Scaling of time-series classification in Big Data

C. Lourido
School of Computer Science
The University of Nottingham

X. Yao
School of Computer Science
The University of Nottingham

Z. Guo
School of Computer Science
The University of Nottingham

D. Abreu
School of Computer Science
The University of Nottingham

***Abstract*—Time series data, characterized by observations over time, play a crucial role in various domains like finance and healthcare due to their potential to reveal trends, dependencies, and correlations. Handling the volume and complexity of big data while extracting timely insights is crucial across these sectors. In this study, we address the challenge of efficiently classifying large-scale time series data within a Big Data framework. We investigate the scalability of two traditional classification techniques—K-Nearest Neighbours (KNN) with Dynamic Time Warping (DTW) and Support Vector Machines (SVM) with DTW—within the context of Big Data. Additionally, we explore the use of PySpark, a distributed computing framework, to parallelize these computations across clusters of machines, enabling faster processing of large datasets alongside several pre-processing techniques such as PCA for dimensionality reduction inherent in time series data.**

To provide a benchmark for scalability, we also evaluate a naturally scalable model, Random Forest, which inherently handles large datasets efficiently. By comparing these approaches, we aim to understand the trade-offs between traditional methods with specialized distance metrics (like DTW) and ensemble learning techniques suited for parallel execution.

Our analysis involves observing experiments to measure the performance, scalability, and computational overhead of each approach. We discuss the practical advantages and limitations of these methods and suggest avenues for further improvement. Ultimately, this research contributes to enhancing the scalability and applicability of time series classification techniques in the context of Big Data analytics.

I. Introduction

Time series data consist of observations collected sequentially over time, making them valuable for modeling and predicting temporal behaviors in various applications. Traditional time series analysis techniques often face limitations when applied to big data due to computational constraints and the sheer size of the datasets [17]. As a result, there is a growing interest in developing scalable methods capable of efficiently handling large-scale time series data.

This research focuses on addressing the scalability of time series classification methods within the realm of Big Data analytics. Specifically, we explore two well-known approaches: K-Nearest Neighbors (KNN) and Support Vector Machines (SVM), both utilizing Dynamic Time Warping (DTW) as a distance metric. The KNN algorithm is chosen due to its simplicity and effectiveness [8] DTW is chosen for its ability to handle variable-length time series, accommodating variations in the timing and speed of underlying processes, making it suitable for real-world applications where data points may not align perfectly in time.

Soft margin Support Vector Machines(SVM) is used to investigate the classification of time series in the big data context due to its suitability for multiclass classification and optimal margin maximisationm, which can lead to better generalization and reduced overfitting compared to other classifiers [5]. It seeks to find a decision boundary, a linear model represented by a line in 2D or a hyperplane in higher dimensions, that maximizes the margin between the classes, ensuring that all data points are correctly classified but at the same time allows for some points to be on the wrong side of the decision boundary, introducing a margin of tolerance for classification errors. SVM has better performance in memory usage for it only needs to store the support vectors, SVM is more effective in high dimensional spaces which is the case with our data.

Building upon this foundation, Spark is integrated into our big data pre-processing, using its parallelized processing capabilities to streamline the handling of massive time-series datasets, ensuring optimal performance on large-scale datasets. In order to focus on the scalability and efficiency of the classification to accommodate larger-scale datasets, the Random Forest algorithm is introduced as a versatile and naturally scalable ensemble learning method, as a comparative benchmark for scalability. Random Forest inherently supports parallel processing and are known for its robustness against overfitting, making it an attractive choice for handling large volumes of time series data. Without the need for feature selection or dimensionality reduction techniques, Random Forest can be easily parallelized, for each decision tree in the forest can be trained independently [2], and predictions can be made concurrently, leading to significant improvements in processing time. Moreover, Random Forest algorithms can be efficiently implemented in Apache Spark, as models can be trained and deployed on massive datasets with millions of samples under Spark clusters [14], making them suitable for large-scale classification tasks, while there isn't a native implementation of SVM in MLlib.

The dataset used in this study is from the "Powering the Nation" initiative, focusing on electricity consumption behaviors in 251 households. Readings were recorded every two minutes over a month and required significant pre-processing.

