Towards Scalable Fraud Detection:The Role of Oversampling and Hybrid Techniques in Distirbuted Data Environment

Elakia Vijayalakshmi Mantharachalam
Department of Computer Science
University of Nottingham
Nottingham,United Kingdom

Dias Kuatbekov
School of Physics and Astronomy
University of Nottingham
Nottingham,United Kingdom

Rustam Guliyev
Department of Computer Science
University of Nottingham
Nottingham,United Kingdom

***Abstract*—The problem of extremely imbalanced datasets in large-scale datasets might severely impact the performace of fraud detection models. The Synthetic Minority Oversampling Technique (SMOTE) is widely considered to be an effective remedy for solving data imbalance scenarios, but the nearest-neighbor search, which lies at the core of the algorithm, presents a scalability challenge in distributed systems. This work presents a comparative analysis of distributed SMOTE implementations such as partition based local SMOTE, LSH-based approximated SMOTE and global SMOTE implementations based on their differences in wall-clock time, scalability, and classification accuracy. The results of the experiments show that partition-based local implementations of SMOTE offer a superior classification accuracy of fraudulent transactions whilst offering very fast execution times. This might provide valuable insights on trade-offs between accuracy and computational cost for fraudulent transaction detection systems.**

***Index Terms*—LSH, Global SMOTE, Approximate SMOTE, K-Means clustering, Bisecting K-Means Clustering, Fraud Detection**

I. Introduction

"An ounce of prevention is worth a pound of cure," principle that holds true even in machine learning, where addressing issues like class imbalance can considerably improve model results. In fraud detection problem, such as those presented by the Credit Card Fraud Detection dataset, the imbalance between two different classes: legitimate and fraudulent transactions, which possessed less than 0.2% of instances represented in this data. This dataset shows transactions that occurred in two days and contains 492 fraudulent transactions out of 284,807 total transactions. Each transaction is described by 11 PCA transformed numerical values and the transaction time. The imbalanced nature of the dataset may cause heavily biased classification algorithms, making then obsolete for detecting rare minority cases.

SMOTE is widely used to address this problem by creating synthetic samples for the minority class. Moreover, exact implementations of SMOTE depend heavily on calculating nearest-neighbor clustering, which becomes computationally heavy and inefficient in distributed environment. This key limitation has led to the development of different partition-based oversampling methods that aim to decrease computational load while preserving classification accuracy.

Partitioning-based SMOTE methods suggest a promising direction for scalable oversampling. Different approaches with techniques like K-means clustering and Bisecting K-Means clustering separate the minority class into various clusters, allowing synthetic samples to be integrated within local partitions. This improves efficiency and avoids grouping unrelated data points under one label. At the same time, Locality-Sensitive Hashing provides an approximate method to classify similar instances without performing whole-distance calculations, minimizing the performance burden in a distributed setup.

As research in distributed systems and data scaling continues to evolve, these partition-based techniques offer an innovative route for scalable and efficient fraud detection. Constant evaluations of these methods in combination with ensemble classifiers provide crucial insights into the trade-off between classification performance, scalability, and computational efficiency in big data environments.

Research Question (RQ): What is the impact of traditional distributed oversampling techniques in combination with ensemble classifiers on the performance, scalability, and computational efficiency of fraud detection models in big data environments?

II. Literature Review

Imbalanced data is the significant issue in fraud detection pipelines: legitimate transactions outnumbering fraudulent transactions, which cause classifiers uncertain to the rare events. SMOTE solving this problem by interpolating synthetic minority samples within each nearest points in clusters [2]. Dal Pozzolo etal. showed that resampling with cost-sensitive learning notably boosts AUC on real credit card streams, specifically the financial value of extremely high recall models[1].

SMOTE strongly relies on exact neighbor queries which gives performance bottleneck as datasets grow.However, naive-

based implementations on Apache Spark trigger costly shuffles, memory loads and limiting scalability[3].

To solve this challenge, researchers are starting to test approximate and partition-based types of SMOTE, which have same performance but require less computation. LSH-SMOTE changes full-scale distance checks to sensitive hashing which cutting neighbor search to subgroups of linear time while keeping F1-score steady[7].Approximate SMOTE, built naively on Spark, combines an approximate KNN index with broadcasting joins and reports near linear scaling to 128 workers and about 90% reductions in execution time with eligible AUC loss[3].

