DEPARTMENT OF COMPUTER APPLICATIONS

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY COCHIN-22

MSc COMPUTER SCIENCE

Specialization in Data Science

PROJECT REPORT

LSTM Autoencoder Based Extreme Rainfall Prediction in Highly Unbalanced Data Using Vector Reconstruction Error

IV SEMESTER APRIL 2023

DEPARTMENT OF COMPUTER APPLICATIONS

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY COCHIN-22

CERTIFICATE

This is to certify that the mini project entitled "· **LSTM Autoencoder Based Extreme Rainfall Prediction in Highly Unbalanced Data Using Vector Reconstruction Error**" is a bonafide record of the project work done by SHINEJIL MUKUND (Reg No: -35921021), SREELESH M (Reg No:- 35921023), THERES MARIYA J (Reg No:- 35921024), TINTO MATHEW(Reg No:- 35921025), BASILA SHERIN P K(Reg No:- 35921012) and MARIYA C JOY (Reg No:- 35921016) under our supervision and guidance in partial fulfilment of the award of Degree of Masters of Science in Computer Science during the year 2021-23 at Department of Computer Applications, Cochin University of science and Technology, Cochin.

Internal Examiner **Head of the department**

Department of Computer Applications
CUSAT

ACKNOWLEDGEMENT

With great pleasure, we hereby acknowledge that the help given by various individuals throughout the project itself is an acknowledgment to the inspiration driven and technical assistance contributed by many individuals.

we convey our reverential salutation to Almighty God, for enabling me to take up and complete the project successfully.

We are pleased our indebtedness to **Dr. M.V Judy**, Head of the Department, Department of Computer Application, CUSAT for her gracious encouragement. We also take this opportunity to express my heartfelt gratitude and thanks to our class coordinator **Mrs. Deepika M.P** and **Dr. Sabu M.K** for the support and guidance. We also thankful to **Ms. Malathi S** our project coordinator.

And also, we are obliged to the teaching staff for being helpful and cooperative during the project period. We extend our heartfelt thanks to parents and friends and well-wishers for their support and timely help.

ABSTRACT

An extreme condition of rainfall is one of those many critical elements of the hydrological cycle that has a direct impact on human life in many aspects. An accurate and early detection of a future rainfall event can help in preventing human and financial losses. Therefore, it is vital to design a framework that can predict the occurrence of an extreme rainfall with significant accuracy. Accordingly, in this project we have proposed Neural network models predicting rainfall that can be done using an autoencoder based forecast model which can predict the rainfall values efficiently using the historical. values. The design of the forecast model is made simple to avoid the heavy training time. Furthermore, a transformation technique was applied on the rainfall dataset to make the data more normal distribution-like. The proposed model detects rainfall and provides warning at an extreme condition. Thus, outperforms the traditional persistence and average forecast models.

Keywords: Rainfall, Prediction, Artificial Neural Networks, Deep Learning, Auto-encoders

CONTENTS

INTRODUCTION

Our project aims to detect extreme scenarios from a highly unbalanced dataset using autoencoders. The dataset contains a significant imbalance between the majority and minority classes, making it challenging to accurately predict extreme cases. To overcome this issue, we are using an autoencoder, which is a type of neural network that can reduce the dimensionality of the data while retaining its most important features. By training the autoencoder on the majority classes and predicting the minority classes, here we predict whether a predicted output is extreme or normal on the basis of the error formed while prediction through this we can identify the extreme scenarios. Here we are using rainfall dataset. The rainfall dataset was collected from the Advanced Centre for Atmospheric Radar Research. It contains information on the amount of rainfall that was recorded at various locations in Kerala. Each record in the dataset represents a single rainfall event and includes details such as wind speed, tpw, and the amount of rainfall in millimetres. The dataset also contains the date of each rainfall record. The wind speed, tpw, and rainfall data were initially separate text files. We created a CSV file that includes columns for date, wind speed, tpw, and rainfall data, and saved this as 'rainfall_data.csv.' for testing the above mentioned methodology.

LITERATURE REVIEW

[1] Deep Learning For Anomaly Detection: A Survey:

When working with real-world data, it is often necessary to identify instances that are significantly different from the rest, known as anomalies or outliers. Anomaly detection aims to identify all such instances in a data-driven way. While some anomalies may be caused by errors in the data, they can also indicate the presence of a new, previously unknown process. Traditional algorithms may struggle to detect outliers in image and sequence datasets due to the complexity of the data structures. Moreover, as the volume of data increases, traditional methods may fail to scale to identify outliers. To overcome these limitations, deep anomaly detection (DAD) techniques learn hierarchical discriminative features from data automatically, which eliminates the need for manual feature development by domain experts. This approach is especially useful for solving problems end-to-end by taking raw input data in domains such as text and speech recognition. However, defining the boundary between normal and anomalous behaviour can be challenging, and it is continually evolving in several data domains. This lack of a well-defined normal boundary presents challenges for both conventional and deep learning-based algorithms. This survey paper discusses various research methods in deep learning-based anomaly detection and their applications across various domains, such as intrusion detection, fraud detection, malware detection etc. It also presents several existing solutions to the challenges posed by deep anomaly detection. For each category of deep anomaly detection techniques, the authors provide assumptions regarding the notion of normal and anomalous data, along with their strengths and weaknesses. The goal of the survey is to identify and evaluate various deep learning models for anomaly detection and assess their suitability for a given dataset. The assumptions presented in the survey can be used as guidelines when selecting a deep learning model for a particular domain or data, to assess its effectiveness.

[2] An overview of anomaly detection techniques: Existing solutions and latest technological trends:

This paper discusses the importance of intrusion detection systems in cyberspace, given the growing threat of spammers, attackers, and criminal enterprises. The prevalence of such threats has made intrusion detection systems join ranks with firewalls as one of the fundamental technologies for network security. The paper provides a comprehensive survey of anomaly detection systems and hybrid intrusion detection systems of the recent past and present. Anomaly detection systems model the normal system/network behaviour which enables them to be extremely effective in finding and foiling both known as well as unknown or “zero day” attacks. However, these systems face challenges such as high false alarm rate, failure to scale to gigabit speeds, among others. The text highlights the vulnerability of today's networks, given their accessibility to employees, partners, and customers. This accessibility makes them more vulnerable to intrusions and attacks, including cybercrime. The text cites statistics showing that network-based attacks are on the rise, and the financial losses incurred by companies due to network attacks/intrusions. The text concludes that intrusion detection systems complement firewalls in detecting possible security breaches by gathering and analysing information from various areas within a computer or a network. Signature detection systems and anomaly detection systems are the two primary approaches, with their share of advantages and drawbacks. The text notes that anomaly detection systems have the advantage of detecting unknown attacks as well as “zero day” attacks, but technological problems need to be overcome before they can be widely adopted.

[3] Machine Learning for Anomaly Detection: A Systematic Review:

The systematic literature review analysed 290 research articles on anomaly detection using machine learning techniques published from 2000-2020. The review focused on four research questions: the application of anomaly detection type, the type of ML technique, the ML model accuracy estimation, and the type of anomaly detection (supervised, semi-supervised, and unsupervised). The findings identified 43 different applications of anomaly detection, with intrusion detection, network anomaly detection, general anomaly detection, and data applications being the most common. SVM was the most commonly used ML model, and PCA and CFS were the most commonly used feature selection/extraction techniques. The review found that 64 of the 290 papers used accuracy or AUC as their main performance metric, and 27% of the selected papers applied unsupervised anomaly detection type, making it the most used approach among the research articles. The review recommends more research on ML studies of anomaly detection to gain more evidence on ML model performance and efficiency, creating a general structure for introducing experiments on ML models, improvement in feature selection/extraction, use of more recent datasets, and consideration of multiple performance metrics.

[4] Machine Learning for Anomaly Detection and Categorization in Multi-cloud Environments:

The paper discusses the use of multi-cloud environments by application service providers (ASPs) and enterprises to reduce costs and improve performance, but also highlights the security concerns associated with this approach. Traditional security techniques are not sufficient to protect user-data in multi-cloud scenarios, and the paper proposes the use of machine learning techniques for intrusion detection and categorization of different types of attacks. The authors use two supervised machine learning techniques, linear regression and random forest, to achieve high detection accuracy and categorization accuracy of 93.6%. The paper concludes by suggesting that these techniques can be applied to multi-cloud environments to improve security. in this study, we used the UNSW dataset to evaluate the effectiveness of supervised machine learning techniques for both anomaly detection and attack categorization. Our results show that the random forest (RF) algorithm, combined with feature selection, can achieve 99% accuracy for anomaly detection. However, we also found that categorization accuracy is lower due to similar attack behaviours. To address this, we developed our own step-wise categorization algorithm, which showed better performance than traditional approaches. However, some attacks were not categorised due to feature similarities and unbalanced data, highlighting the need for more data or features to distinguish them. We believe that these models can be adapted for use in multi-cloud environments with minor adjustments to the learned models.

[5] Machine Learning Techniques for Anomaly Detection: An Overview

The field of intrusion detection is a popular area of research, but challenges still remain, such as reducing false alerts during the detection of unknown attack patterns. One potential solution to this problem is anomaly detection, where deviations from normal behaviour can indicate the presence of attacks or other issues. This paper provides an overview of research directions for applying supervised and unsupervised methods for managing the problem of anomaly detection. The cited references cover major theoretical issues and can guide researchers in interesting research directions. The paper discusses the importance of intrusion detection systems (IDSs) as a second line of defence to protect information systems, in conjunction with other preventive security mechanisms such as access control and authentication. IDSs are classified into signature detection systems and anomaly detection systems, with advantages and drawbacks to each approach. Anomaly detection systems have the capability to detect unknown attacks and customise normal activity profiles for every system, application, and network, but they also face technical challenges such as system complexity, high false alarms, and difficulty in identifying which events trigger those alarms. The paper presents an overview of research directions for applying supervised and unsupervised methods for managing the problem of anomaly detection, including a detailed discussion of the general architecture of anomaly intrusion detection systems and the techniques used in anomaly detection. The use of machine learning techniques has gained attention in intrusion detection research as a means to address weaknesses in knowledge base detection techniques. Anomaly detection can be achieved through supervised and unsupervised techniques, with various algorithms used to achieve good results. This paper provides an overview of machine learning techniques for anomaly detection, with experiments showing that supervised learning methods perform better than unsupervised ones in situations without unknown attacks. Non-linear methods such as SVM, multi-layer perceptron, and rule-based methods perform best among supervised methods, while

techniques such as K-Means, SOM, and one class SVM perform better than others among unsupervised techniques, although they vary in their ability to detect all attack classes effectively.

[6] Survey on Anomaly Detection using Data Mining Techniques:

The amount of data stored and transferred in today's world makes it highly vulnerable to attacks. While various techniques exist to protect data, they are not foolproof. Data mining techniques have emerged to analyze data and detect various kinds of attacks. Anomaly detection is one such technique that uses data mining to detect surprising behaviour hidden within data. Hybrid approaches have also been developed to detect known and unknown attacks more accurately. This paper provides a review of various data mining techniques used for anomaly detection and aims to provide a better understanding of existing techniques for interested researchers to work on in the future. Intrusion Detection Systems (IDS) is used to enhance the security of communication and information systems. IDS can be classified into three types: signature detection systems, anomaly detection systems, and hybrid detection systems. Anomaly detection systems compare activities against normal defined behaviour and have several advantages over other techniques. They can detect insider attacks, are based on custom-made profiles, and can detect previously unknown attacks. The rest of the paper focuses on various anomaly detection techniques. The paper reviews different approaches to anomaly detection and focuses on the four classes of data mining tasks: association rule learning, clustering, classification, and regression. This response specifically discusses the clustering and classification-based approaches. Clustering involves dividing data into groups of similar objects, and clustering algorithms can be used for anomaly detection without prior knowledge. Several clustering-based approaches are discussed, including k-means, k-medoids, EM clustering, and outlier detection algorithms. Outlier detection is a technique to find patterns in data that do not conform to expected behaviour. Classification involves identifying the category of new instances based on a training set of data. In anomaly detection, data can be classified into normal or abnormal categories. Various

classification-based anomaly detection techniques are discussed, including decision trees, support vector machines, and naive Bayes classifiers. Overall, the paper provides an overview of different approaches to anomaly detection and highlights the strengths and weaknesses of each approach.

