POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Thesis

Time series classification models for anomalous transport phenomena

Supervisors
Prof. Lamberto Rondoni
Dr. Sara Bernardi
Dr. Marco Pizzi

Candidate
Giuseppe Pellegrino

Academic Year 2023-2024

To Gloria

Abstract

This thesis investigates time series classification through the application of various algorithms on two distinct datasets, developed in collaboration with Eltek S.p.A. The research aims to assess the performance of these algorithms on both artificially generated and experimentally obtained data. The first case study involves an artificially generated dataset created via simulations to analyze the heat transport of a sensor immersed in a fluid under an applied voltage. The simulations were divided into four groups: one with normal Fourier heat transport and three with different models of anomalous transport, the primary objective was to recognize the four heat transport models using temperature time series. The second case study utilized experimentally obtained data from a capacitor that is discharged through the use of two microelectrodes immersed in a fluid, with a gap of approximately 2 micrometers between them, generating time series of the capacitor's voltage. Each experiment was classified by assigning classes from 0 to 15, where classes 0,4,8,12 stands for normal diffusion while others means anomalous diffusion, the objective was to evaluate the performance of various time series classification models on both real and artificially generated datasets. The study's results highlighted the strengths and limitations of different algorithms in controlled and real-world settings, providing insights into their generalizability and practical industrial applications. This research demonstrates the feasibility of using time series classification algorithms to differentiate between transport models and classify experimental data based on transport parameters. The collaboration with Eltek S.p.A underscores the industrial relevance of these techniques, suggesting future exploration of more sophisticated models and experimental conditions to enhance accuracy and applicability in various domains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) is a subset of artificial intelligence (AI) that empowers systems to learn from data, identify patterns, and make decisions with minimal human intervention. As a rapidly evolving field, machine learning has revolutionized various sectors, including healthcare, finance, transportation, and entertainment. This chapter provides an overview of machine learning, its core principles and methodologies.

The rationale for using machine learning begins with the concept of pattern recognition. Pattern recognition involves the automatic discovery of regularities in data through computer algorithms and using these regularities to take actions such as classifying data into different categories. Consider the example of detecting spam emails. The goal is to develop a system that takes an email as input and determines whether it is spam or not. This task could be approached by crafting specific rules or heuristics to identify spam based on keywords, email structure, and other features. However, in practice, this leads to an overwhelming number of rules and exceptions, resulting in poor performance and constant maintenance as spammers adapt their tactics.

A far more effective approach is to employ machine learning, where a large set of labeled emails (spam and non-spam) is used to train an adaptive model. This training set enables the model to learn the distinguishing features of spam emails through patterns in the data, such as common phrases, sender information, and formatting styles. The model's parameters are adjusted during training to improve its accuracy in identifying spam. This approach not only simplifies the process but also results in significantly better performance as the model continuously improves with more data and evolves with changing spam tactics.

1.1 Machine Learning Process

1.1.1 Problem definition

A machine learning process is divided into several essential phases, each of which must be carefully followed. It begins with **problem definition**, where you clearly identify the problem you need to solve and establish the scope and goals of the study.

1.1.2 Data collection

Next, you proceed to **data collection** phase. Data can originate from various sources such as databases, online repositories, sensors, or user-generated content. The steps you take following this phase are heavily dependent on the data you have gathered. Machine learning is in fact data-driven, this means that the quality and quantity of your data directly influence the performance of the models you select. For this reason, the collection of high-quality data is crucial to the success of your machine learning project.

1.1.3 Data preparation

At this point, you should **prepare your data** to present it in a correct manner to the models you will select. Typically, raw data is incomplete or noisy, making data preparation as crucial as data collection. The philosophy behind data preparation is to uncover and expose the underlying structure of the problem to the learning algorithms as effectively as possible. Data preparation generally involves four phases [15]:

- **Data Cleaning**: Handling missing values, correcting errors, and filtering out noise to ensure the dataset is accurate and reliable.
- **Feature Selection**: Identifying the most relevant features that contribute significantly to the predictive power of the model, removing any redundant or irrelevant features.
- **Data Transformation**: Applying techniques such as normalization, scaling, and encoding to convert data into a suitable format for the algorithms.
- **Dimensionality Reduction**: Reducing the number of features while preserving the essential information, helping to simplify the model and reduce computational cost. Techniques like Principal Component Analysis (PCA) are commonly used for this purpose.

Data cleaning

Data cleaning refers to identifying and correcting errors in the dataset that may negatively impact a predictive model. For example you should remove columns that contain single value, low variance or duplicate data in order to have data that only have some kind of importance in the learning phase of the model [20].

Feature Selection

Feature selection is the process of reducing the number of input variables when developing a predictive model. Reducing the number of input variables is crucial to decrease the computational cost of modeling and, in many cases, improve the performance of the model [15]. Various feature selection methods can be classified into filters, wrappers, embedded, and hybrid methods. **Filter** methods select features based on a performance measure independent of the employed data modeling algorithm. First it is identified the best features and then it is used the modeling algorithms with these selected features. Examples of filter methods include correlation coefficients [92], chi-square tests [90], and fisher scores [28]. **Wrappers** consider feature subsets based on their performance on a specific modeling algorithm, treating the algorithm as a black-box evaluator. For example for classification tasks, a wrapper evaluates subsets based on the classifier's performance, such as Naive Bayes [13] or Support Vector Machines (SVM) [60]. These methods often involve iterative testing and evaluation to identify the optimal subset of features. **Embedded** methods perform feature selection during the execution of the modeling algorithm. These methods are integrated into the algorithm with its normal or extended functionality. Examples include Lasso regression [58], which includes feature selection as part of the training process, and tree-based methods like Random Forests [80], which perform feature importance ranking. **Hybrid** methods combine the advantages of filters and wrappers. Initially, a filter method is employed to reduce the feature space, potentially resulting in several candidate subsets, then, a wrapper method is used to identify the best candidate subset. This two-step approach aims to get the efficiency of filters and the accuracy of wrappers to achieve optimal feature selection. By utilizing these feature selection methods, the process ensures a more efficient and effective modeling approach that lead to improved predictive performance.

Data Transformation

Many machine learning algorithms perform better when numerical input variables are scaled to a standard range [2]. The two most popular techniques for scaling numerical data prior to modeling are normalization and standardization [3]. Normalization scales each input variable separately to the range 0-1, which is the range

for floating-point values where we have the most precision. This technique is particularly useful when the data is bound within a specific range and you want to preserve the relative relationships between the values. Standardization scales each input variable separately by subtracting the mean (called centering) and dividing by the standard deviation. This process shifts the distribution to have a mean of zero and a standard deviation of one. Standardization is useful when the data follows a Gaussian distribution and is often required by algorithms that assume the data is centered around zero with unit variance. By applying these scaling techniques, the performance of machine learning algorithms can be significantly improved, ensuring that the input variables contribute appropriately to the model's learning process.