Another scope of studies is clustering the minority; K-means SMOTE synthesizes only within closest regions, removes noise and outperforms basic SMOTE on 71 public benchmarks [8]. Distributed extension that merging batches in K-means into Spark demonstrates good results which is enhancing recall and reducing overhead on datasets [9].

These studies illustrates some key insights related to this research: Approximate neighbor search and cluster local oversampling can deliver enormous speed ups for fraud detection in big data at the cost of only small accuracy degradation, making them potentially good foundation for ensemble-driven monitoring setups.

III. Methodology

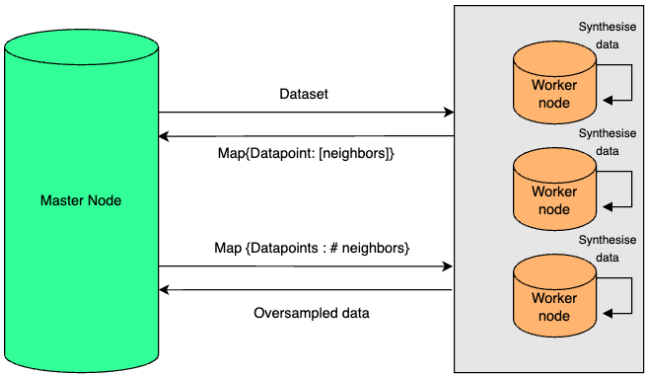

Fig. 1. Architecture

This section explains how we design our experiments to report how different SMOTE implementations behave in distributed systems. For every SMOTE implementation, we test their scalability and computational efficiency by measuring how algorithm wall-clock times change against dataset size. In addition to that, we choose a random forest model as a consistent classifier to measure the effect of those algorithms on classification performance. Our experiment aims to identify the approach that finds the perfect balance between classification accuracy and execution time.

A. Data Preprocessing

We first split the data by dedicating 80 of the data to the training split and the remaining 20 to the testing split. While splitting the data, we ensure that the same proportion of minority classes against majority classes is preserved in both training and testing sets. After data splitting, we standardize our features to have zero mean and unit variance. All of the data pre-processing and oversampling pipeline were performed using PySpark and MLlib APIs, and the feature vectors were converted into VectorUDF format. partitioned sampling.

B. Data Partitioning

Partitioning the data set into meaningful subgroups enables localized oversampling, which helps to preserve the internal structure of the data and prevent the generation of synthetic samples across distinct class boundaries. In the context of Spark-based distributed systems, partitioning also facilitates task parallelism and reduces inter-executor data movement. Three partitioning strategies were implemented using Spark-native tools to support distributed and scalable execution: KMeans Clustering, Bisecting KMeans, and Time-Based Partitioning. A brief comparison of those partitioning techniques can be found on the Table I

*1) **KMeans Clustering**:* KMeans is a centroid-based clustering algorithm that minimizes intra-cluster variance. It partitions the feature space into kk regions, each centered around a computed centroid. This method is effective for grouping data points that are spatially close, allowing oversampling to occur within homogeneous subgroups as shown in the Fig.2. [8], [9]

*a) **Justification**:* The algorithm is highly scalable and well-supported in Spark MLlib. Cluster-local oversampling reduces noise and prevents the generation of unrealistic synthetic instances.

*b) **Use Case**:* Best suited for high-dimensional datasets where spatial distance is a reliable indicator of class separability. Applicable when fraud patterns are consistent across feature space but not time-sensitive.

*c) **Distributed Implementation**:* Spark's MLlib KMeans was used with parallel training. The cluster ID was appended as a new column. Oversampling operations were performed on each cluster using mapPartitions, maintaining executor-locality. An overview of how K-Means work is provided in Algorithm 1.

