Time Series Classification using KNN with DTW under Big Data Schema

Harnitha Suresh
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxhs12@exmail.nottingham.ac.uk

Emaan Bashir
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxeb5@exmail.nottingham.ac.uk

Neeraj Thosar
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxnt2@exmail.nottingham.ac.uk

Shruti Sundaram
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxss34@nottingham.ac.uk

Aishwarya Shahu
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxas30@exmail.nottingham.ac.uk

Kunal Choudhary
Department of Computer Science,
University of Nottingham,
Nottingham, UK
psxkc5@exmail.nottingham.ac.uk

***Abstract*—As the Internet of Things (IoT) expands, sensors in everyday objects are generating vast amounts of data, capturing diverse human activities like sitting, walking, and lying down. Accurately classifying these activities using Human Activity Recognition (HAR) datasets is increasingly vital. Yet, the immense volume and complexity of this data pose significant challenges, highlighting it as a key big data and time-series classification issue. HAR systems are crucial for applications from health monitoring to smart home automation. Traditional data analysis methods falter with the high dimensionality and temporal nature of sensor data. To ensure reliability and efficiency in real-world scenarios, innovative approaches are needed to manage the scale and dynamics of the data.**

The study explores two models for implementing the K-Nearest Neighbors (KNN) algorithm - the RDD-based map model using Euclidean and FastDTW distance measures, and the RDD- based broadcast model with FastDTW - for human activity classification using PySpark. These models were analyzed to assess their performance and operational behaviors in a shared cluster environment.

***Index Terms*—Time-Series Classification, Human Activity Recognition (HAR), k-Nearest Neighbors (KNN), Euclidean measure, Dynamic Time Wrapping (DTW), Big Data Techniques**

I. Introduction

The application of big data techniques to time-series data offers distinct advantages, particularly in scenarios where managing and processing large volumes of chronologically ordered data is crucial. Time-series data, characterized by its sequential nature, poses significant challenges in terms of volume, velocity, and variety, necessitating robust big data solutions. Big data technologies, such as distributed databases and real-time processing frameworks, are essential for handling the scale and complexity of time-series data, enabling organizations to leverage this data for enhanced decision-making and operational efficiency.

Human Activity Recognition (HAR) exemplifies time-series classification, utilizing sensors like accelerometers and gyroscopes in devices to record motion data continuously. These sensors generate large amounts of data, categorizing activities like walking, running, or sitting. The complexity of HAR classification arises from the scale and speed of data collection, coupled with variations from different sensors and user behaviors, making it a significant big data challenge that requires real-time processing and advanced analytics.

In addressing these challenges, the k-Nearest Neighbors (KNN) algorithm has proven to be particularly effective due to its non-parametric and instance-based nature. It utilizes the entire dataset for decision-making and can adapt various distance metrics to better align sequences that differ in timing or phase. However, the traditional Euclidean distance often used in KNN falls short in handling the dynamics of time- series data, where noise, outliers, and time-scaling issues are prevalent.

To overcome these limitations, Dynamic Time Warping (DTW) emerges as a superior alternative for time-series data within HAR. DTW allows for elastic comparisons between sequences, accommodating variations in activity pacing and duration. This flexibility makes DTW particularly effective in environments where activities may not align perfectly over time, providing a more accurate measure of similarity between sequences.

The research investigates two models for deploying the K-Nearest Neighbors (KNN) algorithm: the RDD-based map model, incorporating Euclidean and FastDTW distance measures, and the RDD-based broadcast model with FastDTW, applied to human activity classification using pySpark. These models underwent analysis to evaluate their performance and operational characteristics within a shared cluster setting.

