International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

NIOJS: A Novel Intelligent Model Based on Optimal Jumps for Creating Data Sampling from Big Dataset

Mohammed Mohammed Zayed[1]* , Prof. Fadl Mutaher Ba-Alwi[2] , Hiba ALMarwi[3] and Prof. Gheleb H. AL-Gaphari[4]

Submitted:20/03/2024 **Revised**: 01/05/2024 **Accepted**: 12/05/2024

Abstract: The pervasiveness of big data has revolutionized the landscape of information technology (IT), offering a wealth of insights and opportunities for various sectors, including healthcare, education, and the Internet of Things (IoT). However, the sheer volume and complexity of big data pose challenges in extracting meaningful knowledge. To address this, we propose a novel model for optimal sample selection, enabling efficient extraction of representative subsets from big data. The proposed model, based on optimal jumps, dynamically adapts the clustering process to enhance the efficiency of data sampling. We employ the Adjusted Rand Index (ARI) to evaluate the similarity between clusters and guide the selection of new data in each iteration This model holds the potential to significantly enhance the utilization of big data while reducing computational demands. The proposed could run on big datasets and the samples taken represents the dataset.

Keywords: *Big data sampling, Cluster sampling, DBSCAN, NIOJS , Samples, Optimal Jump*

1. Introduction

The emergence of big data has posed significant challenges in the field of information technology. With its vast volume, intricate nature, and diverse characteristics, big data presents formidable hurdles for traditional data processing techniques. Moreover, inherent issues such as noise, redundancy, imbalance, and false discovery rates further compound the challenges associated with big data. These factors collectively contribute to the computational complexity and burden, rendering many conventional algorithms ineffective.

To tackle these challenges, various approaches have been proposed, including feature selection methods and sampling techniques. Feature selection methods aim to identify and extract the most relevant and informative features from the dataset, thereby reducing dimensionality and improving computational efficiency. On the other hand, sampling techniques focus on extracting representative subsets from big data, enabling the construction of cost-effective models without compromising accuracy.

Our proposed model falls under the category of sampling- based approaches. We aim to alleviate the burden of big data volume by iteratively extracting small samples from the dataset. This approach not only reduces computational demands but also ensures that the extracted samples cover a wide range of cases, introduce an acceptable degree of redundancy, maintain data balance, and reduce noise. Additionally, it enhances practicality and scalability.

This paper introduces "Jump Sampling," an innovative approach to sample selection that leverages the DBSCAN clustering algorithm to efficiently select samples from big datasets. DBSCAN is renowned for its versatility in identifying clusters of various shapes and its adept handling of noisy patterns[1]. However, it faces computational challenges due to the high complexity of its nearest neighbor query[1]. To overcome this hurdle, two strategic approaches are employed.

Firstly, Algorithmic Optimization concentrates on refining the efficiency of the nearest neighbor query algorithm itself. This involves incorporating advanced techniques like spatial indexing structures (e.g., KD-trees) to expedite the search process. However, this method may encounter difficulties when dealing with high-dimensional datasets. Secondly, Data-driven Strategies implement sampling techniques to operate on a representative subset of instances, thereby reducing the overall data processed during the nearest neighbor query. Techniques such as random sampling, stratified sampling, or the application of locality-sensitive hashing (LSH) are deployed, striking a balance between precision and computational efficiency.

In our model, ARI is utilized to measure the similarity between two clusters, assessing the agreement between the labels assigned by a clustering algorithm. This index is often used in the evaluation of clustering algorithms as it considers all pairs of samples and measures how many pairs are assigned to the same or different clusters

[1] *Information system, Sana'a University, Yemen*
ORCID ID : 0000-0002-4961-9467
[2] *Information system, Sana'a University, Yemen*
[3] *Computer Science, Sana'a University, Yemen*
ORCID ID : 0000-0002-0941-1143
** Corresponding Author Email: mzayed@su.edu.ye*

2. Related Work

2.1 The PROTRAS algorithm

introduced by Ros and Guillaume in 2018, stands out for its focus on producing a coreset—a smaller and representative subset of the original dataset. It employs a single parameter, denoted as γ, which signifies the sampling cost and determines the level of approximation for the sample. The parameter is used both in the sampling process and as a stopping criterion. To avoid confusion with the homonymous parameter in DBSCAN, this paper refers to it as γ.