Algorithm 1 KMeans Partitioning

Initialize k centroids randomly
Repeat until convergence:
  Assign each point to its nearest centroid
  Update each centroid as the mean of its cluster
Label each point with its cluster ID

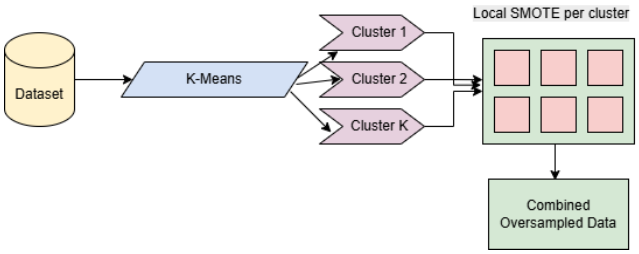

Fig. 2. K-Means Data Movement

*2) **Bisecting KMeans Clustering**:* Bisecting KMeans is a hierarchical variant that recursively splits clusters until the desired number of clusters is reached. It selects the cluster with the highest intra-cluster error and applies a two-way KMeans split like in the Fig.3 [10], [11]

*a) **Justification**:* Produces more balanced cluster sizes and performs better in datasets with skewed distributions or varying densities. Reduces sensitivity to initial centroid selection.

*b) **Use Case**:* Suitable when class clusters differ in size or shape, and when uniformity across partitions is needed for fair synthetic sampling.

*c) **Distributed Implementation**:* Implemented using Spark's BisectingKMeans API. Hierarchical splits were executed in parallel. After clustering, partitioned oversampling was applied via mapPartitions. The logic of Bisecting K-Means is presented in Algorithm 2

Algorithm 2 Bisecting KMeans Partitioning

Start with all data in one cluster
while number of clusters $< k$ **do**
  Select cluster with highest SSE (error)
  Split it into two using KMeans
end while
Assign final cluster labels to all points

Fig. 3. Bisecting K-Means Data Movement

*3) **Time-Based Partitioning**:* This method partitions the dataset into equal-sized time windows based on transaction timestamps. Spark SQL's ntile() was used to assign partition indices to each transaction after sorting like Fig.4. [12].

*a) **Justification**:* Fraud patterns often shift over time. Segmenting data temporally allows oversampling to be sensitive to recent fraud characteristics, improving classifier adaptability.

*b) **Use Case**:* Optimal for streaming or time-series datasets where fraud dynamics evolve and static feature clustering may miss temporal nuances.

*c) **Distributed Implementation**:* Executed with functions of the Spark SQL window. Each time segment was processed independently using executor-local filters and transformations.A pseudocode is depicted in Algorithm 3.

Algorithm 3 Time-Based Partitioning

Compute maximum time value: `max_time`
Set interval: `interval` $\leftarrow$ `max_time`$/p$
For each row: `time_bucket` $\leftarrow \lfloor$`Time`$/$`interval`$\rfloor$
If `time_bucket` $\geq p$, set `time_bucket` $\leftarrow p-1$
Return DataFrame with `time_bucket` column

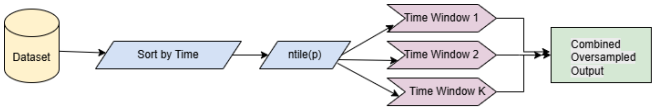

Fig. 4. Time-Based Partitioning Data Movement

TABLE I
COMPARISON OF PARTITIONING METHODS

Method	Partition Basis	Spark Native	Data Movement	Use Case
KMeans Clustering	Spatial feature density	MLlib	Low	Fraud clusters are spatially separable; suitable for general use
Bisecting KMeans	Hierarchical spatial	MLlib	Low	Clusters are uneven or hierarchical in structure
Time-Based Partition	Transaction timestamp	SQL API	Minimal	Temporal drift; useful for evolving fraud behaviors

C. Oversampling Techniques

Oversampling the minority class is a necessary strategy to enhance classifier sensitivity without distorting the decision boundary. This stage, however, becomes computationally intensive when scaled, particularly in distributed systems. Therefore, the oversampling component of the pipeline was designed with a focus on parallelism, partition locality, and Spark-native execution, minimizing memory bottlenecks and network-induced latencies. Four techniques were evaluated—Local SMOTE, SMOTE-ENN, Approximate SMOTE using LSH, and Local SMOTE with KMeans—each with unique design implications and implementation trade-offs.

*1) **Local SMOTE**:* SMOTE (Synthetic Minority Oversampling Technique) is a baseline algorithm that generates new synthetic samples by linear interpolation between a minority instance and one of its k-nearest neighbors. This method increases the decision region of the minority class by populating it with plausible instances. The sketch of how SMOTE works internally is presented in Algorithm 4. [16]

*a) **Technical Rationale**:* Provides foundational understanding of oversampling behavior and model response to balanced input. Useful as a benchmark against distributed alternatives.