II. Literature Survey

This review encapsulates a variety of research studies exploring time series analysis, big data frameworks, and human activity recognition technologies. The studies reviewed offer

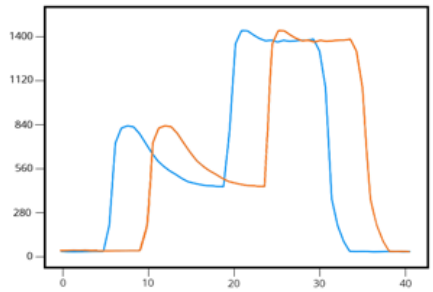

Fig. 1. Point-to-point mapping of Euclidean distance

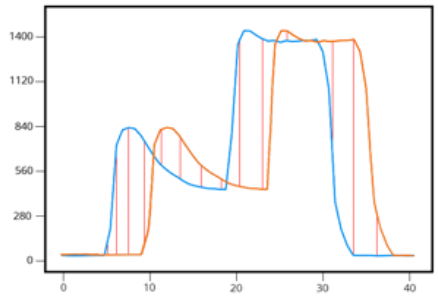

Fig. 2. Alignment of Time Series Data Using Linear Mapping

insights into methodological advancements and their applications in handling large datasets and performing sophisticated data analyses.

Dynamic Time Warping in Time Series: Mahato et al. (2018) discuss the application of k-Nearest Neighbors (k- NN) methods for time series data, highlighting Dynamic Time Warping (DTW) as a superior measure to Euclidean distance for capturing the intrinsic similarities in temporally displaced or morphed data series. Unlike the rigid point-to-point mapping of Euclidean distance, which may result in significant discrepancies as illustrated in Fig.1, DTW constructs a cost matrix to find the shortest path, minimizing the distance between series, as shown in Fig.3. This flexibility allows DTW to more accurately reflect trends in time series data, in contrast to the linear mapping depicted in Fig.2, which often proves ineffective. [1]

FastDTW: A Linear Time and Space Efficient DTW Algorithm: Salvador and Chan introduce FastDTW, an algorithm designed to approximate the optimal warp path between two time series with linear time and space complexity. Unlike traditional DTW, which has quadratic complexity, FastDTW uses a multilevel approach, progressively refining the warp path from coarser to finer resolutions. This method not only maintains high accuracy but also significantly reduces computational demands, allowing for practical application on large datasets. The empirical results demonstrate FastDTW's superior performance over existing DTW approximations, making

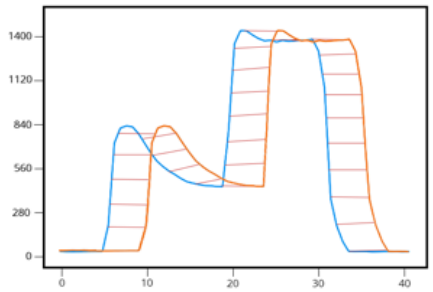

Fig. 3. Alignment of Time Series Data Using Dynamic Time Warping

it highly suitable for real-time applications in various fields such as speech recognition, robotics, and health monitoring. [2]

Comparative Analysis of Euclid and DTW Measures: This study presents a comparison between Euclidean and DTW metrics in discovering motifs within time series data. The findings indicate that despite the higher computational cost, DTW provides more accurate motif detection, which is essential for applications requiring precise pattern recognition, such as anomaly detection in sensor networks or market trend analysis. [3]

Need for Time Series Data Mining Benchmarks: A sur- vey highlighted by KDD '02 emphasizes the importance of benchmarks in time series data mining. It suggests that new approaches like the $O(n^2)$ DFT algorithm could potentially challenge existing methodologies like the radix-2 algorithm in effectiveness, pushing for further exploration in time series data mining benchmarks. [4]

Smartphone Data Analysis for Health Monitoring: Concone et al. (2017) describe the use of smartphone sensors in recognizing human activities, which is pivotal in healthcare applications like monitoring Parkinson's disease symptoms. The paper discusses various machine learning models to enhance the accuracy and efficiency of activity recognition. [5]

Systematic Review of Smartphone-Based HAR Methods: Straczkiewicz et al. (2021) conduct a systematic review of methodologies for human activity recognition using smartphones. The review covers data acquisition, preprocessing, feature extraction, and the challenges posed by sensor placement and orientation, such as variability in data due to different wearing positions, which complicates the transformation of gathered data into meaningful and interpretable outputs. [6]

Comparing the efficiency of Euclidean and DTW measures in Brute-force and MK algorithms for motif discovery in time series: DTW incurs higher costs and requires more time than Euclidean measures but identifies superior motifs. Overall, motifs discovered using DTW outperform those found with Euclidean measurements. [7]