This paper is structured as follows: in Section 2, we provide a comprehensive literature review on time series classification techniques and the utilization of Big Data technologies. Section 3 outlines the methodology, detailing the implementation of KNN, SVM, and Random Forest with DTW, along with the integration of PySpark for parallelization. Section 4 presents

our experiments conducted and their insights. Finally, in Section 5, we conclude with a discussion of findings, focusing on scalability metrics and model performance highlighting the strengths and limitations of each approach and proposing avenues for future research in enhancing time series classification scalability within Big Data environments.

II. Background

The primary objective of this research is to enhance the scalability of k-NN with DTW for time series classification in the context of big data. When considering the challenge of enabling the classic k-NN algorithm with Dynamic Time Warping (DTW) to scale up to big datasets containing lengthy time-series, several significant difficulties arise from a big data perspective. Firstly, one of the primary obstacles is the computational complexity inherent in DTW calculations. DTW involves determining the optimal alignment between two time-series, which becomes increasingly demanding as the dataset size grows and the length of the time-series increases [3]. The need to compute pairwise DTW distances for numerous lengthy time-series can quickly become prohibitive in terms of time and memory requirements [9]. Furthermore, managing the memory and storage demands of large datasets with extensive time-series is critical. Each time-series must be stored in memory, and with k-NN using DTW, the pairwise distances between these time-series must be computed and managed. Efficiently handling these memory and storage requirements becomes a major concern when dealing with big data which can be balanced as SVM has been proven to do so [16]. The technique of combining DTW with SVMs establishes a new SVM kernel known as a Gaussian DTW (GDTW) kernel [1], which has been tested and achieved results comparable to more complex models [12]. Algorithm optimization is another significant challenge. Traditional k-NN with DTW may not be inherently optimized for large-scale datasets. To scale effectively, exploration into optimization techniques such as soft margin support vector machine search methods or parallel processing is necessary. These approaches aim to reduce the number of pairwise DTW computations required while maintaining acceptable levels of accuracy. Additionally, managing the dimensionality and complexity of time-series data poses a significant hurdle. Time-series can be high-dimensional due to their length and complex patterns [15]. Therefore, effective feature extraction and selection are vital for improving the efficiency of k-NN with DTW in big data scenarios.

Employing parallel and distributed computing frameworks, such as Apache Spark, can also aid in tackling these challenges. These frameworks allow for the distribution of the computational workload across multiple nodes or processors, potentially accelerating the DTW computation [4] for large datasets and lengthy time-series. An existing approach by [7] involves the computation of k-NN distributed across Spark clusters using its map-reduce paradigm. During the map phase, different partitions or splits of the training dataset are processed concurrently, calculating the k-nearest neighbors for each partition. This parallel execution optimizes the computation of neighbors within the training data, capitalizing on Spark's inherent distributed computing capabilities. The results from the map phase are merged by multiple reducers, which collect the nearest neighbors identified across the partitions. This approach ensures efficient and scalable processing of k-nearest neighbors across the Spark cluster.

Lastly, achieving the right balance between scalability, performance, and accuracy is crucial. Sometimes, it may be necessary to trade-off a accuracy for scalability in big data settings by leveraging approximation techniques or heuristic approaches.

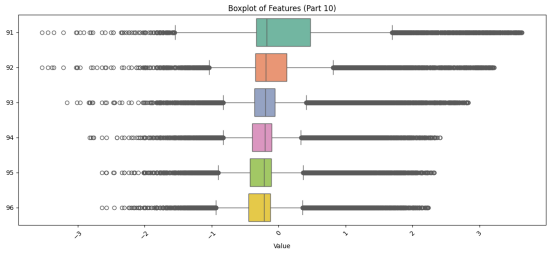
III. Methodology

The methodology begins with loading and preprocessing the dataset using PySpark. A comprehensive data preprocessing pipeline was made, aimed at cleaning, standardizing and preparing the time-series data. The process begins with the extraction of numeric values from the input data using utility functions. Any errors encountered during this stage are corrected to NaN (Not a Number) to facilitate robust data cleaning. Missing values are then forward-filled using the most recent non-null value that occurred before it in the same column or row, ensuring data completeness and consistency.