PROTRAS operates through four key steps:

1. Selection of New Element: In each iteration, a new element is chosen for inclusion in the sample based on the highest probability of cost reduction. This approach aims to limit the sample size.
2. Association of Non-Selected Elements: The second step involves associating each element not selected with the closest element already present in the sample.
3. Cost Computation: The algorithm then computes the cost based on the selected sample and associated non-selected elements.
4. Stopping Criterion: In the final step, it is checked whether the computed cost is below the specified threshold determined by the γ parameter. The algorithm terminates when the cost is sufficiently low.

Unlike clustering algorithms such as DENDIS or DIDES, PROTRAS specifically targets the creation of a coreset. The sampling cost, represented by the γ parameter, plays a crucial role both as a single parameter guiding the sampling process and as a stopping criterion. The algorithm utilizes probability calculations to strike a balance between representing density and achieving spatial coverage of the dataset. This trade-off allows PROTRAS to efficiently generate a representative coreset with a controlled level of approximation.

2.2 The BIRCHSCAN algorithm

proposed by Ros and Guillaume in 2018, is designed to efficiently cluster large datasets by combining the BIRCH and DBSCAN clustering algorithms. The algorithm utilizes biased sampling, aiming to generate a sample that accurately represents the entire dataset without overlooking small partitions.

Here's a breakdown of the BIRCHSCAN algorithm and its key steps:

Biased Sampling with BIRCH (CF-Tree):

BIRCH is applied to the entire dataset using parameters minPTS and ε (used by DBSCAN), along with the parameter δ, defining the Threshold for BIRCH.

The CF-Tree data structure is built during the process, storing subclusters as entries in leaf nodes. Each entry is not just a single element but represents a subcluster with many elements within a spherical neighborhood defined by the Threshold.

BIRCHSCAN uses the centroids of these subclusters as the biased sample for clustering.

DBSCAN on Biased Sample:

DBSCAN is applied to the biased sample obtained from BIRCH, using parameters minPTS, ε, and weights assigned to each sample element based on the size of the corresponding CF subcluster returned by BIRCH.

Weights allow for more consistent clustering by identifying dense elements. Only one element is considered dense, even

if it represents multiple elements in the entire dataset. Labeling

and Final Clustering:

DBSCAN labels each element of the biased sample.

The entire dataset is labeled based on the subclusters returned by BIRCH. Elements belonging to the same subcluster share the same label.

Parameter Tuning (δ):

The size of the CF-Tree generated by BIRCH is influenced by the Threshold parameter (ε × δ).

An empirical study was conducted to determine a reasonable value for δ, with the study indicating that δ = 0.5 achieved a good balance between sample size and clustering results.

The experiments showed that BIRCHSCAN is competitive compared to other methods for clustering large datasets. The algorithm's strength lies in its use of biased sampling with BIRCH to create a representative sample, facilitating efficient clustering with DBSCAN. The parameter δ plays a crucial role in balancing the sample size and achieving satisfactory results.

