Automotive Symptom Detection Using Time Series Analysis

Classifying audio time series to predict engine health

Arshad Ansari
School of Computer Science
University of Nottingham
psxmm16@nottigham.ac.uk

Mrunal Meshram
School of Computer Science
University of Nottingham
psxmm17@nottigham.ac.uk

Sachin Fernando
School of Computer Science
University of Nottingham
psxsf5@nottingham.ac.uk

Shabbir Kutbuddin
School of Computer Science
University of Nottingham
psxsk22@nottingham.ac.uk

***Abstract*—The project focuses on scalable distance-based classification of time series data on Spark in Databricks. The FordA dataset is used to predict an engine's condition as "normal" or "abnormal". The dataset was used by researchers at Ford for studies like Deng et al. (2013) and Bagnall et al. (2017). The task involves classifying automobile engine diagnostics based on sensor signals to classify whether a given engine condition is 'normal' or 'abnormal' based on sensor readings over time. Our experiments focused on model performance and run times for a distributed implementation of DTW-KNN and Proximity forests while using the MLlib implementation as a baseline. The DTW-KNN approach uses a global implementation, which minimizes data movement using broadcast variables and partition-aware execution using RDDs. Through a series of experiments, model run time and classification accuracy are obtained for different partitions and data scales. Performance of size-up and partition impact experiments demonstrate that while DTW-KNN provides strong baseline accuracy, although Proximity Forest achieves better runtime performance and scalability on larger datasets. The study highlights major trade-offs between ease of interpretation and computational efficiency in distributed time series classification, which in turn offers practical insight into Spark-based experimental design for big data.**

***Index Terms*—DTW, KNN, time series, ensemble learning, Spark, RDDs, partitioning, Model evaluation.**

I. Introduction

Time series classification (TSC) is a core problem in machine learning with applications across domains such as predictive maintenance, anomaly detection, and signal processing. The current study focuses on audio-based time series datasets, which can become increasingly large and high-dimensional, traditional classification techniques, especially those relying on instance-based comparisons like Dynamic Time Warping (DTW) struggle to scale. Distributed computing platforms such as Apache Spark offer the opportunity to parallelize these algorithms across large datasets. However, efficient implementation requires thoughtful system design to minimize data movement and ensure effective parallel execution.

Our primary focus is on two Spark-based approaches:

DTW-KNN: A global implementation where the training set is broadcast to all Spark workers, and classification is performed in parallel across partitions using DTW as the distance metric.

Proximity Forest: A scalable ensemble method where each tree is trained using randomized distance-based splitting criteria across distributed partitions.

To contextualize and baseline these distance-based approaches, we also implement classification pipelines using Spark MLlib models specifically, Random Forest and Decision Trees on raw time series data. Although these models do not perform shape-based comparisons and are not optimized for temporal structure, they offer a useful baseline for comparing classification metrics such as accuracy, precision, and recall in a Spark-native, highly optimized setting. While runtime comparisons with DTW-based methods may not be meaningful due to fundamentally different computation patterns, the inclusion of MLlib models strengthens the evaluation by offering a broader perspective on model behavior.

We structure our investigation around the following research questions:

1. Can a global DTW-KNN classifier be efficiently parallelized in Spark while preserving accuracy? Accuracy and runtime performance of the DTW-KNN pipeline is explored for varying number of partitions. Different number of partitions are assessed by their runtime and predictive accuracy.

2. Does Proximity Forest offer a scalable and accurate alternative to DTW-KNN in Spark? A randomized, ensemble-based implementation of Proximity forest is studied for scalability and predictive accuracy.

3. How do distributed distance-based models compare with Spark MLlib classifiers on the same dataset? MLlib classifiers operate on raw time series vectors without temporal alignment is used to compare standard evaluation metrics to assess predictive strength.

This project contributes a fully distributed experimental pipeline, executed on Databricks using PySpark, with detailed runtime and metric-based evaluations. We emphasize Spark-specific concerns such as data movement, partitioning, and size-up testing to align with the distributed systems learning objectives of COMP4124.

