ENSEMBLE METHODS FOR SPATIAL DATA STREAM CLASSIFICATION WITH APPLICATION TO EMERGENCY SERVICES

PRASANTA BHATTACHARJEE
Bachelor of Science, Metropolitan University, 2016

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

ENSEMBLE METHODS FOR SPATIAL DATA STREAM CLASSIFICATION WITH APPLICATION TO EMERGENCY SERVICES

PRASANTA BHATTACHARJEE

Date of Defence: September 27, 2024

Dr. Wendy Osborn Thesis Supervisor	Associate Professor	Ph.D.
Dr. John Anvik Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. John Zhang Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Andrew Fiori Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Abstract

Our research investigates the application of ensemble methods for spatial data stream classification within emergency services, specifically severe weather events. Emergencies demand swift, accurate decisions to mitigate impacts and protect lives. Ensemble methods improve the accuracy of predictions by combining outputs from multiple neural networks, each trained on diverse aspects of the data, including geographic coordinates, weather data, spatial data, and logistical factors. These models collectively contribute to more precise decision-making, particularly in assessing evacuation priorities. We collected and generated relevant data for affected regions and evacuation centres pertinent to severe weather events. For class labeling data, we employed K-means and DBSCAN clustering techniques. Our findings show that the ensemble of neural networks significantly improves the classification accuracy of spatial data stream data, potentially leading to more effective emergency responses. Comprehensive experiments with streams containing both spatial and non-spatial data show the accuracy, precision, and recall of our proposed approach.

Acknowledgments

My profound appreciation goes out to my supervisor, Dr. Wendy Osborn, for her unwavering support of my master's degree and her great knowledge, inspiration, advice, and tolerance for my errors. I am especially grateful for her patience and understanding, as she tolerated my errors and encouraged me to learn and grow from them. Her constant support and dedication are the cornerstone of my academic and personal development.

I am really grateful to my committee members, especially Dr. John Anvik and Dr. John Zang, for their constant support, suggestions, and motivation. I also want to thank Dr. Andrew Fiori for being the Examination chair of my thesis defense.

I am grateful for the financial support provided by the University of Lethbridge's School of Graduate Studies (SGS).

I am incredibly grateful to my parents, sister-in-law, wife, and adorable young niece, Parisha, for their continuous backing and support throughout my life.

I am grateful to my elder brother Pankaj for helping me make all of the greatest decisions and simplifying my life.

I am thankful to one of my close friends, colleagues, and brother, Naveen, for his tremendous support and outstanding knowledge.

Contents

List of Tables

List of Figures

List of Abbreviations

AdaBoost	Adaptive Boosting
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DSP	Data Stream Processor
EPS	Epsilon
MinPts	Minimum Points
Tanh	Hyperbolic Tangent
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
AdaGrad	Adaptive Gradient
RMSProp	Root Mean Square Propagation
RDBMS	Relational Database Management System
DBMS	Database Management System
SEA	Streaming Ensemble Algorithm
ASHA	Asynchronous Successive Halving Algorithm
OzaBag	Online Bagging
DWM	Dynamic Weighted Majority
DCS-DQ	Dynamic Classifier Selection by Divide and Conquer
NumPy	Numerical Python
Sklearn	Scikit-learn
Imblearn	Imbalanced-learn

Chapter 1
Introduction

Emergencies, particularly in the realm of severe weather events, pose unique and immediate threats [12]. They require a specialized set of skills and rapid response techniques. Motivation to serve in emergency services is rooted in a deep sense of duty and a desire to positively impact society. When others are fleeing from danger, emergency responders run toward it, displaying incredible courage and resilience [18]. Their motivation comes from the understanding that their actions can mean the difference between life and death, chaos and order, and despair and hope [18]. An emergency that causes severe environmental damage and constitutes a possible threat to human life may need the use of technical assistance and particular expertise to respond efficiently and mitigate effects.

According to Jin et al. [18], the main functions of emergency management include data collection, assessment, data processing, and decision presentation. To make the most effective use of emergency resources, it is essential to understand the many phases involved in disaster management. The emergency management team aims to coordinate evacuation service members and material shipment, arrange disaster relief system construction, supervise emergency preventive production activities, and assist in aid operations during disasters [18]. In this work, we will investigate the use of ensemble methods, data streams, clustering, and neural networks in severe weather emergency services.

1.1 Technologies

Ensemble methods [9] are methods that create multiple models and combine them to produce a single better predictive model. Their primary objective is to improve the robustness and accuracy of predictions. These methods can lower the risk of overfitting [9], as the combined models often generalize better to unseen data. Although several approaches

to ensemble methods exist [25], the work in this thesis will use Bagging [5]. Bagging, also known as Bootstrap Aggregating, trains multiple versions of a model on different subsets of the training data. In emergency services [9] during severe weather, ensemble methods can significantly enhance the accuracy and reliability of predictive models used for forecasting weather events, estimating their impact, and optimizing resource allocation. By combining outputs from multiple models, ensemble methods can predict class labels, continuous values, probability estimates, and rankings, which are crucial for determining the severity and trajectory of storms. This ensemble ensures more accurate warnings, better preparation strategies, and more effective resource allocation, ultimately improving the overall response to severe weather events.

A data stream [23] refers to a sequence of data elements made available over time. The data streams are continuous and rapidly changing. Handling data streams requires algorithms that can process often with limited memory and computational resources. In emergency services, it is essential to monitor weather conditions in real-time to identify emergency situations. Continuous data from weather stations, satellites, and sensors should be processed in real-time so that severe weather events can be detected. This allows emergency services to update their strategies dynamically based on the latest data.

Neural networks [13] are a subset of machine learning inspired by the structure and function of the human brain. They consist of interconnected layers of nodes, or neurons, where each connection represents a weight-adjusted during training to minimize error. Neural networks are particularly powerful for tasks such as image and speech recognition, natural language processing, and complex pattern recognition. Neural networks can be employed to analyze complex weather patterns and predict severe weather events [32]. Using different types of data such as data streams, neural networks can learn to recognize the precursors of such events and provide early warnings, allowing for timely evacuations and preparations [32].

Clustering [17] is a type of unsupervised learning that involves grouping a set of objects

in such a way that objects in the same group or cluster are more similar to each other than to those in other groups. It helps discover inherent groupings in data without predefined labels. It is widely used for exploratory data analysis, pattern recognition, image analysis, and bioinformatics [8]. Two known clustering algorithms include K-means and DBSCAN. Clustering can be used to label weather data based on current situations present in the data, which can be used for future predictions.

In our research, we combine these methods to yield powerful results. Ensemble methods will combine the predictions of multiple neural networks trained on different aspects of streaming weather data, improving the overall predictive accuracy for severe weather events. Real-time data streams help to predict the severity of weather over time. Clustering can help in segmenting data streams into meaningful groups for further analysis. Clustering can be used to preprocess data, creating class-labeled groups that can then be analyzed by the different neural network models in the ensemble. This can improve the performance of the ensemble by ensuring that each model is trained on a more uniform and consistent subset of data. Neural networks can be trained and updated using real-time data streams, ensuring that the models remain accurate and relevant as new data becomes available and also improving accuracy as more data comes along.

We conducted experiments on our proposed strategies by first creating and class-labeling a severe weather dataset from parts of the province of Alberta. The dataset was then used to simulate a spatial data stream, with the class-labeled data streaming in batches. The ensemble method processes the data in batches, allowing for continuous learning and prediction as new data becomes available. The evaluation outcomes indicate that the ensemble of neural networks outperformed well across various metrics, including accuracy, precision, and recall, particularly at a learning rate of 0.0001. With DBSCAN clustered data, the last batch and average of all batches processed by the ensemble for all the experiments maintained good performance. In contrast, for K-means clustered data ensemble showed more variability in performance, with its results being less consistent across different batches and

learning rates. The findings demonstrate that DBSCAN is more effective in clustering data for our ensemble of neural networks.

1.2 Research Questions

Through these research questions, we aim to identify methodologies that enhance the robustness and reliability of decision-making systems in the context of severe weather events and similar real-time applications. The questions are as follows:

1. Which of the clustering algorithms K-means and DBSCAN produced better class-labeled data in terms of performance?

2. Which sequence for processing a spatial data stream achieves better predictive outcomes?

3. Can using an ensemble of neural networks for processing spatial data streams lead to improved model performance and accuracy?

1.3 Contributions

Our work contributes specifically to the emergency services research field, focusing on severe weather. Additionally, we have implemeneted ensemble methods that handle streaming data containing both spatial and non-spatial information, extending beyond severe weather applications. The contributions are as follows:

1. Previous emergency service-related work did not use a combination of the ensemble method and neural networks. However, in our work, we incorporate both the ensemble method and neural networks. By combining the strengths of ensemble methods, with the powerful predictive capabilities of neural networks, this approach aims to enhance the accuracy and reliability of predictive models in emergency services.

2. Clustering algorithms such as K-means and DBSCAN are not explicitly used in emergency service-related work. However, we use them to group similar data points and identify class labelling.

3. Earlier work in ensemble methods for streaming data did not consider both spatial and non-spatial data within the same data stream. However, in this research, we addressed this gap by integrating both types of data.

1.4 Organization

The rest of this thesis is organized as follows.

Chapter 2 presents background knowledge on existing strategies for processing data streams, ensemble methods, K-means, DBSCAN clusters, and neural networks.

Chapter 3 presents related work on data streams and ensemble methods.

Chapter 4 presents an overview of the data requirements for severe weather, data gathering and generation, clustering, and class labeling. Then, we present the architecture and functionality of our system.

In Chapter 5, we describe the implementation strategy and present specifications related to software and hardware. Then, we present the experimental processes and evaluation results of our research.

Finally, Chapter 6 presents the thesis conclusion and identifies future research directions.

Chapter 2
Background

In this chapter, we present background on the concepts that are related to our research work. This chapter is organized as follows: Sections 2.1 and 2.2 present the research summaries of the data stream and ensemble method. Then, Sections 2.3 to 2.5 present the summaries for K-means clustering, DBSCAN clustering, and neural networks.

2.1 Data Streams

Margara and Rabl [23] define a data stream as a continuously long sequence of objects representing data components that become available over time. This concept is particularly significant in the context of modern data processing and analysis, where data can be generated continuously by various sources such as sensors, social media feeds, financial markets, and network logs. These streams necessitate real-time processing and analysis to extract meaningful insights promptly. According to the authors, data streams are categorized into structured and unstructured [2]. Structured streams follow a specific format, simplifying data modeling and analysis. For example, sensor data from a weather station, where each record includes specific fields like temperature, humidity, wind speed, and timestamp, making it easy to analyze and model. Unstructured streams, often created by merging data from various sources, and lack a consistent format. For instance, social media posts related to weather events, which consist of a mix of text, images, and videos, lacking a uniform structure and requiring more complex processing for analysis.

Structured streams come in three primary types, each varying in the manner their constituent parts are associated with and how they impact one another [23]:

1. Turnstile Model: In this model, a stream is depicted as a list of elements, with each element within the list being updated individually. The length of this list reflects the

domain of the stream elements. This process is akin to traditional database systems, which handle updates, deletions, and insertions in the database.

2. Cash Register Model: Here, elements in the stream are solely additions to the underlying array, meaning that each new element is appended to this array without the possibility of removal. Once an element is added, it remains part of the array permanently. This approach is analogous to the way databases record the history of relational data, capturing every change and addition to maintaining a comprehensive historical record.

3. Time Series Model: In this model, every segment within the stream is regarded as a distinct list, allowing for separate and individual analysis of each segment. This segmentation allows for detailed temporal analysis of data, making it suitable for applications such as stock market analysis, climate data monitoring, and any domain where time-based patterns and trends are critical. Each segment can be independently analyzed, providing insights into how data evolves over time.

In our research work, we used the Time Series Model. This model is particularly suited for our analysis as it allows for a detailed temporal examination of spatial data streams [24]. By treating every segment within the stream as a distinct list, this model facilitates the separate and individual analysis of each segment, which is crucial for understanding temporal patterns and trends in the data. This approach aligns well with our goal of handling the spatial and temporal complexities of data streams using ensemble methods and neural networks. This model also aligns well with severe weather data processing. Severe weather events often involve complex temporal patterns that need to be analyzed over time. By applying this model, it can effectively track and predict changes in weather conditions as they evolve, which is crucial for accurate forecasting and timely responses.

2.2 Ensemble Methods

Ensemble methods [9] refer to a class of machine learning techniques that combine multiple models to improve overall performance and predictive accuracy. These methods leverage the strengths of various models to compensate for their individual weaknesses, ultimately providing a more robust and reliable prediction. These ensemble methods are particularly effective in enhancing the generalization capabilities of machine learning algorithms by aggregating the predictions from multiple models.

The different methods [25] for data-centered ensembles include Bagging, Boosting, Stacking, Averaging and Random forest, each with its unique approach to processing the data and improving model performance.

1. Bagging [25]: Bagging is an ensemble algorithm that is entirely dependent on the given data. The term describes the process of dividing up an enormous amount of data into multiple subsets, which are generated by randomly sampling from the original dataset with replacement. Due to sampling with replacement, the subsets of data may not be disjoint. Each subset is used to train a separate model, and the final prediction is obtained by averaging or majority voting the predictions from all models.

2. Boosting [25]: Boosting is an iterative ensemble technique aimed at identifying and creating a suitable solution to address the mistakes caused by prior algorithms. In boosting, models are trained sequentially, with each new model focusing on correcting the errors made by the previous ones. This is achieved by adjusting the weights of the training samples, increasing the importance of those that were misclassified by earlier models. The final model is a weighted sum of all the individual models, resulting in a strong learner that effectively reduces both bias and variance. Boosting algorithms, such as AdaBoost and Gradient Boosting, are known for their ability to improve the performance of weak learners significantly.

3. Stacking [25]: Stacking arranges data from several algorithms and presents them as

a single algorithm. This method involves training multiple base models on the same dataset and then using their predictions as input features for a meta-model. The meta-model is trained to combine the base model predictions optimally. By leveraging the strengths of different base models and combining them in a meta-model, stacking can achieve better performance than any individual model. This approach is highly flexible and can be applied to various types of base models, making it a powerful tool for complex predictive tasks.

4. Averaging [25]: In the averaging method, predictions from multiple models are combined by calculating the mean of their outputs. This technique is generally used to reduce variance and make predictions more stable. It can be applied in regression tasks by averaging numerical outputs or in classification by averaging class probabilities. While averaging combines predictions from multiple models without necessarily training them on different data samples, bagging uses bootstrapping to create multiple variations of the dataset, trains models independently on these samples, and then aggregates their predictions. Bagging is specifically designed to reduce overfitting by training on diverse data samples, whereas averaging focuses on reducing variance by combining different models' outputs directly.

5. Random forest [25]: A random forest is an extension of bagging applied specifically to decision trees. It creates multiple decision trees using bootstrapped datasets, and at each split in a tree, it randomly selects a subset of features to find the best split. This feature randomness introduces more diversity among the trees, reducing correlation between them and improving overall performance. The final prediction is typically based on the average output of all trees (for regression) or majority voting (for classification). Random Forest is effective at reducing both variance and overfitting.

In our work, we use the bagging method because it reduces the likelihood of overfitting, leading to more reliable and robust models. Bagging helps stabilize the predictions by

averaging the outputs of multiple models trained on different subsets of the data. This ensemble approach not only improves accuracy but also enhances generalization to new data, making the system more effective in handling variability in the data streams from our sensors. Moreover, bagging is straightforward to implement and scales well with the ensemble of neural networks, which is crucial given the limited storage and processing capacity of our data stream processor (DSP).

2.3 K-means clustering

K-means clustering [14] is one of the popular unsupervised learning algorithms, which is used for classifying a set of N-dimensional objects based on their properties into K distinct groups or clusters. This algorithm is devised for partitioning the set of objects by minimizing the variance between the objects. Each object is assigned to the cluster with the centroid closest to that object [22].