3. The Proposed Model

Samples shall cover as most variety of data as possible, to be bias, introduce an acceptable degree of redundancy and reduce noise with the least computational cost possible. In this approach, we employ the DBSCAN clustering algorithm to identify areas of density within the dataset, aided by the proposed jump and ARI metrics. ARI, which does not entail a high computational cost. our primary objective is not to establish clusters but rather to identify representative samples. DBSCAN is chosen for its ability to discern density-based structures within the data, allowing us to pinpoint regions of significance. By incorporating ARI, a measure of clustering similarity, we can evaluate the effectiveness of our approach in identifying these dense areas and selecting suitable samples from them. Although

ARI itself does not impose significant computational overhead, the overall computational complexity of our approach may vary depending on the dataset's size and dimensionality and minPoints parameter of DBSCAN . It's important to note that the efficacy of our approach hinges on the suitability of DBSCAN and ARI for our specific dataset and objectives. In the proposed model, the Jump is a critical aspects as it keeps the probability of finding new minor clusters high , it does not neglect sparse areas. Jump mechanism divides dataset initially to 2 segments (visited, unvisited) the visited segments is further divided into promising areas and weak or sparse areas as the model will collocate the number of cluster created and how many noise points. the jump collects fixed number of records in every iteration fig 1 illustrates the flowchart of jump while fig 2 depicts the model.

3.1 algorithm for jump mechanism

The initial jump is randomly made within the range of the dataset with a specific jump size.

If the data points are good (resulting in many new clusters), this area is considered promising, and the next jump is made next to it.

Jumping will continue to this area until it is fully utilized, meaning the returned data is similar to the data we already have or contains too much noise.

A new jump is executed to a far unvisited area. The model controls the jump, utilizing DBSCAN and ARI to determine whether to accept, exclude, or replace the sample.

In this paper, our approach involves utilizing clustering on jump data to identify clusters and noise points, which are crucial for the functionality of the jump and for enabling comparison between clusters using ARI. The limited data provided by the jump enhances clustering efficiency, although the effectiveness of DBSCAN depends on two key parameters: minPoints and eps. To address this, we manually specify these parameters in our study.

We assess the output of each jump using ARI against all previously accepted samples. Clusters meeting our criteria are accepted and added to the array of accepted clusters. Conversely, if the ARI index falls below a certain threshold, the cluster is excluded. Excluded clusters are then compared to the most relevant clusters from the accepted set. If an excluded cluster demonstrates superior characteristics, it replaces the relevant accepted cluster, while the latter is added to the excluded array ,this process continues until we stopped finding new clusters as the ensures that we get the optimal representatives of the big dataset , and this is the stopping criteria thus there is no need to specify sample size

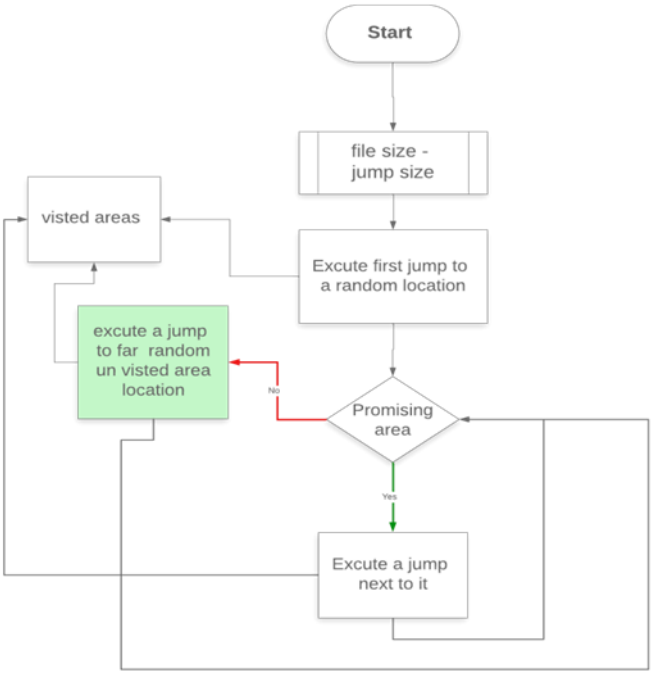

Fig. 1 Jump Algorithm Flow Chart

3.2 Flowchart and proposed model

1 Start

2 Supply minPTS and esp and jump size

3 Execute the initial jump, cluster it, and add it to the accepted cluster since there is nothing yet to compare it to

4 Execute the next jump, cluster it, and then compare it with the accepted clusters for similarity. The Adjusted Rand Index (ARI) comes in handy in this process. If the result of ARI is less than 0, the clusters are not accepted.