Following data cleaning, the characteristics of the cleaned data are recorded, including the number of time-series, their lengths, number of unique values, and any other relevant features. However, to address redundancy in the cleaned data, the starting line number of actual data in a text file is identified, and any comment lines are skipped using a series of utility functions.

Furthermore, the data is parsed to determine the number of columns, and a DataFrame is created. All feature columns are converted to numeric types to facilitate further analysis. Additional extraction functions are applied to each cell, ensuring uniform data representation throughout the dataset. Missing values in both the training and test datasets are filled with the mean of each column to ensure the dataset is complete. Lastly, the cleaned text and arff datasets are merged to ensure consistent column names and remove duplicated features from the arff dataset, streamlining the data integration process.

For our models, subsets of features are divided and presented in multiple subplots using box plots, aiding in the identification of outlier and understand the spread of data within each feature as seen in the example figure bellow.



Looking at the plot it can be seen that the different features have a similar spread and are centered around 0.

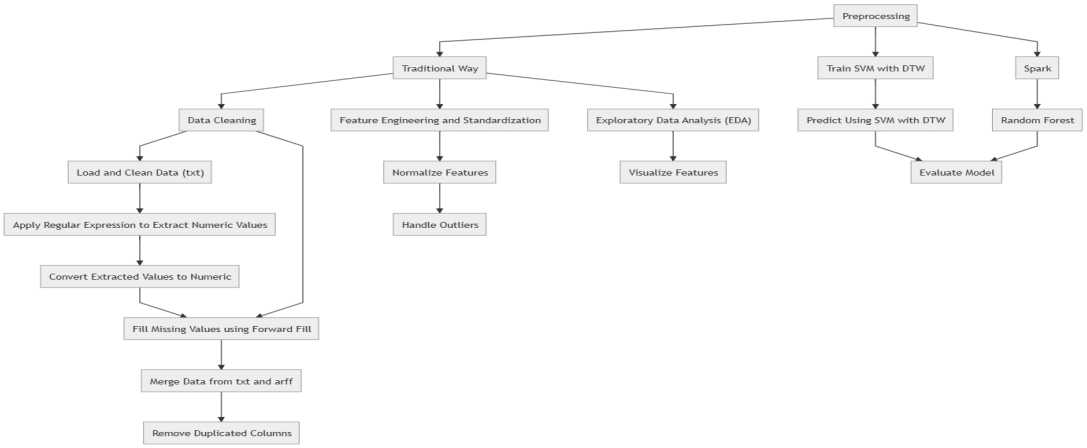
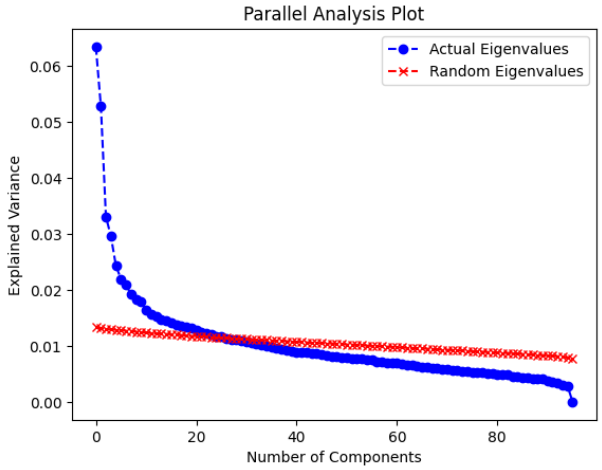


Fig. 1. Workflow diagram

Upon loading the dataset into Spark DataFrames and assembling the features into a vector, we extract the number of features where the covariance ratio is greater than the number random vectors, using this we can then perform PCA with the K value obtained and standardization. Principal Component Analysis (PCA) are performed to reduce dimensionality while preserving variance. The following figure shows the features determined to be kept from PCA.