[7] Prediction of Rainfall Using Intensified LSTM Based Recurrent Neural Network with Weighted Linear Units:

The proposed Intensified LSTM-based RNN model utilizes the historical data of rainfall to train the neural network. The use of LSTM allows the model to capture the long-term dependencies and patterns in the data and helps in the prediction of rainfall. The Intensified LSTM layer intensifies the ability of the LSTM network to predict the rainfall patterns and improve its accuracy. In the evaluation process, the proposed model is compared with various models such as Holt–Winters, Extreme Learning Machine (ELM), Autoregressive Integrated Moving Average (ARIMA), Recurrent Neural Network and Long Short-Term Memory. The performance of the proposed model is analyzed based on various parameters such as Root Mean Square Error (RMSE), accuracy, number of epochs, loss, and learning rate of the network. The results obtained show that the proposed Intensified LSTM-based RNN model outperforms other models in terms of accuracy, number of epochs and loss.

The proposed model provides a robust and efficient solution for the prediction of rainfall, which can be used by organizations responsible for the prevention of disasters to make informed decisions. The use of machine learning and deep learning techniques in the prediction of rainfall helps in enhancing the accuracy of the predictions, which can be used in decision-making processes to mitigate the impacts of natural disasters.

In conclusion, the proposed Intensified LSTM-based RNN model is a promising solution for the prediction of rainfall. The use of deep learning techniques and the improvement in the accuracy of predictions can assist in the prevention of natural disasters and ensure the safety of the community.

[8] Rainfall prediction for the Kerala state of India using artificial intelligence approaches:

The study focuses on the use of three artificial intelligence techniques for forecasting seasonal rainfall patterns in the Kerala state of India. These techniques include K-nearest neighbor (KNN), artificial neural network (ANN), and extreme learning machine (ELM). The purpose of the study is to evaluate the performance of these techniques against observed rainfall data from 2011 to 2016.

The results of the study show that all the three techniques perform reasonably well, but the ELM technique showed the best performance with minimal mean absolute percentage error scores for both the summer monsoon and post-monsoon seasons. The ELM architecture with 8-15-1, which consists of 8 input nodes, 15 hidden nodes, and 1 output node, has proven to be highly effective in providing accurate results.

The prediction accuracy of the techniques is highly influenced by the number of hidden nodes in the hidden layer. A larger number of hidden nodes provides better accuracy but may result in overfitting, while a smaller number of hidden nodes may lead to underfitting. The ELM technique, with its hidden layer containing 15 nodes, has shown the best results in terms of prediction accuracy.

In conclusion, this study highlights the potential of the proposed artificial intelligence approaches in predicting seasonal rainfall patterns in the Kerala state of India with minimal prediction error scores. The results of the study indicate that ELM is the best-performing technique among the three and can be used as a reliable tool for forecasting rainfall patterns.

This study can serve as a reference for future studies on rainfall prediction and can contribute to decision-making processes for managing water resources.

[9] Rainfall Prediction Using Machine Learning & Deep Learning Techniques:

Agriculture is a crucial aspect of survival in India, and rainfall plays a vital role in its success. The prediction of rainfall is becoming increasingly important in recent times, as it helps people take necessary precautions to protect their crops from excessive rainfall. This is where machine learning algorithms come in handy, as they are useful in predicting rainfall patterns.

There are several machine learning algorithms that can be used to predict rainfall, some of the major ones being ARIMA (Auto-Regressive Integrated Moving Average) Model, Artificial Neural Network (ANN), Logistic Regression, Support Vector Machine, and Self-Organizing Map. ARIMA Model is one of the most commonly used linear models for predicting seasonal rainfall, while ANNs are used for non-linear models. In the case of Artificial Neural Networks (ANN), different techniques can be used to predict rainfall such as Back Propagation NN, Cascade NN, or Layer Recurrent Network. ANN is based on the structure of biological neural networks, which makes it similar to the human brain in terms of functionality.

In conclusion, the use of machine learning algorithms for predicting rainfall patterns in India is becoming increasingly important for the success of the agricultural sector. The different models and techniques mentioned above have proven to be effective in predicting rainfall patterns and can be used to make informed decisions about water management and crop protection.

[10] A Data-Driven Approach for Accurate Rainfall Prediction:

There has been a growing interest in utilizing Precipitable Water Vapor (PWV) obtained from Global Positioning System (GPS) signals to predict rainfall in recent years. However, rainfall is influenced by a variety of atmospheric parameters. This study proposes a systematic approach to analyze the various parameters that impact precipitation in the atmosphere. Ground-based weather features such as Temperature, Relative Humidity, Dew Point, Solar Radiation, and PWV, along with Seasonal and Diurnal variables, were identified and a correlation study was performed. All features play a significant role in rainfall classification, however, only a few, such as PWV, Solar Radiation, Seasonal, and Diurnal features, stand out for rainfall prediction. Based on these results, an optimal set of features was used in a machine learning algorithm for rainfall prediction. The experiment using a four-year database (2012- 2015) showed a true detection rate of 80.4%, a false alarm rate of 20.3%, and an overall accuracy of 79.6%. Compared to previous studies, our method significantly reduces false alarm rates.

[11] Rainfall Detection and Rainfall Rate Estimation Using Microwave Attenuation

The study used eight microwave links located in the city of Seoul, South Korea for detecting rainfall and estimating path-averaged rainfall rates. These links operated at frequencies between 6 to 8 GHz and had path lengths ranging from 5.7 to 37.4 km. The aim was to determine if the microwave links could be used to accurately detect rainfall, which was confirmed through comparison with data from rain detectors installed at automatic weather stations. The results showed that the microwave links had an accuracy of ≥80% in detecting rainfall.

Furthermore, the study established the power-law relationships between rain-induced specific attenuation, k (dB km−1), and the rainfall rate, R (mm h−1). These relationships were then cross- validated by estimating the path-averaged rainfall rate. The mean bias of the estimated path-averaged rainfall rate was found to be between -3- and 1-mm h−1, which was considered to be a relatively accurate estimate compared to the rainfall rate measured by ground rain gauges.

In conclusion, the improved accuracy of rainfall detection using microwave links demonstrated that these links can be effectively used for identifying rainy or dry periods and

provide real-time highhigh- time-resolution fall rates. This information is crucial for various organizations and decision-making processes related to the prevention of disasters and the effective management of resources.

[12] Detection of trends in annual extreme rainfall:

This study focuses on the estimation of trends in intensity-duration-frequency of rainfall, which is crucial information needed for various hydrological applications. The research was conducted in Ontario, Canada, using 44 rainfall stations data over a 20-year time frame. To ensure the validity of the results, a systematic approach was taken by employing the Mann-Kendall S trend test and the L- moments procedure to establish homogeneous regions. The study analyzed the annual maximum observations for various durations, including 5, 10, 15, and 30 minutes and 1, 2, 6, and 12 hours. The L-moments procedure helped in delineating 4 or 5 homogeneous regions, depending on the rainfall duration. The results showed that approximately 23% of the regions tested had a significant trend, especially for short-duration storms. Additionally, the study considered the impact of serial and spatial correlation on trend determination. It was observed that serial dependency was present in 2-3% of the data sets, while the spatial correlation was found in 18% of the regions. These findings suggest that the presence of serial and spatial correlation can have a significant impact on trend determination

and must be taken into consideration when estimating trends for intensity- duration-frequency of rainfall.

[13] Nonstationary weather and water extremes: a review of methods for their detection, attribution, and management:

The various drivers that affect the evolution of hydroclimatic extremes, including climate change, land cover change, and human activities such as urbanization, deforestation, and water management practices. These drivers can cause shifts in extreme precipitation patterns, altered atmospheric circulation, and changes in the frequency and severity of droughts, heatwaves, and storms. Metrics used to describe hydroclimatic extremes include indices such as the number of dry days, the intensity of rainfall, and the frequency and duration of floods. These metrics are useful for detecting changes in extreme events and quantifying their impacts.

To assess the nonstationary nature of hydroclimatic extremes, both empirical and simulation-based approaches are used. Empirical methods involve analyzing observational data to detect trends and changes in extreme events. Simulation-based approaches use climate models to simulate extreme events under different future scenarios, including future climate

projections, changes in land use, and other human activities. These approaches can provide insight into the underlying physical processes driving extreme events and the potential impacts of these drivers in the future.

However, the analysis of nonstationary hydroclimatic extremes is not without challenges. Incomplete record lengths, spurious nonstationarities, and limited representation of nonstationary sources in modeling frameworks can result in uncertainty in the analysis. Moreover, the complexity of hydroclimatic extremes and their underlying drivers can lead to difficulties in attributing changes in extreme events to specific drivers.

To address these challenges, future research should focus on improving the accuracy and robustness of methods for detecting and attributing nonstationary extremes, incorporating a better representation of the nonstationary sources of extremes in modeling frameworks, and exploring new methods for quantifying the impacts of hydroclimatic extremes on society.

In conclusion, understanding the drivers and evolution of hydroclimatic extremes is critical for managing the impacts of these events and reducing their devastating effects on society. The review of drivers, metrics, and methods for the analysis of nonstationary hydroclimatic extremes highlights the progress made in recent decades but also identifies key gaps for future research.

[14] Improving an Extreme Rainfall Detection System with GPM IMERG data:

Extreme rainfall events can cause severe impacts such as flash floods, landslides, and soil erosion, leading to loss of life, damage to infrastructure and property, and economic losses. Therefore, having accurate and timely information on the occurrence and location of these events is of utmost importance for emergency response, disaster management, and risk reduction.

Satellite-based precipitation products, such as NASA Global Precipitation Measurement (GPM) IMERG, have the advantage of providing near-global coverage and frequent observations, making them useful for detecting extreme rainfall events. However, their accuracy in representing extreme rainfall events, particularly for short aggregation intervals, has been a subject of concern.

The study mentioned in the passage aimed to address this issue by comparing IMERG data with rain gauge observations and assessing its accuracy in representing extreme rainfall events for different time aggregation intervals. The results showed that IMERG data provides good results for aggregation intervals equal to or greater than 12 hours, with a 24-hour aggregation interval ensuring a probability of detection greater than 80%. The findings of this study were used to develop an updated version of the ITHACA Extreme Rainfall Detection System (ERDS), which is now capable of providing near-real- time alerts about extreme rainfall events using a threshold methodology based on the mean annual precipitation.

In conclusion, the accuracy of satellite-based precipitation products in representing extreme rainfall events is crucial for effective disaster management and risk reduction. The study highlights the importance of using appropriate time aggregation intervals for satellite data and the potential of using satellite-based products for near real-time monitoring of extreme rainfall events.

[15] Estimation of Daily Rainfall Extremes Through the Metastatistical Extreme Value Distribution: Uncertainty Minimization and Implications for Trend Detection:

The accurate estimation of hydrologic extremes is crucial for effective planning and implementation of mitigation and adaptation measures to reduce the impacts of extreme events, such as floods and droughts. However, the traditional extreme value theory has limitations, as it is based on assumptions that preclude the use of all available observations and negatively impact estimation uncertainty. To address this, the Metastatistical Extreme Value Distribution (MEVD) was introduced as an alternative method to make full use of available data and improve estimation uncertainty for large extremes. However, there was a lack of understanding of how to optimally apply the MEVD depending on the statistical properties of the observed variables.

In this study, the authors analyze a large set of long daily rainfall records and synthetic time series to identify the local climatic factors that define the optimal MEVD formulation. They find that in most climates, the MEVD should be based on yearly estimates of the ordinary rainfall distributions, and only in climates with fewer than 20-25 rainy days per year, the estimation of distributional parameters requires samples longer than one year. Additionally, the interannual variability in the distributions of rainfall should be explicitly resolved when there are more than 20-25 rainy days per year.

The optimized MEVD was then used to study the variability of daily rainfall extremes over 294 years in Padova, Italy, and compare it to traditional extreme value estimates. The results show that the MEVD provides improved accuracy for short observations and better resolves high-quantile fluctuations, allowing the emergence of long-term trends over estimation noise.