II. Literature Review

Time series classification is a fundamental problem in pattern recognition, where the objective is to classify temporal sequences into discrete labels. The FordA dataset, drawn from the UCR Time Series Archive, is a known benchmark for univariate time series classification. We have a binary classification problem with 500-length sequences of audio sensor readings with high intra-class variation and label imbalance. This makes the FordA dataset ideal for testing the robustness and scalability of classification algorithms.

An accepted baseline model for time series classification is Dynamic Time Warping (DTW) combined with k-Nearest Neighbors (KNN). In a comprehensive benchmark, Bagnall et al. [1] found that 1-NN with DTW is a competitive benchmark across a diverse range of time series datasets. Despite the implementation's simplicity and interpretability, the DTW-KNN approach has poor runtime scalability since DTW has quadratic complexity. This limitation of DTW is circumvented by parallelizing the ensemble implementation for large-scale datasets - This project explores a broadcasted global model through a distributed setup.

To address the limitations of pairwise DTW at scale, Lucas et al. [2] introduced Proximity Forest, an ensemble of randomized decision trees where each node selects a dissimilarity measure and a reference instance for splitting. This approach enables scalable training and inference by avoiding exhaustive pairwise comparisons and allows diverse models to learn disjoint views of the data. It also supports parallel training which makes it attractive for distributed environments.

While distance-based methods are well-suited to time series, classical machine learning models such as Decision Trees (DT) and Random Forests (RF) have also been applied to time series when treated as fixed-length feature vectors. These models are highly scalable and are natively supported in Spark MLlib. These MLlib models operate on raw values and often ignore temporal structure, which limits their effectiveness on sequences with complex patterns. Nonetheless, they provide a strong baseline for classification metrics in distributed settings when feature engineering or transformations (e.g., TSFresh, Shapelets) are not applied.

This project conducts various experiments on the FordA dataset from the UCR Time Series Classification Archive. The FordA dataset consists of 3,601 training and 1,320 test rows, each one being a univariate time series of length 500. The objective is to classify engine sensor data into two categories: normal vs noisy engine noise.

Here, DTW-KNN and Proximity Forest are implemented as distance-based classifiers, while Random Forest and Decision Tree models are used as MLlib-based baselines to support comparative analysis of accuracy and efficiency in a Spark-native pipeline.

III. Methodology

This section outlines the complete experimental pipeline used to implement and evaluate distance-based classifiers for large-scale time series data using Apache Spark. Our methodology consists of six core stages: data ingestion, distance calculation, distributed data movement, model implementation, experimental design, and evaluation.

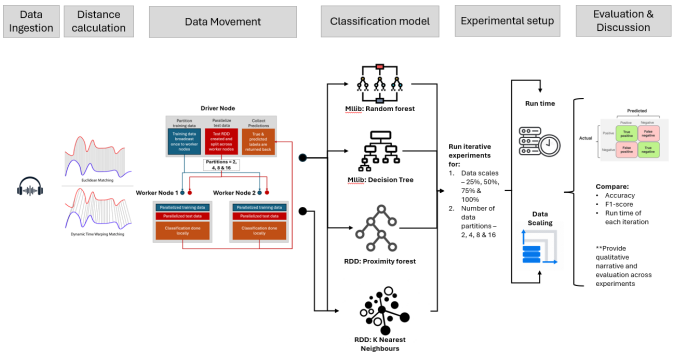

Fig. 1. Experiment Methodology

A. Data Ingestion and Preprocessing

We use the FordA dataset from the UCR Time Series Archive, consisting of univariate sequences of length 500. The dataset is pre-normalized and split into 3601 training and 1320 test instances. Both subsets are loaded from CSV files into Spark DataFrames and converted into RDDs of the format (label, series), where series is a list of float values representing the time series. No feature engineering or windowing is applied; classification is performed directly on the raw signal data to preserve the time-dependent structure.

B. Distance Calculation

Two distance metrics are used to evaluate similarity between time series:

- Dynamic Time Warping (DTW): Aligns sequences non-linearly to account for local time distortions. A custom Python-based DTW implementation is used to preserve transparency and control within Spark transformations.
- Euclidean Distance: Used as a baseline to highlight the advantage of DTW in misaligned sequences (not used in classifier, only visualized).