Dimensionality reduction

Dimensionality represent the number of input variables or features in a dataset. In dimensionality reduction, the objective is to reduce this dimensionality. The problem is that when the number of dimensions increases (i.e., more input variables), the dataset tends to become sparse and less representative of the true data space. Instead of just selecting a subset of features, another approach is to project the data into a lower-dimensional space that preserves the most important properties of the original data. As we saw this process is known as dimensionality reduction and serves as an alternative to feature selection. Applying dimensionality reduction techniques to a dataset offers several advantages: it decreases the number of dimensions and reduces data storage space, requires less computation time, eliminates irrelevant, noisy, and redundant data, optimizes data quality, improve algorithm efficiency, improves accuracy, facilitates data visualization, simplifies classification, and boosts performance [47] [56]. Dimensionality reduction methods such as Principal Component Analysis (PCA) [34], Linear Discriminant Analysis (LDA), and Multidimensional Scaling, transform the original features into a new set of features based on their combinations. The goal is to uncover more meaningful information in this new set. These methods reduce dimensionality by focusing on the features that contribute most to the variance or discriminative power of the data, thus simplifying the dataset while retaining its essential characteristics.

1.1.4 Model selection and evaluation

Now that we have prepared our data, we need to insert them into a model in order to see if the model can learn effectively from the data and predict unseen data. First, we need a model, or better yet, a set of models that could potentially fit our problem. This step is crucial and depends strictly on the nature of the problem and what the literature suggests. Common algorithms include decision trees, support vector machines, neural networks, and ensemble methods. Next, we need a method to evaluate the models in order to compare them and select the best one.

The holdout method is the simplest model evaluation technique and can be summarized as follows: we take a labeled dataset and split it into two parts, a training set and a test set. Then we fit a model to the training data and predict the labels of the test set. The fraction of correct predictions, computed by comparing the predicted labels to the ground truth labels of the test set, gives us an estimate of the model's prediction accuracy. It is important to note that we do not train and evaluate a model on the same training dataset, as this would typically introduce an overly optimistic bias due to overfitting. In other words, we cannot determine whether the model has merely memorized the training data or if it generalizes well to new, unseen data.

Almost every machine learning algorithm comes with some settings that we need to specify, known as hyperparameters. those settings help control the behavior of machine learning algorithms when optimizing for performance. In order to practice hyperparameter tuning with the holdout method, we must modify our initial approach, the "two-way" split into a "three way split", and split the dataset into three parts: a training set, a validation set, and a test set. Reusing the test set multiple times would introduce bias into the final performance estimate and likely result in overly optimistic estimates of generalization performance.

There are different methods for hyperparameter tuning, but the most well-known are k-fold cross-validation and leave-one-out cross-validation. The main idea behind cross-validation is that each sample in our dataset has the opportunity to be tested. In K-fold cross-validation we iterate over a dataset k times. In each round, we split the dataset into k parts: one part is used for validation, and the remaining $k - 1$ parts are merged into a training subset for model evaluation. If we set the number of folds equal to the number of training instances, we refer to this process as Leave-One-Out Cross-Validation (LOOCV). During LOOCV, we fit a model to $n - 1$ samples of the dataset and evaluate it on the single remaining data point. Even if this process is computationally expensive, given that we have n iterations, it can be useful for very small datasets where withholding data from the training set would be too wasteful.

The main difference between the "two-way" holdout method and k-fold cross-validation is that k-fold cross-validation uses all data for training and testing. This approach reduces the pessimistic bias by using more training data.

At this point we must select the best model among them. To approach the model selection phase, we can still use our three-way holdout method. First, we use the training-validation set for hyperparameter tuning. Then, we train the models with the best hyperparameters on the entire training dataset and evaluate them on the test dataset to determine which model performs best. However, the most effective approach when you do not have large size dataset is still k-fold cross-validation [75].The key idea is to keep an independent test dataset that we leave during training and model selection to avoid any leakage of test data into the training stage. Similar to the holdout method, we split the dataset into two

parts: a training set and an independent test set, reserving the test set for the final model evaluation step. We can then experiment with various hyperparameter settings using methods such as Bayesian optimization [81], randomized search [74], grid search [86] and so on. For each hyperparameter configuration, we apply the k-fold cross-validation method on the training set, resulting in multiple models and performance estimates. We then take the hyperparameter settings that produced the best results in the k-fold cross-validation procedure, use the complete training set for train the model with these settings, and perform the final evaluation on the test set that we get earlier. Finally, after completing the evaluation stage, we can optionally fit a model to the entire dataset (both training and test datasets combined). This final model is then ready for deployment.

1.2 Classification Techniques and ML Models

Machine learning can be broadly categorized into four main types: Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

1.2.1 Supervised Learning

This is the most common type of learning where the model is trained on a **labeled** dataset, which means that each training example is paired with an output label. The algorithm learns to map inputs into the desired output. Common applications include classification (e.g., spam detection in emails) and regression (e.g., predicting house prices). Supervised learning is the most common technique in classification problems, since the goal is often to get the model to learn a classification system that we've created. Most commonly, supervised learning leaves the probability for input undefined, such as an input where the expected output is known. This process provides a dataset consisting of features and labels. The main task is to construct an estimator able to predict the label of an object given by the set of features. Then, the learning algorithm receives a set of features as inputs along with the correct outputs and it learns by comparing its actual output with corrected outputs to find errors. In this process, the supervised learning algorithm builds the predictive model. After its training, the fitted model would try to predict the most likely labels for a new set of samples X in the testing set. Depending on the nature of the target y, supervised learning can be classified in: If y has values in a fixed set of categorical outcomes (integers), the task to predict y is called classification. If y has floating point values, the task to predict y is called regression.

1.2.2 Unsupervised Learning

Unsupervised learning uses pattern recognition without the involvement of a target attribute, so, all the variables used in the analysis are used as inputs and because of the approach, the techniques are suitable for clustering and association mining techniques . The goal is to find hidden patterns or intrinsic structures in the input data. Unsupervised learning algorithms are suitable for creating the labels in the data that will be used to implement supervised learning tasks. That is, unsupervised clustering algorithms identify inherent groupings within the unlabeled data and then assign a label to each data value. On the other hand, unsupervised association mining algorithms tend to identify rules that accurately represent relationships between attributes.

1.2.3 Semi-Supervised Learning

Semi-supervised learning (SSL) is a type of Machine Learning (ML) technique where it is half-way between supervised and unsupervised learning, i.e., the dataset is partially labeled. The main objective of SSL is to overcome the drawbacks of both supervised and unsupervised learning. Supervised learning requires a huge amount of training data to classify the test data, which is a cost-effective and time-consuming process. On the other hand, unsupervised learning doesn't require any labeled data and clusters the data based on similarity in the data points by using either clustering or maximum likelihood approach. The main downfall of this approach is that it can't cluster unknown data accurately. To overcome these issues, SSL can learn with a small amount of training data to label the unknown test data. SSL builds a model with few labeled patterns as training data and treats the rest of the patterns as test data.

1.2.4 Reinforcement Learning

This type involves an agent that interacts with its environment by performing actions and receiving rewards or penalties. The main goal of RL is learning through interaction, indeed the agent tries to learns how optimize its behavior in order to maximize the cumulative reward. Applications include game playing (e.g., AlphaGo) and robotics. An RL agent interacts with its environment and, by observing the consequences of its actions, can learn to alter its own behavior in response to rewards received. This paradigm of trial-and-error learning has its roots in psychology and is one of the main foundations of RL. The other key influence on RL is optimal control, which has lent the mathematical formalisms (most notably dynamic programming) that sustain the field.