The formula [22] is used to group data points so that those within the same cluster are as close to each other as possible:

$$J = \sum_{i=1}^{K} \sum_{x \in S_i} \|x - \mu_i\|^2$$

In this formula:

(a) J represents the total sum of squared distances between each data point and the centroid of its respective cluster.

(b) K is the number of clusters.

(c) S_i includes all the data points in cluster i.

(d) μ_i is the centroid (the average position) of all the data points in cluster S_i.

(e) x represents an individual data point within cluster S_i.

2.3.1 K-means Algorithm

MacQueen [22] described and popularized the K-means algorithm as a simple, step-by-step process, although it was originally introduced by Lloyd [21]. The algorithm proceeds as follows.

1. **Initialization**: Start by randomly choosing K distinct data points to be the initial centroids.

2. **Assignment**: For each data point, find the closest centroid based on a distance metric and assign the data point to the corresponding cluster.

3. **Update**: Update the centroids by calculating the mean of all the data points in each cluster.

4. **Repeat**: Continue the assignment and update steps until the centroids changes are very small, indicating convergence.

5. **Compactness Measurement**: After convergence, measure the compactness of the resulting clusters using the within-cluster sum of squares:

$$\text{Compactness} = \sum_{k=1}^{K} \sum_{x \in S_k} \|x - \mu_k\|^2$$

   (a) μ_k is the centroid of cluster S_k.

   (b) x are the data points in S_k.

   (c) K is the total number of clusters.

   This step ensures that the clusters are dense and well-separated, which is desired for good clustering outcomes.

This algorithm cycles through these steps to divide the dataset into K clusters, aiming to minimize the total distance between the data points and their respective cluster centroids. Essentially, it works to make each cluster as tight and distinct as possible.

2.3.2 Distance Metrics

According to Han, Kamber, and Pei [14], distance metrics play a crucial role in the K-means clustering algorithm as they determine how the distance between data points and cluster centroids is calculated. The choice of distance metric can significantly influence the performance and outcome of the clustering process. The commonly used distance metrics are:

1. Euclidean Distance: This is the most commonly used distance metric in K-means clustering. It calculates the straight-line distance between two sets of coordinates, each represented by a vector of coordinates. The distance is computed as the square root of the sum of the squared differences between corresponding coordinates. For example, two coordinates $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ in an n-dimensional space. The Euclidean distance between these points is given by:

$$de(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2. Manhattan Distance: Also known as the ”taxicab” or ”city block” distance, this metric calculates the distance between two points in a multidimensional space by summing the absolute differences of their corresponding coordinates. Conceptually, Manhattan distance is akin to navigating a grid-like street layout, where you can only move horizontally or vertically, similar to how a taxi would drive through the streets of Manhattan. Manhattan distance reflects the cumulative distance traveled along axes-aligned paths. For two points $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, the Manhattan distance is defined as:

$$dm(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$$

   - The sum of these absolute differences across all dimensions gives the Manhattan

distance.

This metric is particularly useful in high-dimensional spaces and for datasets where differences in each dimension are binary or categorical.

3. Minkowski Distance: This is a generalized distance metric that includes both Euclidean and Manhattan distances, depending on the value of the parameter p. For two points $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, the Minkowski distance is:

$$dm(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

The parameter p determines the distance type, where:

- $p = 1$ gives the Manhattan distance,
- $p = 2$ gives the Euclidean distance,

In our work, the Euclidean Distance metric is particularly useful because we often deal with continuous, multi-dimensional data where straight-line relationships are meaningful. Since our objective is to optimize intra-cluster similarity, Euclidean Distance effectively captures the shortest path between data points in a multi-dimensional space. This allows us to minimize distortion when grouping data points into clusters, ensuring that the geometric structure of the data is preserved, making it easier to interpret the relationships between data points within the same cluster. Additionally, this metric aligns well with our use of spatial and non-spatial data, where the relationships between features benefit from this intuitive distance measure.

2.4 DBSCAN clustering

DBSCAN [10] is designed to identify clusters based on the density of data points in a given region. The algorithm works by grouping together points that are closely packed

together, while marking as outliers points that lie alone in low-density regions. DBSCAN classifies points into three categories: core points, border points, and noise points. Core points have dense neighborhoods, border points are within the neighborhood of a core point but do not themselves have dense neighborhoods, and noise points are those that do not meet the density requirements.

1. Neighborhood Definition: Given dataset D, the ε-neighborhood of a point p is defined as:

$$N_\varepsilon(p) = \{q \in D \mid d(p,q) \leq \varepsilon\}$$

   Where d is the distance function.

2. Core Points: A point p is a core point if its ε-neighborhood (N_ε) contains at least MinPts points:

$$|N_\varepsilon(p)| \geq \text{MinPts}$$

   where $|N_\varepsilon(p)|$ is the number of points in the ε-neighborhood of p.

3. Direct Density-Reachable: A point p is directly density-reachable from a point q if p is within the ε-neighborhood of q and q is a core point:

$$p \in N_\varepsilon(q) \text{ and } |N_\varepsilon(q)| \geq \text{MinPts}$$

4. Density-Reachable: A point p is density-reachable from a point q if there is a chain of points $p_1, p_2, \ldots, p_n$ such that:

$$p_1 = q,\ p_n = p, \text{ and each } p_{i+1} \text{ is directly density-reachable from } p_i$$

5. Density-Connected: A point p is density-connected to a point q if there is a point o such that both p and q are density-reachable from o.

Figure 2.1 illustrates the difference between direct density-reachability and density-reachability:

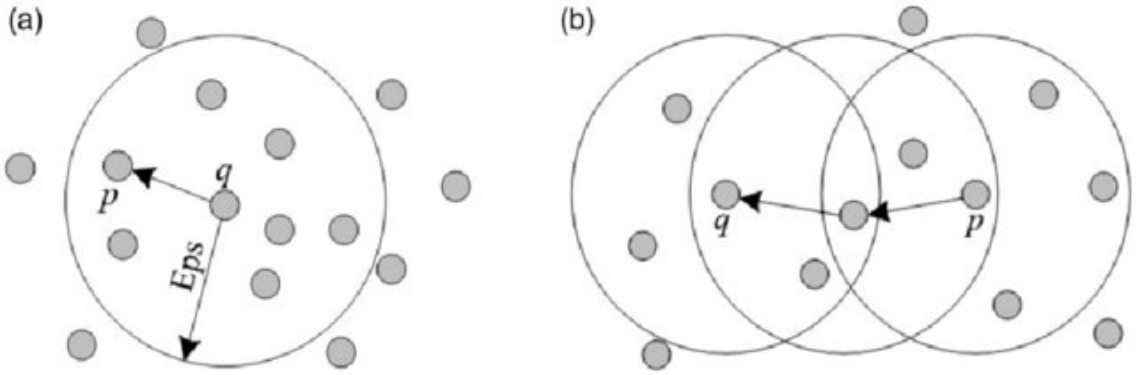

Figure 2.1: Direct Density-Reachability And Density-Reachability. Adapted from Hu et al. [16]

- **Direct Density-Reachability:** In the image (Figure 2.1 (a)), direct density-reachability is illustrated using the relationship between points p and q. A point p is *directly density-reachable* from a core point q if it falls within the ε-neighborhood of q. This means that the distance between p and q is less than or equal to a predefined radius ε, depicted by the circle around q. In the diagram, the arrow shows that p is within the ε-neighborhood of q, indicating that p is directly density-reachable from q.

- **Density-Reachability:** Figure 2.1 (b) of the image demonstrates the concept of density-reachability. A point p is said to be *density-reachable* from another point q if there exists a chain of points $q, p_1, p_2, \ldots, p_n = p$, such that each point is directly density-reachable from the previous one. In other words, p is density-reachable from q if one can traverse through a series of points where each point in the sequence lies within the ε-neighborhood of its predecessor. In the diagram, the overlapping circles show that point p is density-reachable from q through a sequence of direct density-reachability relationships, even though p may not be directly reachable from q in a single step.

2.4.1 DBSCAN Algorithm

The DBSCAN algorithm [10] is particularly effective in identifying clusters of arbitrary shapes and handling noise in large spatial datasets. The algorithm can be summarized in the following steps:

1. **Initialization:** Start with an arbitrary point pp from the dataset DD.
2. **Visit Point:** Check if pp has been visited. If it has, proceed to the next point in the dataset.
3. **ε-neighborhood:** Calculate the ε-neighborhood of p, meaning find all points within a radius of ε around pp.
4. **Core Point Check:** If the number of points in the ε-neighborhood is less than MinPts, mark pp as noise. Otherwise, proceed to the next step.
5. **Cluster Formation:** If pp is a core point, create a new cluster and recursively collect all density-reachable points from pp. For each point qq in the cluster, check its ε-neighborhood. If qq is a core point, add its ε-neighborhood points to the cluster.
6. **Repeat:** Continue the process until all points in the dataset have been visited and either assigned to a cluster or marked as noise.

2.4.2 Silhouette Score

The Silhouette Score is a metric [28] that provides a graphical representation and numerical evaluation of the consistency within clusters of data. It aids in the interpretation and validation of clustering results by measuring how similar an object is to its own cluster compared to other clusters. This score helps in determining the appropriateness of the clustering technique and the number of clusters.

The Silhouette Score combines two aspects of cluster analysis: cohesion and separation. Cohesion refers to how closely related the points in a cluster are, while separation measures how distinct or well-separated a cluster is from other clusters.

1. Cohesion: Measured by the average distance between each point and all other points in the same cluster.

2. Separation: Measured by the average distance between each point and all points in the nearest neighboring cluster.

2.4.3 Silhouette Score Algorithm

The Silhouette Score [28] ranges from -1 to 1, where a higher score indicates better-defined clusters. The algorithm can be summarized in the following steps:

1. **Input:**

   - X: A set of data points.
   - $labels$: A list of cluster labels corresponding to each data point in X.

2. **Output:**

   - The Silhouette Score for each data point and the mean Silhouette Score for the entire dataset.

3. **Steps:**

   (a) **For each data point i in X:**

   - Determine the cluster C_i to which i belongs.

   (b) **Cohesion $a(i)$:**

   - Calculate the average distance between i and all other points in the same cluster C_i:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i, j)$$

   where d is the distance function.

   (c) **Separation $b(i)$:**

- For each cluster C_k different from C_i, calculate the average distance between the centre of cluster i and all points in C_k:

$$b_{C_k}(i) = \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

- The separation $b(i)$ is the minimum average distance to all other clusters:

$$b(i) = \min_{k \neq i} b_{C_k}(i)$$

(d) **Silhouette Score $s(i)$:**

- Compute the Silhouette Score for each point i:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- If $|C_i| = 1$, define $s(i) = 0$.

(e) **Mean Silhouette Score:**

- Compute the mean Silhouette Score for all points:

$$S = \frac{1}{|X|} \sum_{i \in X} s(i)$$

2.5 Neural Networks

Neural networks [15] are a class of machine learning models inspired by the human brain structure and function. These models are designed to recognize patterns, make decisions, and predict outcomes based on data.

2.5.1 Structure of Neural Networks

A neural network [15] is composed of layers of interconnected nodes, or neurons, which process input data and generate output. Figure 2.2 explains the basic structure of neural

networks which includes:

1. Input Layer: The input layer consists of neurons that receive various inputs from the external environment. Each neuron in this layer represents a feature or variable of the input data.

2. Hidden Layers: These layers lie between the input and output layers. They are called hidden because their values are not directly observable. Each neuron in a hidden layer receives input from the previous layer, processes it, and passes it on to the next layer. The complexity and number of hidden layers can vary depending on the specific task and model.

3. Output Layer: The output layer consists of neurons that produce the network's final output. The number of neurons in this layer corresponds to the number of prediction categories or output variables.

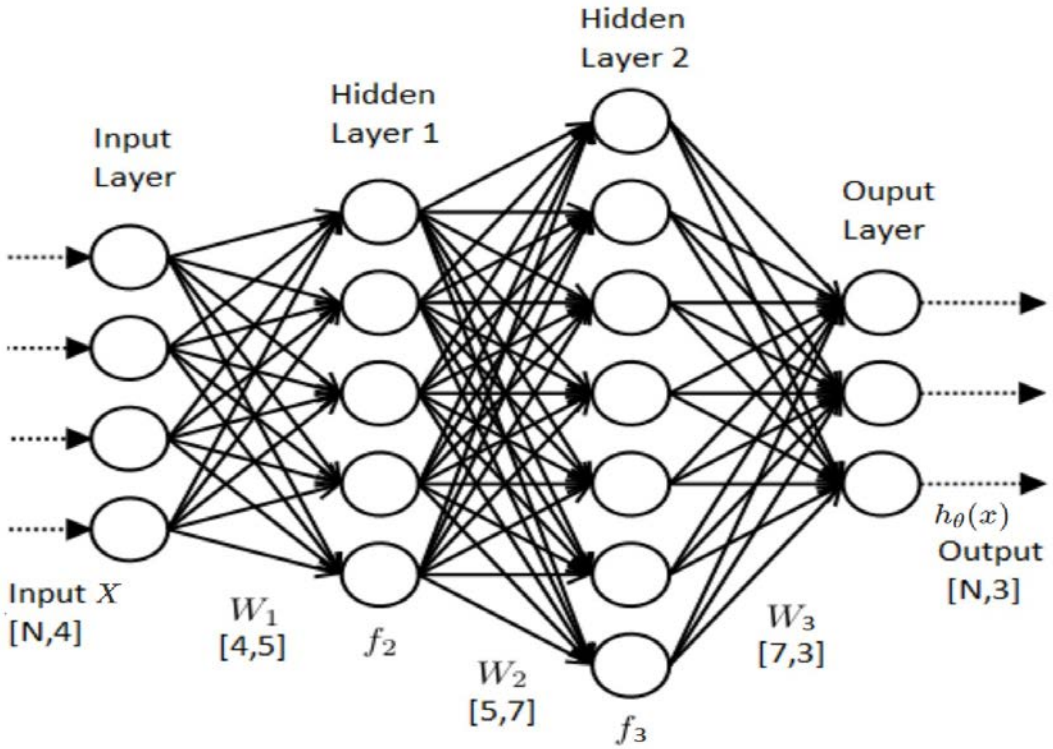

Figure 2.2: Neural Network Structure[1].

[1] Adapted from Google Images, https://images.app.goo.gl/192WJrVGinrLCthp8.

2.5.2 Learning Process

The learning process in neural networks involves adjusting the weights of the connections based on the output error. This process can be summarized in the following steps [15]:

1. Initialization: The weights are initialized randomly or using specific methods like Xavier initialization.

2. Forward Propagation: The input data is passed through the network layer by layer. Each neuron computes a weighted sum of its inputs, applies an activation function to introduce non-linearity, and passes the result to the next layer. This process continues until the output layer produces the final predictions.

3. Loss Calculation: The network output is compared to the actual target values using a loss function, which quantifies the error between the predicted and true values.

4. Backward Propagation: The error is propagated back through the network, and the gradients of the loss with respect to each weight are computed. This involves applying the chain rule of calculus to determine how the error changes with respect to each weight.

5. Weight Update: The weights are updated using an optimization algorithm such as Gradient Descent. The learning rate determines the size of the steps taken in the direction of the negative gradient to minimize the loss.

6. Iteration: Steps 2-5 are repeated for a specified number of epochs or until the loss converges to an acceptable level.

2.5.3 Functions and Optimization Algorithms

The detailed process of the different functionalities is given below [15]:

1. Activation Functions: Activation functions introduce non-linearity into the network, allowing it to learn complex patterns. Common activation functions include:

   (a) Sigmoid Function: Outputs values between 0 and 1, used in binary classification tasks.

   (b) Tanh (Hyperbolic Tangent) Function: Outputs values between -1 and 1, used to center the data. The formula is,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   (c) ReLU (Rectified Linear Unit): It is an activation function that outputs the input value if it is positive. Otherwise, it outputs zero. It is widely used in neural networks due to its simplicity, computational efficiency, and ability to mitigate the vanishing gradient problem, leading to improved model performance.