The next step involves implementing the k-Nearest Neighbors (KNN) with Dynamic Time Warping (DTW) approach. DTW is utilized to calculate the distance between time series sequences. This custom distance function is defined within a Spark environment to compute DTW distances efficiently across all possible combinations of test and train instances. Next, the methodology includes the evaluation phase where the custom KNN with DTW predictions is compared against a baseline KNN model using scikit-learn. The evaluation metrics such as precision, recall, F1-score, and balanced accuracy are computed and compared between the two models to assess performance.

To validate the robustness and scalability of the implemented approach, a subset of the dataset is used for testing both models with different k parameters (using the features extracted from PCA with PA) to receive classification scores for comparison and most importantly time.

Algorithm 1 DTW Distance Calculation and Label Prediction

Calculate DTW distance for all test-train combinations
$cartesian\_df \leftarrow X_{test}.rdd.cartesian(X_{train}.rdd).map(\lambda x : dtw(x)).toDF([\text{"}distance\text{"}, \text{"}train\_id\text{"}, \text{"}test\_id\text{"}])$
Rank distances within each test partition
$ranked\_df \leftarrow cartesian\_df.partitionBy(\text{"}test\_id\text{"}).orderBy(\text{"}dis$
Filter to keep top k nearest neighbors
$k\_nearest\_neighbors\_df \leftarrow ranked\_df.filter(ranked\_df[\text{"}rank\text{"}] \leq k).drop(\text{"}rank\text{"})$
Add alias to train DataFrame
$train\_alias \leftarrow train.select(\text{"}train\_id\text{"}, \text{"}target\text{"}).withColumnRe$
Join with original DataFrame to retrieve neighbor targets
$predictions\_df \leftarrow k\_nearest\_neighbors\_df.join(train\_alias, on = k\_nearest\_neighbors\_df[\text{"}train\_id\text{"}] == train\_alias[\text{"}train\_alias.train\_id\text{"}], how = \text{"}inner\text{"})$
Group by test_id and collect neighbor targets
$predictions\_df \leftarrow predictions\_df.groupBy(\text{"}test\_id\text{"}).agg(F.colle$
Define UDF to determine majority class
$majority\_class\_udf \leftarrow F.udf(\lambda neighbors : max(set(neighbors), key = neighbors.count), F.StringType())$
Apply UDF to predict label for each test point
$predictions\_df \leftarrow predictions\_df.withColumn(\text{"}predicted\_label\text{"}$
Select test_id and predicted_label columns
$predictions\_df \leftarrow predictions\_df.select(\text{"}test\_id\text{"}, \text{"}predicted\_lab$

The approach minimizes data movement across workers by distributing data loading, feature assembly, and standardization operations efficiently within Spark DataFrames. Principal Component Analysis (PCA) for dimensionality reduction and custom KNN with Dynamic Time Warping (DTW) for classification are performed in a distributed manner, optimizing resource utilization and minimizing shuffle operations. Dynamic time warping (DTW) is a specialized metric used

for comparing time series data, offering a solution to the limitations posed by the Euclidean distance metric [13]. In time series classification, a nearest neighbor classifier utilizing the DTW metric is commonly regarded as a foundational baseline algorithm due to its effectiveness in capturing temporal dependencies and variability. We can speed up the DTW algorithm by only taking into account a small portion of the dataset's alignments as described by [11].

In the context of working with a dataset featuring 96 numeric features, the utilization of Principal Component Analysis (PCA) becomes inevitable and beneficial, PCA serves as a fundamental tool for dimensionality reduction and feature extraction. The primary necessity for PCA arises from the inherent challenge of managing high-dimensional data effectively. By applying PCA, we can transform the original set of correlated features into a new set of orthogonal (uncorrelated) principal components. These principal components capture the most significant variance in the dataset, allowing us to represent the data in a more compact form while retaining essential information. This reduction in dimensionality not only simplifies subsequent analyses but also helps mitigate issues associated with the curse of dimensionality, such as increased computational complexity and overfitting. Moreover, PCA facilitates the visualization and interpretation of data patterns by focusing on the most influential features, making it an indispensable technique for pre-processing and preparing datasets with numerous numeric attributes like the one at hand for our big data problem.