III. Proposed Methodology

This study seeks to augment the efficiency and performance of the K-Nearest Neighbors (KNN) algorithm by incorporating Dynamic Time Warping (DTW) as a distance measure using FastDTW library for classifying time series data, each series consisting of 561 points, labeled by human activities. The enhancement lever- ages PySpark's capabilities for parallel processing, database distribution, and handling of big data. Considering the com- putationally intensive nature of DTW and the lazy learning characteristic of KNN, optimization of this combination is crucial. Previous implementations using the Scikit-learn library on a system equipped with 64GB RAM and an AMD EPYC 7763 64-Core Processor required over half a day for execution. Consequently, the objective is to develop an advanced KNN model that utilizes big data methodologies to significantly reduce processing time.

A. Data Pre-processing

The initial raw data, stored as text files containing comma-separated floating-point values, undergoes a series of transformations to prepare for analysis:

- **Data Transformation:** The raw data rows are parsed, with each value extracted and converted to a floating-point format. These values are then organized into structured columns, where each row corresponds to a time series comprised of 561 features.
- **Implementation Details:** These preprocessing steps are executed using PySpark SQL functions applied to DataFrames, augmented by User-Defined Functions (UDFs) for custom transformations. This method harnesses PySpark SQL's advantages, including its distributed processing capabilities and scalability, which are particularly beneficial for large datasets. Furthermore, PySpark SQL integrates seamlessly with the broader Big Data ecosystem, supports SQL queries, and provides fault tolerance.
- **Feature Vectorization and Assembly:** Data vectorization is performed using tools like Vectorizer and Assembler. This process converts data into a vector for- mat suitable for machine learning, enhancing algorithm efficiency and reducing memory usage. The Assembler function consolidates multiple data columns into a single feature vector, simplifying the data structure.
- **Label Integration:** Each time series is assigned a corresponding class label, facilitating subsequent supervised learning tasks.

These preprocessing efforts are critical for optimizing the performance of analytical models, ensuring that the data is in an appropriate format for complex computational tasks within distributed environments.

B. Models

1) **Implementation Using PySpark RDDs for KNN Classification (MODEL 1)**

The first approach involves the implementation of a K-Nearest Neighbors (KNN) model utilizing Resilient Distributed Datasets (RDDs) in PySpark for handling partitioned train and test data efficiently. The model employs advanced techniques to calculate nearest neighbors in a distributed environment, optimizing both the computation and memory usage.

The K-Nearest Neighbors (KNN) algorithm is a simple but effective machine learning method for classification and regression. In big data contexts, traditional KNN faces challenges due to the need to compute distances between all test and training instances. This model explores an optimized approach utilizing PySpark's RDDs and the heapq library, which offers an efficient binary heap implementation.

The model, designated as 'calculateKnnEuclidean', incorporates several parameters essential for its configuration and execution:

- *Parameter Definition:* The function takes an integer k representing the number of nearest neighbors to identify. It also configures 'train partitions' and 'test partitions' for distributing the data across multiple nodes to leverage parallel processing capabilities.
- *Data Preparation:* Initially, feature DataFrames for training and testing are converted into RDDs. These RDDs are then strategically repartitioned to balance the load across the cluster.
- *Distance Calculation:* A Cartesian product of train and test RDDs is formed to facilitate the pairwise distance calculation necessary for the KNN algorithm. The mapPartitions function is employed to apply a defined distance function across each pair of test and train features, computing the Euclidean distances.
- *Aggregation and Sorting:* The groupByKey function groups distances by test points, followed by sorting these distances from nearest to farthest using the mapValues function integrated with heapq. This ensures efficient sorting with a complexity of O(log n), optimizing memory usage by performing in-place operations.
- *Distance Functions:* Two distance functions are defined for flexibility in handling different types of data. *knnEuclidean* function calculates the Euclidean distance between test and train vectors within each partition and employs *heapq* to sort and return the k nearest distances. *knnDTW* utilizes the *fastDtw* Python package to compute Dynamic Time Warping (DTW) distances, providing an alternative for time-series data, with distances sorted similarly using *heapq*.