1.3 Supervised Learning Algorithms

Now we will focus on supervised learning algorithms, because it is the most common approach in machine learning and is actually used in this thesis work.

1.3.1 Naive Bayes

Naive Bayes [11] is a probabilistic classification algorithm based on Bayes' theorem. It is called "naive" because it assumes that the features are independent of each other, which is a strong and often unrealistic assumption. It assumes an underlying probabilistic model and allows capturing uncertainty about the model in a principled way by determining probabilities of the outcomes. The basic purpose of Bayesian classification is to solve predictive problems. This classification provides practical learning algorithms and can combine observed data. Bayesian classification provides a useful perspective for understanding and evaluating learning algorithms. It calculates explicit probabilities for hypotheses and is robust to noise in input data.

1.3.2 Decision Tree

Decision Tree [59] is a non-parametric algorithm used for both classification and regression. It splits the data into subsets based on the feature that results in the most significant information gain or least impurity. A decision tree is a tree-like model used for decision-making and prediction, consisting of nodes that form a hierarchical structure starting from a root node. The root node is the top node with no incoming edges, serving as the starting point of the tree. All other nodes have exactly one incoming edge. Nodes with outgoing edges are referred to as internal nodes or test nodes, while nodes without outgoing edges are called leaves or terminal nodes. In a decision tree, each test node splits the instance space into two or more sub-spaces according to a specific discrete function of the input values. In the simplest case, each test considers a single attribute, partitioning the instance space based on the attribute's value. For categorical attributes, this means splitting the data based on distinct values, while for numeric attributes, the condition involves splitting based on a range of values. Each leaf in the decision tree is assigned to a class that represents the most appropriate target value for the given input conditions. For example, the root node might test the first attribute, and depending on the result, the data is passed down to one of its child nodes. These child nodes, now internal nodes, perform further tests on subsequent attributes. This process continues, with each internal node splitting the data further until the data reaches a leaf node. The leaf node then assigns a class label or target value based on the majority class or most appropriate outcome for the instances that have followed that particular path through the tree. The overall goal of a

decision tree is to create a model that accurately classifies or predicts the target variable by systematically splitting the data into more homogeneous sub-groups at each internal node. This process of splitting and assigning values continues until the stopping criteria are met, which could be a maximum depth of the tree, a minimum number of instances in a node, or another criterion that helps prevent overfitting and ensures generalization to new, unseen data.

1.3.3 Linear Regression

The goal of linear regression [11], as part of the family of regression algorithms, is to identify relationships and dependencies between variables. It models the relationship between a continuous scalar dependent variable y (also known as the label or target in machine learning terminology) and one or more explanatory variables (also referred to as independent variables, input variables, features, observed data, observations, attributes, dimensions, data points, etc.), denoted as X, using a linear function. In regression analysis, the objective is to predict a continuous target variable, in contrast to classification, which aims to predict a label from a finite set. In the case of multiple regression, where the model involves a linear combination of multiple input variables, the relationship is represented as:

$$y = b_0 + b_1 x_1 + \ldots + e.$$

1.3.4 Logistic Regression

Logistic Regression [46] is a classification algorithm used for binary classification problems. Despite its name, it is a linear model for classification rather than regression. Similar to Naive Bayes, logistic regression works by extracting a set of weighted features from the input, taking logarithms, and combining them linearly. This means that each feature is multiplied by a weight and then summed. However, the most significant difference between Naive Bayes and logistic regression is that logistic regression is a discriminative classifier, while Naive Bayes is a generative classifier. Logistic regression is a type of regression that predicts the probability of occurrence of an event by fitting data to a logistic function. Like many forms of regression analysis, logistic regression uses several predictor variables that may be numerical or categorical. The hypothesis in logistic regression is represented by the logistic function (or sigmoid function):

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Here, $h_\theta(x)$ represents the predicted probability that the output is 1 (the event of interest) given the input x. The vector θ contains the weights for the features. The

cost function for logistic regression, often referred to as the logistic loss or log loss, is used to measure the performance of the model. It is given by:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

In this equation: m is the number of training examples, $y^{(i)}$ is the actual label for the i-th training example, and $h_\theta(x^{(i)})$ is the predicted probability for the i-th training example. The goal is to find the values of θ that minimize this cost function, typically using optimization techniques such as gradient descent.

1.3.5 Deep Learning

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input. Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations.

Fully Connected Feed Forward Neural Network (FFNN)

The fundamental unit of neural networks is a *neuron*, which acts as a processing node. In a neural network, neurons are interconnected through synaptic weights, or simply weights. Each neuron receives information weighted by these synaptic connections from the neurons it is linked to and generates an output. This output is produced by passing the weighted sum of input signals (whether from external sources or other neurons) through an *activation function*.

Neural network architectures can be broadly classified into two categories based on the connections between neurons: feed-forward neural networks and recurrent neural networks. If the network does not have feedback loops from the outputs of neurons back to the inputs, it is termed a feed-forward neural network. Conversely, if there are such feedback loops, meaning that outputs loop back as inputs (either to

the same neurons or to others), the network is known as a recurrent neural network. Typically, neural networks are structured in layers. Feed-forward neural networks are further divided based on the number of layers: "single-layer" and "multi-layer" networks.

A single-layer feed-forward neural network is illustrated in 1.1. This structure has two layers, but the input layer is not counted as it does not perform any computation. The input signals are transmitted to the output layer through the weights, and the neurons in the output layer process these signals to produce the final output. In Figure 1.2, a multi-layer feed-forward neural network with one hidden layer is depicted. Unlike the single-layer network, this setup includes at least one layer of hidden neurons between the input and output layers. Hidden neurons play a crucial role by mediating between external inputs and network outputs in a beneficial way [36]. Having one or more hidden layers allows the network to capture higher-order statistics. In the example shown in Figure 1.2, there is only one hidden layer, making it a 3-3-2 network, as it contains 3 input neurons, 3 hidden neurons, and 2 output neurons. Both networks are "fully connected," meaning each neuron in a layer is connected to every neuron in the subsequent layer.

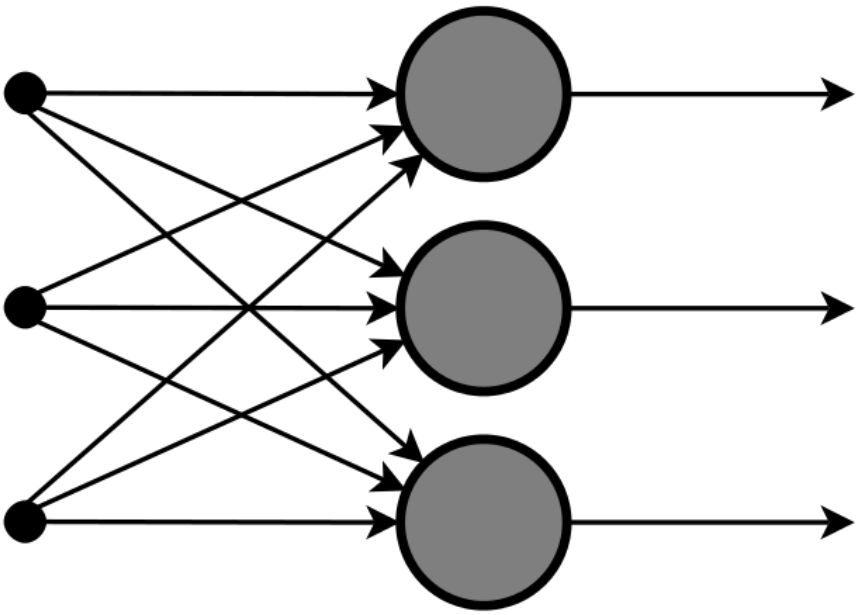


Figure 1.1: Single layer fully connected neural network

Among many other learning algorithms, backpropagation is the most popular and the mostly used one for the training of feed-forward neural networks. The training of a feed-forward neural network using backpropagation involves several key steps: forward propagation, loss calculation, backpropagation, and weight updates. This process is iteratively performed over multiple epochs to minimize the loss and improve the model's performance.