IV. Experiments

In our experiments, we employed Random Forest as an additional machine learning algorithm to assess its scalability and performance in comparison to k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM), which were selected as the primary focus. Random Forest was chosen due to its inherent ability to handle high-dimensional data and large datasets effectively. Unlike k-NN, which relies on storing all training instances and computing distances during prediction, and SVM, which can become computationally intensive with large feature spaces, Random Forest constructs an ensemble of decision trees using random subsets of features and data samples. This ensemble-based approach not only provides robustness against overfitting but also allows for parallelization and efficient processing of big data. By experimenting with Random Forest alongside k-NN and SVM on time-series datasets characterized by numerous features and significant data volume, we aimed to evaluate its suitability and scalability in the context of big data. This comparison helps clarify the trade-offs between the different machine learning algorithms concerning computational efficiency, scalability, and accuracy when dealing with our large-scale time-series datasets.

To investigate scalability, we designed an experiment that involved systematically increasing the size of the time series dataset while measuring the corresponding changes in processing time, resource utilization, and system performance. The objective was to assess how well our model and data processing techniques scaled with growing dataset sizes, simulating real-world scenarios where the volume of data can be substantial and dynamic.

We conducted experiments using varying dataset sizes, ranging from small-scale to large-scale (e.g., 1000 samples to full dataset), and monitored key performance indicators such as:

- **Execution Time:** Measured the time taken to process and analyze the time series data.
- **Memory Utilization:** Assessed the amount of memory required for computations and data storage.

To establish baselines for comparison, we considered the following approaches:

Majority Class Classifier: Predicting the majority class for all instances. Random Classifier: Assigning labels randomly based on the class distribution. Simple Distance-Based Classifier: Using a straightforward distance metric (e.g., Euclidean distance) for k-NN without DTW.

V. Results and Evaluation

We assessed the scalability of each algorithm by varying the dataset size and analyzing the computational resources required. The results demonstrated that k-NN with DTW, despite its accuracy in time-series classification, exhibited limitations in scalability as the dataset size increased. The computational demands associated with calculating pairwise DTW distances for large datasets became a bottleneck, leading to longer processing times and potentially exceeding memory constraints.

In contrast, SVM and Random Forest demonstrated better scalability characteristics. SVM, leveraging kernel methods for classification, efficiently managed larger datasets by transforming data into higher-dimensional space without explicitly computing pairwise distances. Similarly, Random Forest, with its ensemble learning approach, parallelized the construction of decision trees, making it well-suited for handling big data scenarios.

To further investigate scalability, we conducted experiments varying the dataset size from small to large scales (e.g., from 1,000 samples to full dataset) and monitored resource utilization (e.g., memory usage, processing time) for each algorithm. The findings revealed that SVM and Random Forest maintained stable performance across different dataset sizes, showcasing their ability to scale effectively. In contrast, k-NN with DTW exhibited increasing resource requirements and longer execution times as the dataset size grew, highlighting its limitations in managing big data efficiently.

We also observed consistent classification scores between the models across various k values. Interestingly, the key differentiating factor was the computational time required for different k values.

Notably, lower values of k resulted in significantly improved computational efficiency, with faster execution times observed compared to higher k values. This finding can be attributed to the nature of the KNN algorithm, where smaller values of k involve fewer computations to determine the nearest neighbors,

leading to faster predictions. This insight in classification scores across different k values reaffirms the robustness and effectiveness of the custom KNN with DTW approach for time series classification tasks. Despite varying k values, the models exhibited stable performance, suggesting that the choice of k had a more pronounced impact on computational efficiency rather than classification accuracy within the given subset of data. However, the classification scores obtained from all models were notably sub optimal, reflecting the complexity of the dataset and the inherent challenges in time series classification. Despite the lower performance, it's important to acknowledge that achieving high accuracy on such a complex dataset may not be feasible or the primary objective of the study. The confusion matrix analysis revealed significant miss classifications across different classes, emphasizing the complex nature of the dataset and the difficulty in capturing distinct patterns accurately.