In conclusion, the MEVD provides a more accurate estimation of hydrologic extremes and should be considered as an alternative to the traditional extreme value theory. The results of this study provide valuable insights into the optimal application of the MEVD for daily rainfall and its potential for improving the understanding and management of extreme events.

EXISTING METHODOLOGY

One class classification method is a binary classification method where the model is trained on only one type of class instance, say normal class and the model predicts whether a new data belongs to the normal class or the abnormal class. This classification method is different from the traditional classification method where the model is trained on all types of class instances. This method can be used if the data is imbalanced. This is also one of the approaches used in anomaly detection or outlier detection. Some of the methods used for one class classification are:

1.Kernel Density Estimation:

The fundamental concept of KDE involves placing a kernel function at every point in the normal class and then combining these kernel functions to approximate the probability density function. When a new instance is given, its probability density value is calculated based on the estimated probability density function. If the probability density value is above a certain threshold, the instance is classified as normal and otherwise as an abnormal case. KDE has high computational complexity as the probability density function has to be calculated for each new instance. KDE assumes that the target class distribution is unimodal, which means it can only model one peak in the data. If the target class distribution has multiple modes, KDE may not be able to capture all the modes, leading to poor performance. Also, KDE is sensitive to bandwidth parameter and distribution of the data.

2.One-class SVM (Support Vector machine):

One-Class SVM works by transforming the input data points to a higher-dimensional feature space using a kernel function, learning the normal patterns of the data by finding the decision boundary which is a hyperplane that separates the normal data points from the rest of the data points, and classifying new data points based on their position with respect to the decision boundary. During training, One-Class SVM tries to find the hyperplane that maximizes the margin between the normal data points and the decision boundary. This means that the hyperplane should be as far away from the normal data points as possible. If a new data point lies on the same side as the normal data points, it is classified as normal, and if it lies on the other side, it is classified as anomalous or outlier. One-class SVM is very sensitive to hyperparameters. Another limitation is that it assumes that the normal data points are a convex region in the feature space. It is also computationally expensive.

3.Support Vector Data Description (SVDD):

SVDD is similar to One-Class SVM but uses a different optimization objective. A kernel function transforms the data into higher dimensional feature space and tries to find the smallest hypersphere that encloses all the normal data points in the transformed feature space, while minimizing the volume of the hypersphere. Sensitivity to outliers, difficulty in choosing kernel function, high computational complexity are some of the drawbacks of SVDD.

4.Isolation Forest:

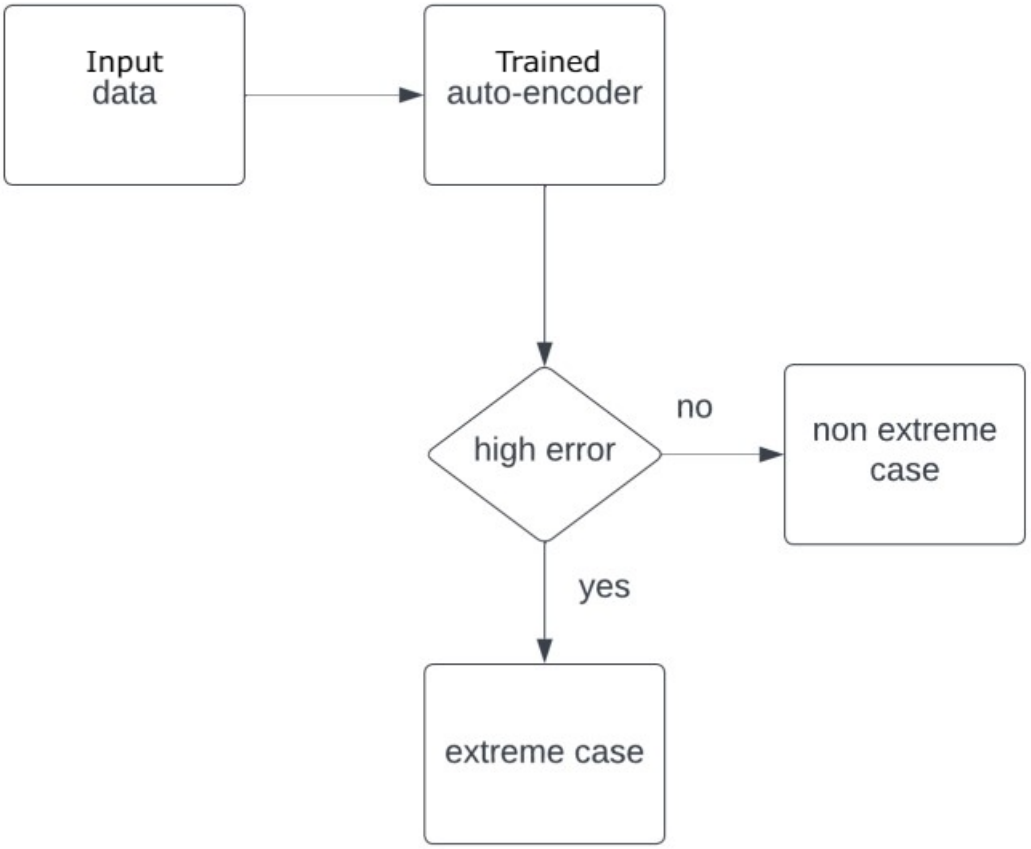
The Isolation Forest algorithm takes as input a set of training data points, where all the data points belong to a single class. A random subset of features and a random threshold are selected, and the data points are partitioned based on whether they are above or below the threshold for the selected feature. This process is repeated recursively to create a binary tree. Multiple such trees are constructed by subsampling features and data points at each node of the tree. For each data point, the length of the path from the root node to the leaf node is calculated. The idea is that normal data points will require more partitions to isolate them, while anomalous data points can be isolated in fewer partitions. An anomaly score is calculated for each data point, which is a function of the path length. Data points with shorter path lengths are assigned higher anomaly scores, indicating that they are more likely to be anomalous. A threshold is set on the anomaly scores to separate normal data points from anomalous data points. If the anomaly score of a new data point is below the threshold, it is classified as normal, and if it is above the threshold, it is classified as an anomalous or outlier. If the number of trees is less, accuracy will be low and if high it can lead to overfitting. Low sample size leads to overfitting and high sample size leads to underfitting. Difficulty in interpreting the model is another limitation.

PROPOSED METHODOLOGY

Our project proposes two methods for detecting extreme scenarios from an imbalanced dataset. The first method involves training the model solely on the majority class and predicting the minority class. The second method trains the model on both majority and minority classes, but assigns a higher error penalty to the majority class and a lower penalty to the minority class, during training and predicts the minority class. These methods are designed to mitigate the impact of class imbalance and improve the accuracy of extreme scenario detection. In both of the above methods we use an autoencoder with custom loss function built specific for the method opted for the training of the model. The trained model is subjected to prediction of the extreme scenarios and the predicted output is classified to

extreme or normal in accordance with the error of each prediction. Autoencoder based method can overcome some of the limitations of other one class classification algorithms like one-class SVM(support vector machine), support vector data description (SVVD) and isolation forest. Lack of assumption of data distribution makes this method more suitable for datasets with complex or unknown data distribution and therefore makes it more robust and flexible. Also, autoencoder based method can efficiently handle high dimensional data and imbalanced data.

ARCHITECTURE DIAGRAM



EXPERIMENT

Data Collection:

The rainfall dataset was collected from the Advanced Centre for Atmospheric Radar Research. It contains information on the amount of rainfall that was recorded at various locations in Kerala. Each record in the dataset represents a single rainfall event and includes details such as wind speed, tpw, and the amount of rainfall in millimeters. The dataset also contains the date of each rainfall record. The wind speed, tpw, and rainfall data were initially separate text files. We created a CSV file that includes columns for date, wind speed, tpw, and rainfall data, and saved this as 'rainfall_data.csv.'

Data Analysis:

To analyze the rainfall data, we added a new column to the dataframe called 'rainfall_category.' We assigned a value to each row based on whether the rainfall amount is greater than a threshold value or not. We set a random rainfall value of 30 and 50 as threshold values. If the rainfall value is greater than the threshold, the data is labeled as Extreme, and if it is less than the threshold, it is labeled as Normal.

Defining Autoencoder:

Once the dataframe data is created with the addition of rainfall category, the next step involves building an autoencoder model. An autoencoder is a neural network used for unsupervised learning. Its purpose is to learn a compressed representation of the input data by reducing the input dimensions and then reconstructing the original data as accurately as possible. The autoencoder is composed of two main parts: the encoder and the decoder. The encoder takes in the input data and maps it to a lower-dimensional representation, referred to as a bottleneck or latent space. The decoder takes the compressed representation and maps it back to the original input space. The objective of the autoencoder is to minimise the difference between the input and the output, which is also known as the reconstruction error.

Hence the model tries to minimise the reconstruction error by making the reconstructed representation as similar as input, the model will be highly trained on the patterns of the minority case which we give as input. Here we create an architecture for the model with

activation function as ReLU(Rectified Linear Unit) ,optimizer as adam and loss function as mean square error as initial hyper parameter.

Reconstruction loss:

For finding the loss of the prediction and labelling the predicted scenarios as extreme and normal according to the loss we load the autoencoder model into two new models: one for the encoder and one for the decoder. The encoder model takes in test data and produces the encoded representation of that data, while the decoder model takes in the encoded representation and produces the reconstructed data. Then we use the encoder model to get the encoded representation of some test data and then use the decoder model to reconstruct that data from the encoded representation. Next, we compute each sample's mean squared error (MSE) reconstruction loss by taking the absolute difference between the original test data and the reconstructed data, squaring it, and summing over all dimensions. The MSE loss for each sample is stored. Finally, the MSE reconstruction loss for each sample is added to the data and a threshold is set for determining the above which is extreme and normal otherwise.

Threshold Analysis 1:

After the preprocessing of data and training of the model we found an error threshold that classifies the rainfall as extreme and normal. For that we performed a threshold analysis on rainfall data. Here we defined a range of threshold values from 0.5 to 10 incremented as 0.1 to classifying rainfall data into normal and extreme categories. Here we iteratively found the best threshold of error that classifies the predicted data as extreme and normal according to the error of each prediction. After classification, the accuracy was obtained from the error. It was also keeping the track of the best threshold and lowest error. Here after some analysis we found out that the error threshold that we got was upto mark.

Grid search method:

Grid search is a hyperparameter tuning method used to find the optimal combination of hyperparameters for a machine learning model. Here four parameters namely activation function, loss function, number of layers and layer size are tested during training with which has a corresponding list of possible values. The activation function that will be used in each layer of the neural network, with the options of relu, tanh, and sigmoid. The loss function that

will be optimised during training, with the options of mse, mae, and logcosh. The number of layers in the neural network, with the options of 1, 2, and 3. Finally, the number of neurons in each layer of the neural network, with the options of 8, 16, and 32. It involves creating a grid of all possible combinations of hyperparameters and testing each combination to determine which one performs best on the basis of a scoring function that utilises mean square error to find the error of each hyperparameter. The model architecture was defined using a range of customizable hyperparameters such as activation, loss, number of layers, and layer size. GridSearchCV function was used to perform a grid search for optimal hyperparameters for the model. The results show that the model with one layer and 16 neurons in the layer, with relu activation function and logcosh loss function, gave the best performance.

Threshold Anlysis 2:

In the preprocessed data, an error threshold of 8.6 has been applied. This threshold value was determined by using a trial and error approach on the provided extreme and normal data sets. By testing various threshold values, it was found that a threshold of 8.6 produced the closest desired results.

Regularisation:

Regularisation is a technique used to prevent overfitting of a model to the training data, and to improve generalisation performance on new, unseen data. Overfitting occurs when a model is too complex or flexible and learns the noise in the training data, resulting in poor generalisation to new data. Regularization techniques add additional constraints or penalties to the model during training, to encourage it to learn simpler patterns that are more likely to generalise well to new data.