2. Loss Functions: Loss functions measure the discrepancy between predicted and true output. Examples include:

   (a) Mean Squared Error (MSE): Used for regression tasks, calculates the average of the squared differences between predicted and true values.

   (b) Cross-Entropy Loss: Used for classification tasks, measures the performance of a classification model whose output is a probability value.

3. Optimization Algorithms: These algorithms adjust the network's weights to minimize the loss. Common optimizers include:

   (a) Gradient Descent: Updates weights by moving in the direction of the negative gradient of the loss function.

   (b) Stochastic Gradient Descent (SGD): A variant of Gradient Descent that updates weights using a random subset of the data at each step.

(c) Adam (Adaptive Moment Estimation): Combines the advantages of two other extensions of gradient descent, namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The formula is:

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

- $\hat{m}_t$: Bias-corrected first moment estimate.
- $\hat{v}_t$: Bias-corrected second moment estimate.
- θ_t: The parameter vector at time step t.
- α: The learning rate.
- ε: A small constant to prevent division by zero (typically 10^{-8}).

In our work, we used Tanh as an activation function because it outputs values between -1 and 1, which can make learning easier and faster. The loss function utilized in this research is Cross-Entropy Loss. It is effective because it penalizes incorrect classifications heavily, which helps the model learn and adjust its weights more effectively, thereby improving the accuracy of the predictions. Finally, as an optimizer, we used Adam (Adaptive Moment Estimation). It adjusts the learning rate for each parameter dynamically, which helps in faster convergence and better performance on large datasets and complex neural networks.

2.6 Chapter Summary

In this chapter, we presented the theoretical background of data streams and ensemble methods, including clusters and neural networks. In the next chapter, we summarize related work.

Chapter 3
Related Work

In this chapter, we summarize the work related to our research. The chapter is organized as follows: Sections 3.1 presents the King and Osborn [27] [19] works that our thesis is based on these work, and Sections 3.2 and 3.3 present the data stream and ensemble method related works.

3.1 Classification Approaches for Spatial Streaming Data

In this section, we discuss the most relevant works related to our research.

3.1.1 An Iterative Strategy for Deep Learning Classification on Spatial Data Streams

King and Osborn [27] addresses the problem of classification of data in a spatial data stream, which has received very little attention compared to the classification of spatial objects in a static data set.

According to the authors, prior research in geographic data stream mining has mostly concentrated on k-nearest neighbour search and grouping. The existing categorization of streaming geographic data utilizes a tree structure to expedite entropy computations for decision tree construction. It does not delve into the use of neural networks.

The authors propose employing deep neural networks for the continuous classification of geographic input streams. The approach processes incoming spatial objects in batches using a sliding window. The classifier is trained and tested iteratively on small portions of the stream, alternating between training on one portion of the batch and testing on another. This incremental method aims to achieve a classification accuracy close to what would be obtained if the entire dataset was available all at once. As more objects are processed, the classifier becomes more accurate, offering an efficient solution for real-time classification of streaming spatial data.

The proposed method was evaluated experimentally by training the neural network gradually using a stream of spatial objects as input and comparing the results to a fully trained network to assess accuracy. Three distinct splits were employed by the authors to prepare and evaluation: 90/10, 80/20, and 70/30. According to the findings, the incremental strategies accuracy increased with the proportion of training items. But the 80:20 split was the most accurate when compared to the accuracy obtained when the complete dataset was evaluated. There was a greater discrepancy between cumulative and total accuracy in the splits 90:10 and 70:30, despite having the greatest average cumulative accuracy.

The suggested method may be enhanced, according to the authors, to accommodate real-time information mining of geographical information streams.

The authors work aligns with our ensembles in its emphasis on incremental neural network training using data streams, managing limited storage, and splitting data for training and testing. Both approaches address real-time spatial data stream classification through structured batch processing and incremental learning techniques to enhance model accuracy.

3.1.2 Ensemble Methods for Spatial Data Stream Classification

Building on their earlier work, which focused on the use of a single deep neural network for classifying spatial data streams, King and Osborn [19] extend their approach by exploring ensemble methods for improving classification accuracy in real-time geographic data streams. While their initial strategy employed iterative training and testing of a single neural network on small batches of spatial objects, the authors now propose an ensemble technique that combines the strengths of multiple models to enhance classification performance.

In this follow-up work, the authors address the limitations of single-model classification by using an ensemble of three neural networks, each with three hidden layers. This ensemble method, optimized with hyper-parameter tuning, aims to reduce overfitting and increase

robustness in processing vector-based spatial data streams, which are typically challenging due to the unavailability of all data at once.

The authors propose an iterative ensemble technique for deep learning of a spatial data stream classification that employs three neural networks with three hidden layers each, optimized using the Tune library with ASHA algorithm for hyper-parameter tuning. Utilizing the tanh activation function for hidden layers and log-softmax for the output layer, each model is independently optimized to reduce overfitting and improve robustness. Data is processed in batches of 100 objects, with an 80/10/10 split for training, validation, and testing, and models are trained using the Adam optimizer. The ensemble approach combines predictions from the three models through a simple voting mechanism where each model votes on the label for a given input, and the label receiving the most votes is selected; in the event of a tie, a random selection is made among the tied labels.

The coordinate system, including rectangles represented by a tuple containing coordinates for the rectangle's top-right and bottom-left corners, is utilized by the authors to evaluate the ensemble approach and hyper-parameter optimization. Results revealed that the ensemble approach's average accuracy outperformed the single, unoptimized models by 2.77 percent. In addition to producing a 5.12 percent lower measure of variability at 22.03 percent, the ensemble advance also delivers a much higher accuracy of 7.89 percent variance compared to the single, unoptimized model.

The authors suggested that future research concentrate on customising non-simulated datasets to make them suitable for the suggested strategy.

Our research is mainly based on the King and Osborn [19] works, and it aligns with our ensemble of neural networks.

3.2 Data Stream Related Work

This section reviews existing methodologies for processing data streams, highlighting approaches.

3.2.1 Continuous Queries over a Data Stream

In order to effectively handle and process an incoming data stream, efficient frameworks and algorithms are needed. In the initial work on continuous query procesing, Babu and Widom [3] demonstrate the difficulty of processing the streaming data with conventional mechanisms of a traditional Relational Database Management System (RDBMS), like triggers and materialized views.

The authors found that some existing methods of stream processing have a specified query programming language and a certain range of functionality. Other methods deal with scalability concerns for combining many queries or monitor continuous queries based on incremental view maintenance. All of these methods do not use any novel architecture, triggers, or materialised concepts.

According to the authors, a streaming query is ongoing access to a single-supplied object. They developed an architecture with four components (Stream, Store, Scratch, and Throw) and six steps to ensure that the data objects that are requested are always consistent. In the first step of the architecture, the Stream component receives the incoming tuples into the system. Stream also contains the topics that will be part of the result "forever". Tuples that are part of the result currently but may not remain in the result in the future are moved to the Store. The Scratch component contains elements that may be needed in the future but are not part of the current result. The final element, Throw, collects and deletes all unnecessary tuples which creates empty space in the system.

The authors described the techniques employed, which included employing triggers and materialized views to store and evaluate the incoming stream, rather than evaluating the effectiveness of the recommended architecture. Incoming data is processed using the trigger and components like Stream and Store may be left unfilled, with Scratch being used to handle any necessary data for monitoring complex events or assessing conditions. Scratch is used as a search tool to evaluate the query conditions. Similar to this, materialized views keep the views in Store and the data which is not included in conventional tables are

stored in Scratch.

The network monitoring model is one of the alternatives to traditional stream processing research that the authors proposed. They state that it will support the resolution of computational and system research problems.

Our spatial data stream processing system aligns with the proposed architecture of Babu and Widom [3] regarding dynamic data handling and efficient stream processing. Their query management principles of bagging and neural network ensemble approach for robust model training and evaluation.

3.2.2 Other related works on data stream

The work by Krempl et al. [20] highlights eight core challenges in data stream mining: data privacy, managing legacy systems, handling incomplete and delayed information, and analyzing complex data streams. The authors emphasize the limitations of traditional data mining methods for dynamic data streams and the lack of systematic techniques to address these challenges. While they provide a detailed description of these issues, no specific solutions or evaluation results are presented. Instead, the focus is on outlining the gaps and challenges in current data stream mining approaches. Their work aligns with ours by highlighting the limitations of traditional data mining methods for dynamic data streams and emphasizing the need for tailored approaches like the ensemble model, which uses bagging and batch processing to handle real-time data efficiently. Our ensemble addresses the challenges of handling complex data streams and ensuring reliable predictions by employing multiple neural network models to enhance accuracy and robustness.

Towne et al. [31] address efficient query processing in data streams and relations, aiming to maximize approximation results while considering system resource limits. Existing methods only support static relation queries or convert relations into streams. The authors introduce techniques for joining data streams with relations directly, focusing on star-streaming joins and using sliding window semantics. They apply load shedding and seman-

tic load shedding to handle resource constraints, prioritizing tuples with higher matching probabilities. Experiments show that pre-filtering enhances performance significantly as fact relation sizes decrease. The authors state that future research should focus on reducing fact relation sizes, combining various data streams, and addressing DBMS system issues. The authors work focuses on efficient query processing in data streams using star-streaming joins and handling resource constraints, which aligns with the ensemble use of a DSP with limited storage that processes batches of records. Both systems emphasize resource-efficient handling of data streams and maximizing performance through techniques like pre-filtering or batch-based processing.

3.3 Ensemble Methods Related Work

This section explores existing research on ensemble methods, focusing on strategies that leverage multiple neural networks to enhance model robustness, reduce variance, and improve overall predictive performance.

3.3.1 Bagging Predictors

Breiman [5] addressed the problem of how to improve the accuracy of prediction models, especially when the prediction method is unstable. The author proposes the "Bagging predictors" method as a solution to this problem which can give substantial gains in accuracy.

The author does not provide a literature survey but instead focuses on introducing and demonstrating the effectiveness of the "Bagging predictors" method for improving accuracy in predicting numerical outcomes or classes. The author mentions that the method may not work well if the prediction method is already stable or if the bootstrap samples are similar to each other. Additionally, the method may not be suitable for very large datasets due to computational constraints.

The proposed Bagging method by Breiman is designed to improve the accuracy and

robustness of predictive models, particularly decision trees, by combining multiple models trained on different subsets of the training data. The process is divided into different steps. Firstly, the data must be prepared. The preprocess dataset is cleaned and categorical variables are encoded if needed. The dataset is split into two parts: a training set and a testing set. Bootstrap Sampling involves randomly selecting data points from the training set with replacements. The resulting sample may contain duplicate data points. Secondly, Model Training is based on selecting the base model for prediction. The author states that decision trees are a common choice due to their ability to handle both categorical and numerical data, making them versatile for various types of datasets. Thirdly, Aggregating Predictions are used to make predictions on the testing set or new data once all the individual models are trained. It works with two types of problems. One is the classification problem, which use majority voting. Each model votes for a class, and the class with the most votes is chosen as the final prediction. The second one is the regression problem, which can average the predictions made by each model to obtain the final prediction. Before final deployment, tuning parameters is essential as it will optimize the performance of the bagged ensemble. Once the performance of the bagged ensemble is satisfactory, it is ready to be deployed for making predictions on new, unseen data.

The author uses commonly used datasets for classification and regression tasks. For classification, the author used waveform (simulated), breast cancer (Wisconsin), ionosphere, diabetes, glass, soybean, and only heart data from the UCI Machine Learning Repository. For Regression tasks, the author used datasets Boston Housing, Ozone, Friedman 1, Friedman 2, and Friedman 3. The method gives substantial gains in accuracy when it performed the test using classification and regression trees and subset selection in linear regression. It consistently reduced the prediction error compared to a single model. For classification, the reduction rates range from 6 percent to 77 percent, and for regression trees, the rate ranges from 21 percent to 46 percent. It performs well even when the base model is unstable or prone to over-fitting.

The author work aligns with our ensemble by emphasizing bagging to improve the stability and accuracy of predictive models. Here, we applied bagging to generate multiple subsets for training distinct neural networks, ensuring robustness and reducing overfitting—principles consistent with the author approach to handling instability in prediction models.

3.3.2 Other related works on ensemble methods

Street and Kim [30] addressed the challenge of large-scale streaming classification and the limitations of traditional ensemble methods like Boosting and Bagging. They proposed the Streaming Ensemble Algorithm (SEA), which processes data in small blocks to build classifiers. The ensemble size is fixed, and new classifiers are added only if they improve performance, replacing existing ones to maintain a set size of 25 trees. Predictions are combined using majority voting, with accuracy improvements observed in 90% of runs for adult data, 84% for SEER data, and 58% for anonymized data. Their approach aligns with ours in handling large-scale streaming data through batch processing, limited storage, and voting mechanisms for efficient stream classification.

Gomes et al. [11] address several key challenges in data stream learning: concept drift, dynamic data distribution, temporal relationships, and the emergence of new classes. They highlight the limitations of static learning methods and propose a taxonomy of over 60 ensemble algorithms, including OzaBag, DWM, and SEA, to handle evolving data streams. The study identifies issues like scalability, resource constraints, and handling incomplete or imbalanced data. While no new experimental results are provided, the work aggregates and summarizes existing research, offering a comprehensive overview of ensemble methods. Future research is expected to focus on big data stream learning, parallelization, and adapting traditional methods to complex real-world scenarios, such as partially supervised and imbalanced data streams. The authors addressed challenges like scalability, resource constraints, and handling evolving data streams. Our ensemble handles these same challenges

by using a Data Stream Processor (DSP) with limited storage and three neural networks trained on bagged subsets of data.

Bagheri and Gao [4] note that classifier selection in ensemble methods is often overlooked due to high computing costs, and existing works are shifting towards classifier combination. They propose Dynamic Classifier Selection by Divide and Conquer (DCS-DQ) to enhance classifier selection by breaking down complex tasks into binary sub-classifications. Compared with popular ensemble methods and single classifiers, DCS-DQ significantly reduces execution time but only slightly improves accuracy. Evaluated on 14 UCI datasets, the authors suggest future research should focus on incorporating a more effective first guesser classifier and assessing other algorithms, like decision trees or Bayes classifiers, as base learners. Their work aligns with our ensemble as both emphasize efficient classifier selection and performance enhancement using ensemble methods. While DCS-DQ focuses on binary sub-classifications to streamline execution time, our ensemble uses bagging and multiple neural network configurations to optimize processing within limited storage constraints, highlighting a shared goal of resource-efficient classifier combination strategies.

3.4 Chapter Summary

This chapter summarizes the existing work on data streams and ensemble methods related to our research. In the next chapter, we present our data requirements and system overview.

Chapter 4
Data and System Overview

In this chapter, we discuss our proposed strategy, including the data requirements, gathering and generation, and clustering. The chapters structure is as follows: Section 4.1 discusses the data requirements for severe weather, Section 4.2 discusses the data gathering and generation process, Section 4.3 discusses class labeling and rules, and Section 4.4 presents our system architecture and its functionality.

4.1 Data Requirements

In this section, we present and discuss the data requirements for tracking severe weather events. Due to the unavailability of a comprehensive dataset that meets the specific needs of our analysis, it was necessary to curate our own dataset. This custom dataset ensures that we have precise and relevant data tailored to the objectives of our research.

Accurate and timely data, including time, location, and specific conditions, enables emergency services to make informed decisions and allocate resources efficiently [1]. Although different emergency types exist, we chose to focus on severe weather due to its significant impact, frequency, and the need for continued advancements in prediction, preparedness, and response. The following attributes are used to address severe weather situations, focusing on the environmental conditions and areas of emergency evacuation planning. These attributes are not limited to a specific geographical area - they are applicable to any region where severe weather events, such as hurricanes, tornadoes, or heavy storms, might occur. All the values are floating point values except for the Primary ID, which is an integer value.

1. Primary ID: This is a unique value within the table to ensure that no two records share the same identifier.

2. Evacuation centre (City latitude and longitude): These coordinate systems determine the precise location of an evacuation center in the weather data.