DTW is used within the DTW-KNN classifier and for randomized splitting in Proximity Forest trees.

C. Distributed Data Movement and Design

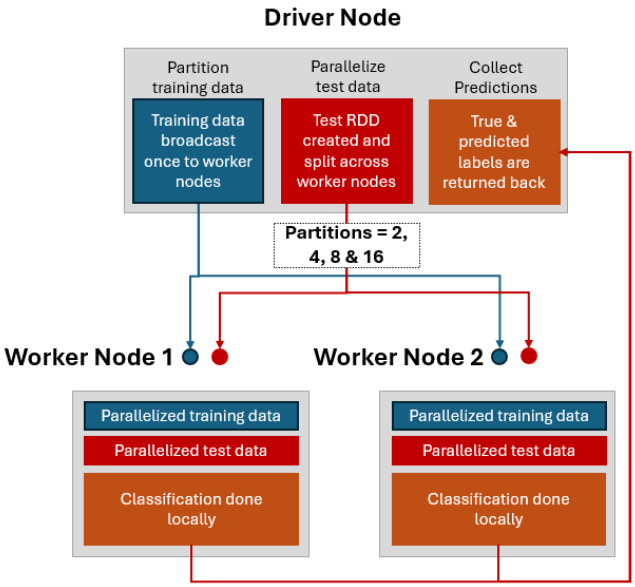

Fig. 2. Data movement

To minimize communication overhead and comply with Spark best practices, we design a global model architecture:

- Broadcast: The training RDD is broadcast to all worker nodes once.
- Test Partitioning: The test set is parallelized and repartitioned across executors (2, 4, 8, 16 partitions).
- Local Classification: Each worker classifies its local test subset using the full training set.

The use of broadcast variables ensures reduced data shuffling, and test predictions are returned to the driver node for aggregation and evaluation.

D. Classification Models

We implement four classifiers using different design philosophies:

- DTW-KNN (RDD-based):
  Implements global 1-NN or k-NN with DTW. Training data is broadcasted; predictions are computed in parallel using mapPartitions.
- Proximity Forest (RDD-based):
  An ensemble of decision trees trained with randomized distance-based splits. Each tree chooses a random distance metric and reference sample per node. Ensemble predictions are based on majority voting.
- Random Forest (MLlib-based):
  Spark-native implementation using decision trees trained on raw time series vectors. Offers a fast, scalable baseline without time-alignment.
- Decision Tree (MLlib-based):
  Single-tree classifier trained on raw vectors for interpretability comparison.

E. Experimental Design

We design our experiments to evaluate both performance and scalability:

- Data Size Scaling: We run all models on 25%, 50%, 75%, and 100% of the training data to test their ability to scale in terms of runtime and accuracy.
- Partition Count Variation: We evaluate each method using 2, 4, 8, and 16 Spark partitions to measure runtime sensitivity to data parallelism.
- Evaluation Metrics: We record accuracy and Runtime (measured via time library).

Each experimental run is repeated with fixed random seeds and consistent test sets to ensure reproducibility and fair comparison.

F. Output Collection and Comparison

All predictions and metrics are saved as CSV files within the Databricks workspace. Evaluation results are visualized using bar plots, confusion matrix heatmaps, and runtime vs. data size graphs. DTW-KNN and Proximity Forest are compared both against each other and against the MLlib models on classification performance.

This methodology ensures a rigorous evaluation of interpretability, scalability, and runtime behavior across different types of time series classifiers in a distributed setting.

IV. Experimental Study

The experimental study evaluates the accuracy, efficiency, and scalability of four time series classification approaches implemented on the FordA dataset. Our objective is to assess how distance-based methods such as DTW-KNN and Proximity Forest perform in a distributed computing environment, and how they compare against traditional machine learning classifiers and a sequential baseline implementation.