In this scenario two common regularisation techniques called L1 and L2 regularisation are used to minimise the overfitting. L1 regularisation, also known as Lasso regularisation, adds a penalty term to the loss function of the model that is proportional to the absolute values of the model's coefficients. This penalty encourages the model to produce sparse coefficients, which means that some of the coefficients will be set to zero. This can be useful in situations

where the dataset has many irrelevant features, as the model can automatically select the most important features and ignore the rest.

Mathematically:

$$L1\ loss = Loss + \lambda * ||w||1$$

where Loss is the original loss function, ||w||1 is the L1 norm of the model's coefficients, and λ is a hyperparameter that controls the strength of the regularisation.

L2 regularisation, also known as Ridge regularisation, adds a penalty term to the loss function of the model that is proportional to the square of the model's coefficients. This penalty encourages the model to produce small coefficients, which helps to prevent overfitting by reducing the complexity of the model. L2 regularisation can be expressed mathematically as follows:

$$L2\ loss = Loss + \lambda * ||w||2^2$$

where ||w||2 is the L2 norm of the model's coefficients, and λ is a hyperparameter that controls the strength of the regularisation.

In the first few lines of the code, an autoencoder model is defined. The autoencoder model consists of an input layer of shape (3,), a hidden layer of 16 neurons with ReLU activation function and L1 regularisation, and an output layer of 3 neurons with no activation function. The autoencoder is compiled with the Adam optimizer and the loss function is set to 'logcosh'. The model is trained on the normal data with 100 epochs, a batch size of 32, and a validation split of 0.2. The training data consists of three features: windspeed, tpw, and rainfall. Once the model is trained, it is used to predict the rainfall values for all the data points. The difference between the predicted and actual rainfall values is calculated and stored in a new column called 'error'. The data points are classified into two categories based on the threshold value of the error. If the error is greater than the threshold, the rainfall is classified as 'Extreme'. Otherwise, it is classified as 'Normal'. The actual rainfall class is defined based on the threshold of 30. If the actual rainfall is greater than 30, it is classified as 'Extreme'. Otherwise, it is classified as 'Normal'. Finally, the accuracy of the model is calculated by comparing the predicted and actual rainfall classes. The number of correct

classifications is divided by the total number of data points and multiplied by 100 to get the percentage accuracy. The result is printed to the console.

Loss function:

Normal Data:

Here we introducing a loss function consist of two parts a mean squared error (MSE) loss and a Kullback-Leibler (KL) divergence term that penalises sparsity in the predictions.The function takes two arguments, sparsity_weight and sparsity_target, which control the strength of the sparsity penalty and the desired level of sparsity, respectively. The MSE loss term calculates the mean squared difference between the true and predicted values for the third column of the input data. The KL divergence term measures the difference between the predicted distribution and a target distribution determined by the sparsity_target parameter. The penalty is stronger for values farther away from the target level, as determined by the ratio of the actual value to the mean value of the predictions. Overall, this loss function encourages the model to make accurate predictions while also favouring sparse predictions, with the strength of the sparsity penalty controlled by the sparsity_weight parameter.

Using Normal and Extreme Data:

This is a custom loss function where the function takes two arguments, normal_weight and extreme_weight, which control the weighting of the loss terms for normal and extreme values, respectively. The loss function consists of two terms, normal_loss and extreme_loss, which are calculated based on whether the true value for the rainfall amount column of the input data is less than or equal to the percentile value.

The normal_loss term calculates the mean squared difference between the true and predicted values for the first three columns of the input data, but only for values where the true value in the third column is less than or equal to the threshold. The extreme_loss term calculates the same mean squared difference, but only for values where the true value in the third column is greater than the threshold. The final loss is the sum of the absolute values of normal_loss and extreme_loss, each multiplied by the corresponding weight. The abs function is used to ensure that the loss is always positive.

Discretization:

Discretization is a technique used in machine learning to transform continuous variables into categorical variables by dividing a range of values into smaller intervals or bins. Discretization can be used to address data sparsity in high-dimensional datasets, where continuous variables may have few or no observations in some regions of the feature space. By discretizing the variables, the model can use the counts of observations in each bin as a proxy for the continuous variable. Three discretization techniques are implemented and compared here they are:

Decision Tree discretization:

Here we perform discretization on the continuous variables "rainfall", "windspeed", and "tpw" using a decision tree. The decision tree algorithm is used to identify the cut-off values that create the intervals. In this case, the decision tree is trained on the "rainfall" variable and then applied to the "windspeed" and "tpw" variables to create new columns with discretized values. The max_depth parameter of the decision tree is set to 4, which limits the tree to 4 levels of decision-making. The random_state parameter is set to 42 to ensure the reproducibility of results. We created equally spaced bins for each continuous variable. The bins parameter is set to 6, which creates 6 intervals for each variable. We then predict the interval for each observation in the dataset for each variable and stored in new columns "rainfall_tree", "windspeed_tree", and "tpw_tree" respectively. The result is a dataset with the original continuous variables and new categorical variables representing the discretized values.

K-means clustering:

K-means clustering is another technique that can be used for discretization of continuous variables. The k-means algorithm is a unsupervised learning algorithm that groups similar data points together into clusters based on their similarity. To perform discretization using k-means clustering, we would first select the number of clusters, K, to create. This would determine the number of intervals or bins that the continuous variable will be divided into. We would then apply the k-means algorithm to the continuous variable, using the K value as the number of clusters to create. The algorithm would group similar values together into clusters, and each data point would be assigned to the cluster that it is closest to. The cluster

assignments can then be used to create new categorical variables representing the discretized values. For example, we could create a new variable with labels ranging from 1 to K, where each label represents one of the K clusters. In summary, the k-means clustering algorithm can be used for discretization by grouping similar data points into clusters based on their similarity.

Fuzzy discretization:

Fuzzy discretization is another technique that can be used to discretize continuous variables. Unlike k-means clustering and decision tree-based discretization, fuzzy discretization assigns continuous values to multiple overlapping intervals or bins. To perform fuzzy discretization, we would first determine the number of bins or intervals to create, as well as the degree of overlap between the intervals. This can be specified using a parameter called the fuzzifier, which determines the degree of overlap between the intervals. Next, we would define the intervals or bins by specifying a set of fuzzy membership functions. These functions describe how much each data point belongs to each interval, based on the similarity between the data point and the centre of the interval. The fuzzy membership functions can be created using various methods, such as clustering, gradient descent, or expert knowledge. Once the membership functions are defined, we can apply them to the continuous variable to determine the degree of membership for each data point in each interval. The resulting membership values can be used to create new categorical variables representing the discretized values. For example, we could create a new variable with labels ranging from 1 to K, where each label represents one of the K intervals, and the membership values for each data point indicate the degree of membership to each interval. In summary, fuzzy discretization is a technique that assigns continuous values to multiple overlapping intervals or bins, based on the degree of membership to each interval. This is determined by a set of fuzzy membership functions, which describe the degree of similarity between each data point and the centre of each interval. The resulting membership values can be used to create new categorical variables representing the discretized values. The advantage of fuzzy discretization over other techniques is that it allows for a more fine-grained representation of the data, capturing the degree of membership to each interval.

If the data has clear thresholds or ranges that define extreme rainfall events, decision tree discretization may be a good method to use. Decision tree discretization can handle both categorical and continuous variables and can be adapted to capture complex relationships in

the data. If the data has well-defined clusters that correspond to extreme rainfall events, k-means clustering may be a good method to use. K-means clustering can group similar values of a continuous variable into clusters and may be able to capture the patterns in the data that correspond to extreme rainfall events. If the relationship between the variables in the dataset is complex and non-linear, fuzzy discretization may be a good method to use. Fuzzy discretization can handle uncertainty and non-linearities in the data and can be used when the relationship between the variables is complex.

For our dataset, the data is extremely complex and nonlinear, so the discretization method adopted here is fuzzy discretization.

Error Threshold: AUPRC and ROC Curves:

As we came forward, we discovered that assigning a random value for threshold was not an effective approach for obtaining optimum results. Therefore, we decided to automate the process and explore various methods for determining the threshold. Our earlier method of obtaining threshold for different datasets did not yield satisfactory results, hence the need to implement different approaches.

One such method that we have begun implementing is AUPRC(The Area Under the Precision-Recall Curve).It is a metric used to evaluate the performance of a binary classification model. Unlike other metrics such as accuracy or F1 score, AUPRC takes into account the trade-off between precision and recall, which is especially important when dealing with imbalanced datasets, where the number of examples from one class is much larger than the other. The precision-recall curve is created by plotting the precision and recall values obtained for different probability thresholds used to classify examples as positive or negative. The AUPRC is then calculated as the area under this curve. A perfect classifier would have a precision-recall curve that passes through the point (1,1) and the AUPRC value would be equal to 1. On the other hand, a random classifier would have a curve that is a horizontal line at the level of the base rate of the positive class, and the AUPRC value would be equal to the ratio of the number of positive examples to the total number of examples. We choose any threshold values to predict extreme and lower rainfall conditions, such as 30, 40, or 50, the error threshold value becomes very high. Due to this high error threshold value, we are unable to predict the rainfall accurately. Therefore, the AUPRC method is not applicable in the given dataset.

Another important methode we've tried to evaluate the performance is Receiver Operating Characteristic (ROC) curve. An ROC curve is a graphical representation of the performance of a classification model at all possible classification thresholds. It is used to determine the trade-off between true positive rate (TPR) and false positive rate (FPR) of a classifier model. In an ROC curve, we input the data and try to find the area under the curve (AUC) of the data. The AUC represents the degree to which a classifier is capable of distinguishing between classes. Maximising the AUC is important for obtaining the best possible results. The ROC curve has an AUC (area under the curve) of 1. We have identified the best threshold for different data points by setting rainfall values above 30 as extreme and below normal. For this threshold, the best value we obtained was 28.4. Next, we assigned rainfall values above 50 as extreme and below normal, and the best threshold for this value was found to be 54.45419348421875, which is a value with multiple decimal places. Using such a value may result in slight errors at certain points, so it is recommended to round it off to one or two decimal places to obtain a suitable threshold value. By doing so, we can achieve better classification using this method.

By comparing both ROC and AUPRC curves we understood that, ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) for different threshold values. The point on the curve closest to the top-left corner represents the best threshold for the classification task, which maximises the TPR while minimising the FPR. On the other hand, AUPRC curve plots the precision-recall trade-off for different threshold values. While it can also be used to find the best threshold for classification, it may not be as effective as ROC curve in situations where the positive class is rare, which is often the case with extreme rainfall events. This is because AUPRC curve places more emphasis on precision, which may not be as important in this specific task as correctly identifying all extreme rainfall events.

Loss Function:

Kl divergence:

Kullback-Leibler (KL) divergence is a mathematical measure that quantifies how different two probability distributions are from each other. In machine learning, it is often used as a regularisation term in the loss function to encourage the model to make more "sparse" predictions. This is because sparse predictions are often more desirable than non-sparse predictions, especially in cases where the output variable has a lot of noise or where there is

limited training data. By adding a KL divergence term to the loss function, the model is encouraged to make predictions that are closer to zero and thus more "sparse". This can help prevent overfitting and improve the generalisation performance of the model.

Mean square Error:

MSE stands for Mean Squared Error, which is a commonly used loss function in regression problems. It measures the average squared difference between the predicted and actual values of the output variable. Specifically, MSE is calculated by taking the average of the squared differences between the predicted and actual values of each sample in the dataset. A lower MSE indicates that the model is making more accurate predictions. MSE is a popular choice of loss function because it is differentiable, computationally efficient, and well-suited for gradient descent optimization algorithms.

The code defines a custom loss function for a neural network. The loss function combines two terms: mean squared error (MSE) and Kullback-Leibler (KL) divergence. The MSE term measures the difference between the predicted and true values for a particular output variable of the neural network. The KL divergence term penalises the model for having too many "sparse" predictions (values close to zero), by comparing the proportion of "sparse" predictions in the model to a desired sparsity target. The weight of the sparsity term can be adjusted using the sparsity_weight parameter. By combining the MSE and KL terms, the loss function encourages the model to make accurate predictions while also avoiding overly sparse outputs.

