrices Amt and Btn. Given that the MapReduce process accepts a file containing Key Value Pair and I and j denote the element's index subscript. If the value of M13 is 7, for example, it is recorded as 'M125.' We develop the Map and Reduce processes as described in Algorithm based on the aforementioned premise.

Algorithm 3: Implementation of MapReduce for Matrix Multiplication

```
Object key, Context context
// value: M_i_j_M_{i,j}
if M = 'B'  //value is an element of B
  for c=1:n
    map_key. set(i_c);
    map_value. set(A_j_B_{i,j});
    context1.write(map1_key, map1_value);
  end for
  for r=1:m
    map_key. set(r_j);
  end for
Reduce(Object key, Context)
//values: A_j_A_{i,j} or B_i_B_{i,j}
sum:=0; A_row:=0; B_col:=0;
for values
  if value==B_j_B_{i,j}
    A_row[j]. set(B_{i,j});
  else
    B_col[i]. set(B_{i,j});
  end if
end for
```

The Map process simply copies the elements of A and B, which are then sent to the Reduce process to be calculated. Each of A's elements, Ai, j, must be multiplied by Bj,c (c=1, 2,..., n). Ci, c will result from the addition of Ai, j*Bj, c. As a result, the term I c) is utilised to identify. It will yield m*t*n elements in the end. Every element of B, Bi, j, must be multiplied by Ar,i (r=1, 2,..., m). Ar, i*Bi, j will add up to Cr,j (r, j), which is utilised as an identification keyword. It will eventually yield t*n*m components. The key, value>, which is the same as keyword, is used during the shuffling phase.

The Map and Reduce functions on Hadoop can be regarded the smallest parallel execution units. We'll look at the time complexity of the algorithm if the aforesaid application is running in a distributed cluster of p computers. Because each Map function has an O(n) time complexity, when p = max(mt/p,1), the mt Map functions can be distributed to a maximum of mt machines.

It must run mn Reduce routines throughout the Reduce phase. Each Reduce function has an O time complexity (t). As a result, the Reduce phase's overall time complexity is O(max(mn/p, 1)t. During the MapReduce process, the time cost of network transmission and the cost of file reading and writing are not taken into account in the above study. There may be significant variances during the experiment process due to the various circumstances.

Because of AC-CNN imposes sparse constraints on the network, most matrices formed of parameters are sparse as well. We can make the matrix multiplication above even better. At initially, we just store nonzero elements while saving the matrix.

Algorithm 4: Process for Reducing Matrix Multiplication

```
Reduce(Object key, Context)
//key: r_c
//values: B_j_B_{i,j} or C_i_C_{i,j}

B_row:= 0;
C_col:=0;
for value in values
  if value==B_j_B_{i,j}
    A_row[j]. set(B_{i,j});
  else
    C_col[i]. set(C_{i,j});
  end if
end for
for j=1:r
  if A_row[j]>0 and B_col[j]>0
    sum+=B_row[j]*C_col[j];
  end if
end for
if sum > 0
  context.write(B_k_c_sum);
end if
```

Since, the Map phase only duplicates the elements, the file already contains a sparse matrix. There is no need to update the Map function. If the corresponding position elements of A and B are both nonzero, it just has to perform multiplication during the Reduce phase, and it only needs to write the results if the corresponding position element of C is nonzero. The aforementioned strategy can improve the algorithm's efficiency even further when the matrix is sparse.

4. EVENT DETECTION

The standard data set namely KTH is often used to test the performance of algorithms for event recognition. It is compiled by nine people and contains ten different sorts of events.

Table 3: Comparison of obtained results with other methods

	recognition rate	frames
Instance-R [32]	99. 6%	10
Instance-L [38]	55. 0%	12
Instance-P [44]	93. 8%	9
Our work 1-frames	87. 0%	1
Our work 3-frames	90. 7%	3
Our work 7-frames	94. 9%	7

It should be mentioned that they cannot be directly compared due to the fact that they employ two separate methodologies. In our method and literature, each continuous media clip is treated as a separate unit and given an event label [11]. The feature is calculated using a temporal window in the literatures, but a label is only allocated to the centre frame of the window. Each continuous media clip, which consists of ten frames. JHUANG [30] retrieves the features from each of the nine frames in the sequence. Although the proposed method recognition rate isn't the best in the best-case scenario.