A. Overview of Experimental Setup

Experiments are conducted in the Databricks environment using Apache Spark with Python (PySpark). All models are evaluated on the full FordA test set, while varying the training data size and number of Spark partitions. To assess consistency and minimize randomness, fixed seeds and deterministic broadcasted training subsets are used across runs. The following classifiers are included:

- DTW-KNN (Spark RDD): Global model with DTW distance and broadcasted training data. Classification is parallelized using .map() on the test RDD.
- Proximity Forest (RDD-based): Randomized ensemble of distance-based decision trees. Each tree uses a different dissimilarity measure and operates independently on partitions.
- Random Forest (MLlib): Spark-native classifier applied to vectorized time series.
- Decision Tree (MLlib): Fast baseline using a single tree on raw time series vectors.

B. Fast DTW-KNN implementation

To provide a baseline for performance comparison, we implemented a sequential DTW-KNN classifier in pure Python using NumPy. This implementation:

- Uses k = 1, 3 and 5; with standard DTW (no window constraints)
- Compares each test instance to all training instances one by one
- Measures runtime on a single core without Spark

This reference model provides insight into the computational bottlenecks of DTW-KNN without any parallelism. Its results are compared to the distributed version to assess speedup, cost of parallelism, and accuracy preservation.

C. Experiment 1: Accuracy and Runtime Comparison

Each model is tested on the full test set using 100%, 75%, 50%, and 25% of the training data. For DTW-KNN and Proximity Forest, we additionally vary k and tree depth to measure sensitivity. The metrics recorded include:

- Classification accuracy
- Runtime using time() from job start to collection of predictions

Results show that Distributed DTW-KNN maintains poor accuracy even when parallelized and runtime for Sequential KNN is much higher compared to Distributed KNN. Proximity Forest achieves comparable accuracy with much lower inference time due to its ensemble and parallel nature.

D. Experiment 2: Partition Impact Study

To evaluate Spark execution efficiency, we fix the dataset size and vary the number of partitions (2, 4, 8, 16). This experiment tests:

- How workload distribution affects runtime
- Whether over-parallelization leads to performance degradation
- Accuracy consistency across partitioning levels

DTW-KNN benefits from increased partitions up to 8, after which communication overhead begins to reduce performance gains. Proximity Forest maintains stable performance across partition sizes.

E. Experiment 3: Scalability (Size-Up Behavior)

We perform size-up tests by fixing test data and measuring how runtime increases with larger training sets. We record time for training set sizes: 100%, 75%, 50%, and 25%.

The sequential DTW-KNN shows quadratic scaling ($O(n^2)$), while the Spark version shows linear growth in partitions up to a threshold. Proximity Forest demonstrates sublinear scaling due to independent tree training and early stopping heuristics.

F. Result Outputs and Visualization

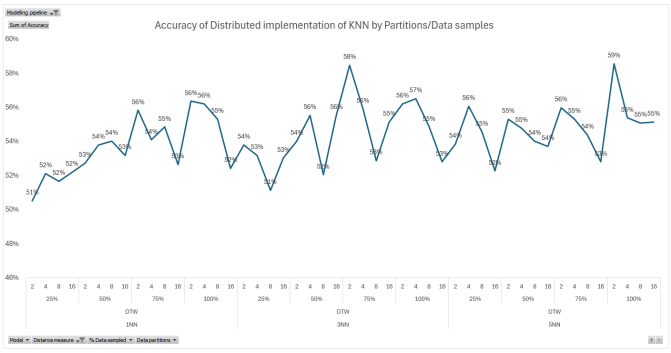

Fig. 3. Accuracy of Distributed KNN implementation

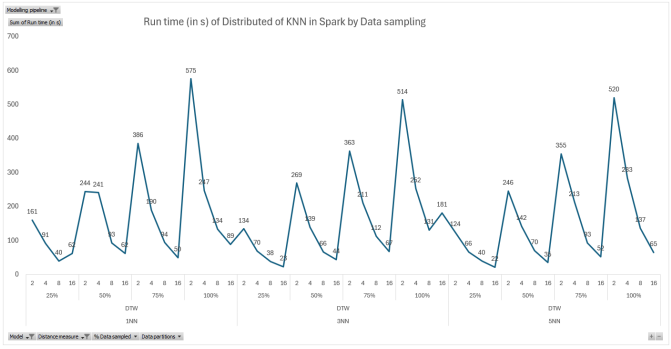

Fig. 4. Run time of Distributed KNN implementation

In Figure 3: We implement a distributed RDD for KNN using DTW, we see that,

- Accuracy increases as K increases in KNN. K=5 yields significantly better results than K=1
- Run time decreases significantly as number of partitions is increased