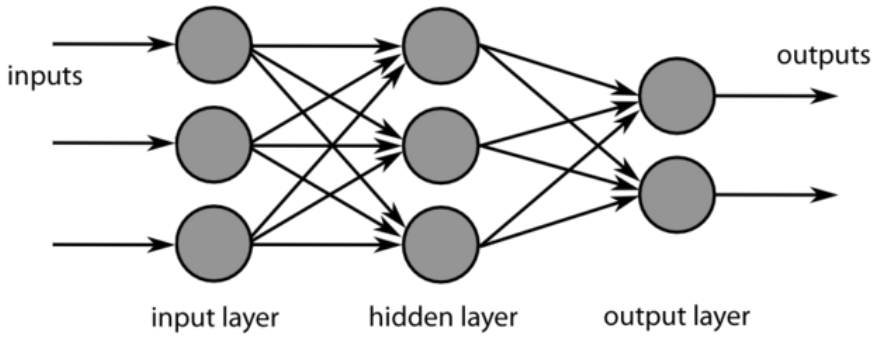


Figure 1.2: Two layer fully connected neural network by Chrislb, licensed under CC BY-SA 3.0. No changes were made.

In forward propagation, the input data is passed through the network, layer by layer, to generate an output. The input data are fed into the input layer. The data are then passed through each hidden layer. For each neuron in a hidden layer, the following steps occur: compute the weighted sum of inputs, $z = \sum(w_i \cdot x_i) + b$, and apply an activation function, $a = \phi(z)$, to introduce non-linearity. Here, w_i are the weights, x_i are the input values, b is the bias term, and ϕ is the activation function. Finally, the processed data reaches the output layer, where it generates the output predictions using a similar process (weighted sum and activation function).

The network's output is compared with the true target values to calculate the loss (error). A common loss function for classification problems is the cross-entropy loss, while mean squared error (MSE) is often used for regression problems. For instance, if using cross-entropy loss,

$$L = -\frac{1}{m}\sum_{m}[y_i \log(y^i) + (1 - y_i)\log(1 - y^i)],$$

where y^i is the predicted output, y_i is the true label, and m is the number of samples.

Backpropagation is the process of calculating the gradient of the loss function with respect to each weight by the chain rule, propagating this error backward through the network. Starting from the output layer, compute the gradient of the loss function with respect to the output of the neurons.

The weight updates are performed using an optimization algorithm such as stochastic gradient descent (SGD) or Adam. The weights are adjusted in the opposite direction of the gradient to minimize the loss function.

This entire process (forward propagation, loss calculation, backpropagation, and weight updates) is repeated for multiple epochs, where an epoch is a single pass through the entire training dataset. The goal is to minimize the loss function, thereby improving the model's predictions. The backpropagation algorithm is central to training feed-forward neural networks, enabling the model to learn from the

data by iteratively adjusting weights and biases to minimize the error. Through this process, the network improves its ability to make accurate predictions on new, unseen data.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs), also known as ConvNets, are specifically designed to process data that come in the form of multiple arrays. For instance, a color image is composed of three 2D arrays, each containing pixel intensities for one of the three color channels. Various data modalities fit into this framework: 1D arrays for signals and sequences, such as language; 2D arrays for images or audio spectrograms; and 3D arrays for video or volumetric images.

There are four key concepts that enable ConvNets to effectively leverage the properties of natural signals: local connections, shared weights, pooling, and the use of multiple layers. The architecture of a typical ConvNet is structured as a series of stages, with the initial stages consisting of convolutional layers and pooling layers.

In a convolutional layer, units are organized into feature maps. Each unit within a feature map is connected to local patches in the feature maps of the previous layer through a set of weights known as a filter bank. The result of this local weighted sum is then passed through a non-linear function, called activation function, such as ReLU (Rectified Linear Unit) and many others [27]. All units in a feature map share the same filter bank, while different feature maps within the same layer use different filter banks.

This architecture is designed for two main reasons. First, in array data like images, local groups of values are often highly correlated, forming distinctive local patterns that can be easily detected. Second, the local statistics of images and other signals are invariant to location. This means that a pattern appearing in one part of an image could potentially appear anywhere in the image. Therefore, having units at different locations share the same weights allows the network to detect the same pattern in various parts of the array.

Mathematically, the filtering operation performed by a feature map is a discrete convolution, which is the origin of the term "convolutional" in Convolutional Neural Networks.

Recurrent Neural Network (RNN) and ResNet

For tasks involving sequential inputs, such as speech and language, RNNs often outperform other types of neural networks. RNNs process an input sequence one element at a time, maintaining a 'state vector' in their hidden units that implicitly contains information about the history of all past elements in the sequence. This mechanism allows RNNs to effectively handle temporal dependencies in sequential

data.

To understand how backpropagation is applied to RNNs, consider the outputs of the hidden units at different discrete time steps. These outputs can be viewed as analogous to the outputs of different neurons in a deep multilayer network. This perspective makes it clear how the backpropagation algorithm can be extended to train RNNs by propagating errors backwards through time. Despite their theoretical power, training RNNs has historically been challenging. This difficulty arises because the backpropagated gradients tend to either grow or shrink at each time step, leading to the problems of exploding or vanishing gradients [39, 7] over many time steps. These issues hinder the effective training of RNNs, particularly for long sequences. However, advancements in RNN architectures [37] and training methods [88, 72] have significantly improved their performance. Techniques such as Long Short-Term Memory (LSTM) [40] and Gated Recurrent Units (GRUs) have been developed to address the vanishing and exploding gradient problems. As a result, RNNs have become highly effective for tasks such as predicting the next character in a text sequence or the next word in a sentence. They are also applied in more complex tasks, including machine translation and time series forecasting. In parallel with advances in RNNs, another significant development in neural network architecture is the Residual Network (ResNet). ResNet was introduced to address the difficulties in training very deep neural networks, particularly the degradation problem where adding more layers leads to higher training error. ResNet introduces the concept of residual learning by using skip connections, which allow the network to learn residual functions with reference to the layer inputs, instead of learning unreferenced functions.