Interpolation:

Interpolation is a mathematical technique used to estimate a value between two known values. It involves constructing a function that passes through the given data points, and then using this function to estimate the value of a point that lies within the range of the data. There are several methods of interpolation, including linear interpolation, polynomial interpolation, and spline interpolation. The method used here is linear interpolation. Linear interpolation

involves drawing a straight line between two data points and estimating the value of the unknown point based on where it lies along that line.

The dataset contains both extreme and normal rainfall values. To preprocess the data, we will replace all extreme rainfall values with NaN values and randomly select some normal rainfall values to also replace with NaN values. To predict future rainfall amounts and classes, we are currently employing linear interpolation since we do not have access to precise data on future rainfall amounts. Here missing data points in the specified column is identified, and then for each missing data point, it finds nearby data points with valid values for specified columns. It uses these nearby data points to estimate the missing value using linear interpolation. The code then updates the missing value in the DataFrame. The result is an updated DataFrame with interpolated missing values. Then a for loop is used to iterate over each missing value, and it computes the slope between the neighbouring data points using simple linear regression. It uses this slope to estimate the missing value and updates the DataFrame with the estimated value. The interpolation is performed using the 'max_diffs' parameter, which specifies the maximum difference allowed between the neighbouring data points for interpolation. Through this method, we have obtained a maximum deviation of 55.16381877217265 and a minimum deviation of 0.013460542172641254 between the predicted and actual rainfall amounts.

IMPLEMENTATION

In [168…
```
import numpy as np
import pandas as pd

# create example data
data = pd.read_csv("rainfall_data.csv")
data=data.iloc[:,3:]
# calculate 95th percentile for each column
percentiles = data.quantile(0.95)

# find rows with values above the 95th percentile
above_95th = data[(data > percentiles).all(axis=1)]

print(above_95th)
```

```
       rainfall
6     24.932436
7     23.246956
41    33.846075
64    23.561225
83    22.738915
...         ...
2534  21.900691
2538  24.230811
2541  22.366934
2551  34.073259
2552  21.649361

[129 rows x 1 columns]
```

In [169…
```
import pandas as pd
data = pd.read_csv("rainfall_data.csv")
data=data.iloc[:,3:]
# calculate 95th percentile for each column
percentiles = data.quantile(0.95)

# filter rows above 95th percentile for each column
filtered_data = data[(data > percentiles).any(axis=1)]




print(filtered_data)
```

```
       rainfall
6     24.932436
7     23.246956
41    33.846075
64    23.561225
83    22.738915
...         ...
2534  21.900691
2538  24.230811
2541  22.366934
2551  34.073259
2552  21.649361

[129 rows x 1 columns]
```

In [ ]:
```

```

In [112…
```
percentiles
```

Out[112]:
```
rainfall    20.807473
Name: 0.95, dtype: float64
```

In [22]:
```
import pandas as pd
data = pd.read_csv("rainfall_data.csv")
data=data.iloc[:,3:]
# calculate 95th percentile for each column
percentiles = data.quantile(0.95)

# filter rows above 95th percentile for each column
filtered_data = data[(data > percentiles).any(axis=1)]




print(filtered_data)
```

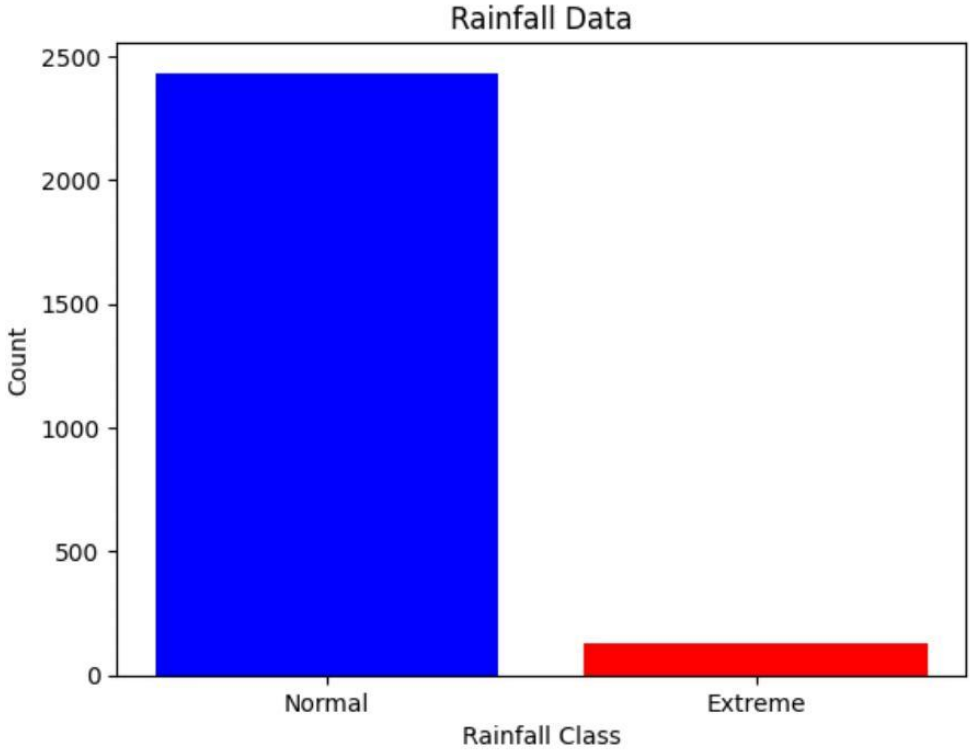
```
       rainfall
6     24.932436
7     23.246956
41    33.846075
64    23.561225
83    22.738915
...         ...
2534  21.900691
2538  24.230811
2541  22.366934
2551  34.073259
2552  21.649361

[129 rows x 1 columns]
```

In [23]:
```
percentiles
```

Out[23]:
```
rainfall    20.807473
Name: 0.95, dtype: float64
```

In [24]:
```
data = pd.read_csv("rainfall_data.csv")
# label extreme rows in original dataframe
data['rainfall_class'] = 'normal'
data.loc[data.index.isin(filtered_data.index), 'rainfall_class'] = 'extreme'
```

In [79]:
```
import matplotlib.pyplot as plt

# Count the number of extreme and normal rainfall values
extreme_count = len(data[data['rainfall_class'] == 'extreme'])
normal_count = len(data[data['rainfall_class'] == 'normal'])

# Plot the counts as a bar chart
plt.bar(['Normal', 'Extreme'], [normal_count, extreme_count], color=['blue', 'red'])

# Add labels and title
plt.xlabel('Rainfall Class')
plt.ylabel('Count')
plt.title('Rainfall Data')

# Show the plot
plt.show()
```



In [25]:
```
import numpy as np
import copy
import tensorflow as tf
from sklearn import preprocessing
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from keras import backend as K
import os
import pandas as pd
import matplotlib.pyplot as plt
from numpy.random import seed
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

In [6]:
```
extreme_data=data[data["rainfall_class"]=="extreme"]
normal_data=data[data["rainfall_class"]=="normal"]
```

In [26]:
```
def weighted_loss(sparsity_weight=0.01, sparsity_target=0.001):
    def loss(y_true, y_pred):
        mse_loss = K.mean(K.square(abs(y_true[:,2] - y_pred[:,2])), axis=-1)
        kl_divergence = sparsity_weight * K.sum(sparsity_target * K.log(abs(sparsity_tar
                        + abs((1 - sparsity_target)) * K.log(abs((1 - sparsity_target) /
        return mse_loss + kl_divergence
    return loss
```

In [27]:
```
def find_best_reg_strength(train_data, reg_strengths, num_epochs, batch_size):
    best_reg_strength = None
    best_loss = float('inf')
    for reg_strength in reg_strengths:
        # Define the autoencoder architecture
```

```
        input_layer = Input(shape=(3,))
        encoded = Dense(16, activation='relu', kernel_regularizer=tf.keras.regularizers.
        decoded = Dense(3, activation=None)(encoded)

        # Create the autoencoder
        autoencoder = Model(input_layer, decoded)

        # Compile the autoencoder with the weighted loss function
        #autoencoder.compile(optimizer='adam', loss="mse")
        autoencoder.compile(optimizer='adam', loss=weighted_loss())
        # Train the model
        history = autoencoder.fit(train_data, train_data, epochs=num_epochs, batch_size=
        #history = autoencoder.fit(train_data,epochs=num_epochs, batch_size=batch_size,

        # Calculate the validation loss
        val_loss = np.mean(history.history['val_loss'])

        # Update the best regularization strength and loss
        if val_loss < best_loss:
            best_reg_strength = reg_strength
            best_loss = val_loss

    print('Best regularization strength:', best_reg_strength)
    return best_reg_strength
```

In [29]:

```
reg_strengths = [0.1, 1, 10, 100]
num_epochs = 3000
batch_size = 100
#train_data = normal_data[["windspeed_discretized","tpw_discretized"]].values.astype("fl
train_data = normal_data[["windspeed","tpw","rainfall"]].values.astype("float32")

#best_reg_strength = find_best_reg_strength(train_data, reg_strengths, num_epochs, batch

# Define the autoencoder architecture with the best regularization strength
input_layer = Input(shape=(3,),name='input')
encoded = Dense(16, activation='relu', kernel_regularizer=tf.keras.regularizers.L2(l2=20
decoded = Dense(3, activation=None, name='decoder')(encoded)

# Create the autoencoder
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder with the weighted loss function
#autoencoder.compile(optimizer='adam', loss="mse")
autoencoder.compile(optimizer='adam', loss=weighted_loss())

# Train the model with the majority class
history = autoencoder.fit(train_data, train_data, epochs=num_epochs, batch_size=batch_si
#history = autoencoder.fit(train_data,epochs=num_epochs, batch_size=batch_size, validati
```

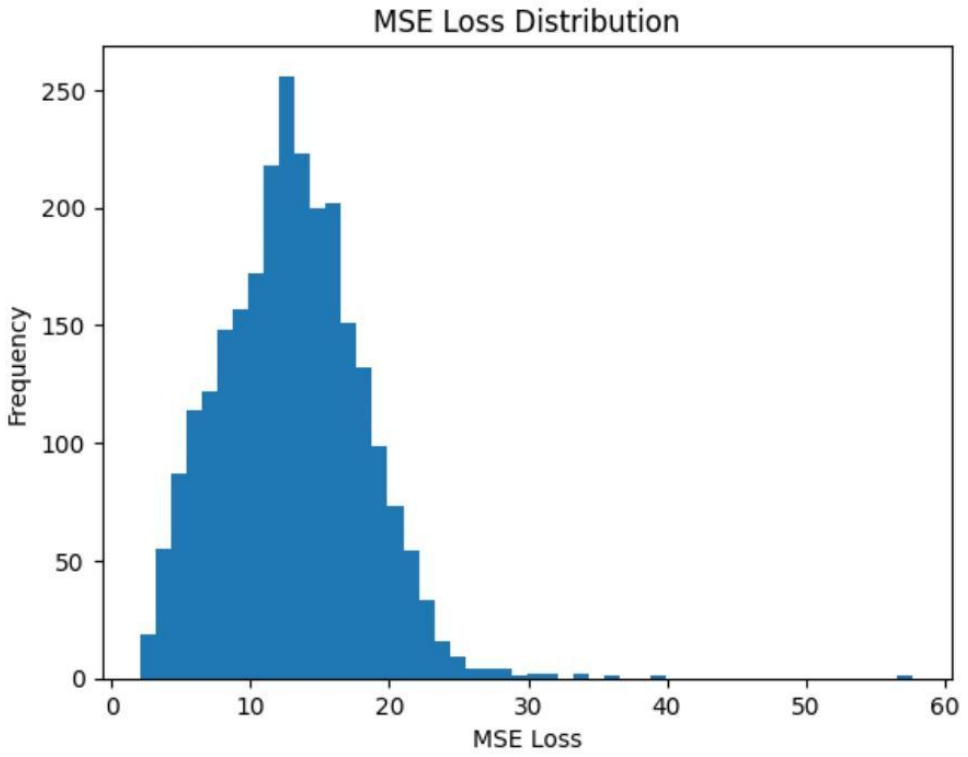
```
Epoch 1/3000
20/20 [==============================] - 1s 10ms/step - loss: 182.8936 - val_loss: 169.6
488
Epoch 2/3000
20/20 [==============================] - 0s 2ms/step - loss: 159.3827 - val_loss: 148.54
97
Epoch 3/3000
20/20 [==============================] - 0s 2ms/step - loss: 139.5323 - val_loss: 130.46
30
Epoch 4/3000
20/20 [==============================] - 0s 2ms/step - loss: 122.3490 - val_loss: 115.03
75
Epoch 5/3000
20/20 [==============================] - 0s 2ms/step - loss: 107.6919 - val_loss: 101.46
04
Epoch 6/3000
```