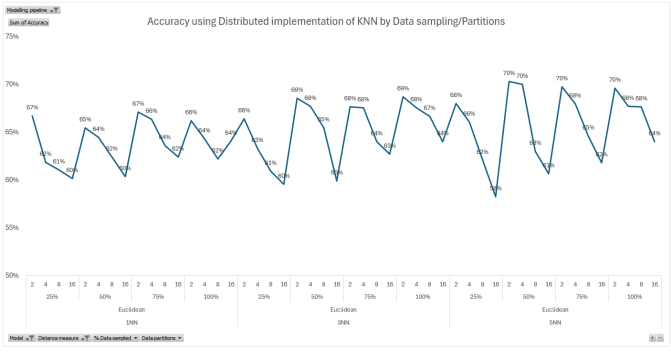

Fig. 5. Accuracy of Distributed KNN (Euclidean) implementation

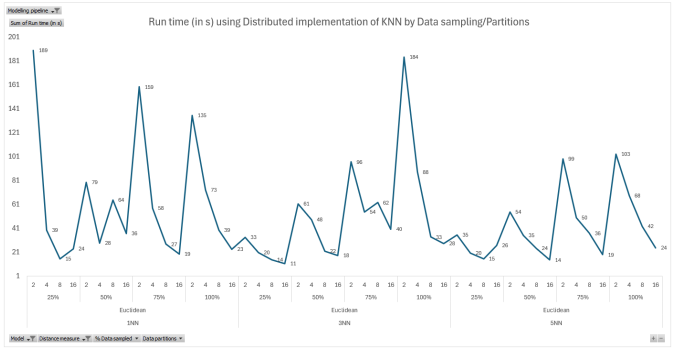

Fig. 6. Run time of Distributed KNN (Euclidean) implementation

In Figure 5 & 6: We implement a distributed RDD for KNN using Euclidean distance, we see that,

- Accuracy increases as K increases in KNN. K=5 yields significantly better results than K=1
- Run time decreases significantly as number of partitions is increased

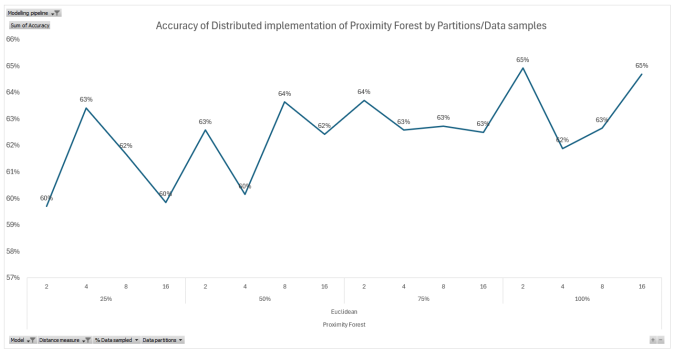

Fig. 7. Accuracy of Distributed Proximity Forest implementation

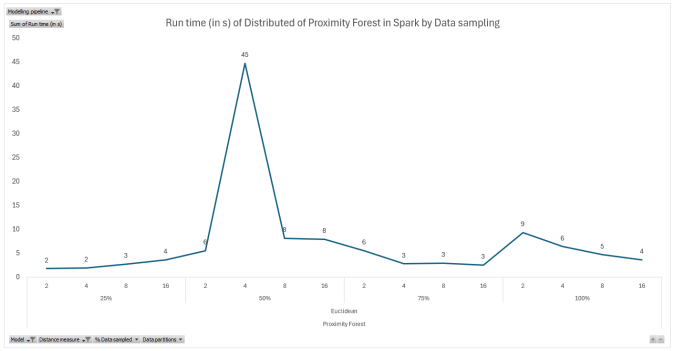

Fig. 8. Run time of Distributed Proximity Forest implementation

In Figure 7 & 8: We implement a distributed RDD for KNN using Euclidean distance, we see that,

- Accuracy decreases for higher number of partitions.
- Accuracy increases as sample size increases.
- Run time decreases significantly as number of partitions increases.