5. PERFORMANCE OF MAPREDUCE ALGORITHM FOR EVENT DETECTION

Both KTH and related data sets were used in the trials to validate the validity of AC-CNN-MR after employing Hadoop MapReduce. The results of the parallel experiment were compared to the results of the serial experiment. The experiment was performed 81 times. Finally, we calculated an average of all the results. Figure 16 shows that the ability of ER-MR is nearly identical to the ability of AC-CNN.

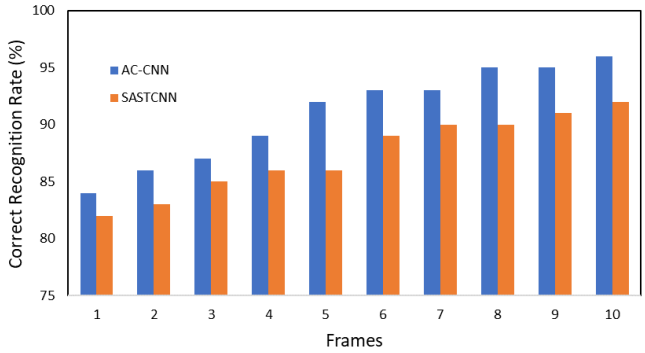

Figure 16: Accuracy of AC-CNN and SASTCNN on the KTH dataset.

This demonstrates the method's portability and verifies the feasibility and correctness of AC-CNN-MR parallelization in this dissertation. We use the same settings for the KTH data set as we did before. The 5 fold cross-validation approach is used for all experiment assessments in the KTH data set, which means that all samples are randomly divided into 5 groups of equal size. One data set is chosen at random as the testing set, while the others are used as training sets. This technique was repeated five times, with the average of the five experiments provided. The comparison trials on the KTH data set, as shown in Figure 17, also corroborate the validity of the AC-CNN-MR algorithm.

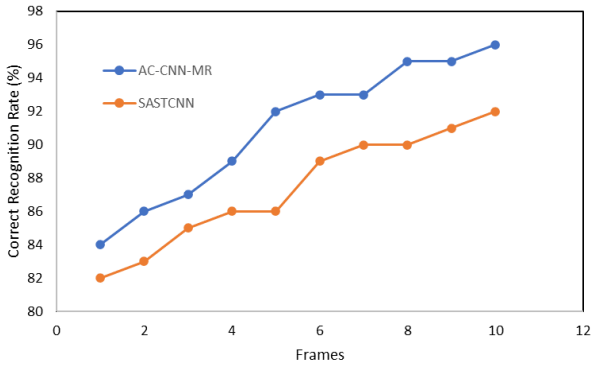

Figure 17: The accuracy of AC-CNN-MR and SASTCNN on the KTH dataset.

We examined the training and testing times of AC-CNN and AC-CNN-MR, respectively, to analyse the speed-up ratio after AC-CNN-MR parallelization. The AC-CNN training and testing times correspond to the time it takes to run AC-CNN-MR on a single machine with Hadoop. We retrieved a continuous media segment of 81 samples from the KTH data set. We also replicated each of them 100 times, yielding a total of 8100 samples. The data samples were trained ten times and the training time results were averaged. We tested 900 samples at random and repeated the process ten times.

We created 65000 samples in total using the same experimental procedure on the KTH data set. A total of 10,500 samples were picked at random for the testing set. The training and testing processes are separated because they have substantial differences. For example, the training process can only accelerate the matrix operation, whereas the testing process can fully accelerate each testing sample. The results are shown in Table 4, which show that the speed-up ratio is not very high.

Table 4: A comparison of the computational costs.

		AC-CNN	AC-CNN-MR	speedup
KTH	training	97854ms	79485ms	1. 23
	testing	15652ms	10422ms	1. 50
KTH	training	793412ms	643776ms	1. 23
	testing	117167ms	77478ms	1. 51

The theoretical speed-up ratio for a Hadoop cluster with three machines is three, although the actual speed-up ratios are significantly less than three. The cause for this is most likely due to network transmission delays and reading and writing data taking up the majority of the time. The speed-up ratio of the training process is too low, which is due to the fact that the training process primarily relies on the matrix operation to accelerate, and because the matrix size is not very large, the network transmission delay overwhelms the matrix parallel algorithm's acceleration effect. The testing process has a higher speed-up ratio than the training process since it

involves the Map process whereas the training process does not.