The implementation of KNN with PySpark RDDs and heapq sorting provides a robust solution for managing large datasets in distributed environments. This approach reduces computational overhead and enhances perfor-

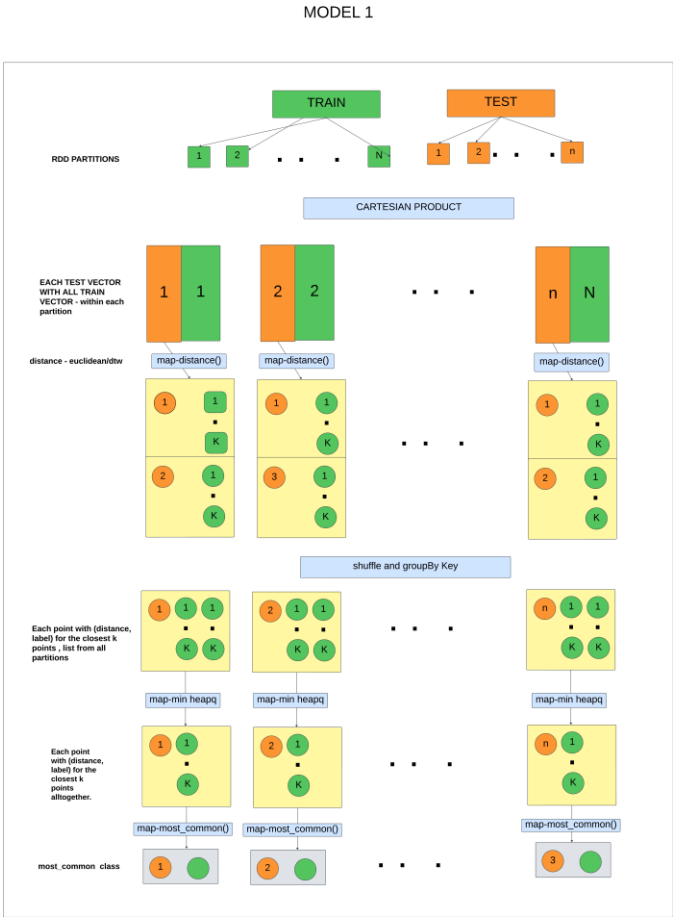

Fig. 4. Model 1: Implementation Using PySpark RDDs for KNN Classification

mance, making it suitable for scalable and real-time machine learning applications, ensuring essential scalability and efficiency for big data challenges.

Fig. 4 shows the visual representation of this model.

2) Implementation Using MapReduce and Broadcast Variables for K-Nearest Neighbors with Dynamic Time Warping (MODEL 2)

This model implements the K-Nearest Neighbors (KNN) algorithm enhanced with the Dynamic Time Warping (DTW) distance measure, designated as KNNwithDTW. The key objective is to efficiently process test data in a distributed system using MapReduce paradigms, coupled with the strategic use of broadcast variables for the training data.
Technical Implementation involves the following steps:

- *Data Repartitioning and Distribution:* The test dataset features are converted into a Resilient Distributed Dataset (RDD) and repartitioned to balance the load across the cluster. This step ensures that the computation related to distance measurement is evenly distributed, optimizing resource utilization.

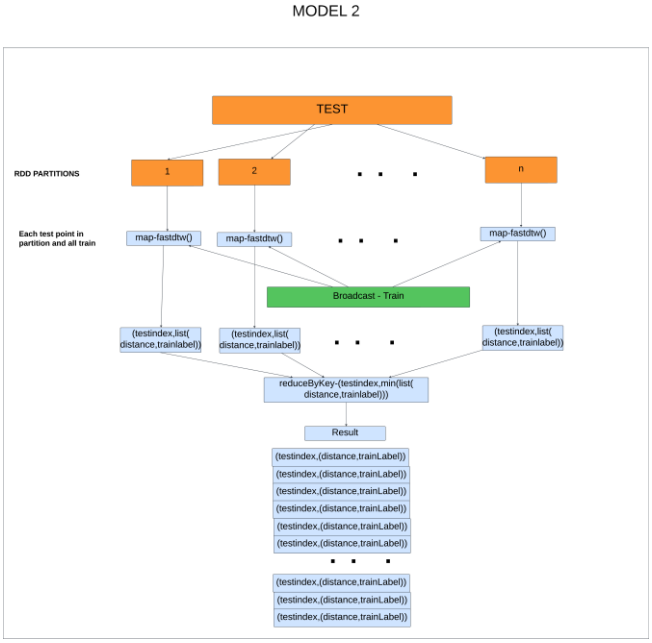

Fig. 5. Model 2: Implementation Using MapReduce and Broadcast Variables for K-Nearest Neighbors with Dynamic Time Warping