A typical ResNet block includes a skip connection that bypasses one or more layers by connecting the input of a layer directly to the output of a subsequent layer. This kind of architecture is able to train much deeper networks effectively because it helps in maintaining the flow of gradients through the network. The success of ResNet has led to its adoption in numerous applications, including image classification, object detection, and segmentation.

Chapter 2

Time Series Classification Algorithms

Time series (TS) are a type of data that represent the evolution of a variable over time, where observations are recorded in chronological order. This kind of data appears in many fields such as economics, social sciences, environmental sciences and finance, to name a few.

Time series have a temporal dependency structure, where successive observations can be influenced by previous ones, indeed compared to other types of data, TS are unique and requires the use of specific techniques and models for their analysis and forecasting.

time series can display various types of behavior: trends, seasonality, cyclicity, and randomness. The trend represents the general direction of the series' movement over time, while seasonality refers to regular variations that occur at specific intervals, such as seasons of the year. Cyclicity indicates fluctuations that do not follow a regular pattern but occur at irregular intervals. Lastly, randomness refers to random behavior in the series.

Analyzing time series could involve several activities, including visualizing the data to identify temporal patterns, identifying statistical models that capture the series' characteristics, and forecasting future observations. Techniques such as series decomposition, autocorrelation, ARIMA (AutoRegressive Integrated Moving Average) models, and neural network-based models are widely used to analyze and model time series to make accurate predictions.

2.1 Time series definition

Time series classification (TSC) involves modeling a discrete response variable based on a continuous, ordered sequence of real-valued observations (a time series). These time series can be either univariate (with a single variable observed at each time point) or multivariate (involving multiple variables at each time point). For instance, daily stock prices can be treated as univariate time series for tasks like predicting market trends, while weather data with temperature, humidity, and wind speed could form a three-dimensional multivariate time series for forecasting weather conditions.

TSC problems span diverse domains, including electroencephalograms, electrocardiograms, motion data (such as HAR), image outlines, spectrograms, light curves, audio, traffic levels, electricity usage, and more. The wide range of problem domains characterizes TSC research.

2.1.1 Time Series (TS)

A time series $T = (t_1, t_2, \ldots, t_n)$ is an ordered sequence of n data points. We represent the j-th value of T by t_j. If each point $t_j \in T$ represents a single value ($t_j \in \mathbb{R}$), the series is a univariate time series (UTS). If each point represents the observation of multiple variables at the same time point then each point is a vector $t_j \in \mathbb{R}^d$ of length d, and we can refer to it as a multivariate time series (MTS).

2.1.2 Multivariate Time Series (MTS)

A multivariate time series $T = (t_1, \ldots, t_n) \in \mathbb{R}^{d \times n}$ is a sequence of n vectors, where each t_j is a vector with d components (sometimes called dimensions). We denote the j-th observation of the k-th component by the scalar $t_{k,j} \in \mathbb{R}$. Note that an MTS can also be viewed as a collection of d separate time series, as this is often how they are processed in practice. However, the vector representation emphasizes that we assume the dimensions are synchronized, i.e., all observations in t_j occur at the same time point or spatial location.

2.1.3 Dataset

A dataset $\mathcal{D} = (X, Y) = \{(T^{(i)}, c^{(i)})\}_{i=1}^{m}$ consists of m time series, where each $T^{(i)}$ is paired with a label $c^{(i)}$ from a predefined set of discrete class labels $\mathcal{C}$. For example, $T^{(i)}$ could represent the daily sales data for a store, and $c^{(i)}$ could be a label indicating whether the sales are above or below a certain threshold.

2.2 Time Series Classification Models

[68] defined the taxonomy of time series classification algorithms as following: Distance-based, feature-based, interval-based, Shapelet-based, dictionary-based, convolution-based, deep learning based and hybrid approaches.

2.2.1 Distance-Based algorithms

Distance-based methods play a crucial role in time series classification defining a dissimilarity measure between time series, which is then incorporated into classification algorithms such as k-nearest neighbor (k-NN) or Support Vector Machines (SVMs). However, several challenges arise due to the unique characteristics of time series data:

- Temporal Nature: Time series data are inherently ordered, requiring careful consideration of their temporal relationships.
- High Dimensionality: Time series often have many data points, making distance computation complex.
- Noise: Real-world time series data can be noisy, affecting the choice of distance metric.
- Varying Lengths: Different time series may have varying lengths, posing challenges for direct comparison.

Two main types of time series distance measures exist:

1. **Lock-Step Measures:** These compare corresponding points between two time series (e.g., Euclidean distance).
2. **Elastic Measures:** Elastic measures create non-linear mappings to align series, allowing one-to-many comparisons (e.g., Dynamic Time Warping).

Additionally, there are structure-based and edit-based distance measures. It was explored three primary approaches for time series classification:

1. **Direct Distance Use:** Distances are directly employed with k-NN classifiers.
2. **Feature Transformation:** Distances are used to transform time series into feature vectors.
3. **Kernel Methods:** Distances contribute to kernel functions for time series classification.

Choose an appropriate distance measure is crucial for accurate classification.

K-Nearest Neighbor (k-NN)

The k-NN approach utilizes existing time series distances within classifiers. Specifically, the 1-NN classifier is commonly used in time series classification, thanks to its performance and simplicity [26, 54]. Given a distance measure and a time series, the 1-NN classifier predicts the class by identifying the closest object from the training set. However, it's important to note that this method is sensitive to noise in the training data, a common characteristic of time series datasets.

Distance Features

In this group of features, methods employ time series distance measures in order to create feature vectors. these approaches handle the challenges presented in time series classification (e.g., handling ordered sequences and varying instance lengths) by transforming series into order-free vectors in R^n. The goal is to fill the gap between time series and conventional classification by using classifiers based on vector while still using the potential of time series distances. Calculating distance features is a preprocessing step, compatible with any classifier.

Distance Kernel

Methods in this category don't directly transform time series using existing distances. Instead, they construct kernels specifically for time series. These kernels capture underlying structures and enable powerful learning algorithms to operate directly on pairwise relationships between time series instances.

2.2.2 Feature-Based algorithms

When dealing with time series data, one common approach is instance-based classification, like distance-based algorithm, where the focus is on comparing time series directly by measuring the similarity or distance between their raw data points. For instance, if you have short time series with meaningful patterns, you can compare them to known instances by matching them based on similarity. This approach involves finding the nearest neighbors for a given time series. However, it can be computationally expensive, especially for large datasets or long time series. Examples of instance-based classifiers as we saw before include k-nearest neighbors (k-NN) and dynamic time warping (DTW) algorithms. These methods directly operate on the raw time series data without feature extraction. An alternative approach is feature-based classification. Instead of comparing raw data points, a time series can be represented using a set of derived properties (features). These features summarize the entire time series, allowing us to characterize them as series-to-vector transforms, as shown in the Figure 2.1 . Common features include statistical moments (such as mean and variance) [70], frequency domain features (e.g., Fourier

coefficients) [69], and autocorrelation properties [31]. By extracting relevant features, it simplify the classification process and reduce the dimensionality of the data. The typical pipeline involves feature extraction followed by a classifier (e.g., decision trees, neural networks). Examples of feature-based classifiers include Random Forests and Support Vector Machines (SVMs). In summary, feature-based classification uses informative features extracted from time series, making it computationally efficient and providing insights into the dataset.