*b) **Use Case**:* Appropriate for small-scale, local experimentation and comparative benchmarking during early model evaluation stages as shown in fig.2.

*c) **Implementation Constraints**:* Executed via imblearn.SMOTE on pandas DataFrames, requiring .toPandas() conversion and interrupting the distributed Spark pipeline. Computationally expensive and non-scalable beyond moderate data volumes.

Limitation Global memory usage and centralized computation. Not applicable in Spark-native environments without major redesign.

Algorithm 4 Local SMOTE

for each minority instance x **do**
  Identify k nearest neighbors of x
  for each neighbor x_n **do**
    $x_{\text{new}} \leftarrow x + \lambda \cdot (x_n - x)$
  end for
end for

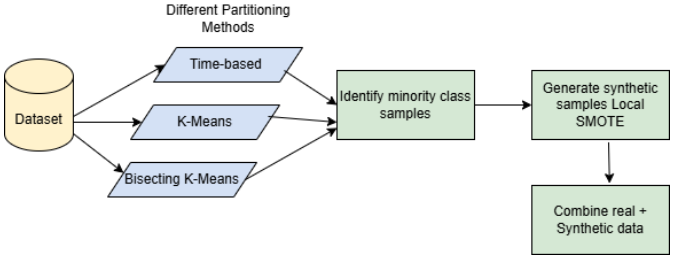

Fig. 5. Local SMOTE Data Movement

*2) **SMOTE-ENN**:* SMOTE-ENN combines SMOTE with Edited Nearest Neighbors, a noise-filtering technique that removes samples misaligned with their local neighborhood majority. This improves precision by eliminating overlapping or ambiguous instances post-synthesis as is shown in Algorithm 5. [13], [14]

*a) **Technical Rationale**:* Increases signal-to-noise ratio in the minority class by suppressing outliers and enhancing boundary sharpness. Ensures that synthetic samples do not contribute to class confusion.

*b) **Use Case**:* Valuable as a quality-focused benchmark to compare against scalable, distributed techniques. Suitable in small-batch experimentation or where minority class integrity is critical.

*c) **Implementation Constraints**:* Non-distributed. Executed via imblearn.combine. SMOTE-ENN, requiring full collection of the dataset in driver memory. Suitable only for benchmarking sample quality and algorithmic evaluation this can be understood using the fig.3.

Algorithm 5 SMOTE-ENN

Apply SMOTE to generate synthetic minority points
for each point x in the dataset **do**
  if label(x) $\neq$ majority label of k nearest neighbors **then**
    Discard x
  end if
end for

Limitation: Unscalable due to reliance on full neighbor graph. Spark-incompatible in its native form.

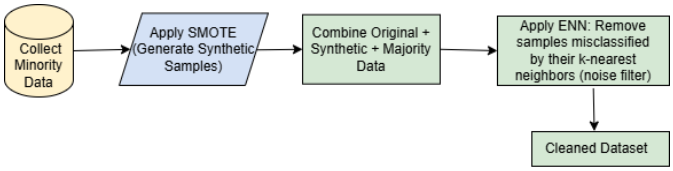

Fig. 6. Smote-ENN data movement

*3) **Approximate SMOTE with LSH**:* To address scalability, an approximate kNN-based SMOTE was developed using Locality Sensitive Hashing (LSH). The idea behind approximated SMOTE with LSH is presented in Algorithm 6. Spark MLlib's BucketedRandomProjectionLSH was employed to hash minority instances into buckets based on feature proximity, enabling efficient neighbor lookup without exhaustive pairwise comparisons. [15]

*a) **Technical Rationale**:* Reduces the computational complexity of neighbor search from $\mathcal{O}(n^2)$ to sublinear. Ensures scalability for high-dimensional and large-scale transactional data.

*b) **Use Case**:* Ideal for high-volume, latency-sensitive fraud detection pipelines that require scalable oversampling without sacrificing minority coverage.

*c) **Distributed Execution**:* LSH model fit on minority instances. Buckets generated and neighbor retrieval performed using approxNearestNeighbors(). Oversampling applied within each partition via mapPartitions where is explained in the fig.4.