- *Broadcast Variable Utilization:* Training dataset features are encapsulated within a broadcast variable, which significantly reduces data transfer across the network. Broadcast variables cache data on each node rather than transferring it with every task, thus improving memory usage and the efficiency of joins in the distributed computing environment.
- *KNNwithDTW Computation:* It involves a Distance Calculation and Label Prediction. A flatMap transformation applies a KNN function to each partition of the test dataset RDD. For every test feature within the partition, the KNN function computes the 'fastdtw' distance to all features in the broadcasted training dataset. Then, the distances, along with the indices and labels of the training features, are emitted from the KNN function. Subsequently, the reduceByKey function aggregates these outputs to associate each test feature with the nearest training feature based on the calculated DTW distance, effectively determining the predicted label for each test instance.

The integration of 'fastdtw' with KNN enhances time-series distance computations, reducing overhead while maintaining accuracy. Using broadcast variables and RDDs leverages Spark's in-memory computing for efficient large dataset management. This strategy effectively deploys scalable machine learning algorithms in distributed settings, ensuring high performance and accurate time-series classification.

Fig. 5 shows the visual representation of this model.

IV. Experimental Setup

A. Dataset Description

The dataset comprises studies conducted with 30 participants aged between 19 and 48, engaging in six distinct activities: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, and LAYING. Each participant wore a Samsung Galaxy S II around their waist, which used its built-in accelerometer and gyroscope to record 3-axial linear acceleration and 3-axial angular velocity at a frequency of 50Hz. The data was visually recorded to facilitate manual annotation.

Data features include:

- Triaxial acceleration from the accelerometer along with estimated body acceleration.
- Triaxial angular velocity measured by the gyroscope.
- A vector of 561 features derived from time and frequency domain variables.
- The dataset contains 7352 train features and labels and 2947 test features and labels.

B. Metrics

To evaluate the performance of both models, several key metrics are considered such as:

1) Accuracy
   This measure evaluates the proportion of total predictions that were correctly classified by comparing the predicted labels against the true labels in the test dataset.
2) Execution Time
   The total computational time required to execute each model from initial data processing to final output is critically assessed.
3) Scalability (size-up)
   The ability of the models to handle increasing volumes of data without significant degradation in performance is analyzed.
4) Resource Utilization (Scale-up)
   Evaluating how effectively each implementation utilizes computational resources, such as CPU and memory, underlines the cost-efficiency of the models.
5) Fault Tolerance
   Given the distributed nature of the computations, the resilience of the models to node failures or network issues is also considered.

   PySpark RDDs are expected to enhance data handling by improving data partitioning and in-memory process- ing, potentially reducing execution times and improving scalability. In contrast, using MapReduce and broadcast variables aims to optimize network data transfers, potentially lowering overhead and enhancing computation speed and efficiency. This comparative assessment helps identify the optimal approach for KNN classification in distributed settings, contributing to the optimization of big data workflows in challenging environments.

k value	Time
k = 5	26 mins
k = 10	14 mins
k = 20	20 mins
k = 50	30 mins
k = 70	18 mins

TABLE I
Computational Time Results for Model 1 (Euclidean measure)

k value	Time
k = 5	18 mins
k = 10	17 mins
k = 20	18 mins
k = 50	16 mins
k = 70	16 mins

TABLE II
Computational Time Results for Model 2 (Fast DTW)

C. Validation Procedure

A manual for-loop is used for hyper-parameter fine-tuning of k by selecting k values based on the square root of the number of training samples, 85. Accuracy and model run time is captured.