3. Temperature: It quantifies the sensation of warmth or chilliness in an object or environment.

4. Precipitation: It refers to any form of water, liquid or solid, that falls from the atmosphere and reaches the ground, including rain, snow, sleet, and hail.

5. Wind speed: It refers to the rate at which air moves horizontally through the atmosphere at any given point.

6. Affected region (Bounding Box): The bounding box represents a defined rectangular area around an affected region within 100km of the evacuation center, using their latitude and longitude coordinates. This area helps in identifying and analyzing the affected region for precise weather impact assessments and evacuation planning.

7. From the affected region to the evacuation centre (Driving Distance): It refers to the total length of a route from one location to another. From our research perspective, it is the length from the centre of an affected area to the evacuation centre.

8. Priority for evacuation (Class Labels:) The class labels are extremely high, high, medium, low, and extremely low. These class labels represent the priority for evacuation.

These attributes were selected after extensive discussions with several individuals from the city corporation in Sylhet, Bangladesh. These individuals work closely with this data and provided an initial list of requirements. They also offered a comprehensive overview of the factors influencing evacuation planning and execution during severe weather events.

Temperature, precipitation, and wind speed are fundamental meteorological parameters that help in assessing the severity and nature of the weather conditions. The driving distance to evacuation centers is essential for planning safe and efficient routes for evacuees. Class

labels indicating the priority for evacuation, derived from neural network outputs, helping in prioritizing areas based on the severity of the threat, ensuring that resources are allocated effectively and those in the most danger are evacuated first. These attributes collectively enable a data-driven approach to managing and responding to severe weather emergencies.

4.2 Data Gathering and Generation

The existing severe weather dataset from Environment Canada [6] does not have the vast range of attributes that we mentioned in the data requirements section. In addition, many of the attributes have missing data. Open-Meteo [26] contains similar attributes such as temperature, precipitation, and wind speed to Environment Canada [6]. Therefore, we obtained temperature, precipitation and wind speed data from Open-Meteo [26]. For particular attributes are complete, but we cannot guarantee its accuracy.

Neither Environment Canada [6] nor Open-Meteo [26] have any data on evacuation centers and driving distance, so it was created manually. We used four cities to act as evacuation centres: Lethbridge, Medicine Hat, Calgary, and Edmonton. For each city, we fetched daily data for two years from 2020-2021 of each of the cities for 731 records and a total of 2924 records. For each city, four towns within 100km of the evacuation centre are chosen. The driving distance between each town and its evacuation center, which was obtained using Google Maps, will be used as the shortest distance. Within 100kms, we chose towns of varying driving distance from their evacuation centre. Table 4.1 refers to town data for Calgary, while Table 4.2 refers to Edmonton, Table 4.3 refers to Lethbridge and Table 4.4 refers to Medicine Hat.

Table 4.1: Town data related to Calgary.

Attributes	Carstairs	Kananaskis	Nanton	Bowden
Latitude	51.5333	51.0482	50.35	51.9167
Longitude	-114.1	-115.022	-113.767	-114.033
Driving Distance (KM)	50.05	65.79	84.57	92.75
Lat1	51.3083	50.8232	50.125	51.6917
Long1	-114.445	-115.367	-114.112	-114.378
Lat2	51.7583	51.2732	50.575	52.1417
Long2	-113.755	-114.677	-113.422	-113.688

Table 4.2: Town data related to Edmonton.

Attributes	Camrose	Lamont	Redwater	Vegreville
Latitude	53.0167	53.75	53.95	53.5
Longitude	-112.817	-112.783	-113.11	-112.05
Driving Distance (KM)	74.87	52.13	51.29	96.08
Lat1	52.7917	53.525	53.725	53.275
Long1	-113.162	-113.128	-113.455	-112.395
Lat2	53.2417	53.975	54.175	53.725
Long2	-112.472	-112.438	-112.765	-111.705

Table 4.3: Town data related to Lethbridge.

Attributes	Cardston	Coaldale	Fort Macleod	Magrath
Latitude	49.1987765	49.733333	49.4173086	49.1987765
Longitude	-113.30186	-112.61666	-113.40312	-112.86855
Driving Distance (KM)	80	12	55	38
Lat1	48.9737	49.5083	49.1923	48.9737
Long1	-113.6468	-112.9616	-113.74812	-113.2135
Lat2	49.4237765	49.958333	49.6423086	49.4237765
Long2	-112.9568	-112.2716	-113.0581	-112.5235

Table 4.4: Town data related to Medicine Hat.

Attributes	Bow Island	Brooks	Maple Creek	Redcliff
Latitude	49.8667	50.5667	49.9167	50.0833
Longitude	-111.367	-111.9	-109.467	-110.783
Driving Distance (KM)	52.42	104.45	88.03	9.04
Lat1	49.6417	50.3417	49.6917	49.8583
Long1	-111.712	-112.245	-109.812	-111.128
Lat2	50.0917	50.7917	50.1417	50.3083
Long2	-111.022	-111.555	-109.122	-110.438

A bounding box is formed around each town to create an affected region. The bounding box is centered on the town and extends approximately 25 kilometers in each direction from the town, forming a 50 km by 50 km square region around the town. The values lat1, long1, lat2, and long2 are defined as the coordinates of the southwestern and northeastern corners of the bounding box around each town. The values 0.225 and 0.345 were obtained using the NOAA Great Circle Distance Calculator [7] by measuring the latitude and longitude differences from a central point. They represent the degree shifts used to define a bounding of distance of approximately 25kms from a central point. To calculate the bounding box, we used the following formulas:

1. **lat1**: This is the latitude of the southwestern corner of the bounding box. It is calculated by subtracting 0.225 degrees from the town's latitude.

$$\text{lat1} = \text{town latitude} - 0.225$$

2. **long1**: This is the longitude of the southwestern corner of the bounding box. It is calculated by subtracting 0.345 degrees from the town's longitude.

$$\text{long1} = \text{town longitude} - 0.345$$

3. **lat2**: This is the latitude of the northeastern corner of the bounding box. It is calcu-

lated by adding 0.225 degrees to the town's latitude.

$$\text{lat2} = \text{town latitude} + 0.225$$

4. **long2**: This is the longitude of the northeastern corner of the bounding box. It is calculated by adding 0.345 degrees to the town's longitude.

$$\text{long2} = \text{town longitude} + 0.345$$

For example, if a town is located at a latitude of 53.5 and a longitude of -113.5, the bounding box coordinates would be calculated as follows:

- **lat1** = 53.5 - 0.225 = 53.275
- **long1** = -113.5 - 0.345 = -113.845
- **lat2** = 53.5 + 0.225 = 53.725
- **long2** = -113.5 + 0.345 = -113.155

After completing the bounding box creation for each town for the individual cities, we merged all the data records, totaling 2,924 records for each city. Therefore, the total number of data records across the four cities is 11,696 records.

4.3 Class Labelling

Clustering techniques K-means and DBSCAN are used to assign predefined categories or class labels to our data points for classification purposes. Tables 4.5 and 4.6 present the final outputs after completing the clustering process for class labeling using K-means and DBSCAN, respectively.

4.3.1 K-means Clustering

We applied the K-means to identify distinct groups within our dataset. We chose $k = 2$, 3, and 5 to explore different levels of clustering detail within our dataset. Starting with $k = 2$ allows us to observe the most distinct binary separation, while $k = 3$ introduces a moderate level of complexity, potentially revealing more nuanced groupings. By examining $k = 5$, we aim to identify even finer subgroups, ensuring a thorough understanding of the inherent patterns and relationships within the dataset. We calculate the compactness of a cluster using some-of-squares. After we completed our experiments with different cluster sizes, we decided that $k = 5$ clusters gave the most compact clusters. A higher value of k may have produced more compact clusters. However, we do not have many evacuation priorities, which is why we did not go higher than $k = 5$. Therefore, we have chosen this configuration for our class label classification.

For each cluster we use the formula $(\text{long1}_{\text{max}} - \text{long0}_{\text{min}}) \times (\text{lat1}_{\text{max}} - \text{lat0}_{\text{min}})$, where we use the min and max values from the bounding data to calculate the area size for the entire cluster. We then calculate the average values for the attribute, temperature, precipitation, and driving distance, using their minimum and maximum values. These averages, along with the calculated area size, are specifically used to assign class labels (as shown in Table 4.5). Wind speed was excluded from this analysis because there is no significant variance in wind speed between the clusters, which is why we did not use it.

Once the clusters have been formed, for each cluster the records in it are assigned the same class label. However, simply assigning class labels does not make the data balanced. Data balancing can be achieved through techniques such as oversampling the minority class and undersampling the majority class, ensuring that each class has a similar number of data points. Applying random over sample and random under sample techniques [34] for above mentioned data records and we obtained 24,830 oversampled data records and 3,615 undersampled data records. We perform all of our experiments by using oversampled data records because oversampled data preserves the original data distribution, leading to more

generalizable and accurate results. Undersampling, on the other hand, risks losing important information.

4.3.2 DBSCAN Clustering

We also applied the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm to cluster the data for class-labelling purposes. Unlike K-means, DBSCAN does not require specifying a predefined number of clusters (k). Instead, DBSCAN identifies clusters based on the density of data points, where dense regions form clusters and sparse regions are considered noise or outliers. We experimented with different parameters for DBSCAN, such as the epsilon (ε) value, which defines the maximum distance between two points to be considered neighbors, and the minimum number of points required to form a dense region (*minPts*).

After running DBSCAN, the algorithm formed 6 clusters and assigned records to various clusters and marked some records as outliers. These outliers were merged into the nearest cluster based on the city latitude and longitude to ensure all data points were accounted for. During the clustering process, we found one cluster that only had 12 records assigned to it, which is cluster 1. Given the small size of this cluster, we decided to merge it with another cluster (cluster 0) to create a more robust group. Cluster 0 had similar evacuation and affected region data records to those of Cluster 1, indicating a potential overlap in characteristics or patterns between the two clusters. This combined cluster was renamed to cluster 10. For the calculation of area size and the subsequent class label classification, we applied the same formula and process used in our K-means analysis. This included calculating the bounding box around all towns in the cluster and determining the average values for the attributes temperature, precipitation, and driving distance. The results of this process and the final class-label assignments are summarized in Table 4.6.

The dataset has an imbalanced class distribution, which necessitates the application of balancing techniques before model training. After applying the balancing techniques, we

obtained 29,210 data records from oversampling and 3,655 data records from undersampling. We used oversampled DBSCAN clusters for our experiments. The variation in the number of oversampled data records between K-means and DBSCAN is the reason for the difference in cluster sizes. K-means tend to create more uniform, spherical clusters, while DBSCAN forms clusters of varying densities and shapes. This leads to different imbalances in the cluster sizes, which leads to over-sampling generating more data for DBSCAN, where some clusters might be sparser or smaller than others, requiring more data to balance.

4.3.3 Class Labelling Rules

Our if-then rules for calculating the class labels by using both K-means and DBSCAN clustered data are shown in Figure 4.1. The variables **temperature_average**, **precipitation_average**, and **driving_distance_average** represent the average values of their respective attributes (temperature, precipitation, and driving distance) calculated from the minimum and maximum values within the bounding box around each town. The *area_size* variable is derived from the clusters bounding box coordinates using the formula referred to in Section 4.3.1.

The other threshold variables, such as **temp_threshold_extremely_high**, **precip_threshold_extremely_high**, **distance_threshold_extremely_high**, **area_threshold_high**, **temp_threshold_extremely_low**, **precip_threshold_extremely_low**, **distance_threshold_extremely_low**, and **area_threshold_extremely_low**, represent predefined thresholds for each attribute and the area size. Although the same rules are applied to both clustered sets, their values depend on the final cluster properties and are different for K-means and DBSCAN. These thresholds are used to determine the class labels, such as **extremely_high**, **high**, **medium**, **extremely_low**, and **low**, based on whether the combined attribute values and area size exceed these set thresholds.

```
if (row['temperature_average'] > temp_threshold_extremely_high and
    row['precipitation_combined'] > precip_threshold_extremely_high and
    row['driving_distance_average'] > distance_threshold_extremely_high
        and
    row['area_size'] > area_threshold_extremely_high):
    return 'extremely_high'
elif (row['driving_distance_average'] > distance_threshold_high and
      row['precipitation_average'] > precip_threshold_extremely_high and
      row['temperature_average'] > temp_threshold_high and
      row['area_size'] > area_threshold_high):
    return 'high'
elif (row['precipitation_average'] > precip_threshold_medium and
      row['driving_distance_combined'] > distance_threshold_medium and
      row['temperature_average'] > temp_threshold_medium and
      row['area_size'] > area_threshold_medium):
    return 'medium'
elif (row['area_size'] > area_threshold_extremely_low and
      row['precipitation_average'] > precip_threshold_extremely_low and
      row['driving_distance_average'] > distance_threshold_extremely_low
          and
      row['temperature_average'] > temp_threshold_extremely_low):
    return 'extremely_low'
else:
    return 'low'
```

Figure 4.1: Assigning class labels based on average attributes and area size.

The low-priority class was characterized by mostly sub-zero temperatures and moderate precipitation, while the extremely low-priority class exhibited higher precipitation but above-zero temperatures. The medium-priority class displayed a mix of sub-zero and above-zero temperatures, with precipitation levels varying across the spectrum. The high-priority class was defined by predominantly above-zero temperatures and significantly higher levels of precipitation. Finally, the extremely high-priority class represented regions with the most extreme conditions, typically warmer temperatures and the highest precipitation levels, indicating areas that require immediate attention.

Table 4.5: K-means Clustered Data.

Attributes	Cluster-0	Cluster-1	Cluster-2	Cluster-3	Cluster-4
City Latitude Min	49.66608	49.66608	49.66608	49.66608	49.66608
City Latitude Max	53.532513	53.532513	53.532513	50.017574	50.017574
City Longitude Min	-114.03227	-114.03227	-114.03227	-112.88928	-112.88928
City Longitude Max	-110.73299	-110.73299	-110.73299	-110.73299	-110.73299
Temperature Min	-37.6	-0.6	-37.6	-27.9	7.1
Temperature Max	6.1	30.1	30.1	7.5	30.1
Precipitation Min	0	0	0	0	0
Precipitation Max	23.1	60.5	60.5	23.1	60.5
lat1 Min	48.9737765	48.9737765	48.9737765	49.508333	49.508333
lat1 Max	53.725	53.725	53.275	49.8583	49.8583
long1 Min	-115.367	-115.367	-114.378	-112.9616	-112.9616
long1 Max	-111.712	-111.712	-109.812	-111.128	-111.128
lat2 Min	49.4237765	49.4237765	49.4237765	49.958333	49.958333
lat2 Max	54.175	54.175	53.725	50.3083	50.3083
long2 Min	-114.677	-114.677	-113.688	-112.2716	-112.2716
long2 Max	-111.022	-111.022	-109.122	-110.438	-110.438
Area Size	15.421627	15.421627	18.415742	0.9148958	0.9148958
Driving Distance Min	38.00	38.00	74.87	9.04	9.04
Driving Distance Max	84.57	65.79	104.45	12.00	12.00
Class Label	medium	high	extremely high	low	extremely low

Table 4.6: DBSCAN Clustered Data.