Trade-offs: Loses exact neighbor precision in favor of runtime performance. Suitable when neighbor fidelity is less critical than execution efficiency.

Spark Benefits: Executes fully in parallel, respects partition boundaries, and avoids inter-node shuffling.

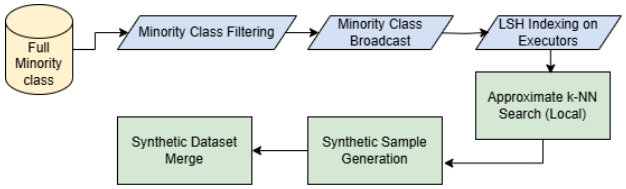

Fig. 7. Approx Smote with LSH data movement

Algorithm 6 Approximate SMOTE with LSH

Fit LSH on minority class samples
for each point x in the partition **do**
  Retrieve approximate neighbors of x using LSH
  for each neighbor x_n **do**
    Generate $x_{\text{new}} \leftarrow x + \lambda \cdot (x_n - x)$
  end for
end for

*4) **Global SMOTE**:* Global SMOTE presents an exact solution of SMOTE, meaning that it generates synthetic points based on true neighboring points. To ensure exactness, it distributes the minority class instances across all worker nodes. Each node then calculates the nearest neighbors for each

minority instance. In the driver node, we create a sampling map that indicates how many neighbors should be generated for each point; this map is broadcasted to all workers. This information is used to synthesize new points in parallel across the workers. The in-detail design decisions behind global SMOTE can be found in Algorithm 7. The exactness of this implementation means generation of higher quality synthetic samples. However, this comes at the cost of parallelization. It is important to note that this approach assumes we are dealing with extremely unbalanced datasets, where the total number of minority instances can fit within the worker nodes. [17]

Algorithm 7 Core Logic: Global SMOTE

Require: Dataset D, label identifier L, minority label c_{min}, neighbors k, ratio η
Ensure: Balanced Dataset D_{bal}

procedure GLOBALSMOTE(D, L, c_{min}, k, η)
  Partition D into $D_{minority}$ and $D_{majority}$
  Extract $V_{min} = \{(id_i, v_i)\}$ from $D_{minority}$
  Initialize KNN_{map}
  for each $v_i \in V_{min}$ **do**
    Find k nearest neighbors of v_i, excluding v_i
    Store mapping $id_i \mapsto N_i$ in KNN_{map}
  end for
  Define sampling plan $S_{plan} : id_i \mapsto n_{gen\_i}$
  Initialize $S_{syn} \leftarrow \emptyset$
  for each $(id_i, v_i) \in V_{min}$ **do**
    for $j = 1$ to n_{gen_i} **do**
      Pick random neighbor $id_{nn} \in N_i$
      Get v_{nn} for id_{nn}
      Compute $v_{syn} \leftarrow v_i + \alpha \cdot (v_{nn} - v_i)$
      Add (v_{syn}, c_{min}) to S_{syn}
    end for
  end for
  $D_{bal} \leftarrow D \cup S_{syn}$
  return D_{bal}
end procedure=0

IV. Experimental Study

All experiments were conducted using Apache Spark 3.5.0 running in a distributed environment on a Databricks cluster with the following specifications: 4 worker nodes, each equipped with 16 vCPUs and 64 GB RAM, and one driver node with 32 vCPUs and 128 GB RAM.

The Spark environment was configured to optimize performance by enabling data locality and minimizing shuffles. Specifically, `spark.sql.shuffle.partitions` was tuned to reduce unnecessary data movement.

Each SMOTE variant was evaluated across multiple data fractions (10%, 20%, 30%, 50%) to assess scalability. Execution time was measured using Spark's internal job metrics, and classification performance was evaluated using a consistent Random Forest classifier with 100 trees and a maximum depth of 10.

To ensure reproducibility, each experiment was repeated three times, and average values were reported for both execution time and performance metrics (PR AUC, ROC AUC, Accuracy). For non-distributed methods such as SMOTE-ENN, the operations were executed in a local environment using Pandas and `imbalanced-learn`, noting their limitations in scaling beyond memory-constrained workloads.