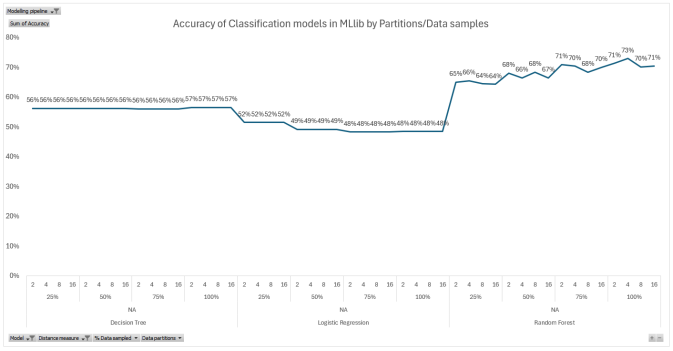

Fig. 9. Accuracy of MLlib Classification implementation

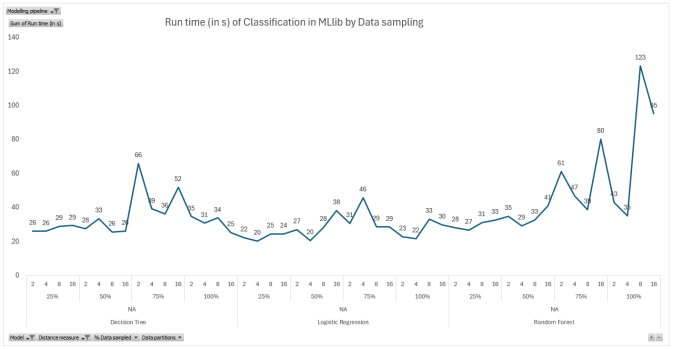

Fig. 10. Run time of MLlib Classification

In Figure 9 & 10: We implement a distributed RDD for KNN using Euclidean distance, we see that,

- Accuracy increases as K increases in KNN. K=5 yields significantly better results than K=1
- Run time decreases significantly as number of partitions is increased

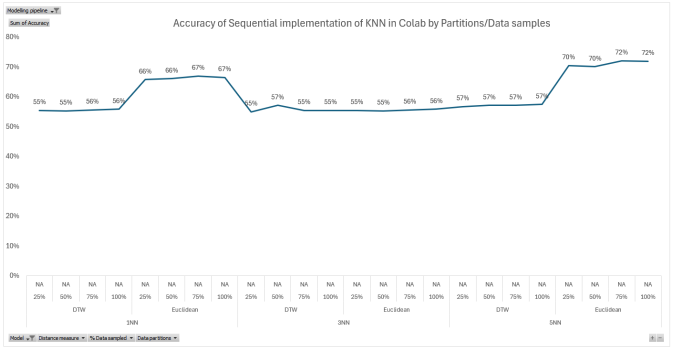

Fig. 11. Accuracy of KNN using Sequential implementation in Colab

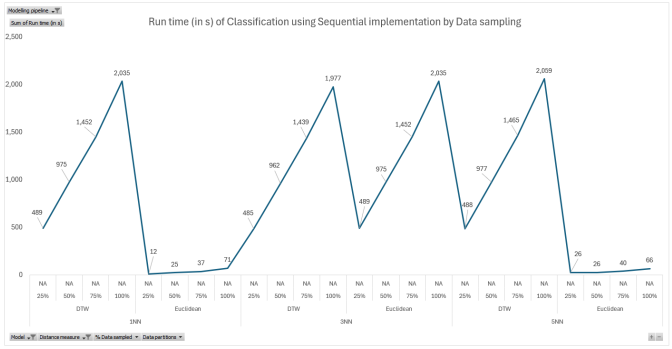

Fig. 12. KNN using Sequential implementation in Colab

In Figure 11 & 12: We implement a distributed RDD for KNN using Euclidean distance, we see that,