5 If the result is greater than or equal to 0, then the cluster is further checked for better sampling with the most similar cluster.

5.1 If the current cluster (doomed to be excluded) has better clusters, it is replaced with the already accepted weaker cluster. The replaced cluster is then added to the rejected.

5.2 If the current cluster is not better than the already accepted most similar cluster, it is added to the rejected clusters.

6 Check if the exit limit has been reached .

6.1 If no execute a jump .

6.2 If yes stop .

7 Stop.

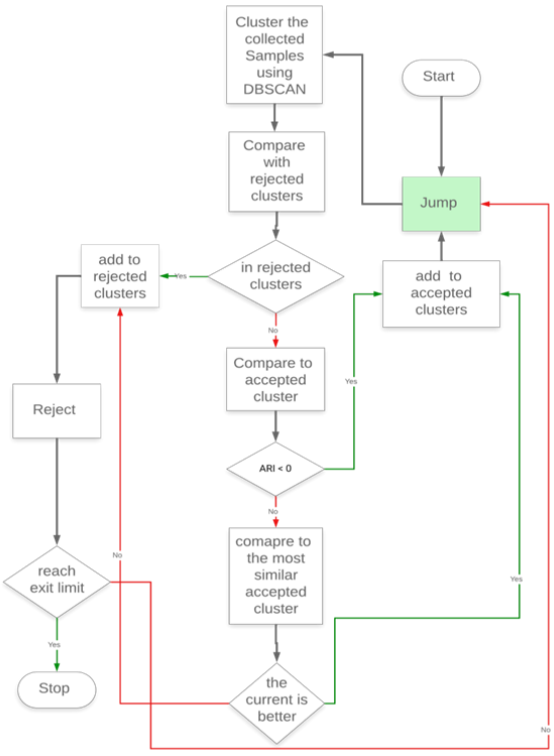

Fig. 2 Jump Model Flow Chart

3.2.1 Pseudocode for Clustering with Jump Algorithm

```
        // Step 5.1: Check if new cluster has better quality
        if NewClusterIsBetter(NewCluster, MostSimilarCluster) then
            // Replace weaker cluster
            ReplaceCluster(AcceptedClusters, MostSimilarCluster, NewCluster)
            RejectedClusters.add(MostSimilarCluster)
        else
            // Step 5.2: Add current cluster to rejected
            RejectedClusters.add(NewCluster)
        end if
    else
        // Clusters not similar, reject
        RejectedClusters.add(NewCluster)
    end if
    // Step 6: Check exit condition
    if ExitLimitReached() then
        Stop
    end if
  end while
// Step 7: End process
Stop
```

```
Start
  // Step 2: Supply parameters
  Input: minPTS, esp, jump_size

  // Step 3: Execute the initial jump
  ExecuteJump()
  ClusterResult = ClusterData(minPTS, esp)
  AcceptedClusters = [ClusterResult] // Add first cluster to accepted

  // Step 4: Loop for further jumps
  While (ExitLimitNotReached) do
    ExecuteJump()
    NewCluster = ClusterData(minPTS, esp)

    // Step 4.1: Compare with accepted clusters using ARI
    Similarity = CompareClustersWithARI(NewCluster,
AcceptedClusters)

    // Step 5: Check similarity result
    if Similarity >= 0 then
      MostSimilarCluster = FindMostSimilarCluster(NewCluster,
AcceptedClusters)
```

3.3 Model Preserving Dataset Original Shape

Our Model Preserve the same shape of the dataset after sampling fig3 to fig 5 shows the shapes of dataset sample's shapes of

Abalone , Isolet and Machinery Fault\imbalance respectively while fig 6 shows where samples were taken , that proves wild coverage of the samples over the dataset.