Attributes	Cluster-2	Cluster-3	Cluster-4	Cluster-5	Cluster-10
City Latitude Min	53.532513	50.017574	50.017574	50.017574	49.66608
City Latitude Max	53.532513	50.017574	50.017574	50.017574	51.072056
City Longitude Min	-113.4034	-110.7329	-110.7329	-110.7329	-114.0322
City Longitude Max	-113.4034	-110.7329	-110.7329	-110.7329	-112.8892
Temperature Min	-37.6	-27.2	-27.2	-27.2	-34
Temperature Max	29.4	30.1	30.1	30.1	29.1
Precipitation Min	0	0	0	0	0
Precipitation Max	32	27.7	24.4	24.4	41.9
lat1 Min	52.7917	49.6417	50.3417	49.6917	48.9737765
lat1 Max	53.725	49.8583	50.3417	49.6917	51.6917
long1 Min	-113.455	-111.712	-112.245	-109.812	-115.367
long1 Max	-112.395	-111.128	-112.245	-109.812	-112.9616
lat2 Min	53.2417	50.0917	50.7917	50.1417	49.423776
lat2 Max	54.175	50.3083	50.7917	50.1417	52.1417
long2 Min	-112.765	-111.022	-111.555	-109.122	-114.677
long2 Max	-111.705	-110.438	-111.555	-109.122	-112.2716
Area Size	2.420775	0.8492484	0.3105	0.3105	9.8057781
Driving Distance Min	51.29	9.04	104.45	88.03	12
Driving Distance Max	96.08	52.42	104.45	88.03	92.75
Class Label	medium	extremely low	extremely high	high	low

4.4 System Architecture

In this section, we present our proposed system architecture, as illustrated in Figure 4.2. Given the approaches summarized earlier, our work is based on the strategy proposed by King and Osborn reffered in Chapter 3, Section 3.1. We have chosen to focus on the bagging predictors method and artificial neural networks because they offer robust performance and improved accuracy through ensemble learning and nonlinear modeling capabilities.

Our ensemble consists of three neural networks and a data stream processor (DSP) with limited storage. We have four sensors - one per evaluation centre - that send information to the DSP. Upon receiving a certain number of objects or tuples (*S*), the DSP processes the set of S objects before discarding it. We have configured three different neural network architectures with an input layer, an output layer and varying numbers of hidden layers. The first architecture consists of 2 hidden layers, each with 50 neurons. The second architecture has 3 hidden layers with 100, 50, and 25 neurons, respectively. The third architecture features 4 hidden layers with 200, 100, 50, and 25 neurons, respectively. The number of epochs is a hyperparameter that needs to be chosen before the training process starts, and in this case, we set it to 10 for training the neural networks. Also, we employed the Adam optimizer for optimization and used the Hyperbolic Tangent (tanh) as the activation function.

In the DSP, each batch of data is divided into training and testing sets. Specifically, for a batch of S=100 records, 90 records are allocated for training and 10 for testing. To process the training subset, we apply the bagging technique [5], which involves randomly sampling with replacement from the 90 training records to create three distinct subsets, each consisting of 60 records. This method allows some records to appear in multiple subsets while others may not be selected at all. Each of these bagged subsets is used to train separate neural networks, resulting in three models capable of making predictions.

Once the models are trained, they are evaluated on the 10 testing records from each batch. Predictions from all three models are collected, and a majority voting process is

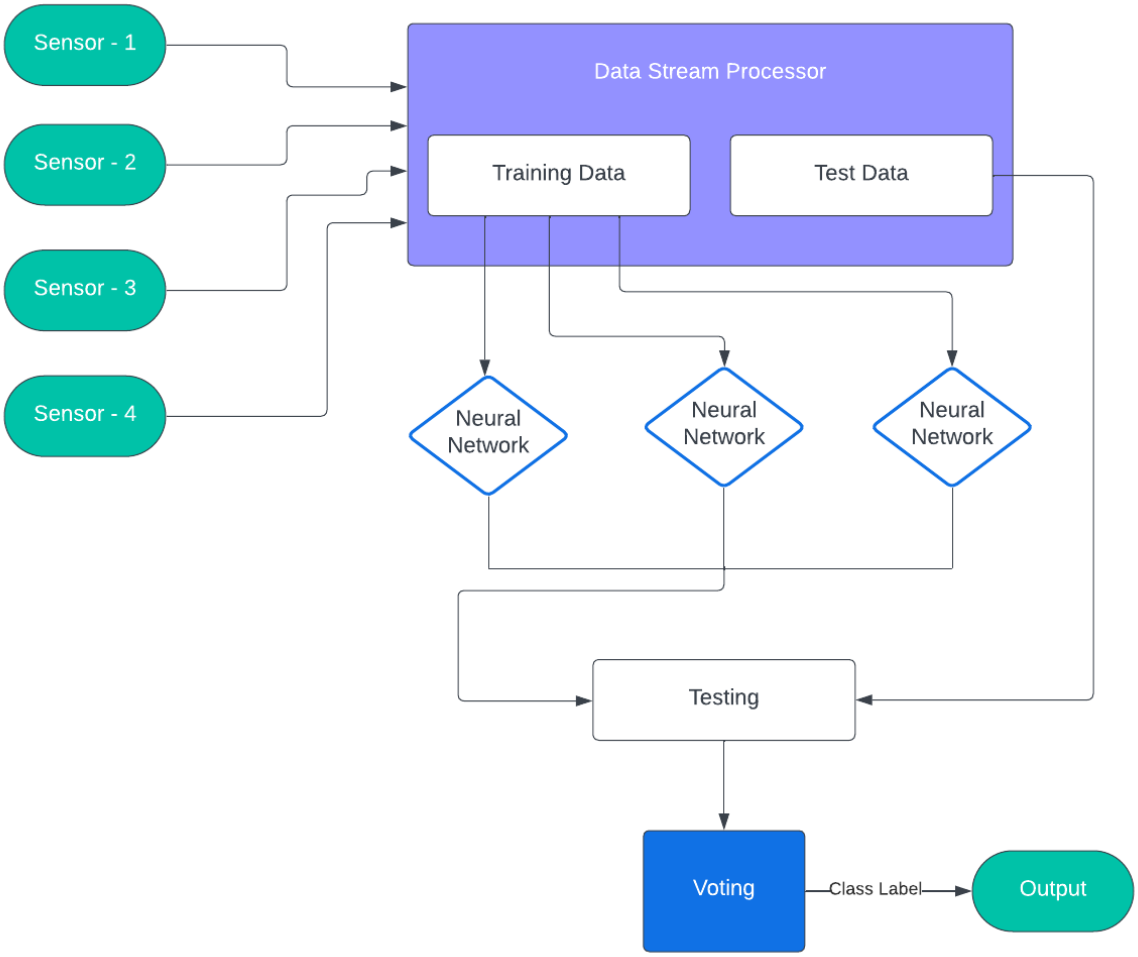

Figure 4.2: System Architecture.

applied to determine the final prediction for each testing record. If a tie occurs in the voting process, the class with the highest priority is selected. The batch accuracy is then computed based on the number of correct predictions from the combined model outputs.

Algorithm 1 is designed to process every single batch of 100 records, preparing the data for training and testing. Given an input batch, the algorithm divides it into a training and testing set, then performs bagging to create three distinct training subsets by sampling from the training data. The final output includes the bagged training subsets and the testing set for training and evaluation.

Algorithm 1: Processing Data for Training with Bagging

Input: B = batch of 100 records

Output: Three distinct subsets F_{j1}, F_{j2}, F_{j3} and testing set E for batch B

$T \leftarrow$ **GetTrainingRecords**$(B, 90)$;

// Get the first 90% of B for training

$E \leftarrow$ **GetTestingRecords**$(B, 10)$;

// Get the last 10% of B for testing

for $j = 1$ ***to*** 3 **do**

    $F_j \leftarrow$ **BagSample**$(T, 60)$;

    // Bagging: randomly sample 60% from T with replacement

    return F_j;

return E;

Algorithm 2 uses the three bagged subsets generated by Algorithm 1 to train three models. These models make predictions on the testing set, and the final prediction for each record is determined by majority voting. The batch accuracy is calculated as the ratio of correct predictions to the total number of testing records, providing an evaluation of the models performance.

Algorithm 1 and Algorithm 2 are repeated for every incoming batch in the DSP, ensuring consistent processing as new data arrives.

Algorithm 2: Batch Training and Evaluation

Input: B = batch of 100 records

Output: Accuracy for the single batch

$\{F_1, F_2, F_3\}, E \leftarrow$ **Algorithm 1**(B);

// Use Algorithm 1 to obtain three bagged subsets and testing records for batch

$CorrectPredictions \leftarrow 0$;

for $j = 1$ ***to*** 3 **do**

    $M_j \leftarrow$ **TrainModel**(F_j);

    // Train model M_j **on bagged subset** F_j

foreach r_k ***in*** E **do**

    $Predictions \leftarrow [M_1(r_k), M_2(r_k), M_3(r_k)]$;

    // Get predictions from all models

    $FinalPrediction \leftarrow$ **MajorityVote**$(Predictions)$;

    // Determine final prediction using majority voting

    if $FinalPrediction =$ ***TrueLabel***(r_k) **then**

        $CorrectPredictions \leftarrow CorrectPredictions + 1$;

        // Increment correct predictions counter

$BatchAccuracy \leftarrow \frac{CorrectPredictions}{|E|}$;

// Calculate accuracy as ratio of correct predictions to total test records

return $BatchAccuracy$;

// Return accuracy for the single batch

4.5 Chapter Summary

In this chapter, we presented our overall data requirements, gathering, and generation process, and the architecture and functionality of our system. In the next chapter, we will discuss our evaluation process and outcomes.

Chapter 5
Experiments and Evaluations

In this chapter, we present the categories of experiments and their results, along with the specifications of software and hardware. The rest of the chapter is organized as follows. First, Section 5.1 presents information about software and hardware-related specifications. Then, Section 5.2 discusses our three experiments. Furthermore, Section 5.3 discusses the performance metrics. Finally, Section 5.4 describes our three experiment results.

5.1 Implementation Details

We present the software and hardware specifications that played a crucial role in our research, ensuring the reliability and reproducibility of our results.

5.1.1 Software and Testing Specification

We implemented three experiments such as random data, interleaved clustered data and interleaved date wise data to perform testing for the ensemble approach. We used Python 3.11.2 as the programming language. For data manipulation and analysis, we utilized Pandas for handling structured data in DataFrames and NumPy for support with large, multi-dimensional arrays and matrices. Our deep learning models were built and trained using PyTorch 2.2.1. For clustering, we used Scikit-learn (Sklearn), and to address imbalances in the datasets, we employed Imbalanced-learn (Imblearn), which are classical machine learning algorithms. The following testing environment, along with the programs and data, is used for this research. The Hardware specifications are as follows. The main computing device in the testing environment is a Lenovo IdeaPad 3, equipped with the AMD Ryzen 7 5825U CPU with Radeon Graphics and 2.00 GHz, RAM: 16 GB, Storage: 512 GB Intel SSD.

5.2 Experimental Strategy

In the experiments, we evaluated the performance of our proposed system architecture under three experimental setups. The first two experiments were conducted using the oversampled clustered data records, and the third experiment was conducted with a clustered dataset that maintains the original time sequence of data records. The three experiments are structured as follows:

1. Random Data: The primary objective of this experiment is to incrementally train and test the ensemble on a random order of the data. To ensure variety in the data, the datasets are shuffled before training and testing. The randomness of the data order aims to ensure that no inherent patterns from the data order influences the results. For this experiment, we used oversampled clustered data records of 24,830 for K-means 29,210 for DBSCAN.

2. Interleaved Clustered Data: Interleaving refers to a method of arranging data from multiple individual sets into one set in an interleaved manner. For this work, the records from different clusters are mixed in a specific pattern rather than being grouped by cluster. Data was processed by interleaving records from each cluster, ensuring that the first two records from each cluster were fetched sequentially, and this pattern continued until all records were included. For example, we have five clusters each with four rows such as Cluster 1: A1, A2, A3, A4, Cluster 2: B1, B2, B3, B4, Cluster 3: C1, C2, C3, C4, Cluster 4: D1, D2, D3, D4 and Cluster 5: E1, E2, E3, E4. Now to interleave, we take the first 2 rows from Cluster 1 (A1, A2), followed by the first 2 rows from Cluster 2 (B1, B2), the first 2 rows from Cluster 3 (C1, C2), the first 2 rows from Cluster 4 (D1, D2), and finally the first 2 rows from Cluster 5 (E1, E2). This process is repeated with the remaining rows. Finally, the resulting interleaved data would be A1, A2, B1, B2, C1, C2, D1, D2, E1, E2, A3, A4, B3, B4, C3, C4, D3, D4, E3, E4. By interleaving records from each cluster, we ensure a balanced representation of clusters throughout the dataset and each batch that is processed.

If we do not use interleaving and instead use all the data from each cluster one at a time, several issues might arise:

(a) Training Bias: Models trained on grouped data might become biased, learning patterns specific to each cluster rather than generalizing across the entire dataset.

(b) Overfitting: The model might overfit to the unique characteristics of each cluster, reducing its ability to perform well on new, unseen data.

(c) Poor Generalization: The model might face this issue when the model fails to capture the underlying patterns in the data and thus performs poorly on unseen data. While the order of the data does not directly cause poor generalization, it can influence the learning process, especially if the model is sensitive to the sequence in which it sees the data.

In this experiment, we used the oversampled data records of 24,830 for K-means and 29,210 records for DBSCAN.

3. Interleaved Data by Date: We utilized the original date-wise sequence of records for this experiment. We have a clustered dataset of 11,696 records. After applying DBSCAN and K-means to the data, each of the cities data needed to be put back into order by date and then interleaved similarly to data set two. Each of the cities, including their individual towns, contains 731 records, a total of 2,924 records. From each town, we fetched the first record sequentially until all records were processed. This approach ensures that the temporal dependencies and patterns in the data are preserved. By using this experiment, the proposed models perform over time and across different affected areas.

The experiments conducted utilized specific input parameters that were critical for data processing, model training, and evaluation, as discussed in the Chapter 4. The data was shuffled, as determined by the shuffle_data parameter set to True, with a random_state value of 42. The dataset was then divided into batches, each containing 100 records. The data

was further split into training and testing sets, with 90% allocated for training (train_size = 0.9) and 10% for testing (test_size = 0.1). Within each batch, three subsets of 60 samples (train_subset_size = 60) were used for training each of the neural networks, with sampling performed with replacement (replace = True). The models were trained over 10 epochs, with the training process iterating over each batch. To find the best-performing model, we experimented with ten different learning rates: 0.01, 0.001, 0.0001, 0.05, 0.005, 0.0005, 0.08, 0.008, 0.0008, and 0.15.

5.3 Performance Metrics

Performance metrics are quantitative measures used to evaluate the effectiveness and efficiency of a machine learning model. These metrics help to understand how well a model performs in making predictions based on the data it was trained on. Cumulative metrics are running totals that accumulate the results over time, giving insight into how the models performance evolves as more data is processed. Average cumulative metrics take the cumulative results at each point and average them, providing a single summary statistic that reflects the models overall performance across the entire data stream. All of our proposed experiments used the following metrics [29]:

1. Accuracy: Accuracy is the ratio of correctly predicted instances to the total instances in the dataset. It gives an overall effectiveness of the model. Accuracy is a useful metric when the classes in the dataset are balanced.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TP (True Positives): The number of correct positive predictions.
- TN (True Negatives): The number of correct negative predictions.

- FP (False Positives): The number of incorrect positive predictions.
- FN (False Negatives): The number of incorrect negative predictions.

2. Precision: Precision is the ratio of correctly predicted positive instances to the total predicted positive instances. It indicates the quality of positive predictions made by the model.

$$\text{Precision} = \frac{TP}{TP+FP}$$

3. Recall: Recall is the ratio of correctly predicted positive instances to all instances in the actual class. It measures the models ability to capture all relevant cases.

$$\text{Recall} = \frac{TP}{TP+FN}$$

5.4 Experimental Results

In this section, we evaluated the classification performance with an ensemble of neural networks. Accuracy, Precision, and Recall were used to assess the cumulative performance. We performed all the tests on the K-means and DBSCAN clustered data.

5.4.1 Experiment 1: Random Data

Tables 5.1 and 5.2 summarize the performance metrics for the K-means clustered data, showing the results for different learning rates for the last batch and the averages across all batches, respectively. Tables 5.3 and 5.4 present the corresponding results for DBSCAN clustered data.