V. Results

In the conducted experiments, we evaluated the performance of our models across a range of k values, specifically selecting k=5, 10, 20, 50, and 70 to evaluate their impact on completion time. TABLE I and II showcase the results we got for Model 1 and Model 2 respectively.

Our experiments showed that using the Fast Dynamic Time Warping (Fast DTW) technique within the K-Nearest Neighbors (KNN) framework consistently outperformed the traditional Euclidean distance approach in accuracy. KNN with Fast DTW (Model 1) demonstrated superior performance across all test instances, proving highly effective for time-series classification tasks like Human Activity Recognition (HAR).

The computational times for KNN models using Euclidean distance and Fast DTW were similar, ranging from 16 to 26 minutes as detailed in TABLE I and II. Despite DTW's reputation for being computationally intensive, the use of Fast DTW has become more efficient due to advances in parallel and distributed big data methodologies, making it viable for large-scale applications.

In terms of computational speed, Model 1 using FastDTW averaged 16 minutes, while Model 2, also using Fast DTW, completed in about 19 seconds. These results were influenced by the conditions of the shared cluster during testing—Model 1 was run under heavy load, and Model 2 under minimal traffic. The significant reduction in time for Model 2 is attributed to efficient broadcasting techniques, enhancing both time complexity and scalability, making it suitable for real- time applications requiring fast and reliable processing.

In assessing two KNN-based models for big data scenarios, our analysis compared Model 1, a PySpark implementation with k=50, against Model 2. Model 1 showed superior performance, especially when compared to non-PySpark KNN implementations using Dynamic Time Warping (DTW).

A. Accuracy and Complexity

The accuracy of Model 2, using a k-value of 5, was found to be lower than that of Model 1 with k=50. After extensive testing, it was determined that Model 1 with k=50 was the optimal choice due to several factors:

- The difference in accuracy between k=50 and k=70 was minimal, suggesting that a lower complexity, achieved with k=50, does not compromise accuracy significantly.
- Model 1 achieved a runtime of only 16 minutes, which is substantially faster compared to the more than half-day runtime required by the non-PySpark KNN implementation.

B. Final Testing and Scalability

During final testing, Model 2 with k=50 reached an 80% accuracy, showcasing its robustness with practical datasets. Scalability tests on Model 1 (k=50) showed it completed in 34 seconds with 10 data points and 16 minutes with 2947 test points, demonstrating its ability to scale effectively with increasing data size while maintaining reasonable computation times.

C. Fault Tolerance

The architecture of Model 1 enhances fault tolerance by distributing computations across multiple partitions, where each operates independently to prevent single-point failures. If a partition's job fails, the driver nodes efficiently reallocate tasks to other workers. This design ensures system integrity and performance even if a worker node fails, allowing for continuous operation with minimal downtime.

In summary, the results underscore the effectiveness of Model 1 with k=50 in terms of accuracy, computational efficiency, scalability, and fault tolerance, making it a superior choice for big data applications over the tested alternatives.

VI. Discussion

In the comparative analysis of two models—RDD-based and DataFrame SQL-based implementations of the K-Nearest Neighbors algorithm—the study finds that RDDs perform better in crowded cluster environments, where the DataFrame SQL approach sometimes faces bottlenecks and stalls. This underscores RDDs' efficiency in data distribution and management under high concurrency, typical in big data settings.

It was observed that the distance metrics, commonly utilized in these models, is particularly sensitive to noise. This sensitivity is problematic given the high-dimensional nature of time-series data, which inherently possesses high feature correlation and significant noise levels.

This may impede the effectiveness of the KNN algorithm under standard configurations. As a result, there is a compelling case for exploring alternative distance measures that could offer resilience against the distortive effects of noise and the intricate relationships found in time-series datasets.

VII. Conclusion & Future Work

In this study, we successfully explored various parallelization strategies for implementing the K-Nearest Neighbors algorithm using Apache Spark. The optimized approach leveraged PySpark's resilient distributed datasets (RDDs) and the efficient binary heap implementation provided by the heapq library, demonstrating significant enhancements in handling large-scale data efficiently.