The experimentation set-up includes testing of the scalability and accuracy. To test first, we check how the wall-clock time of an algorithm changes as we increase the dataset.

V. Discussion and Results

The objective of this study was to examine the trade-offs between execution time, scalability, and classification performance of various SMOTE-based oversampling techniques implemented in Apache Spark. The experimental results provide several key insights into how these techniques behave in distributed environments.

TABLE II
Execution Time (s) Comparison for All Methods by Data Fraction (Lower is Better)

Method	Fraction			
	0.1	0.2	0.3	0.4
Label Aware Local SMOTE	4.45	4.11	4.56	7.01
TimeBased Local SMOTE	**2.85**	**2.60**	**2.74**	**4.54**
KMeans Local SMOTE	2.95	4.49	3.60	5.54
Bisecting K-Means Local SMOTE	3.24	3.95	3.62	5.12
Approx_SMOTE_LSH	6.91	7.01	7.68	7.16
Global SMOTE	6.52	6.69	7.88	6.11
Local SMOTE with ENN	558.27	589.46	529.73	513.72

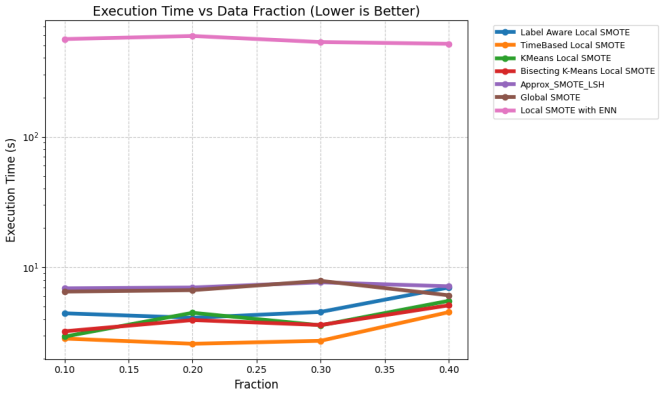

Fig. 8. Execution time VS Data Fraction

TABLE III
Performance Metrics for Various Methods and Combinations (Higher is Better)

Method / Combination	PR AUC	ROC AUC	Accuracy
Label Aware Local SMOTE	0.7273	0.9355	0.9993
Time Based Local SMOTE	0.7286	0.9437	0.9993
K-Means Local SMOTE	**0.8150**	0.9675	**0.9994**
Bisecting K-Means Local SMOTE	0.7971	0.9605	**0.9994**
SMOTE ENN	0.7367	0.9752	0.9952
LSH SMOTE	0.7791	0.9694	0.9993
Global_SMOTE	0.7541	**0.9728**	0.9957

The comparison of the execution time (in seconds) for different partitioning methods for various data fractions are presented in Table II. As one could expect, the local implementations of the SMOTE take the lead in execution time,

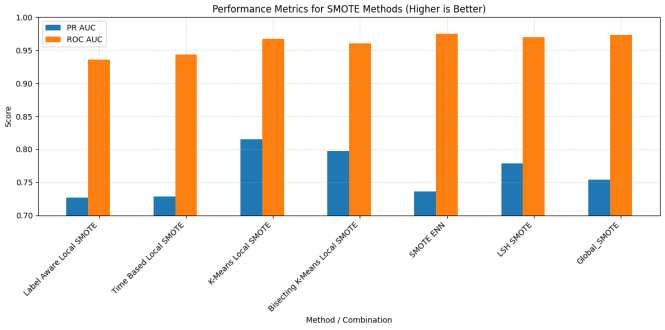

Fig. 9. Performance Metrics VS Various Methods

with time-based partitioning SMOTE being the fastest overall. [18]

Apart from, the comparison of classification accuracies and execution times of different SMOTE implementations are presented in Table III. Against our expectations, global implementation of SMOTE methods does not offer a significant boost on classification accuracy, with K-Means-based partitioning local SMOTE implementations taking the lead in PR AUC scores by a huge margin.