```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import load_model
from tensorflow.keras.models import load_model

# Define the custom loss function

# Load your autoencoder model
autoencoder = load_model("my_autoencoder_model.h5", custom_objects={"loss": weighted_los
#autoencoder = load_model("my_autoencoder_model.h5")
# Create a Keras model to get the output of the encoder and decoder layers
encoder = Model(inputs=autoencoder.input, outputs=autoencoder.get_layer("encoder").outpu
decoder = Model(inputs=autoencoder.get_layer("decoder").input, outputs=autoencoder.outpu

# Get the encoded representation of the input data
encoded_X = encoder.predict(test_data)

# Reconstruct the input data from the encoded representation
reconstructed_X = decoder.predict(encoded_X)
#print(test_data)
print(reconstructed_X)
# Compute the MSE reconstruction loss for each sample
mse_loss = np.sum((abs(test_data - abs( reconstructed_X))), axis=1)
#mse_loss = abs(np.sum(test_data)-np.sum(reconstructed_X), axis =1)
data["mse_loss"]=mse_loss
# Print the MSE reconstruction loss for each sample
print(data["mse_loss"])
top10_indices = mse_loss.argsort()[::-1][:10]

# Print the MSE reconstruction loss for each of the top 10 samples
for i in top10_indices:
    print('MSE for sample', i+1, ':', mse_loss[i])
sample_idx = 2387  # 0-based index of sample 5
print('MSE loss for sample 2387:', mse_loss[sample_idx])
sample_idx = 2388  # 0-based index of sample 5
print('MSE loss for sample 2388:', mse_loss[sample_idx])
```

```
81/81 [==============================] - 0s 563us/step
81/81 [==============================] - 0s 550us/step
[[ 0.2862509  -1.3080093   4.657294  ]
 [ 0.13247353 -1.218514   11.272638  ]
 [ 0.23515391 -1.2782762   6.8540053 ]
 ...
 [ 0.35295653 -1.3468888   1.8042814 ]
 [ 0.34419057 -1.3417888   2.1819296 ]
 [ 0.10071635 -1.2000943  12.654963  ]]
0       13.265363
1       13.924608
2       13.995251
3       16.166170
4       19.459686
          ...
2557     6.612772
2558     6.187747
2559     5.476810
2560     5.271360
2561     6.669413
Name: mse_loss, Length: 2562, dtype: float64
MSE for sample 2388 : 57.716224716840365
MSE for sample 2389 : 39.57602340111249
MSE for sample 877 : 36.0362178513443
MSE for sample 2273 : 34.04263466497363
MSE for sample 2387 : 33.893836112859375
MSE for sample 553 : 31.648751951855854
MSE for sample 2386 : 31.047322926619508
MSE for sample 2506 : 30.70819402798314
MSE for sample 2272 : 29.97858930104309
```

```
MSE for sample 1389 : 29.413159858731646
MSE loss for sample 2387: 57.716224716840365
MSE loss for sample 2388: 39.57602340111249
```

In [80]:
```
import matplotlib.pyplot as plt

# Create a histogram of the MSE loss distribution
plt.hist(mse_loss, bins=50)
plt.xlabel("MSE Loss")
plt.ylabel("Frequency")
plt.title("MSE Loss Distribution")
plt.show()
```



In [33]:
```
df_sorted = data.sort_values('mse_loss', ascending=False)
```

In [34]:
```
df_sorted
```

Out[34]:

	date	windspeed	tpw	rainfall	rainfall_class	mse_loss
2387	8/9/2019	21.575483	0.042829	80.636074	extreme	57.716225
2388	8/10/2019	20.276800	0.043114	58.431061	extreme	39.576023
876	6/23/2007	23.908970	0.331606	49.356381	extreme	36.036218
2272	8/16/2018	22.673685	0.060061	47.362358	extreme	34.042635
2386	8/8/2019	22.589184	0.024273	47.165503	extreme	33.893836
...	...	...	...	...	...	...
1831	6/2/2015	1.298863	0.008272	4.517705	normal	2.333776
1097	9/30/2008	1.304361	0.000761	0.347961	normal	2.289957
957	9/12/2007	1.079963	0.002913	8.226346	normal	2.170729
1832	6/3/2015	1.063474	0.004362	4.542275	normal	2.102499

1334	9/23/2010	1.015756	0.053173	9.300362	normal	2.070439

2562 rows × 6 columns

In [70]:
```
df_max = df_sorted.nlargest(129, 'mse_loss')
threshold=np.mean(df_max["mse_loss"])
```

In [71]:
```
data['predicted_rainfall_class'] = np.where(data['mse_loss'] > threshold, 'extreme', 'no
data[data["mse_loss"]>threshold]
```

Out[71]:

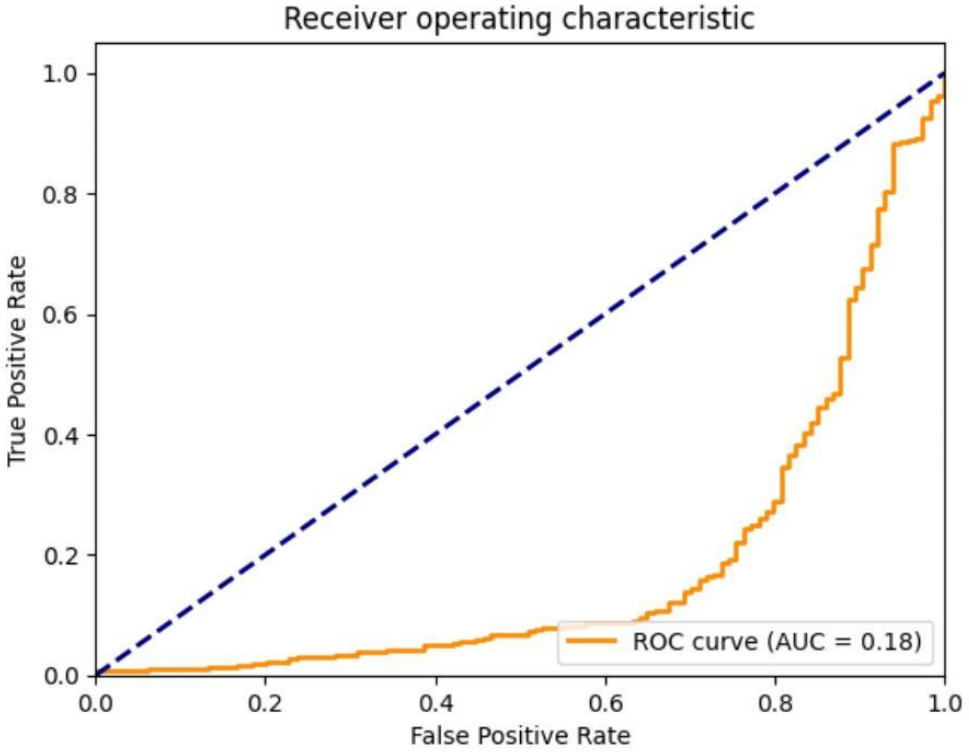
	date	windspeed	tpw	rainfall	rainfall_class	mse_loss	predicted_rainfall_class	actual_rainfall_t
41	7/12/2000	23.003010	0.016358	33.846075	extreme	27.530543	extreme	
85	8/25/2000	22.623075	0.092310	22.514946	extreme	23.929330	extreme	
158	7/7/2001	23.248583	0.019830	13.876615	normal	24.415082	extreme	
159	7/8/2001	23.883757	0.021224	23.128264	extreme	25.353500	extreme	
501	6/14/2004	22.810444	0.010272	17.305405	normal	24.020653	extreme	
552	8/4/2004	21.769238	0.043448	44.404364	extreme	31.648752	extreme	
763	7/2/2006	24.526026	0.007815	12.667222	normal	25.689046	extreme	
764	7/3/2006	23.953388	0.010426	19.001666	normal	25.108775	extreme	
876	6/23/2007	23.908970	0.331606	49.356381	extreme	36.036218	extreme	
1033	7/28/2008	20.585272	0.014900	32.559895	extreme	24.460867	extreme	
1142	7/15/2009	22.801527	0.062284	26.020044	extreme	24.676454	extreme	
1143	7/16/2009	21.342918	0.037500	38.754120	extreme	28.350547	extreme	
1144	7/17/2009	21.451813	0.105873	39.511546	extreme	28.776564	extreme	
1388	7/17/2011	19.214113	0.057870	45.056388	extreme	29.413160	extreme	
1481	6/18/2012	16.490360	0.060908	47.977786	extreme	28.176472	extreme	
1647	8/1/2013	23.105871	0.013754	14.409518	normal	24.285670	extreme	
1769	8/1/2014	20.457104	0.029008	35.467946	extreme	25.799962	extreme	
1849	6/20/2015	24.939993	0.007733	19.569918	normal	26.080169	extreme	
1850	6/21/2015	25.501999	0.008071	17.963338	normal	26.694657	extreme	
1851	6/22/2015	23.407738	0.004632	24.639903	extreme	25.127390	extreme	
1980	6/29/2016	21.279696	0.040968	30.296499	extreme	23.975678	extreme	
2239	7/14/2018	24.540573	0.024890	12.268159	normal	25.680935	extreme	
2240	7/15/2018	24.236286	0.041971	10.012399	normal	25.328427	extreme	
2241	7/16/2018	24.174835	0.032131	13.464989	normal	25.323967	extreme	
2270	8/14/2018	21.997007	0.067254	29.882922	extreme	24.462188	extreme	
2271	8/15/2018	21.421936	0.086819	41.892355	extreme	29.978589	extreme	
2272	8/16/2018	22.673685	0.060061	47.362358	extreme	34.042635	extreme	
2385	8/7/2019	21.010150	0.027716	44.681781	extreme	31.047323	extreme	
2386	8/8/2019	22.589184	0.024273	47.165503	extreme	33.893836	extreme	
2387	8/9/2019	21.575483	0.042829	80.636074	extreme	57.716225	extreme	

2388	8/10/2019	20.276800	0.043114	58.431061	extreme	39.576023	extreme
2504	8/4/2020	25.645590	0.063940	23.203779	extreme	27.080824	extreme
2505	8/5/2020	27.123528	0.037294	32.042267	extreme	30.708194	extreme
2506	8/6/2020	24.605556	0.025061	27.547421	extreme	26.749458	extreme
2507	8/7/2020	19.529327	0.108583	42.017340	extreme	28.129189	extreme

In [72]:
```
wrong_predictions = data[data['predicted_rainfall_class'] != data['rainfall_class']]

percentage_of_error=(wrong_predictions.shape[0]/data.shape[0])*100

# Print the wrong predictions
print("wrong predictions = ",wrong_predictions.shape[0])
print("total data = ",data.shape[0])
print("percentage of error = ",percentage_of_error)
```

```
wrong predictions =  114
total data =  2562
percentage of error =  4.449648711943794
```

In [73]:
```
data['actual_rainfall_binary'] = (data['predicted_rainfall_class'] == data['rainfall_cla

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(data['actual_rainfall_binary'], data['mse_loss'])
roc_auc = auc(fpr, tpr)
# Calculate TNR
tnr = 1 - fpr

# Calculate Youden's J statistic for each threshold
j_stat = tpr + tnr - 1

# Find the index of the threshold that maximizes J
best_threshold_idx = np.argmax(j_stat)

# Get the best threshold
best_threshold = thresholds[best_threshold_idx]

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()