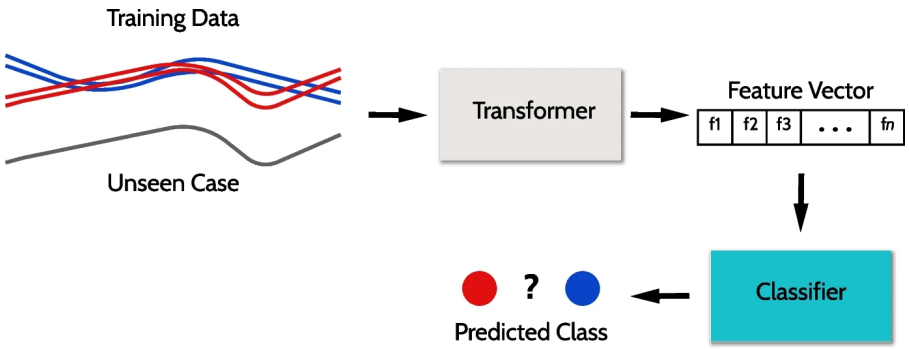


Figure 2.1: Figure of the process of feature extraction of a time series followed by a classifier, image taken from [68] with permission

2.2.3 Interval-Based algorithms

What we actually saw is two categories of algorithms: instance-based and feature based. Instance-based classifiers predict the class of a test instance by comparing it directly to training instances, relying on measures of similarity or distance between time series. Feature-based classifiers, on the other hand, build models using derived features from the time series. An alternative approach within feature-based methods is the use of interval-based features. Interval features are calculated from specific segments or intervals of the time series, such as "the interval between time 10 and time 30". Various types of features can be derived from these intervals, but simple and interpretable features like the mean and standard deviation are often preferred. For example, one might consider "the average value of the time series segment between time 10 and time 30".

Interval-based classifiers, as introduced by [23], extract intervals of fixed lengths and offsets from the time series and compute summary statistics on these intervals. Many approaches to interval-based classification use some form of random selection to choose intervals, essentially employing a randomized feature selection method. These randomly selected interval locations are consistent across all series in the dataset, and many interval-based classifiers combine features from multiple intervals to improve robustness and accuracy. The motivation for using intervals is to

reduce the impact of noise and capture more localized, phase-dependent properties of the time series that might be informative for classification. Interval-based classifiers often adopt a random forest ensemble model, where each base classifier consists of a pipeline of transformation followed by a tree classifier. This approach, injects diversity into the model by randomizing the intervals for each tree in the ensemble. By focusing on interval-based features, classifiers can mitigate the effects of noise and capture important local temporal patterns that are missed by global feature methods, due to this we can have potentially more accurate and interpretable models.

2.2.4 Shaplet-Based algorithms

Shapelet-based algorithms offer a powerful approach to time series classification, particularly in cases where traditional instance-based methods struggle. According to [91], the poor performance of instance-based classifiers is often attributed to the presence of noise in the data, which can obscure the subtle differences in the shapes of time series. Shapelet-based methods address this issue by focusing on smaller, more discriminative subsections of the time series, known as shapelets.

Shapelets are essentially small "sub-shapes" extracted from the training data that are highly indicative of class membership. shapelet-based classifiers, by comparing these local features rather than the entire time series, can provide more robust and accurate results than other classic approaches, especially in noisy datasets, where global features used by other state-of-the-art classifiers might be less effective. One of the key benefits of shapelets is their interpretability, indeed they are derived from actual subsections of the time series, so they can be directly related to specific patterns or events within the data. This can help domain experts understand which features are most important for classification and provide insights into the underlying processes generating the data. In practice, shapelets are subseries from the training data that are independent of the phase of the time series. This means that they can be used to discriminate between classes based on their presence or absence, regardless of where they appear in the series. To evaluate a shapelet, the algorithm slides the subseries across the time series and calculates the z-normalized Euclidean distance between the shapelet and the underlying window. The distance between a shapelet and any time series, denoted as sDist(), is the minimum distance over all such windows. Figure 2.2 provides a visualization of the sDist() process. In this figure, a shapelet S is shifted along a time series A. At each position, the distance between the shapelet and the corresponding window in the time series is calculated. The minimum distance and its corresponding offset are recorded as the sDist() for that shapelet and time series pair. This process ensures that the shapelet is matched to the most similar segment of the time series, providing a robust measure of similarity.

The quality of a shapelet is evaluated based on its ability to discriminate between

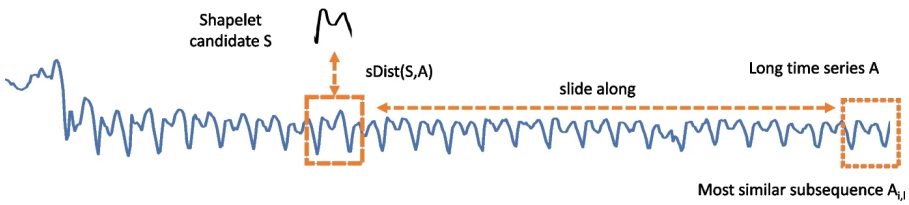


Figure 2.2: Visualization of shapelet distance operation sDist(), image taken from [68] with permission

different classes. Once the distances between a shapelet and all training series are computed, these distances can be used as features in a classifier.

2.2.5 Dictionary-Based algorithms

Dictionary-based methods offer another effective approach to time series classification by focusing on phase-independent subseries. These methods differ from shapelet-based algorithms in that, instead of measuring the distance to a specific subseries, they convert each window of the time series into a sequence of discrete symbols, commonly referred to as words. The classification is then based on the frequency of these words, which is why these methods are often described as bag-of-words approaches.

The fundamental structure of dictionary-based algorithms involves several key steps. Initially, a window of a fixed length w is moved across each time series. Each subseries captured by this window is then transformed into a string or pattern that represents it. To achieve this, the subseries is first compressed from length w to a shorter length l. This compression reduces the dimensionality of the subseries, making the following steps more manageable.

Once compressed, the subseries is discretized, meaning that each of the l data points is converted into one of a predefined set of α discrete values. This process, that is entirely showed in figure 2.3,effectively converts the continuous time series data into a sequence of symbols or words. The occurrence of each resulting word r is then recorded in a histogram, also known as a bag. During a stage called numerosity reduction, contiguous series of identical words are counted as a single occurrence to avoid over-representation of repetitive patterns.

Each time series is thus represented by a separate histogram that captures the frequency of different words. To classify a new instance, the algorithm compares the histogram of the new time series to the histograms of the training set. This comparison is typically done using a distance measure, with the 1-nearest neighbor classifier being a common choice. However, other classification methods can also be employed depending on the specific requirements of the problem.