Looking ahead, there are several avenues to further enhance the performance and functionality of our models. Future research exploring non-Euclidean metrics, which are less sensitive to noise, could lead to more sophisticated and adaptable machine learning models better suited for complex, large-scale time-series data analysis.

Since we focused mainly on parallelization approaches and Spark implementation, future efforts may focus on integrating feature selection techniques to refine the model's accuracy.

Structuring the implementation into more coherent pipelines and transformers can streamline the processing workflow, and improve scalability and ease of integration with other data processing operations.

Experimenting with different k values using the broadcast method presents an opportunity to optimize the accuracy of Model 2. Adjusting k values dynamically based on the dataset characteristics could lead to more precise predictions.

To address more complex time-series challenges, implementing advanced algorithms such as Time Warped Edit Distance (TWE) and Minimum Snap Matching (MSM) could provide more nuanced handling of temporal distortions and variations in data sequences.

References

[1] Mahato, Vivek, Martin O'Reilly, and Pa´draig Cunningham. "A Comparison of k-NN Methods for Time Series Classification and Regression." AICS. 2018.

[2] Salvador, Stan, and Philip Chan. "Toward accurate dynamic time warping in linear time and space." Intelligent Data Analysis 11.5 (2007): 561-580.

[3] Yoshida, Sho, and Basabi Chakraborty. "A comparative study of similarity measures for time series classification." New Frontiers in Artificial Intelligence: JSAI-isAI 2015 Workshops, LENLS, JURISIN, AAA, HAT-MASH, TSDAA, ASD-HR, and SKL, Kanagawa, Japan, November 16-18, 2015, Revised Selected Papers. Springer International Publishing, 2017.

[4] Radovanovic´, Milosˇ. High-dimensional data representations and metrics for machine learning and data mining. Diss. University of Novi Sad (Serbia), 2011.

[5] Concone, Federico, et al. "Smartphone data analysis for human activity recognition." AI* IA 2017 Advances in Artificial Intelligence: XVIth International Conference of the Italian Association for Artificial Intelligence, Bari, Italy, November 14-17, 2017, Proceedings 16. Springer International Publishing, 2017.

[6] Straczkiewicz, Marcin, Peter James, and Jukka-Pekka Onnela. ”A systematic review of smartphone-based human activity recognition methods for health research.” NPJ Digital Medicine 4.1 (2021): 148.
[7] DU, NGUYEN TAI, and P. H. A. M. VAN CHUNG. ”COMPARING EFFICIENCY BETWEEN TWO MEASURES OF EUCLID AND DTW USED IN DISCOVERY MOTIF IN TIME SERIES.” Journal of Science and Technology-IUH 38.02 (2019).
[8] Hussain, Fiza Gulzar, Muhammad Wasim, and Ayesha Nasir. ”A Comparative Study of Parallel and Distributed Big Data programming models: Methodologies, Challenges and Future Directions.” Lahore Garrison University Research Journal of Computer Science and Information Technology 7.3 (2023).
[9] Zhang, Jianbo, Zhuangzhuang Ye, and Kai Zheng. ”A parallel computing approach to spatial neighboring analysis of large amounts of terrain data using spark.” Sensors 21.2 (2021): 365.
[10] Junaid, Muhammad, et al. ”Performance evaluation of data-driven intelligent algorithms for big data ecosystem.” Wireless Personal Communications 126.3 (2022): 2403-2423.
[11] Gupta, Yogesh Kumar, and Surbhi Kumari. ”A study of big data analytics using apache spark with Python and Scala.” 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS). IEEE, 2020.
[12] Salloum, Salman, et al. ”Big data analytics on Apache Spark.” International Journal of Data Science and Analytics 1 (2016): 145-164.
[13] Zadrozny, B., et al. ”Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining.” (2002): 694.
[14] Anguita, Davide, et al. ”Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine.” Ambient Assisted Living and Home Care: 4th International Workshop, IWAAL 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings 4. Springer Berlin Heidelberg, 2012.
[15] Xi, Xiaopeng, et al. ”Fast time series classification using numerosity reduction.” Proceedings of the 23rd international conference on Machine learning. 2006.
[16] Mitsa, Theophano. Temporal data mining. Chapman and Hall/CRC, 2010.