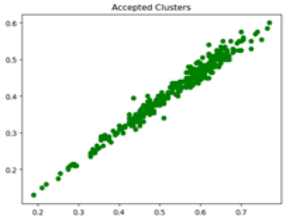

Fig. 3 Abalone Samples Shape

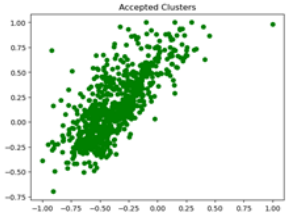

Fig. 4 Isolet Samples Shape

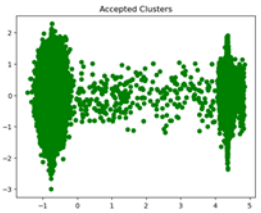

Fig.5 Machinery Fault\ Samples Shape

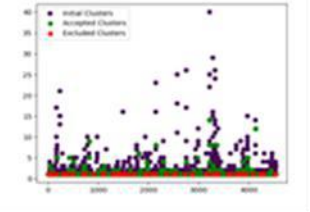

Fig.6 shows samples coverage all over the dataset

Dataset Name	*Size*	*Features*	*Example*	*DBSCAN MinPt*	*ARI*	*Time in Seconds*
Abalone	184 kb	8	4177	50	80.05	0.30
Isolet	34881 kb	617	7797	1	70.86	1.37
Machinery Fault\imbalance	5.37 GB	3056	11749953	100	82.02	481.51

Table 1. Experimental parameters and time

4. Results and Discussion

Experiments were repeated with different DBSCAN parameters, specifically MinPoints, until the optimal value was found. The epsilon (eps) value was kept fixed at 0.5. Table 1 lists the final parameters used for DBSCAN, and the reported time is the average time of 5 runs.

The experiment was conducted on a PC running Windows 10 Enterprise with 8 GB of RAM and an Intel® Core™ i5-10400 CPU, utilizing an HDD hard disk. The experiments demonstrated that the model could handle big datasets, such as Machinery Fault\imbalance, which was 5.37 GB in size. The execution time is affected by the MinPoints parameter; it should be set low because the data is divided into many chunks, thereby reducing redundancy. The jump sample size parameter does not have a significant effect on the result. The model demonstrates excellent performance in selecting samples across datasets of varying sizes and densities.

5. Future Work

In future work, this research could be expanded to address the challenges of real-time streaming data. The current model focuses on static datasets, but real-time data requires continuous sampling and clustering, which would involve developing dynamic and adaptive techniques. Integrating the hybrid model into streaming frameworks like Apache Kafka or Apache Flink could allow for real-time analytics while managing large data flows.

Additionally, further investigation into automatic hyperparameter tuning for DBSCAN could improve its scalability and robustness across different datasets. Techniques such as Bayesian optimization or genetic algorithms could be explored to fine-tune DBSCAN's parameters in a more automated and efficient manner, reducing the need for manual intervention.

Another promising direction would be to apply the model in domain-specific areas such as healthcare, finance, and IoT, where big data is increasingly prevalent. In healthcare, for instance, this model could help analyze large patient datasets to identify patterns for disease prediction or treatment outcomes. The integration of advanced machine learning algorithms, such as deep learning or reinforcement learning, with this hybrid model could further enhance predictive analytics and anomaly detection, enabling more effective decision-making in high-dimensional and complex data environments.

References

[1] Deng, Dingsheng. "DBSCAN clustering algorithm based on density." 2020

[2] 7th international forum on electrical engineering and automation (IFEEA). IEEE, 2020.

[3] Warrens, Matthijs J., and Hanneke van der Hoef. "Understanding the adjusted rand index and other partition comparison indices based on counting object pairs." Journal of Classification 39.3 (2022): 487-509.