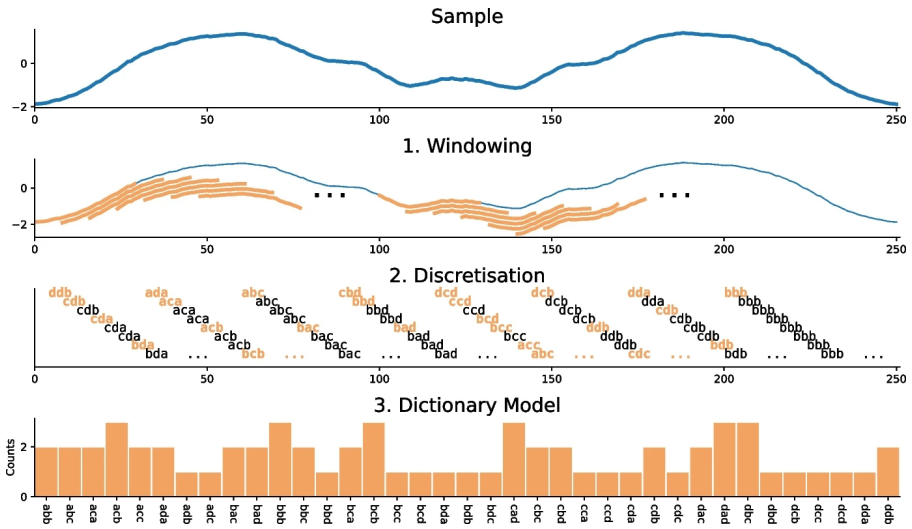


Figure 2.3: Visualization of the process of transformation of a TS into the dictionary model, image taken from [68] with permission

2.2.6 Convolution-Based algorithms

Convolutional neural networks (CNNs), that has some convolutional layers, have shown good results in time series classification. These networks contain a large number of trainable parameters that are optimized through stochastic gradient descent or its variants. Compared to classic algorithms like logistic regression or support vector machines, CNNs typically require a larger sample size to effectively train these parameters due to their complexity.

for this reason, some algorithms [21, 22] have been developed to extract features from time series using numerous random convolutional kernels. Here, all the parameters of the kernels (including length, weights, bias, dilation, and padding) are generated randomly from predefined distributions. Each kernel is then applied to a time series through a sliding dot product operation, producing a series-to-series transformation known as an activation map. Instead of extracting a single feature from each kernel (such as the maximum or mean, which is common in traditional CNNs), these algorithms extract two features: the maximum value and the proportion of positive values within the activation map.

Convolution-based algorithms typically follow a standard approach as shown in Figure 2.4, the process starts with the formation of activation maps for each convolution. These maps then undergo pooling operations to extract relevant features, specifically the maximum and the proportion of positive values. The resulting features are concatenated into a single feature vector, which is then used for classification. A ridge regression classifier is usually used for this purpose.

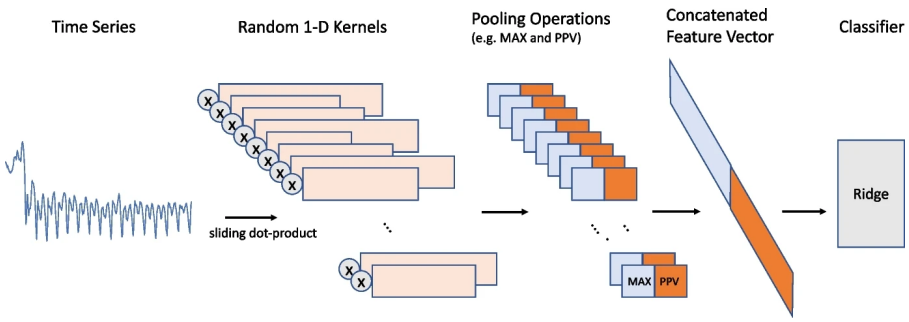


Figure 2.4: Visualization of a tipical convolution based approach, image taken from [68] with permission

2.2.7 Deep Learning-Based algorithms

Deep learning has significantly advanced time series classification by offering methods to automatically learn complex patterns from data. Among various deep learning techniques, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), including advanced forms like Long Short-Term Memory (LSTM) are particularly prominent.

CNNs, initially designed for image processing, have been adapted effectively for time series classification, using convolutional layers to extract local patterns and features from time series data. These layers apply filters to the input time series, performing convolutions to produce feature maps that capture specific patterns such as trends or seasonal variations. Pooling operations, like max pooling or average pooling, reduce the dimensionality of these feature maps, retaining the most important features while reducing computational complexity. After several convolutional and pooling layers, fully connected layers combine the extracted features to make final classification decisions.

RNNs are specifically designed to handle sequential data, making them well-suited for time series classification. They maintain a hidden state that captures information from previous time steps, allowing them to model temporal dependencies. However, vanilla RNNs can suffer from problems like vanishing and exploding gradients when dealing with long sequences. LSTMs address these limitations by incorporating memory cells and gating mechanisms (input, forget, and output gates) to control the flow of information, making them capable of learning long-term dependencies.

Combining CNNs and RNNs we can have the strengths of both architectures. A common approach is to use CNN layers to extract spatial features from the time series data, followed by RNN layers to capture temporal dependencies. This hybrid model can be particularly powerful for complex time series classification tasks.

2.2.8 Hybrid methods

The nature of the data and the specific problem often dictate which category of algorithm is most appropriate for time series classification. Without prior knowledge of the best approach, the most accurate algorithms on average tend to combine multiple transformation types into a hybrid model. This strategy uses the strengths of different methods to create a more robust predictive model. A common approach to enhance the performance of a final model is to average the predictions of several independently trained models. Traditional ensemble methods, such as random forests, consist of base classifiers that all belong to the same type of algorithm, such as decision trees, however, this approach limits the model's advantages. In contrast, using multiple types of algorithms within an ensemble allows for learning a more diverse representation of the data. This diversity can capture various aspects of the time series, which might be missed by a single type of algorithm. For time series classification, ensemble models that integrate different types of algorithms, such as dictionary approaches, shapelet-based algorithms, and convolutional neural networks, have been developed [6, 5, 67]. These hybrid models often achieve state-of-the-art performance in terms of predictive accuracy at the cost of high computational complexity.

2.3 Conclusion

In this chapter, we have explored and presented a comprehensive overview of various time series classification types of models. The models covered include traditional machine learning algorithms, as well as more advanced methods and hybrid models that combine multiple approaches. We conducted extensive evaluations on these models using two distinct datasets based on anomalous transport phenomena, which present unique challenges due to their complex and often unpredictable nature. The first dataset was synthetically generated, the second dataset was experimental, focusing on real-world data. We evaluated seven models on these datasets, aiming to use the best model for each type of algorithm, so, as described in [68], we employed the following models: HIVE-COTE 2.0 (HC2) [67], Hydra+MultiROCKET (multi-Hydra) [22], RDST [33], drCIF [67], Weasel 2.0 (weasel-d) [85], freshPRINCE [64], and inceptionTime [42].

Chapter 3

Experiments

Anomalous transport refers to nonequilibrium processes that cannot be described using standard methods of statistical physics. This novel class of transport phenomena has recently been observed in a wide variety of complex systems. A transport process $x(t)$ is considered anomalous if its mean square displacement does not increase linearly over time [50, 35, 94]. Anomalous transport is observed in a wide range of phenomena, including the movement of molecules within living cells [82], dynamics on cell membranes [71], disordered solid-state systems [12], telomeres within the nuclei of mammalian cells [14], soil transport [61], and heat transfer in low-dimensional systems [52]. Extensive research has been conducted to investigate its microscopic origins [50, 43, 44, 49, 41]. The statistical properties of transport are described by the probability density of the displacement $P(\Delta x, t)$. However, this density is often unknown, so transport is typically studied in terms of the asymptotic behavior of its moments over time:

$$\langle |\Delta x(t)|^p \rangle \sim t^{\gamma(p)}$$

where $\langle \cdot \rangle$ mean the average over a group of trajectories, $p \in \mathbb{R}$ is the order of the moment, and $\gamma(p)$ is referred to as the spectrum of the moments of the displacement.