Secondly, a notable observation was the computational efficiency of scikit-learn's KNN implementation with PySpark DataFrame compared to the custom KNN with DTW approach. The scikit-learn model exhibited significantly faster execution times, suggesting that PySpark may already optimize certain aspects of distributed computations, such as nearest neighbor search and distance calculations, more efficiently than the custom implementation. This highlights the potential for further customization and optimization of the proposed custom model to enhance its performance and scalability.

By utilizing Spark, specifically on a cluster of 10 nodes, we observed significant improvements in data processing speed and scalability compared to traditional single-node processing. Spark's ability to parallelize computations and leverage in-memory processing enabled efficient handling of the large dataset, particularly beneficial for tasks like feature extraction, model training, and prediction. In addition to Apache Spark, we explored other techniques aimed at enhancing scalability in our time-series analysis such as data partitioning where we divided the dataset into smaller partitions distributed across multiple nodes, allowing parallel processing and reducing the memory footprint of individual computations.

While each approach demonstrated strengths in scalability and performance, there were also areas where improvements and trade-offs were observed. The implementation of k-NN with DTW in PySpark leveraged distributed computing capabilities to parallelize DTW computations across multiple nodes or clusters, effectively scaling with increasing dataset sizes.Additionally, by distributing computations, our approach reduced the overall processing time and memory requirements, mitigating the computational complexity associated with pairwise DTW calculations. However, despite its scalability, k-NN with DTW suffers from reduced accuracy when dealing with high-dimensional and complex time-series data. The reliance on nearest neighbors for classification can be sensitive to noise and outliers, impacting overall predictive performance.

The SVM when implemented with PySpark's MLlib, demonstrated efficient scaling capabilities by leveraging kernel methods and distributed training across clusters while inherently handling the high-dimensional data well, making them suitable for the time series classification task with its complex feature spaces. The downside is that it is a computationally expensive model, especially with non-linear kernels, which impacts its scalability and real-time processing of large-scale time-series datasets. Furthermore, the SVM is sensitive to its parameter tuning, greatly varying the performance which required careful tuning and experimentation.

Method	TS Length	Precision	Recall	F1-score	Time(m)
KNN-DTW	6202	0.55	0.56	0.56	80.7
SVM-DTW	6968	0.35	0.52	0.41	43.1
RF	7711	0.47	0.47	0.43	12.9

A. Improvements

A potential future improvement for our study could involve exploring the use of more complex deep learning models, such as N-BEATS, for time series classification. N-BEATS is an MLP-based deep neural architecture with backward and forward residual links known for its effectiveness in capturing intricate patterns within data through hierarchical feature learning. By adapting N-BEATS to time series analysis and deploying it in a distributed computing environment with a mini-batch size optimized for scalability, like 64K, we could potentially achieve improved scalability and efficiency without compromising accuracy. Research has demonstrated the benefits of larger batch sizes in deep learning tasks [6]. However, successful implementation would require careful optimization of model training, hyperparameter tuning, and resource management to fully harness the scalability advantages of deep learning for time series classification within Big Data frameworks.

In order to test and achieve improved results with the k-Nearest Neighbors (k-NN) algorithm, we could leverage a technique known as Locality Sensitive Hashing (LSH) which was first introduced towards removing the curse of dimensionality [10]. LSH offers a powerful approach to approximate nearest neighbor search, particularly beneficial for datasets characterized by high dimensionality and extensive size such as ours. The motivation behind employing LSH lies in its ability to efficiently identify candidate nearest neighbors with a high probability of belonging to the same class, therefore accelerating the k-NN classification process.

The key advantage of LSH is its capability to handle large-scale datasets and lengthy time-series by reducing the computational burden associated with exact nearest neighbor searches. By employing hash functions that map similar data points to the same or nearby hash buckets, LSH efficiently identifies potential neighbors without exhaustively comparing every data point against all others. This approach not only saves computational resources but also mitigates the scalability challenges inherent in conventional k-NN methods.

In the context of time-series data, where the length and complexity of sequences can pose significant computational demands, LSH offers an effective strategy to expedite the identification of nearest neighbors for classification tasks. By

approximating nearest neighbor searches with high probability guarantees, LSH enables us to navigate the vast search space inherent in the big data setting while maintaining competitive accuracy levels. Ultimately, integrating LSH with k-NN provides a sensible solution to the big data time-series problem by enhancing scalability, reducing computational overhead, and facilitating efficient classification of lengthy time-series data.