[4] Chacón, José E., and Ana I. Rastrojo. "Minimum adjusted Rand index for two clusterings of a given size." Advances in Data Analysis and Classification 17.1 (2023): 125-133.

[5] de Moura Ventorim, Igor, et al. "BIRCHSCAN: A sampling method for applying DBSCAN to large datasets." Expert Systems with Applications 184 (2021): 115518.

[6] Ros, Frédéric, and Serge Guillaume. "DENDIS: A new density-based sampling for clustering algorithm." Expert Systems with Applications 56 (2016): 349-359.

[7] Ros, Frédéric, and Serge Guillaume. "DIDES: a fast and effective sampling for clustering algorithm." Knowledge and information systems 50 (2017): 543-568.

[8] Zhu, Lu, et al. "Improvement of DBSCAN algorithm based on adaptive Eps parameter estimation." Proceedings of the 2018 international conference on algorithms, computing and artificial intelligence. 2018.

[9] Xianting, Qi, and Wang Pan. "A density-based clustering algorithm for high-dimensional data with feature selection." 2016 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII). IEEE, 2016.

[10] Alwosheel, Ahmad, Sander van Cranenburgh, and

Caspar G. Chorus. "Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis." Journal of choice modelling 28 (2018): 167-182.

[11] Silva, José, Bernardete Ribeiro, and Andrew H. Sung. "Finding the critical sampling of big datasets." Proceedings of the Computing Frontiers Conference. 2017.

[12] Luchi, Diego, Alexandre Loureiros Rodrigues, and Flávio Miguel Varejão. "Sampling approaches for applying DBSCAN to large datasets." Pattern Recognition Letters 117 (2019): 90-96.

[13] Berndt, Andrea E. "Sampling methods." Journal of Human Lactation 36.2 (2020): 224-226.

[14] Li, Mingyang, et al. "A method of two-stage clustering learning based on improved DBSCAN and density peak algorithm." Computer Communications 167 (2021): 75-84.

[15] Iliyasu, R., & Etikan, I. (2021). Comparison of quota sampling and stratified random sampling. Biom. Biostat. Int. J. Rev, 10(1), 24-27.

[16] Sharma, Gaganpreet. "Pros and cons of different sampling techniques." International journal of applied research 3, no. 7 (2017): 749-752.

[17] Stratton, Samuel J. "Population research: convenience sampling strategies." Prehospital and disaster Medicine 36, no. 4 (2021): 373-374.

[18] Berndt, Andrea E. "Sampling methods." Journal of Human Lactation 36, no. 2 (2020): 224-226.

[19] Mahmud, Mohammad Sultan, Joshua Zhexue Huang, Salman Salloum, Tamer Z. Emara, and Kuanishbay Sadatdiynov. "A survey of data partitioning and sampling methods to support big data analysis." Big Data Mining and Analytics 3, no. 2 (2020): 85-101.

[20] Pandey, Kamlesh Kumar, and Diwakar Shukla. "Stratified sampling-based data reduction and categorization model for big data mining." In Communication and Intelligent Systems: Proceedings of ICCIS 2019, pp. 107-122. Springer Singapore, 2020.

[21] Djouzi, Kheyreddine, Kadda Beghdad-Bey, and Abdenour Amamra. "A new adaptive sampling algorithm for big data classification." Journal of Computational Science 61 (2022): 101653.

[22] Hasanin, Tawfiq, Taghi M. Khoshgoftaar, Joffrey L. Leevy, and Richard A. Bauder. "Severely imbalanced big data challenges: investigating data sampling approaches." Journal of Big Data 6, no. 1 (2019): 1-25.

[23] Pandey, Kamlesh Kumar, and Diwakar Shukla. "Euclidean distance stratified random sampling based clustering model for big data mining." Computational and Mathematical Methods 3, no. 6 (2021): e1206