For the K-means clustered data, the performance in the last batch was notably higher when using a lower learning rate. Specifically, a learning rate of 0.0001 resulted in the highest performance metrics, achieving 80.00% Accuracy and 70.00% Recall. However,

69.33% Precision is lower when compared to the other two metrics (Table 5.1). Precision is critical because it reflects the proportion of true positives out of all predicted positives, ensuring that the model minimizes false positives. In contexts where misclassifications carry significant costs, Precision plays a key role in determining model effectiveness, particularly when high Accuracy and Recall may still overlook the impact of false positives. While the last batch with a learning rate of 0.0001 achieved high performance, the averages for this learning rate are significantly lower, with 50.93% Accuracy, 45.92% Precision, and 48.45% Recall (Table 5.2). However, compared to other learning rates, 0.0001 has the highest performance in terms of averages. Learning rates, such as 0.001 and 0.0008, also showed good performance in the last batch but were lower than 0.0001. The results are summarized in Figure 5.1, where the performance of the K-means clustered data for the last batch and the average of all batches at a learning rate of 0.0001 are visualized.

For DBSCAN clustered data, the last batch with a learning rate of 0.0001 achieved perfect scores across all metrics: 100.00% Accuracy, Precision, and Recall (Table 5.3). This is an exceptional outcome, and when we examine the averages for the same learning rate, we find that they are also relatively high at 72.66% Accuracy, 65.21% Precision, and 70.59% Recall (Table 5.4). This suggests that, with a learning rate of 0.0001, it performs exceptionally well in individual instances but performance consistency across multiple batches for DBSCAN clustered data is below 80%. In contrast, other learning rates that performed well in the last batch, such as 0.001, showed a significant drop in the averages, indicating less stable performance. Figure 5.2 visualizes the trends and results in DBSCAN clustered data for the last batch and the average of all batches.

Table 5.1: K-means last batch results for random data.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	16 seconds	50.00%	28.89%	30.00%
0.001	15 seconds	70.00%	46.67%	50.00%
0.0001	**15 seconds**	**80.00%**	**69.33%**	**70.00%**
0.05	16 seconds	40.00%	39.34%	37.74%
0.005	16 seconds	30.00%	47.59%	30.00%
0.0005	16 seconds	60.00%	53.33%	60.00%
0.08	17 seconds	40.00%	40.49%	33.42%
0.008	15 seconds	50.00%	28.89%	30.00%
0.0008	16 seconds	70.00%	46.67%	50.00%
0.15	14 seconds	40.00%	37.70%	48.84%

Table 5.2: K-means all the batches cumulative results for random data.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	18 seconds	38.79%	38.69%	34.90%
0.001	15 seconds	45.95%	40.71%	43.55%
0.0001	**17 seconds**	**50.93%**	**45.92%**	**48.45%**
0.05	16 seconds	38.32%	40.57%	34.52%
0.005	18 seconds	38.84%	38.83%	35.77%
0.0005	15 seconds	48.61%	43.81%	47.22%
0.08	17 seconds	37.70%	39.51%	34.86%
0.008	16 seconds	38.92%	39.23%	35.58%
0.0008	16 seconds	44.00%	39.76%	42.29%
0.15	15 seconds	37.85%	39.79%	35.07%

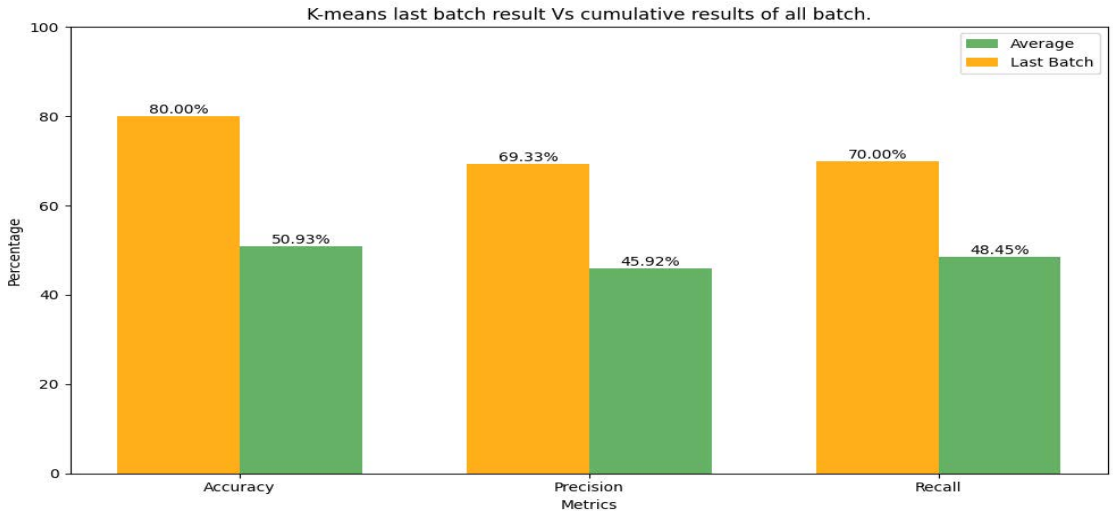

Figure 5.1: K-means result of random data for learning rate 0.0001.

Table 5.3: DBSCAN last batch results for random data.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	19 seconds	30.00%	38.32%	25.00%
0.001	19 seconds	70.00%	62.50%	60.50%
0.0001	**19 seconds**	**100.00%**	**100.00%**	**100.00%**
0.05	18 seconds	39.32%	45.97%	25.00%
0.005	18 seconds	30.00%	38.17%	25.00%
0.0005	18 seconds	70.00%	52.00%	50.00%
0.08	18 seconds	30.37%	31.14%	34.97%
0.008	18 seconds	30.00%	30.96%	25.00%
0.0008	17 seconds	70.00%	62.50%	58.50%
0.15	18 seconds	30.00%	38.43%	25.00%

Table 5.4: DBSCAN all the batches cumulative results for random data.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	14 seconds	38.65%	39.78%	34.44%
0.001	16 seconds	53.57%	44.32%	52.61%
0.0001	**18 seconds**	**72.66%**	**65.21%**	**70.59%**
0.05	19 seconds	38.69%	40.82%	34.61%
0.005	15 seconds	39.33%	39.73%	35.14%
0.0005	17 seconds	64.97%	55.29%	63.02%
0.08	17 seconds	37.70%	39.70%	34.97%
0.008	16 seconds	38.13%	38.88%	34.69%
0.0008	18 seconds	51.92%	43.30%	51.31%
0.15	16 seconds	38.17%	40.01%	34.79%

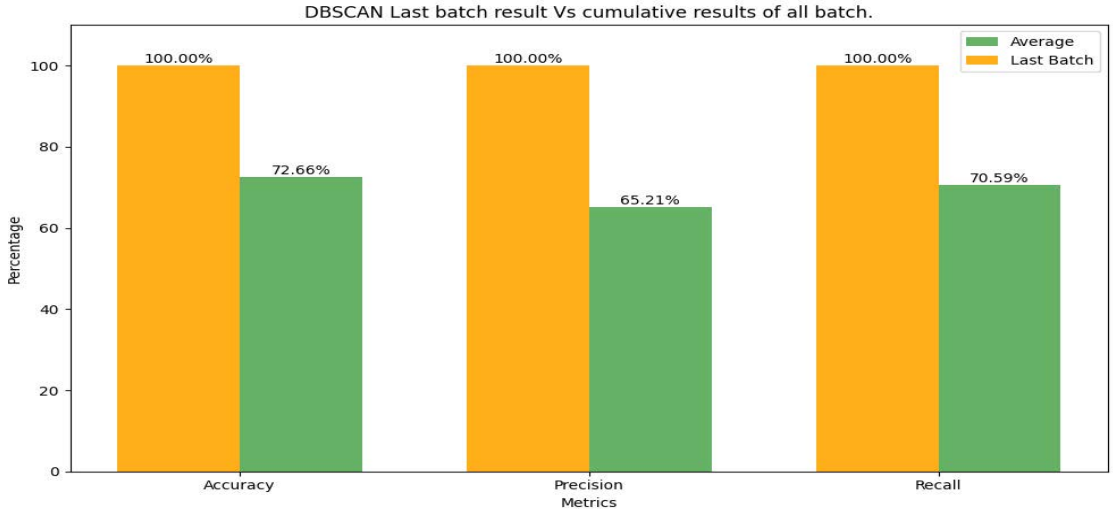

Figure 5.2: DBSCAN results of random data for learning rate 0.0001.

5.4.2 Experiment 2: Interleaved Clustered Data

Tables 5.5 and 5.6 present the performance metrics for K-means clustered data, showing the last batch results and averages across all batches, respectively. Tables 5.7 and 5.8 provide the corresponding results for DBSCAN clustered data.

The analysis of interleaved clustered data using K-means revealed that the last batch with a learning rate of 0.0001 achieved the highest metrics: 80.00% Accuracy, 83.33% Precision, and 80.00% Recall (Table 5.5). The average results for the same learning rate are also strong, with 76.47% Accuracy, 69.80% Precision, and 76.47% Recall (Table 5.6). This indicates that this learning rate is consistently reliable across different instances. Learning rates like 0.0005 and 0.0008, which performed well in the last batch, have lower average metrics, suggesting that while they may perform well in specific instances, their overall performance is less consistent. Figure 5.3 visualizes the trends, showing the K-means interleaved cluster data results for the learning rate of 0.0001.

For DBSCAN clustered data, the last batch again showed that a learning rate of 0.0001 is highly effective, achieving perfect scores (Table 5.7). The average results for this learning rate are also high, with 89.80% Accuracy, 86.81% Precision, and 89.80% Recall (Table 5.8), confirming that this learning rate offers both high performance and consistency across multiple batches. Other learning rates, such as 0.005 and 0.0005, which showed relatively good last-batch performance, had lower averages, indicating less reliability across instances. The trends are depicted in Figure 4.4, which shows the DBSCAN interleaved cluster data results for the last batch and the average of all batches at a learning rate of 0.0001.

Table 5.5: K-means last batch interleaved cluster data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	15 seconds	54.26%	43.17%	47.78%
0.001	15 seconds	30.00%	59.57%	30.00%
0.0001	**15 seconds**	**80.00%**	**83.33%**	**80.00%**
0.05	15 seconds	44.47%	40.68%	48.85%
0.005	15 seconds	49.95%	42.08%	42.80%
0.0005	15 seconds	60.00%	61.33%	60.00%
0.08	15 seconds	41.28%	49.80%	40.68%
0.008	15 seconds	48.83%	44.86%	49.52%
0.0008	15 seconds	30.00%	48.15%	30.00%
0.15	14 seconds	47.12%	49.78%	42.67%

Table 5.6: K-means cumulative all the batches interleaved cluster data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	19 seconds	50.27%	46.70%	50.06%
0.001	15 seconds	48.96%	49.06%	48.90%
0.0001	**17 seconds**	**76.47%**	**69.80%**	**76.47%**
0.05	18 seconds	44.33%	42.20%	44.22%
0.005	18 seconds	47.08%	44.49%	47.28%
0.0005	17 seconds	72.09%	66.87%	72.09%
0.08	16 seconds	44.66%	44.31%	44.67%
0.008	19 seconds	46.47%	44.36%	46.31%
0.0008	15 seconds	51.10%	45.17%	51.10%
0.15	16 seconds	44.79%	45.12%	44.90%

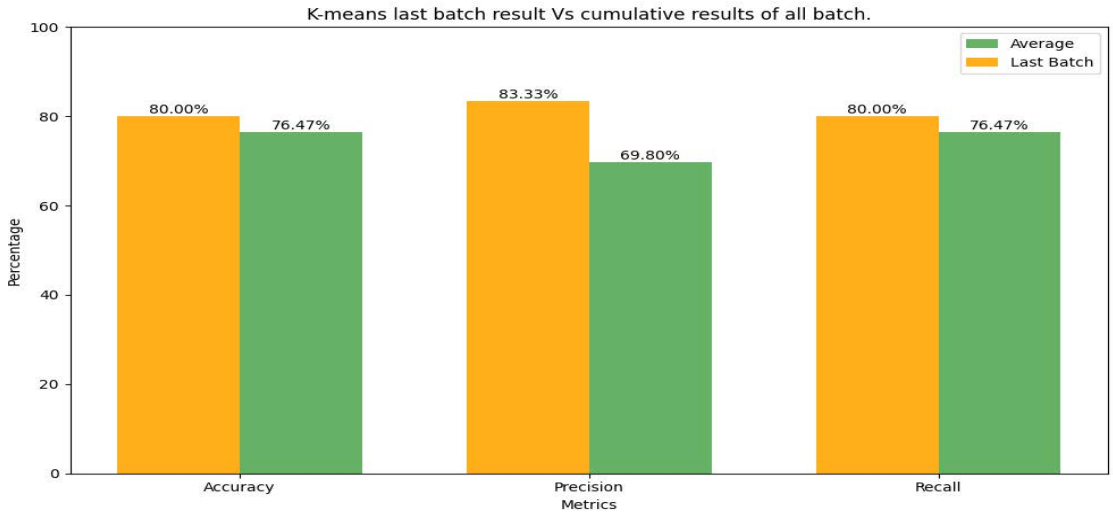

Figure 5.3: K-means interleaved cluster data result for learning rate 0.0001.

Table 5.7: DBSCAN last batch interleaved cluster data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	18 seconds	70.96%	66.40%	66.87%
0.001	18 seconds	64.22%	66.06%	65.68%
0.0001	**18 seconds**	**100.00%**	**100.00%**	**100.00%**
0.05	18 seconds	62.47%	67.31%	61.39%
0.005	18 seconds	72.75%	74.57%	68.99%
0.0005	18 seconds	67.99%	68.34%	73.22%
0.08	18 seconds	69.77%	70.40%	74.06%
0.008	18 seconds	68.49%	62.16%	68.37%
0.0008	17 seconds	62.96%	64.88%	67.74%
0.15	19 seconds	64.73%	70.50%	69.82%

Table 5.8: DBSCAN cumulative all the batches interleaved cluster data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	13 seconds	56.09%	55.83%	55.98%
0.001	14 seconds	56.48%	56.61%	56.33%
0.0001	**18 seconds**	**89.80%**	**86.81%**	**89.80%**
0.05	17 seconds	47.84%	47.62%	47.99%
0.005	15 seconds	56.54%	55.93%	56.09%
0.0005	16 seconds	67.30%	64.82%	66.88%
0.08	16 seconds	45.69%	45.84%	45.48%
0.008	18 seconds	56.80%	55.63%	56.44%
0.0008	14 seconds	66.74%	63.25%	66.67%
0.15	18 seconds	44.95%	44.89%	45.10%

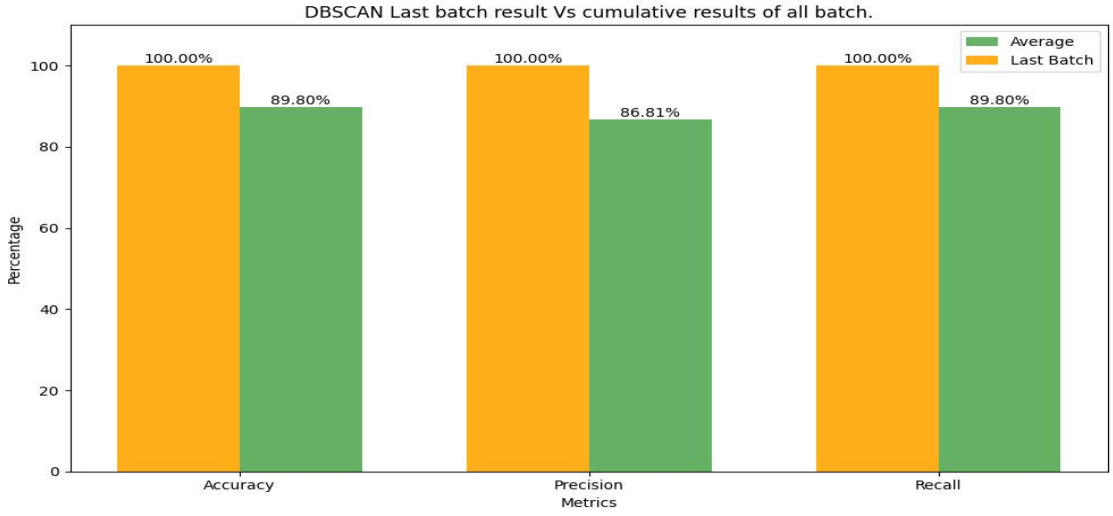

Figure 5.4: DBSCAN interleaved cluster data result for learning rate 0.0001.