6. CONCLUSION

In this research study, we have introduced AC-CNN-MR, which parallelizes matrix multiplication using Hadoop MapReduce. The AC-CNN-MR algorithm was first implemented on a Hadoop cluster. On both systems, the performance of AC-CNN-MR was examined, and the experimental findings were analysed in terms of accuracy and computation efficiency.

REFERENCES

[1] Wang, L., Suter, D.: Recognizing human activities from silhouettes: Motion subspace and factorial discriminative graphical model. In: Proceedings of IEEE Conference on Computer Vision and Pattern Detection, pp. 1-8 (2007)

[2] Turaga, P., Chellappa, R., Subrahmanian, V., Udrea, O.: Machine recognition of human activities: A survey. IEEE Trans. Circ. Syst. Vid. 18(11), 1473-1488 (2008)

[3] Weinland, D., Ronfard, R., Boyer, E.: Free viewpoint Detection using motion history volumes. Comput. Vis. Image Und. 104(2-3), 249-257 (2006)

[4] Weinland, D., Boyer, E., Ronfard, R.: Detection from arbitrary views using 3D exemplars. In: Proceedings of IEEE International Conference on Computer Vision, (2007)

[5] Atmosukarto, I., Ahuja, N., Ghanem, B.: Action recognition using discriminative structured trajectory groups. In: Proceedings of IEEE Winter Conference on Applications of Computer Vision, pp. 899-906 (2015)

[6] Yan, Y., Ricci, E., Subramanian, R., Liu, G., Sebe, N.: Multitask linear discriminant analysis for view invariant action recognition. IEEE Trans. Image Process. 23(12), 5599- 5611 (2014)

[7] Bulbul, M., Jiang, Y., Ma, J.: DMMs-Based multiple features fusion for human action recognition. INT. J. Multimedia DATA EN. Manage. 6(4), 23-39 (2015)

[8] Lawrence, S., Giles, C., Tsoi, A., Back, A.: Face recognition: a convolutional neural-network approach. IEEE Trans. Neural Networ. 8(1), 98-113 (1997)

[9] Moez, B., Franck, M., Christian, W., Christophe, G., Atilla, B.: Spatio-temporal convolutional sparse auto-encoder for sequence classification. In: Proceedings of the 23rd British Machine Vision Conference 2012. (2012)

[10] Ranzato, M., Huang, F. J., Boureau, Y., Lecun, L.: Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Detection, pp. 1-8 (2007)

[11] Ranzato, M., LeCun, Y.: A sparse and locally shift invariant feature extractor applied to document images. In: Proceedings of International Conference on Document Analysis and Detection, pp. 1213-1217 (2007)

[12] Hsieh, L., Wu, G., Hsu, Y., Hsu, W.: Online image search result grouping with MapReduce-based image clustering and graph construction for large-scale photos. J. Vis. Commun. Image R. 25(2), 384-395 (2014)

[13] Han, J., Liu, Y., Sun, X.: A scalable random forest algorithm based on MapReduce. In: Proceedings of 2013 4th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 849-852 (2013)

[14] Chen, T., Zhang, X., Jin, S., Kim, O.: Efficient classification using parallel and scalable compressed model and Its application on intrusion detection. Expert Syst. Appl. 41(13), 5972-5983 (2014)

[15] Tewari , N. C., Koduvely, H. M., Guha, S., Yadav, A.: MapReduce implementation of variational Bayesian probabilistic matrix factorization algorithm. In: Proceedings of IEEE International Conference on Big Data, pp. 145-152 (2013)

[16] Nobakht, B., Boer, F.: Multi-core programming. Technology Foundation. 104(4), 430-445 (2006)

[17] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating MapReduce for multi-core and multiprocessor systems. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, 13-24 (2007)

[18] Mao, Y., Morris, R., Kaashoek, M. F.: Optimizing MapReduce for multicore architecture. MIT-CSAIL-TR-2010-020, (2010)

[19] Jiang, W., Ravi, V. T., Agrawal, G.: A Map-Reduce system with an alternate API for multi-core environments. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 84-93 (2010)

[20] Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, R.: MapReduce for machine learning on multicore. Adv. Neural Inform. Process Syst. 19. 281-288 (2006)

[21] Kovoor, G., Singer, J., Luján, M.: Building a Java MapReduce framework for multi-core architectures. In: Proceedings of Multiprog, pp. 87-98 (2010)

[22] Chen, R., Chen, H., Zang, B.: Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. ACM, pp. 523-534 (2010)

[23] Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In BigLearn, NIPS Workshop (2011)

[24] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia. ACM, pp. 675–678 (2014)

[25] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., Bengio, Y.: Theano: new features and speed improvements. In arXiv preprint arXiv:1211.5590.