- Accuracy increases as K increases in KNN. K=5 yields significantly better results than K=1
- Run time decreases significantly as number of partitions is increased

All results (predictions, runtime logs, confusion matrices) are saved to /mnt/2025-team19/ and visualized through:

- Accuracy vs. data size & partitions plots
- Runtime vs. data size & partitions plots
- Accuracy vs. k plots
- Run time vs. k plots

V. Discussion

A. Overall Performance

The experimental results underscore the effectiveness and limitations of distributed classification models when applied to large time series datasets. The distributed DTW-KNN approach benefited from higher K values, as larger neighborhoods led to more robust predictions. However, the strategy of selecting only one random partition per test instance, although efficient in reducing computation, sometimes limited the model's access to diverse and representative training samples, especially when the number of partitions was high. This explains the trade-off observed between improved runtime and reduced accuracy. Euclidean-KNN, in contrast, showed more stable performance across partitioning configurations and consistently outperformed DTW-KNN in both speed and accuracy. The simpler nature of Euclidean distance likely contributed to the lower computational cost, and its less sensitivity to data warping made it more scalable in the distributed setting. Interestingly, the inverse relationship between KKK and runtime suggests that larger neighborhoods may result in more decisive predictions that require less computational overhead in the voting process. The Proximity Forest model leveraged the advantages of ensemble learning in a distributed environment. Its increasing accuracy with higher partition counts—except for very small data samples—demonstrates its robustness in partitioned training environments. Its superior runtime efficiency compared to DTW-KNN makes it a promising candidate for large-scale deployments where both speed and accuracy are critical. The results from the MLlib classifiers highlight the importance of model selection in distributed settings. While Decision Trees and Random Forests showed predictable improvements with larger datasets and more partitions, Logistic Regression suffered from declining accuracy, possibly due to data sparsity within partitions or its reliance on global data patterns not easily captured in partitioned subsets. Finally, the sequential implementations confirmed the value of distributed processing. Although they achieved competitive accuracy when trained on larger datasets, their runtime was significantly higher, reaffirming the necessity of parallelism for scalable machine learning workflows. The designed distributed approach—where training data is randomly partitioned, broadcasted, and accessed in parallel—proved to be a practical solution for handling large-scale time series classification, balancing trade-offs between runtime efficiency and classification accuracy.

B. Distributed Implementation

The distributed implementation of the DTW-KNN classifier revealed a consistent improvement in accuracy with higher values of K, where K=5 achieved the best performance, followed by K=3 and K=1. The runtime efficiency improved significantly as the number of partitions increased, although a lower number of partitions within each data sample led to better accuracy. This indicates that the granularity of partitioning affects both speed and performance. In case of the Euclidean-KNN classifier, accuracy improved with larger data samples and higher values of K, consistently showing that K = 5 outperformed K = 3, which in turn outperformed K = 1. However, an increase in the number of partitions per data sample segment negatively impacted accuracy. Euclidean-KNN outperformed distributed DTW-KNN in both accuracy and runtime. Furthermore, the runtime for Euclidean-KNN decreased with higher values of KKK, which contrasts with typical expectations and highlights its efficiency.

C. Comparing KNN with Proximity Forest implementation

For the distributed Proximity Forest classifier, an increase in accuracy was observed as the number of partitions increased, except when the data sample size was less than 25%, in which case the accuracy slightly decreased. The runtime of the Proximity Forest was significantly lower than that of the distributed DTW-KNN, making it a more efficient model in terms of computational cost.

D. Comparing implementation pipelines: Spark vs MLlib vs Sequential (in Colab)

Using Apache Spark's MLlib, Decision Trees and Random Forests showed increasing accuracy as both sample size and number of partitions grew. In contrast, Logistic Regression experienced a decrease in accuracy under the same conditions. Across all MLlib models, the runtime scaled with both partition count and sample size, reflecting the overhead introduced by distributed computation.