# Print the best threshold
print("Best threshold:", best_threshold)
```




```
Best threshold: 26.74945766797516
```

In [74]:
```
threshold=best_threshold
```

In [75]:
```
data['predicted_rainfall_class'] = np.where(data['mse_loss'] > threshold, 'extreme', 'no
data[data["mse_loss"]>threshold]
```

Out[75]:

	date	windspeed	tpw	rainfall	rainfall_class	mse_loss	predicted_rainfall_class	actual_rainfall_t
41	7/12/2000	23.003010	0.016358	33.846075	extreme	27.530543	extreme	
552	8/4/2004	21.769238	0.043448	44.404364	extreme	31.648752	extreme	
876	6/23/2007	23.908970	0.331606	49.356381	extreme	36.036218	extreme	
1143	7/16/2009	21.342918	0.037500	38.754120	extreme	28.350547	extreme	
1144	7/17/2009	21.451813	0.105873	39.511546	extreme	28.776564	extreme	
1388	7/17/2011	19.214113	0.057870	45.056388	extreme	29.413160	extreme	
1481	6/18/2012	16.490360	0.060908	47.977786	extreme	28.176472	extreme	
2271	8/15/2018	21.421936	0.086819	41.892355	extreme	29.978589	extreme	
2272	8/16/2018	22.673685	0.060061	47.362358	extreme	34.042635	extreme	
2385	8/7/2019	21.010150	0.027716	44.681781	extreme	31.047323	extreme	
2386	8/8/2019	22.589184	0.024273	47.165503	extreme	33.893836	extreme	
2387	8/9/2019	21.575483	0.042829	80.636074	extreme	57.716225	extreme	
2388	8/10/2019	20.276800	0.043114	58.431061	extreme	39.576023	extreme	
2504	8/4/2020	25.645590	0.063940	23.203779	extreme	27.080824	extreme	
2505	8/5/2020	27.123528	0.037294	32.042267	extreme	30.708194	extreme	
2507	8/7/2020	19.529327	0.108583	42.017340	extreme	28.129189	extreme	

In [76]:
```
wrong_predictions = data[data['predicted_rainfall_class'] != data['rainfall_class']]
percentage_of_error=(wrong_predictions.shape[0]/data.shape[0])*100

# Print the wrong predictions
print("wrong predictions = ",wrong_predictions.shape[0])
print("total data = ",data.shape[0])
print("percentage of error = ",percentage_of_error)
```

```
wrong predictions =  113
total data =  2562
percentage of error =  4.410616705698673
```

In [49]:
```
num_correct = len(data[data['predicted_rainfall_class'] == data['rainfall_class']])
num_total = len(data)
accuracy = num_correct / num_total * 100
print('Accuracy: {:.2f}%'.format(accuracy))
```

```
Accuracy: 95.59%
```

In [ ]:

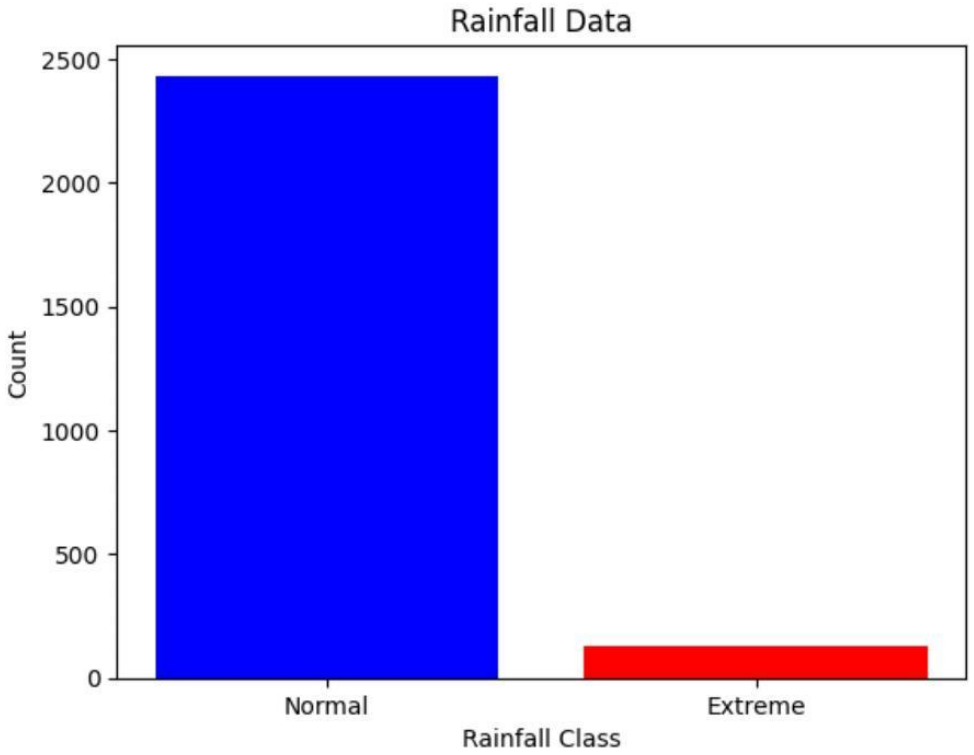
In [ ]:

In [ ]:

In [ ]:

In [170…
```
data = pd.read_csv("rainfall_data.csv")
# label extreme rows in original dataframe
data['rainfall_class'] = 'normal'
data.loc[data.index.isin(filtered_data.index), 'rainfall_class'] = 'extreme'
```

In [214…
```
import matplotlib.pyplot as plt

# Count the number of extreme and normal rainfall values
extreme_count = len(data[data['rainfall_class'] == 'extreme'])
normal_count = len(data[data['rainfall_class'] == 'normal'])

# Plot the counts as a bar chart
plt.bar(['Normal', 'Extreme'], [normal_count, extreme_count], color=['blue', 'red'])

# Add labels and title
plt.xlabel('Rainfall Class')
plt.ylabel('Count')
plt.title('Rainfall Data')

# Show the plot
plt.show()
```



In [171…
```
(data[data["rainfall"]>20.807473]).count()
```

Out[171]:
```
date             129
windspeed        129
tpw              129
rainfall         129
rainfall_class   129
dtype: int64
```

In [172…
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your data into a Pandas DataFrame
df = data
```

DISCUSSION

The project proposes two methods for detecting extreme scenarios from an imbalanced dataset. Extreme scenarios are rare events that occur and predicting of extreme events are important. The imbalance in the dataset arises because the majority of instances belong to the normal class, while only a small number of instances belong to the extreme class. This imbalance can make it difficult for accurately predicting.

Method 1: The first method involves training the model solely on the majority class and predicting the minority class. This approach takes advantage of the fact that the model which is trained on the majority class will produce a larger error for the minority class than the majority class while prediction. We utilise this variation in error for accurately labelling the predicted scenarios as extreme and normal.

Method 2: The second method trains the model on both majority and minority classes, but assigns a higher error penalty to the majority class and a lower penalty to the minority class during training and predicts the minority class. This approach takes advantage of the fact that the model which is trained on the majority class will produce a large error for the minority class than the majority class while prediction. We utilize this variation in error for accurately labelling the predicted scenarios as extreme and normal.

In both of the above methods we use an autoencoder with custom loss function built specific for the method opted for the training of the model. The trained model is subjected to prediction of the extreme scenarios and the predicted output is classified to extreme or normal in accordance with the error of each prediction. This approach is suitable for imbalanced datasets, as it allows for accurate detection of extreme scenarios while minimising the impact of the class imbalance.

Here, while training the model with the same hyper parameters for both the method1 and method 2, we found out that method 1 had accuracy of 95.6% and method 2 had an accuracy of 95.3%.

Method 3: In this method we train the model using windspeed and tpw data for predicting whether the rainfall amount is extreme or normal. Here we found out that there is no direct link between the features and the predictions. Hence we opted for further future feature collection.

CONCLUSION

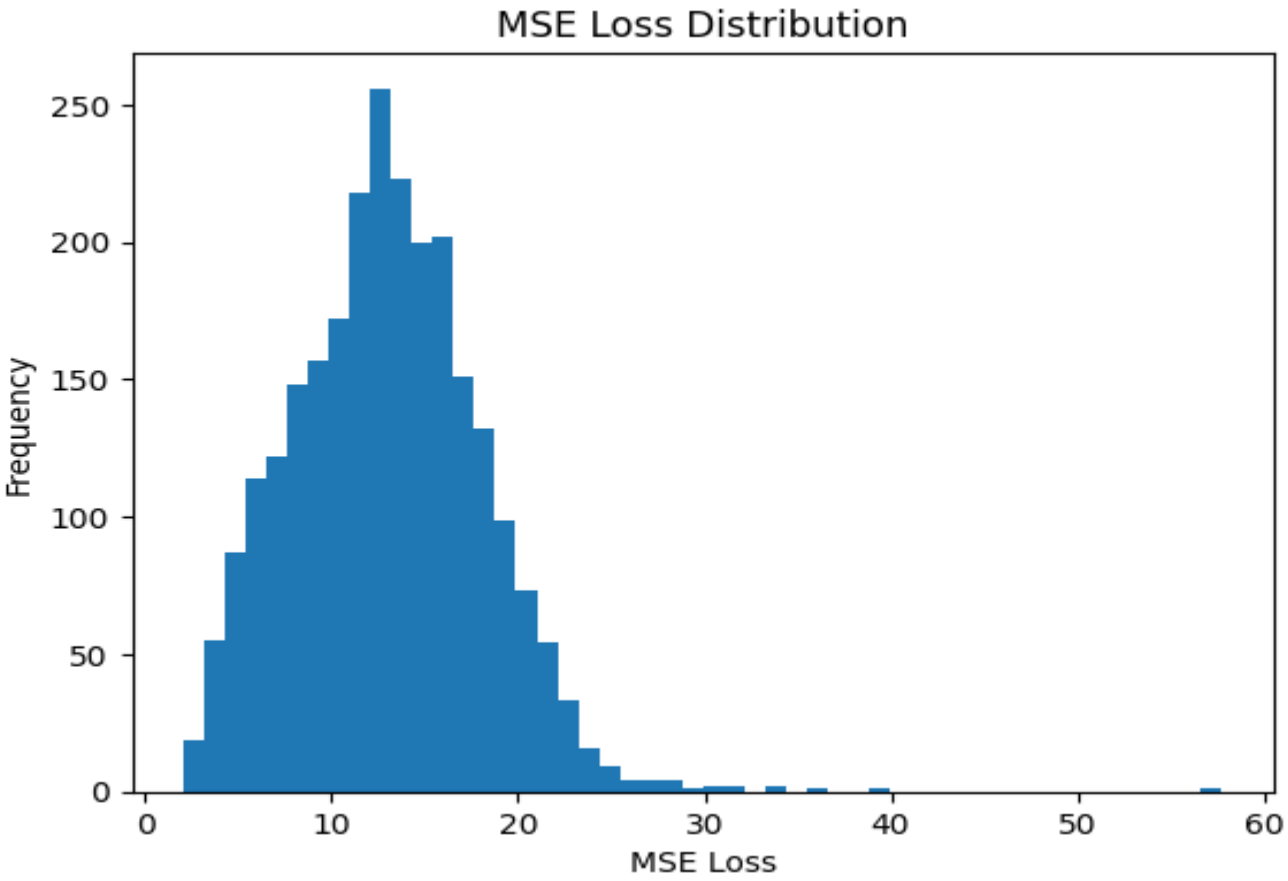
Here we tried out two approaches. In the first one we trained the model with only the majority class and tried predicting the minority class and in the second one we trained the model with both the majority and minority classes. In both scenarios we got around 95% accuracy with the first method having a slight increase in performance. Here, while training the model with the same hyper parameters for both the method 1 and method 2, we found out that method 1 had accuracy of 95.6% with 113 out of 2562 predictions are wrong. There for the error rate is 4.41%. Best error threshold value is 26.74. Method 2 had an accuracy of 95.3% with 24 out of 513 predictions are wrong. There for the error rate is 4.67%. Best error threshold value is 30.70. The interpolation and training of the model using only wind speed and tpw features proved that the features aren't linear to the prediction. hence further analysis and improvements of feature collections are needed.