[26] Yang, J., Zhang, D., Frangi, A.F., Yang, J.: Two-

Dimensional PCA: a new approach to appearance-based face representation and recognition. IEEE Trans. Pattern Anal. Machine Intell. 26(1), 131-137 (2004)

[27] Sheng, Z., Shan, Z.: Face recognition by LLE dimensionality reduction. In: Proceedings of International Conference on Intelligent Computation Technology & Automation.1, pp. 121-123 (2011)

[28] Lanaaya, H., Martin, A., Aboutajdine, D., Khenchaf, A.: A new dimensionality reduction method for seabed characterization: Supervised6 Curvilinear Component Analysis. Oceans-europe.1, 339-344 (2005)

[29] Blank, M., Gorelick, L., Shechtman, E., Irani, M., Basri, R.: Actions as space-time shapes. In: Proceedings of Tenth IEEE International Conference on Computer Vision, pp. 1395-1402 (2005).

[30] Jhuang, H., Serre, T., Wolf, L., Poggio, T.: A biologically inspired system for action recognition. In: Proceedings of IEEE 11th International Conference on Computer Vision. IEEE, pp. 1-8 (2007)

[31] Niebles, J. C., Fei-Fei, L.: A hierarchical model of shape and appearance for human action classification. In: Proceedings of IEEE Conference on Computer Vision and Pattern Detection. IEEE, pp. 1-8 (2007)

[32] Bergstra, J., Yamins, D., & Cox, D. D. (2013, June). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in science conference (Vol. 13, pp. 20). Citeseer.

[33] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011, December). Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems (NIPS 2011) (Vol. 24). Neural Information Processing Systems Foundation.

[34] Caruana, R., & Niculescu-Mizil, A. (2006, June). An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning (pp. 161-168).

[35] Breiman, L. (2001) 'Random Forests'. Machine Learning 45, 5–32.

[36] Sirikulviriya, N., & Sinthupinyo, S. (2011, May). Integration of rules from a random forest. In International Conference on Information and Electronics Engineering (Vol. 6, pp. 194-198).

[37] Lim, C., Lee, S. R., & Chang, J. H. (2012). Efficient implementation of an SVM-based speech/music classifier by enhancing temporal locality in support vector references. IEEE Transactions on Consumer Electronics, 58(3), 898-904.

[38] Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295-316.

[39] Tay, B., Hyun, J. K., & Oh, S. (2014). A machine learning approach for specification of spinal cord injuries using fractional anisotropy values obtained from diffusion tensor images. Computational and mathematical methods in medicine, 2014.

[40] Hosmer Jr, et al. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

[41] Angel, L., Viola, J., Vega, M., & Restrepo, R. (2016, August). Sterilization process stages estimation for an autoclave using logistic regression models. In 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA) (pp. 1-5). IEEE.

[42] Kibriya, A. M., et al. (2004, December). Multinomial naive bayes for text categorization revisited. In Australasian Joint Conference on Artificial Intelligence (pp. 488-499). Springer, Berlin, Heidelberg.

[43] Lowd, D., & Domingos, P. (2005, August). Naive Bayes models for probability estimation. In Proceedings of the 22nd international conference on Machine learning (pp. 529-536).

[44] Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

[45] Cui, Z., & Gong, G. (2018). The effect of machine learning regression algorithms and sample size on individualized behavioral prediction with functional connectivity features. Neuroimage, 178, 622-637.

[46] Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the royal statistical society: series B (statistical methodology), 67(2), 301-320.

[47] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267-288.

[48] Park, T., & Casella, G. (2008). The bayesian lasso. Journal of the American Statistical Association, 103(482), 681-686.

[49] Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In Automated Machine Learning (pp. 3-33). Springer, Cham.

[50] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011, December). Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems (NIPS 2011) (Vol. 24). Neural Information Processing Systems Foundation.

[51] Tanay Agrawal (2020). Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient.