The sequential implementations of the classifiers demonstrated improved accuracy with larger data sample sizes and

higher partition counts. However, the runtime was considerably greater compared to the distributed versions, confirming the limita///;tions of non-parallel approaches for large-scale time series data.

VI. Conclusions and Future work

This study investigated the scalability and effectiveness of distributed time series classification models in handling large-scale audio and speech data. The experiments demonstrated that while distributed DTW-KNN benefits from higher values of K, its performance is hindered by the randomness of partitioning, which can reduce training diversity and limit accuracy. Euclidean-KNN proved to be a more stable and computationally efficient alternative, offering better scalability and runtime advantages in distributed environments.

Proximity Forest emerged as a promising model for distributed classification, combining ensemble robustness with computational efficiency. Its consistent performance across varying partition configurations indicates strong adaptability for large-scale deployments. Among the MLlib classifiers, Decision Trees and Random Forests maintained predictable accuracy improvements with increasing data, whereas Logistic Regression struggled—likely due to its dependence on globally coherent data, which is disrupted by partitioning.

Sequential implementations, although accurate with larger data samples, revealed significant runtime overheads, reinforcing the necessity of distributed strategies for real-time or large-scale applications. The overall findings confirm that parallelism and intelligent partitioning are essential to balance the trade-offs between accuracy and computational efficiency in time series classification tasks.

A. Future Work

Intelligent Partitioning: Future studies could explore data-aware partitioning strategies instead of random sampling, to ensure representative training subsets and minimize accuracy loss in distributed settings.

Distributed DTW Optimization: DTW-based classifiers could benefit from optimizations such as lower-bounding, early abandoning, or approximate DTW variants to reduce computational cost.

Integration of Feature Extraction: Incorporating feature engineering techniques (e.g., MFCCs, spectrograms) into the distributed pipeline may improve performance, particularly for simpler models like Logistic Regression.

Scaling Beyond Spark: Evaluating the models using cloud-native distributed systems (e.g., Dask, Ray, or Kubernetes with Spark) could provide insights into real-world deployment performance.

Acknowledgment

The authors would like to express their sincere gratitude to the module leader and teaching staff of COMP4124: Big Data Learning and Technologies for their invaluable guidance and support throughout the due course of this project. They also extend their appreciation to the University of East Anglia and the UCR Time Series Classification Archive for their generosity in providing access to the FordA dataset, which formed the fundamental basis of this study. Furthermore, the authors acknowledge the availability of the Databricks Spark platform, which played a crucial role in enabling the scalable implementation and experimentation of the methodologies explored in this work.

References

[1] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017, doi:10.1007/s10618-016-0483-9.

[2] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, "Proximity forest: An effective and scalable distance-based classifier for time series," *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 607–635, 2019, doi:10.1007/s10618-019-00617-3.

[3] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera, "kNN-IS: An iterative Spark-based design of the k-Nearest Neighbors classifier for big data," *Knowledge-Based Systems*, vol. 117, pp. 3–15, 2017, doi:10.1016/j.knosys.2016.06.012.

[4] R. Tavenard, J. Faouzi, G. Vandewiele, et al., "tslearn: A machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-091.html

[5] Apache Spark MLlib, "Classification and regression - MLlib," 2025. [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html

[6] Aeon Toolkit, "ProximityForest Classifier," 2023. [Online]. Available: https://aeon-toolkit.org/...ProximityForest.html

[7] Keras Team, "Time series classification from scratch," Keras.io Examples, 2023. [Online]. Available: https://keras.io/examples/timeseries/timeseries_classification_from_scratch/

[8] JAXStack Team, "JAX time series classification," JAXStack Documentation, 2023. [Online]. Available: https://docs.jaxstack.ai/en/latest/JAX_time_series_classification.html

[9] A. Gupta, A. Das, and M. Naskar, "A hybrid model for audio classification using MFCC and BiLSTM," in *Proc. of the 1st Int. Workshop on Audio Processing using Machine Learning (APML 2023)*, CEUR Workshop Proc., vol. 3554, pp. 114–122, Dec. 2023. [Online]. Available: https://ceur-ws.org/Vol-3554/paper13.pdf