FUTURE WORK

- Add more features to the dataset to improve the accuracy of predicted rainfall amounts.
- Experiment with different interpolation techniques such as polynomial and spline interpolation.
- Test the model's performance on other datasets similar to the current scenario to validate its accuracy.
- Implement the LSTM architecture to predict future rainfall amounts.

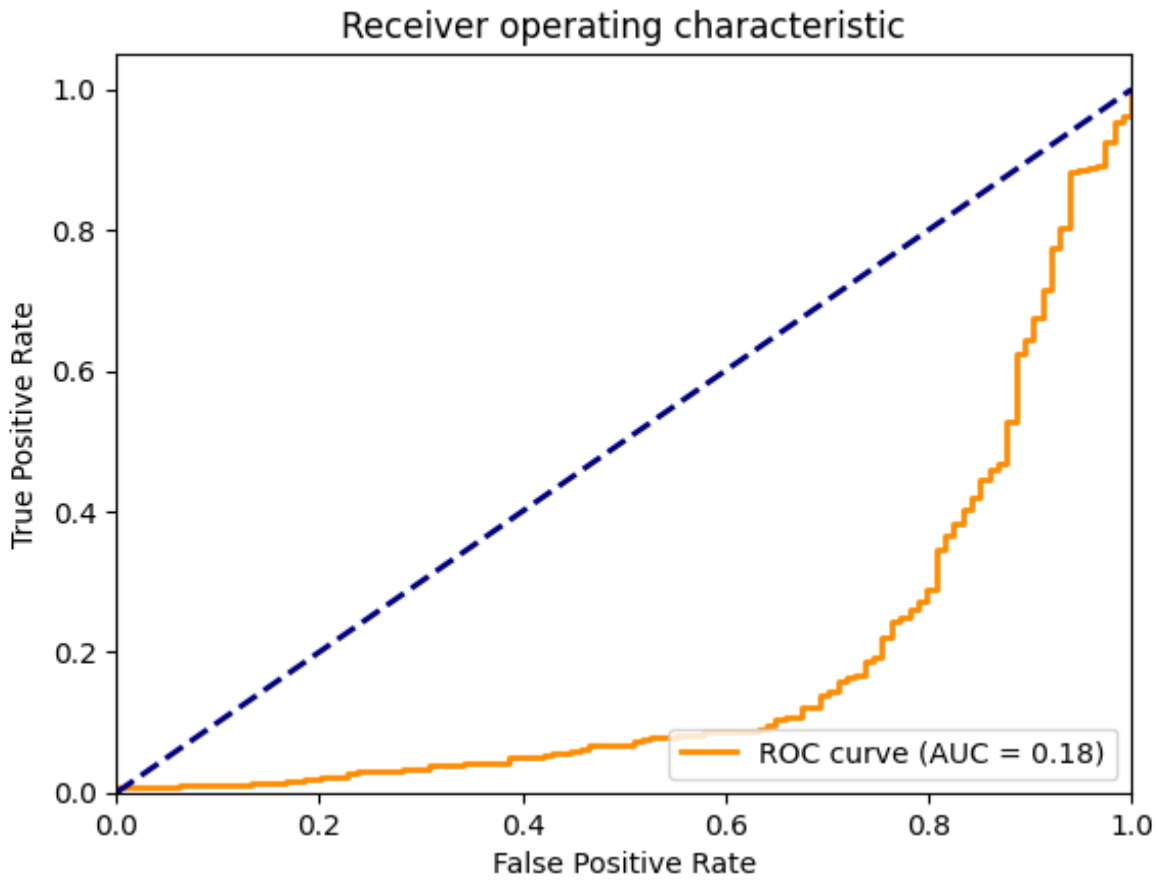
RESULT

Method 1: using Normal data



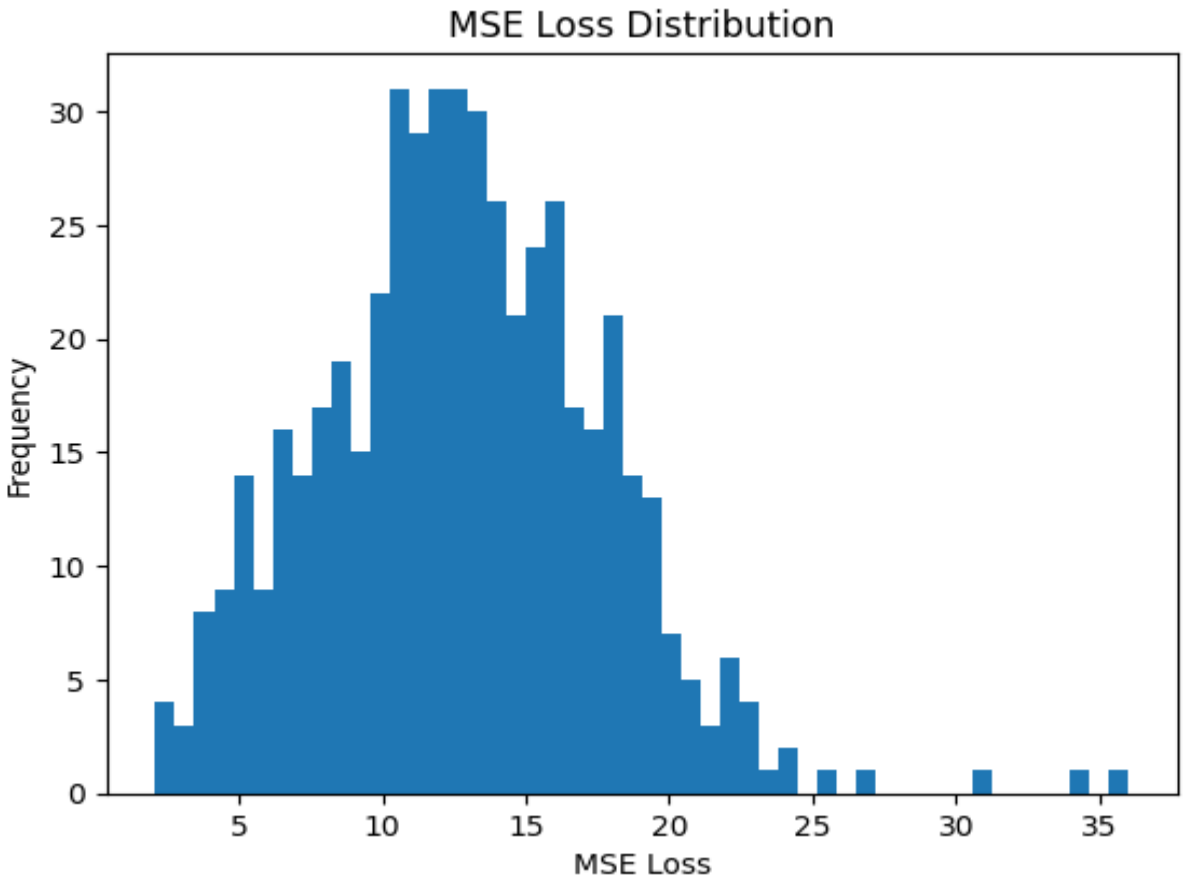
For the given dataset 113 out of 2562 predictions are wrong. There for the error rate is 4.41%

Error Threshold:



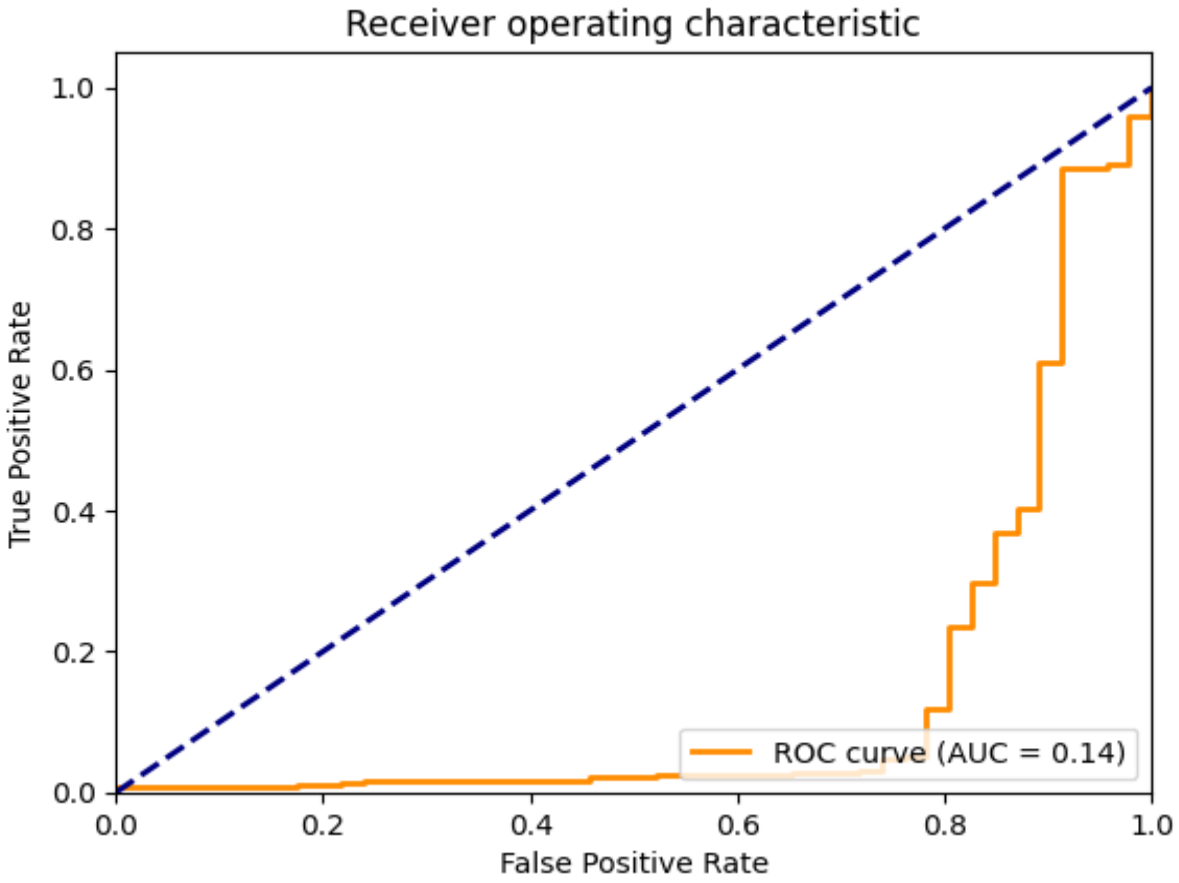
Best error threshold value is 26.74

Method 2: Using Normal and Extreme Data



For the given dataset 24 out of 513 predictions are wrong. There for the error rate is 4.67%

Error threshold:



Best error threshold value is 30.70

REFERENCES

1. Raghavendra,University of Sydney, Capital Markets Cooperative Research Centre (CMCRC), Sanjay Chawla, Qatar Computing Research Institute (QCRI), HBKU 2019, January "DEEP LEARNING FOR ANOMALY DETECTION: A SURVEY"

2. Animesh Patcha, Jung-Min Park "An overview of anomaly detection techniques: Existing solutions and latest technological trends", Volume 51, Issue 12, 22 August 2007, Pages 3448-3470

3. ALI BOU NASSIF, (Member, IEEE), MANAR ABU TALIB, (Senior Member, IEEE), QASSIM NASIR3, AND FATIMA MOHAMAD DAKALBAB "Machine Learning for Anomaly Detection: A Systematic Review", doi:10.1109/ACCESS.2021.3083060

4. Tara Salman, Deval Bhamare, Aiman Erbad, Raj Jain, Mohammed Samaka "Machine Learning for Anomaly Detection and Categorization in Multi-cloud Environments"

5. Salima Omar, Asri Ngadi, Hamid H. Jebur, International Journal of Computer Applications (0975 – 8887), Volume 79 – No.2, October 2013

6. Shikha Agrawal, Jitendra Agrawal,2015 19th International Conference on Knowledge Based and Intelligent Information and Engineering Systems doi:10.1016/j.procs.2015.08.220