Partition-based techniques, specifically *KMeans Local SMOTE* and *Bisecting KMeans Local SMOTE*, demonstrated strong performance in both execution time and predictive accuracy. These methods leveraged Spark's parallel processing by using `mapPartitions` to preserve executor locality, thereby minimizing data shuffles and memory bottlenecks. Notably, KMeans Local SMOTE achieved the highest PR AUC (0.815) and ROC AUC (0.9675), while maintaining a low runtime, making it an optimal choice for scalable fraud detection. We suggest that this strong classification performance is due K-Means being able to capture high localized nature of the fraudulent transaction samples.. [19]

LSH SMOTE, which employs approximate nearest neighbors through Locality Sensitive Hashing, offered a practical trade-off by significantly reducing execution time while maintaining strong classification performance. Although slightly less precise in PR AUC compared to KMeans, it provided substantial improvements in runtime over Global SMOTE. [20]

In contrast, *Global SMOTE*, while delivering good ROC AUC scores, was the most computationally expensive due to full pairwise distance calculations and cross-node data movement. Similarly, *SMOTE-ENN*, though effective at noise reduction, suffered from poor scalability and high memory usage, making it unsuitable for large-scale Spark environments. [21]

These findings highlight the effectiveness of partition-aware strategies in distributed data environments. By minimizing communication overhead and enabling local synthetic sampling, partitioned methods scale better and retain high model fidelity. [22]

Several limitations must be acknowledged. The SMOTE-ENN and baseline SMOTE methods required `.toPandas()`, limiting their scalability. Additionally, the experiments were conducted on a single dataset, and generalizability to other domains requires further validation. Future work should explore adaptive partitioning, real-time streaming support, and broader dataset evaluation.

VI. Conclusion

This research explored the implementation and evaluation of various SMOTE-based oversampling methods within a distributed Spark framework for imbalanced fraud detection tasks. The results demonstrate that partition-based techniques, such as *KMeans Local SMOTE* and *Bisecting KMeans Local SMOTE*, achieve a favorable balance between execution time and classification performance, making them particularly well-suited for big data applications.

Among all methods tested, KMeans-based SMOTE emerged as the most efficient and accurate approach, outperforming other techniques in both PR AUC and runtime. Approximate methods like LSH SMOTE also showed promise in reducing execution latency without significant performance degradation.

In contrast, methods requiring full dataset access (Global SMOTE, SMOTE-ENN) were less scalable and incurred high computational overhead. These results reinforce the importance of distributed design, data locality, and algorithmic approximation in scalable machine learning pipelines.

Future directions include expanding the study to multi-source or real-time datasets, integrating with ensemble classifiers for robust fraud detection, and investigating adaptive or dynamic partitioning methods that can respond to evolving data distributions.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[8] Jin, X. and Han, J. (2011) 'K-means clustering', Encyclopedia of Machine Learning, pp. 563–564. doi:10.1007/978-0-387-30164-8 425.
[9] Giorgini, L.T., Bischoff, T. and Souza, A.N. (2025) KGMM: A K-means clustering approach to gaussian mixture modeling for score function estimation, arXiv.org. Available at: https://arxiv.org/abs/2503.18054 (Accessed: 06 May 2025).
[10] Ahmed, M., Seraj, R. and Islam, S.M.S. (2020) The K-Means Algorithm: A comprehensive survey and performance evaluation, MDPI. Available at: https://www.mdpi.com/2079-9292/9/8/1295 (Accessed: 06 May 2025).
[11] JMLR. Available at: https://www.jmlr.org/papers/volume24/18-080/18-080.pdf (Accessed: 06 May 2025).
[12] UNC. Available at: https://www.cs.unc.edu/ anderson/papers/rtss21c.pdf (Accessed: 06 May 2025).
[13] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, vol. 16, pp. 321–357.

[14] He, H. and Ma, Y. (2013). Imbalanced Learning: Foundations, Algorithms, and Applications. 1st ed. Wiley.
[15] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 604–613.
[16] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, pp. 321–357.
[17] He, H. and Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), pp. 1263–1284.
[18] M. Zaharia et al., "Spark: Cluster Computing with Working Sets," in *Proc. 2nd USENIX Conf. HotCloud*, 2010.
[19] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *FOCS*, 2006, pp. 459–468.
[20] A. Fernández et al., *Learning from Imbalanced Data Sets*, Springer, 2018.
[21] X. Y. Liu, J. Wu, and Z. H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Syst., Man, Cybern. B*, vol. 39, no. 2, pp. 539–550, 2009.
[22] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, 2018.