5.4.3 Experiment 3: Interleaved Date By Data

Tables 5.9 and 5.10 present the performance metrics for K-means clustered data, showing the last batch results and the averages across all batches, respectively. Tables 5.11 and 5.12 provide the corresponding results for DBSCAN clustered data.

In the case of interleaved date data with K-means clustering, the last batch with a learning rate of 0.0001 achieved the highest metrics, with 70.00% Accuracy, 52.38% Precision, and 58.33% Recall (Table 5.9). The average results of all batches for this learning rate have good performance, with 64.53% Accuracy, 51.06% Precision, and 49.87% Recall (Table 5.10), indicating consistent effectiveness. Learning rates such as 0.0005, which showed decent last-batch performance, have slightly lower averages, pointing to variability in performance across instances. The trends and results are visualized in Figure 5.5, which shows the K-means interleaved date data results for the last batch and the average of all batches at a learning rate of 0.0001.

For DBSCAN clustered data, the last batch with a learning rate of 0.0001 achieved the best metrics, with 80.00% Accuracy, 53.33% Precision, and 60.00% Recall (Table 5.11). The average results of all batches are 65.17% Accuracy, 61.13% Precision, and 59.39% Recall (Table 5.12), indicating that this learning rate is effective and consistent. Other learning rates that performed well in the last batch, such as 0.001, showed a significant drop in averages, indicating lower stability across instances. The trends are depicted in Figure 5.6, which shows the DBSCAN interleaved date data results for the last batch and the average of all batches at a learning rate of 0.0001.

Table 5.9: K-means last batch interleaved date by data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	7 seconds	40.00%	43.39%	33.33%
0.001	7 seconds	40.00%	39.06%	33.33%
0.0001	**6 seconds**	**70.00%**	**52.38%**	**58.33%**
0.05	6 seconds	40.00%	31.24%	33.33%
0.005	7 seconds	40.00%	46.48%	33.33%
0.0005	6 seconds	40.00%	49.82%	33.33%
0.08	7 seconds	40.00%	30.24%	33.33%
0.008	7 seconds	40.00%	48.88%	33.33%
0.0008	7 seconds	40.00%	47.72%	33.33%
0.15	7 seconds	40.00%	40.74%	33.33%

Table 5.10: K-means cumulative all the batches interleaved date by data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	8 seconds	49.00%	42.93%	34.98%
0.001	7 seconds	54.53%	45.64%	41.71%
0.0001	**7 seconds**	**64.53%**	**51.06%**	**49.87%**
0.05	6 seconds	44.95%	41.14%	33.74%
0.005	7 seconds	51.69%	42.68%	38.04%
0.0005	8 seconds	59.27%	46.50%	42.85%
0.08	7 seconds	43.04%	39.50%	32.02%
0.008	6 seconds	50.09%	42.69%	36.67%
0.0008	6 seconds	54.70%	44.61%	39.83%
0.15	7 seconds	41.35%	39.49%	31.70%

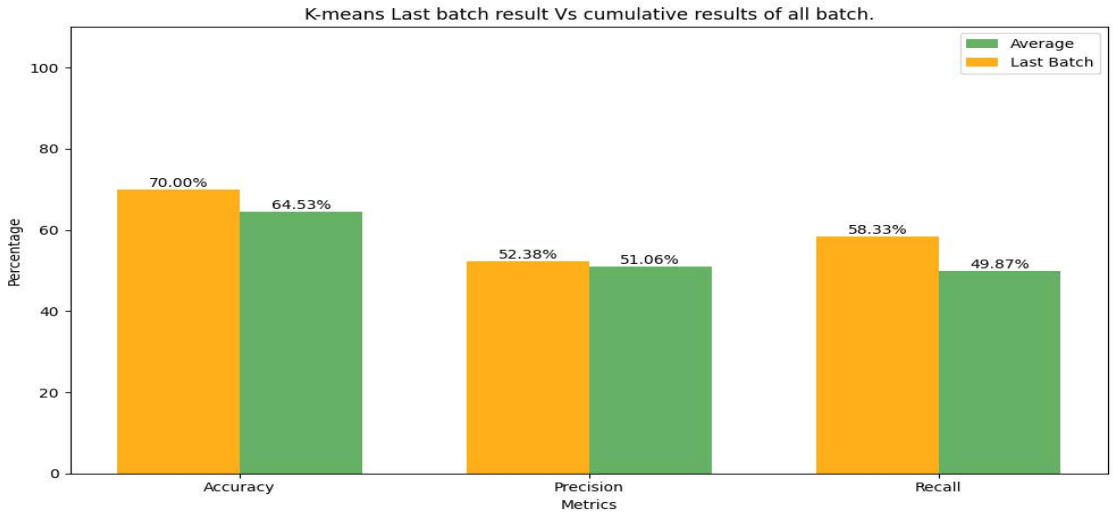

Figure 5.5: K-means interleaved date data result for learning rate 0.0001.

Table 5.11: DBSCAN last batch interleaved date by data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	7 seconds	40.00%	36.47%	32.05%
0.001	7 seconds	37.96%	41.07%	49.40%
0.0001	**6 seconds**	**80.00%**	**53.33%**	**60.00%**
0.05	7 seconds	40.00%	35.35%	34.33%
0.005	7 seconds	46.67%	31.97%	44.20%
0.0005	7 seconds	32.03%	32.23%	32.46%
0.08	7 seconds	40.00%	42.37%	42.34%
0.008	7 seconds	40.00%	34.20%	42.76%
0.0008	7 seconds	60.00%	30.00%	40.00%
0.15	7 seconds	40.00%	48.67%	38.33%

Table 5.12: DBSCAN cumulative all the batches interleaved date by data results.

Learning Rates	Time	Accuracy	Precision	Recall
0.01	5 seconds	51.45%	37.77%	38.20%
0.001	7 seconds	61.16%	44.47%	45.74%
0.0001	**7 seconds**	**65.17%**	**61.13%**	**59.39%**
0.05	8 seconds	49.26%	37.56%	35.41%
0.005	8 seconds	51.60%	39.57%	39.86%
0.0005	7 seconds	57.05%	45.50%	46.21%
0.08	7 seconds	47.58%	37.72%	35.63%
0.008	7 seconds	49.46%	39.71%	38.52%
0.0008	6 seconds	56.50%	43.42%	46.11%
0.15	6 seconds	48.01%	38.57%	35.59%

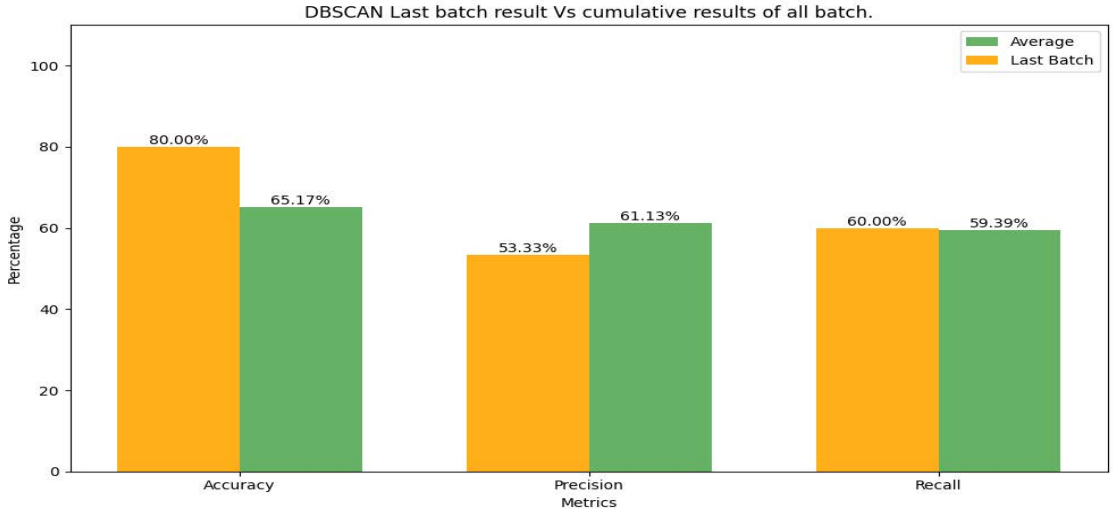

Figure 5.6: DBSCAN interleaved date by data result for learning rate 0.0001.

5.4.4 Discussion

Our system architecture, utilizing an ensemble of neural networks, excels in performance with both K-means and DBSCAN clustered data, with a 0.0001 learning rate in achieving high recall, accuracy, and precision. The ensemble approach combines the strengths of multiple neural networks. K-means clustering effectively organizes data into distinct groups based on similarity, while the DBSCAN density-based method adeptly handles varied data distributions, making it suitable for class labeling tasks. Overall, the experiments demonstrate that lower learning rates generally result in better performance, particularly with DBSCAN clustered data. The last batch results highlight the peak performance achievable, while the averages provide insight into the consistency and reliability of different learning rates across multiple instances. DBSCAN consistently outperforms K-means in both the last batch and average metrics, suggesting that DBSCAN is more effective in clustering the data for the ensemble of neural networks used in these experiments. The time (seconds) taken for each experiment was relatively consistent, with minor variations across different learning rates.

We have grouped every 25 batches and summarized the minimum and maximum results for all experiments conducted at a learning rate of 0.0001. This approach helps illustrate how the results evolve incrementally across different stages of the experiment. The analysis reveals that data clustered using the DBSCAN algorithm achieves higher cumulative accuracies much earlier compared to data clustered by the K-means algorithm. This indicates that DBSCAN-based clustering is better suited for providing early and stable improvements in model performance. Furthermore, the variance across different batches remains low, suggesting that the models predictions are relatively consistent and stable when applied to various subsets of the same dataset or when introduced to new, unseen data. A low variance indicates that the model is not overly sensitive to small fluctuations or changes in the input data, which is a desirable characteristic for ensuring robustness and reliability in real-world applications.

In each experiment, the results from the first 25 batches for K-means clustered data are consistently lower compared to the results for DBSCAN clustered data. As the experiments progress beyond the initial 25 batches, the performance results start to vary depending on the clustering technique used. This suggests that while DBSCAN provides an early advantage in cumulative accuracy, the subsequent results may depend on additional factors, such as the evolving nature of the data or the specific characteristics of each clustering method. Overall, the results include accuracy metrics evaluated using minimum, maximum, average, and variance values, providing a comprehensive view of model performance. However, precision and recall metrics are assessed based solely on average values, which serve as key indicators of the models capability to correctly identify relevant instances and minimize false positives.

For the first simulation random data, the results for K-means clustered data (Tables 5.13 and 5.14), there is significant variation in accuracy, with values ranging from 30.00% to 100.00%, and averages between 37.66% and 63.33%. Because the ensemble was trained on fewer data items at the beginning, then later on, they worked on other data items. The same issue happened with DBSCAN clustered data. The high variance (e.g., 275.36 for batches 226-249) suggests that our ensemble on the K-means clustered data performs inconsistently. This happened throughout all the simulations for K-means clustered data. Because is likely due to its sensitivity to initial cluster centroids and batch-to-batch variability in the data distribution. The high variance reflects the models instability, indicating poor generalization across different parts of the data stream. The high variance in each range indicates that there is a high dispersion of accuracies in that range. A low variance indicates that most accuracies are close to the average accuracy. Precision and recall also fluctuate, with lower values in early batches (e.g., 38.30% precision and 36.16% recall) and gradual improvements in later batches (e.g., 56.94% precision and 58.51% recall), further highlighting K-means difficulty in consistently identifying true positives and avoiding false positives. In contrast, our ensemble on the DBSCAN clustered data (Tables 5.15 and

5.16) shows more consistent performance, with accuracy averages converging faster from 43.24% to 98.33% and significantly lower variance in later batches (e.g., 14.71 for batches 276-293). This lower variance indicates more stable and reliable performance, as our ensemble on the DBSCAN clustered data handles the variability in random data streams more effectively. For lower variances, there is lower dispersion of values and most are close to the average for the range of 25 batches. Precision and recall results for DBSCAN clustered data are also more consistent and higher (e.g., 96.39% precision and 96.88% recall for batches 276-293), demonstrating its ability to minimize false positives and effectively identify true positives. Overall, while our ensemble on the K-means clustered data can perform well in certain batches, its high variance and fluctuating precision and recall indicate inconsistent performance. In contrast, DBSCAN clustered data lower variance and more stable precision and recall suggest superior generalization and reliability in handling random data streams, making it the more effective method.

For the second simulation involving the interleaved clustered data, for the K-means clustered data (Tables 5.17 and 5.18), the results show a wide range of accuracy across batches, with min-max values fluctuating between 20.00% and 100.00%. The average accuracy improves from 56.80% in early batches to 92.50% in later batches, but the high variance in some batches (e.g., 239.33 in batches 1-25, 225.00 in batches 26-50 and 191.67 in batches 176-200) indicates inconsistency in performance. This high variance suggests that our ensemble on the K-means clustered data struggles to maintain stability, likely due to its sensitivity to initial cluster centroids and the changing structure of the data. This is a similar issue with variance as with the first experiment. Precision and recall similarly improve as batches progress, starting with 39.80% precision and 56.80% recall in early batches, increasing to 93.19% and 94.40% precision and 92.50% and 92.80% recall in later batches. However, the lower precision in earlier batches reflects a higher rate of false positives. In contrast, DBSCAN (Tables 5.19 and 5.20) demonstrates more stable and consistent performance, with min-max values in the later batches narrowing between 90.00%

and 100.00%. The variance is also much lower, especially in later batches (e.g., 0.00 for batches 176-200), indicating that our ensemble on the DBSCAN clustered data maintains a high level of consistency. The precision and recall results for DBSCAN clustered data are also higher and more stable, reaching 100.00% in 176-200 batches, suggesting that our ensemble on the DBSCAN cluster is better at handling noise and irregular data clusters. Thus, while K-means clusters can perform well, its higher variance reflects greater instability, whereas DBSCAN lower variance and consistent precision and recall demonstrate superior generalization and reliability in processing interleaved cluster data.

For the third simulation interleaved data by date, the results for K-means clustered data (Tables 5.21 and 5.22) show a wide range in accuracy, with min-max values fluctuating between 30.00% and 100.00%, and averages increasing from 53.80% to 78.82%. However, high variance in certain batches (e.g., 399.78 for batches 51-75) indicates inconsistent performance, suggesting that K-means struggles with stability, particularly in handling temporal variations in interleaved date data. Precision and recall improve over batches, starting at 42.44% and 38.30%, respectively, and reaching 67.05% and 70.52%, but the fluctuations in early batches point to higher rates of false positives and inconsistent identification of true positives. In contrast, DBSCAN (Tables 5.23 and 5.24) demonstrates more consistent performance, with accuracy ranging from 50.45% to 100.00%, and an average improving from 60.05% to 79.71%. DBSCAN also has significantly lower variance in earlier batches (e.g., 21.82 for batches 1-25), reflecting greater stability, although variance increases slightly in later batches as accuracy approaches its maximum. DBSCAN precision and recall are also higher and more stable, starting at 61.82% and 57.29% and reaching 63.64% and 64.46%, respectively. The lower variance and more consistent precision and recall suggest that DBSCAN is better suited for handling temporal variations in interleaved date-wise data, providing more reliable and stable results compared to K-means.