To further enhance efficiency, strategies such as optimized data partitioning, caching, and broadcast variables could be explored to reduce unnecessary data shuffling and improve overall performance.

VI. Conclusion

Our experiments revealed important insights into the scalability of these methods. KNN with DTW, while effective in capturing temporal dependencies, exhibited limitations as dataset sizes increased. The computational demands associated with pairwise DTW distance calculations posed challenges in terms of processing time and memory utilization. In contrast, SVM and Random Forest demonstrated better scalability characteristics, leveraging kernel methods and ensemble learning techniques for efficient handling of high-dimensional data.

Notably, despite the advantages in scalability, it's essential to acknowledge the sub optimal accuracy scores obtained from all models, reflecting the complexity of the dataset and the inherent challenges in time series classification. The confusion matrix analysis revealed significant misclassifications across different classes, emphasizing the difficulty in capturing distinct patterns accurately within this context.

One notable finding was the computational efficiency of scikit-learn's KNN implementation with PySpark DataFrame compared to the custom KNN with DTW approach. The scikit-learn model exhibited significantly faster execution times, highlighting the optimization potential of distributed computing frameworks like PySpark.

Moving forward, there are several avenues for improvement. Exploring deep learning models like N-BEATS for time series classification within a distributed computing environment could offer enhanced scalability and efficiency. Additionally, leveraging techniques such as Locality Sensitive Hashing (LSH) with KNN could expedite nearest neighbor searches and mitigate scalability challenges associated with large datasets.

In conclusion, this research contributes to advancing the scalability and applicability of time series classification techniques in Big Data analytics. By understanding the trade-offs between traditional methods and ensemble learning techniques, we can better optimize model performance and scalability. Applying these techniques to diverse domains beyond electricity consumption, such as healthcare and manufacturing, could yield valuable insights into temporal behaviors and facilitate data-driven decision-making on a broader scale. By embracing innovation and collaboration across disciplines, we can unlock the full potential of Big Data for time series classification and predictive modeling.

References

[1] Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. Online handwriting recognition with support vector machines-a kernel approach. In *Proceedings eighth international workshop on frontiers in handwriting recognition*, pages 49–54. IEEE, 2002.

[2] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[3] Bowen Duan, Donghui Zhao, Junyou Yang, and Shuoyu Wang. A novel posture recognition based on time series supervised learning algorithm. In *2021 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, pages 394–398. IEEE, 2021.

[4] Zhipeng Gao, Ting Wang, Qian Wang, and Yang Yang. Execution time prediction for apache spark. In *Proceedings of the 2018 International Conference on Computing and Big Data*, pages 47–51, 2018.

[5] Babacar Gaye, Dezheng Zhang, and Aziguli Wulamu. Improvement of support vector machine algorithm in big data background. *Mathematical Problems in Engineering*, 2021:1–9, 2021.

[6] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[7] Jesus Maillo, Sergio Ramírez, Isaac Triguero, and Francisco Herrera. knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data. *Knowledge-Based Systems*, 117:3–15, 2017.

[8] Francisco Martínez, María Pilar Frías, Francisco Charte, and Antonio J Rivera. Time series forecasting with knn in r: the tsfknn package. *R J.*, 11(2):229, 2019.

[9] Abdullah Mueen and Eamonn Keogh. Extracting optimal performance from dynamic time warping. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2129–2130, 2016.

[10] Indyk Piotr. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of 30th Symposium on Theory of Computing, 1998*, pages 604–612, 1998.

[11] Lawrence R Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Tsinghua University Press, 1999.

[12] Juan J Rodrıguez and Ludmila I Kuncheva. Time series classification: Decision forests and svm on interval and dtw features. In *Proc. of the Workshop and Challenge on Time Series Classification*, 2007.

[13] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[14] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016.

[15] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

[16] Vo Quang Viet, Hoang Minh Thang, and Deok-Jai Choi. Balancing precision and battery drain in activity recognition on mobile phone. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 712–713. IEEE, 2012.

[17] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350–361, 2017.