Table 5.13: K-means batch-wise random data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	30.00-47.92%	37.66%	39.23
26-50	30.00-90.00%	46.26%	174.01
51-75	30.00-80.00%	49.20%	157.67
76-100	30.00-80.00%	52.99%	202.51
101-125	30.00-70.00%	45.22%	142.80
126-150	30.00-80.00%	48.79%	168.04
151-175	30.00-90.00%	53.93%	263.74
176-200	40.00-90.00%	58.80%	186.00
201-225	30.00-100.00%	53.59%	257.43
226-249	30.00-90.00%	63.33%	275.36

Table 5.14: K-means batch-wise random data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	38.30%	36.16%
26-50	44.77%	45.57%
51-75	45.69%	48.77%
76-100	47.04%	48.47%
101-125	38.76%	42.11%
126-150	40.83%	45.53%
151-175	47.45%	51.74%
176-200	52.86%	58.20%
201-225	47.00%	49.84%
226-249	56.94%	58.51%

Table 5.15: DBSCAN batch-wise random data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	30.00-70.00%	43.24%	176.78
26-50	30.00-100.00%	59.55%	463.15
51-75	30.00-100.00%	59.60%	354.00
76-100	30.00-100.00%	61.60%	289.00
101-125	40.00-100.00%	71.60%	197.33
126-150	50.00-90.00%	75.60%	142.33
151-175	50.00-100.00%	73.60%	165.67
176-200	50.00-100.00%	80.00%	150.00
201-225	50.00-100.00%	76.80%	156.00
226-250	40.00-100.00%	83.20%	306.00
251-275	80.00-100.00%	96.00%	33.33
276-293	90.00-100.00%	98.33%	14.71

Table 5.16: DBSCAN batch-wise random data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	45.80%	41.77%
26-50	53.25%	60.55%
51-75	49.31%	55.00%
76-100	53.17%	59.41%
101-125	63.23%	69.10%
126-150	65.58%	71.91%
151-175	62.68%	74.40%
176-200	66.67%	74.55%
201-225	64.58%	74.35%
226-250	77.60%	82.22%
251-275	92.98%	94.33%
276-293	96.39%	96.88%

Table 5.17: K-means batch-wise interleaved cluster data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	20.00-80.00%	56.80%	239.33
26-50	40.00-100.00%	70.00%	225.00
51-75	60.00-100.00%	70.80%	166.00
76-100	40.00-100.00%	62.40%	135.67
101-125	40.00-90.00%	69.60%	145.67
126-150	60.00-100.00%	86.40%	140.67
151-175	50.00-100.00%	80.00%	116.67
176-200	60.00-100.00%	84.00%	191.67
201-225	80.00-100.00%	92.80%	62.67
226-249	70.00-100.00%	92.50%	71.74

Table 5.18: K-means batch-wise interleaved cluster data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	39.80%	56.80%
26-50	56.93%	70.00%
51-75	58.53%	70.80%
76-100	50.08%	62.40%
101-125	59.01%	69.60%
126-150	84.93%	86.40%
151-175	79.47%	80.00%
176-200	82.53%	84.00%
201-225	94.40%	92.80%
226-249	93.19%	92.50%

Table 5.19: DBSCAN batch-wise interleaved cluster data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	40.00-80.00%	57.71%	107.06
26-50	60.00-100.00%	72.00%	191.67
51-75	60.00-100.00%	84.40%	109.00
76-100	70.00-100.00%	79.60%	54.00
101-125	70.00-100.00%	92.80%	96.00
126-150	80.00-100.00%	98.40%	30.67
151-175	90.00-100.00%	99.60%	4.00
176-200	100.00-100.00%	100.00%	0.00
201-225	90.00-100.00%	99.60%	4.00
226-250	90.00-100.00%	99.20%	7.67
251-275	90.00-100.00%	99.20%	7.67
276-293	90.00-100.00%	97.22%	21.24

Table 5.20: DBSCAN batch-wise interleaved cluster data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	44.90%	57.85%
26-50	62.00%	71.80%
51-75	78.05%	84.40%
76-100	73.57%	79.60%
101-125	92.24%	92.80%
126-150	97.60%	98.40%
151-175	99.73%	99.60%
176-200	100.00%	100.00%
201-225	99.73%	99.60%
226-250	99.47%	99.20%
251-275	99.47%	99.20%
276-293	98.15%	97.22%

Table 5.21: K-means batch-wise interleaved date by data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	30.00-80.00%	53.80%	209.11
26-50	30.00-70.00%	58.40%	230.67
51-75	35.42-100.00%	60.62%	399.78
76-100	40.00-90.00%	75.60%	250.67
101-117	40.00-100.00%	78.82%	273.53

Table 5.22: K-means batch-wise interleaved date by data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	42.44%	38.30%
26-50	47.45%	44.44%
51-75	48.21%	45.13%
76-100	55.28%	57.57%
101-117	67.05%	70.52%

Table 5.23: DBSCAN batch-wise interleaved date by data accuracy results for learning rate 0.0001.

Batch	Min-Max	Average	Variance
1-25	50.45-68.93%	60.05%	21.82
26-50	50.49-69.21%	58.67%	22.73
51-75	50.02-90.00%	63.22%	77.03
76-100	52.70-100.00%	68.87%	193.31
101-117	60.00-100.00%	79.71%	138.97

Table 5.24: DBSCAN batch-wise interleaved date by data precision and recall average results for learning rate 0.0001.

Batch	Precision	Recall
1-25	61.82%	57.29%
26-50	59.57%	57.61%
51-75	60.67%	58.80%
76-100	60.77%	60.41%
101-117	63.64%	64.46%

Overall, our ensemble on the DBSCAN clustered data demonstrated the best performance across all three experiments, consistently outperforming K-means clustered data in terms of accuracy, precision, recall, average, and variance. DBSCAN clustered data showed greater stability, with lower variance in later batches and more reliable accuracy, precision, and recall values. Particularly in the second simulation, involving interleaved cluster data,

DBSCAN clustered data consistently achieved higher accuracy with lower variance, indicating better generalization and robustness in handling structured data streams. In contrast, K-means clustered data exhibited high variability and instability, especially in earlier batches, with its performance improving only in later stages. These results suggest that DBSCAN clustered data is better suited for scenarios with complex and noisy data, offering more consistent and reliable clustering performance across diverse datasets.

5.5 Chapter Summary

In this chapter, we presented our experiments and the evaluation process. The results indicate that our ensemble of neural networks, including three experiments, performs well for clustered data, showing that DBSCAN clustered data records outperformed the K-means clustered data.

Chapter 6
Conclusion

6.1 Conclusion

The research presented in this thesis focuses on applying ensemble methods for spatial data stream classification, specifically for severe weather data. The primary objective was to improve the accuracy and timeliness of decision-making during severe weather emergencies by leveraging advanced machine learning techniques. Ensemble methods, such as bagging and neural networks, were employed to enhance classification tasks, while clustering algorithms like K-means and DBSCAN were used to preprocess spatial data, segmenting it effectively based on similarities. The performance of the ensemble of neural networks was evaluated using three types of data simulations: random data, interleaved clustered data, and interleaved data by date.

The following research questions guided the study:

1. **Which of the clustering algorithms K-means and DBSCAN produced better class-labeled data in terms of performance?**

   The experiments demonstrated that DBSCAN outperformed K-means in terms of classification accuracy, precision, and recall. Specifically, DBSCAN clustered data, when processed using ensemble neural networks, resulted in more accurate and consistent outcomes. For example, DBSCAN achieved 100.00% accuracy, precision, and recall for the last batch of interleaved clustered data, with cumulative results of 89.80% accuracy, 86.81% precision, and 89.80% recall. In contrast, K-means showed lower performance metrics, achieving a cumulative accuracy of 76.47%, precision of 69.80%, and recall of 76.47% for the same dataset.

   Overall, DBSCAN ability to handle varying densities and complex data structures

made it a more effective clustering algorithm for this study, producing better class-labeled data and enabling more accurate predictions in ensemble models.

2. **Which sequence for processing a spatial data stream achieves better predictive outcomes?**

   Among the three types of data streaming sequences evaluated—random data, interleaved clustered data, and interleaved data by date—the interleaved clustered data yielded the best results for both DBSCAN and K-means clustering methods. The interleaved clustered data scenario resulted in more stable and accurate predictions across batches, reflecting the ensemble models robustness in handling dynamic changes in spatial data streams. DBSCAN, paired with interleaved clustered data, reached 100.00% accuracy, precision, and recall in the last batch, further proving the efficacy of this streaming sequence.

   This finding underscores the importance of choosing appropriate data streaming sequences when dealing with spatial data streams, as interleaved clustered data scenarios enable better alignment with real-world applications by capturing temporal and spatial dependencies.

3. **Can using an ensemble of neural networks for processing spatial data streams lead to improved model performance and accuracy?**

   Yes, the use of ensemble methods significantly improved model performance and accuracy when processing spatial data streams. Bagging, combined with neural networks, helps to reduce variance and prevent overfitting, thereby enhancing classification accuracy. For example, the ensemble approach with DBSCAN clustered data at a learning rate of 0.0001 resulted in 100.00% accuracy, precision, and recall for the last batch in the random data scenario.

   These results confirm that ensemble methods, particularly those using neural networks, are effective in improving model performance and accuracy for spatial data

stream classification. The ability to leverage multiple models strengths and reduce the impact of individual model weaknesses led to more robust and reliable classification outcomes, especially for complex datasets like DBSCAN-clustered severe weather data.

This study demonstrates that ensemble methods, combined with clustered data, significantly enhance spatial data stream classification.

For DBSCAN clustered data, all the experiment results are follow as:

1. For random data experiments, the last batch results are 100.00% accuracy, 100.00% precision, and 100.00% recall, and maintaining superior cumulative results of all batches 72.66% accuracy, 65.21% precision, and 70.59% recall.

2. For interleaved clustered data experiments, 100.00% accuracy, 100.00% precision, and 100.00% recall for the last batch, and cumulative results of all batches 89.80% accuracy, 86.81% precision, and 89.80% recall.

3. For interleaved clustered data by date experiments, the last batch outputs are 80.00% accuracy, 53.33% precision, and 60.00% recall, and the average of all batches are 65.17% accuracy, 61.13% precision, and 59.39% recall.

For K-means clustered data, all the experiment results are follow as:

1. For random data experiments, the last batch outputs are 80.00% accuracy, 69.33% precision, and 70.00% recall, and cumulative results of all batches are 50.93% accuracy, 45.92% precision, and 48.45% recall.

2. For interleaved clustered data experiments, 80.00% accuracy, 83.33% precision, and 80.00% recall for the last batch, and the final average results of all batches are 76.47% accuracy, 69.80% precision, and 76.47% recall.

3. For interleaved clustered data by date experiments, 70.00% accuracy, 52.38% precision, and 58.33% recall belongs to the last batch, and the average of all batches are 64.53% accuracy, 51.06% precision, and 49.87% recall.

6.2 Future Works

In the future, we intend to do the following:

1. Investigate using more sophisticated neural network architectures such as Graph Neural Networks (GNNs) and Recurrent Neural Networks (RNNs) [33] for better handling temporal dependencies and spatial relationships in severe weather data.

2. Implement anomaly detection mechanisms to identify and handle outliers in real-time data streams, enhancing the reliability of the clustering and classification results.

3. Develop and deploy real-time predictive analytics platforms that leverage advanced ensemble methods and clustering algorithms to provide timely and accurate severe weather forecasts and alerts.

4. Ensuring that all data in transit and at rest is encrypted using robust encryption algorithms such as Advanced Encryption Standard (AES) to prevent unauthorized access.

5. Investigate the integration of federated learning to enable collaborative model training across multiple decentralized devices or locations while preserving data privacy and security.

Bibliography

[1] Federal Emergency Management Agency. Emergency management guide for business and industry, 2020. https://www.fema.gov.

[2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, 2002.

[3] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.

[4] Mohammad Ali Bagheri and Qigang Gao. An efficient ensemble classification method based on novel classifier selection technique. In Dumitru Dan Burdescu, Rajendra Akerkar, and Costin Badica, editors, *2nd International Conference on Web Intelligence, Mining and Semantics, WIMS '12, Craiova, Romania, June 6-8, 2012*, pages 22:1–22:7. ACM, 2012.

[5] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, 1996.

[6] Environment Canada. Environment canada, 2023. https://climate.weather.gc.ca/historical_data/search_historic_data_e.html.

[7] NATIONAL HURRICANE CENTER and CENTRAL PACIFIC HURRICANE CENTER. Noaa great circle distance calculator, 2023. https://www.nhc.noaa.gov/gccalc.shtml.

[8] Marcel J. T. Reinders Dick de Ridder, Jeroen de Ridder. Pattern recognition in bioinformatics. *Briefings in Bioinformatics*, 14(5):633–647, 04 2013.

[9] Thomas G Dietterich. Ensemble methods in machine learning. *International workshop on multiple classifier systems*, pages 1–15, 2000.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[11] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Comput. Surv.*, 50(2):23:1–23:36, 2017.

[12] A. C. R. Gonçalves, X. Costoya, R. Nieto, et al. Extreme weather events on energy systems: a comprehensive review on impacts, mitigation, and adaptation measures. *Sustainable Energy res. 11*, 11(4), 2024.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, Burlington, MA, 3rd edition, 2011.

[15] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999.

[16] Xiaojuan Hu, Lei Liu, Ningjia Qiu, Di Yang, and Meng Li. A mapreduce-based improvement algorithm for dbscan. *Journal of Algorithms Computational Technology*, 12:174830181773566, 10 2017.

[17] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[18] Wenbo Jin, Yudong Fang, Yongqiang Liu, Jixing Yang, and Junhui Yu. Research on emergency management information business and information system framework. In *2023 IEEE 13th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 53–59. IEEE, 2023.

[19] Liam King and Wendy Osborn. Ensemble methods for spatial data stream classification. *Procedia Computer Science*, 224:155–162, 2023.

[20] Georg Krempl, Indre Zliobaite, Dariusz Brzezinski, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou, and Jerzy Stefanowski. Open challenges for data stream mining research. *SIGKDD Explor.*, 16(1):1–10, 2014.

[21] STUART P. LLOYD. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[22] JAMES MACQUEEN. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, page 281–297. University of California Press, 1967.

[23] Alessandro Margara and Tilmann Rabl. Definition of data streams. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019.

[24] Israel Martínez-Hernández and Marc G. Genton. Surface time series models for large spatio-temporal datasets. *Spatial Statistics*, 53:100718, 2023.

[25] Ammar Mohammed and Rania Kora. A comprehensive review on ensemble deep learning: Opportunities and challenges. *J. King Saud Univ. Comput. Inf. Sci.*, 35(2):757–774, 2023.

[26] Open-Meteo. Historical weather data, 2023. `https://open-meteo.com/en/docs/historical-weather-api`.

[27] Wendy Osborn and Liam King. An iterative strategy for deep learning classification on spatial data streams. In Eric Pardede, Maria Indrawan-Santiago, Pari Delir Haghighi, Matthias Steinbauer, Ismail Khalil, and Gabriele Kotsis, editors, *iiWAS2021: The 23rd International Conference on Information Integration and Web Intelligence, Linz, Austria, 29 November 2021 - 1 December 2021*, pages 532–537. ACM, 2021.

[28] Peter Rousseeuw. Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987.

[29] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45(4):427–437, 2009.

[30] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant, editors, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 377–382. ACM, 2001.

[31] Kristine Towne, Qiang Zhu, Calisto Zuzarte, and Wen-Chi Hou. Window query processing for joining data streams with relations. In Bruce Spencer, Margaret-Anne D. Storey, and Darlene A. Stewart, editors, *Proceedings of the 2007 conference of the Centre for Advanced Studies on Collaborative Research, October 22-25, 2007, Richmond Hill, Ontario, Canada*, pages 188–202. IBM, 2007.

[32] Shikha Verma, Kuldeep Srivastava, Akhilesh Tiwari, and Shekhar Verma. Deep learning techniques in extreme weather events: A review, 2023.

[33] Lilapati Waikhom and Ripon Patgiri. Recurrent convolution based graph neural network for node classification in graph structure data. In *2022 12th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 201–206, 2022.

[34] Yuxuan Yang, Hadi Khorshidi, and Uwe Aickelin. A review on over-sampling techniques in classification of multi-class imbalanced datasets: insights for medical problems. *Frontiers in digital health*, 6:1430245, 07 2024.