The exponent $\eta = \gamma(2)$ denotes the mean-square displacement. We can have four different kind of transport: **subdiffusive** when $0 < \eta < 1$, **diffusive** when $\eta = 1$, **superdiffusive** when $1 < \eta < 2$, and **ballistic** when $\eta = 2$.

Transport is termed scale-invariant when the probability density of displacements Δx follows the scaling $P(\Delta x, t) = t^{-\nu} F(\Delta x / t^{\nu})$ with constant ν, meaning all moments of the displacement are characterized by a single scale t^{ν}. The spectrum of the moments of the displacement will then be a linear function of p: $\gamma(p) = \nu p$.

When the spectrum $\gamma(p)$ is nonlinear, the transport is called strong anomalous diffusion [17]. This phenomenon has been observed in various simple stochastic systems [4], polygonal billiard channels [79], one-dimensional maps [73], running sand piles [16], stochastic models of inhomogeneous media [8], diffusion in laser-cooled atoms [1], experiments on the mobility of particles inside living cancer cells

[32], particles passively advected by dynamic membranes [87], and bulk-mediated diffusion on lipid bilayers [51]. Another studied case involves different scalings of the bulk and the tail of the probability distribution, resulting in a piecewise linear form of the scaling exponent $\gamma(p)$:

$$\gamma(p) = \begin{cases} \nu p & \text{for } p < p_c, \\ p - (1-\nu)p_c & \text{for } p \geq p_c. \end{cases}$$

Strong anomalous diffusion is thought to be generic [76, 93] for dynamics with fat-tailed waiting-time distributions. In those years, its dynamical basis has been analyzed through generalizations of the central limit theorem and non-normalizable densities [30, 76]. Concurrently, numerous studies have explored the relationship between the properties of deterministic dynamics and transport. Generally, chaos is associated with rapid decay of correlations and normal diffusion, whereas non-chaotic dynamics often leads to anomalous transport and slow decay of correlations [50, 94, 49, 45, 78]. Stochastic processes, while often resembling chaotic dynamics, can result in either normal or anomalous diffusion depending on their correlation decay rate [25, 18].

In this chapter, we will focus specifically on both the anomalous transport of heat and anomalous discharge of a capacitor. Transport in materials can exhibit anomalous behavior due to various factors such as fractal-like structures, varying thermal conductivity, and non-uniform energy distribution. Understanding these phenomena is crucial for applications in thermal management, energy storage, and the design of materials with tailored thermal properties. Our objective is to study the anomalous transport through time series data analysis. This thesis explores the domain of time series classification through the application of various algorithms on two distinct datasets and case studies where we aims to evaluate the performance of these algorithms on both artificially generated and experimentally obtained data. The first dataset was created through simulations to study the heat transport of a sensor immersed in a fluid under an applied voltage, the second task utilized experimentally obtained data, specifically the discharge of a capacitor where each experiment resulted in a time series of the capacitor's voltage.

3.1 Sensor Case Study

I conducted my thesis work at ELTEK GROUP S.p.A, that is a company based in Casale Monferrato, Italy, specialized in the design, development, and production of advanced electronic and electromechanical components. They focuses on creating high-precision parts for various industries, including automotive, medical, and industrial applications. There, we studied the heat transport in a sensor immersed in a fluid. The sensor was modeled in a simplified manner, consisting of two parallelepipeds of 500 x 500 x Z μm^3 connected by a filament of 100 x 200 x Z μm^3.

The same material was used for both the main bodies and the filament, with two materials tested: platinum and copper. Our objective was to simulate an applied voltage to the sensor in order to measure the temperature of the filament, and then analyze the data using artificial intelligence to determine the type of transport occurring. The sensor was immersed in various fluids for testing. Specifically, five different fluids were tested: water, ethanol, a 50% mixture of water and ethanol, hydrogen, and air. By conducting these experiments, we aimed to understand how different fluid environments affect the heat transport properties of the sensor and to explore the capability of AI in recognizing the transport mechanisms involved.

3.1.1 COMSOL Simulations

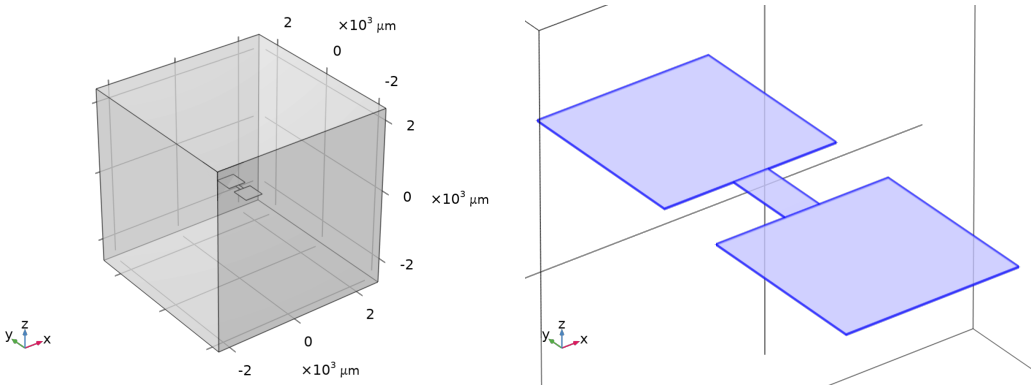
COMSOL Multiphysics is a simulation software platform used for modeling and solving complex scientific and engineering problems. It is widely recognized for its versatility and powerful capabilities in multiphysics simulations, where multiple physical phenomena interact with each other. The core strength of COMSOL Multiphysics lies in its ability to handle coupled physics problems seamlessly, whe users can combine various physical models such as fluid dynamics, heat transfer, structural mechanics, electromagnetics, chemical reactions, and more within a single simulation environment. The software provides an intuitive graphical user interface (GUI) that simplifies the setup, simulation, and post-processing of models. The tree-like model builder guides users through the simulation process, from defining geometry and materials to specifying physics and meshing. COMSOL comes with a wide range of physics modules that can be added as needed, each designed to handle specific types of physical phenomena. These modules include the Heat Transfer Module, AC/DC Module, CFD Module, among others.

Geometry

As previously mentioned, the setup is fairly simple. We have a sensor composed of two squares and a rectangle situated within a cube, as shown in the figures 3.1(a) and 3.1(b). This sensor undergoes extrusion to render the entire structure three-dimensional. The squares have dimensions of 500 x 500 micrometers, while the filament represented by the rectangle measures 100 x 200 micrometers. The extrusion thickness, denoted as s, is variable, as different thicknesses were tested to understand how heat transport behaves as a function of this parameter. In the initial analysis, five thicknesses were examined: 100 nm, 200 nm, 500 nm, 1000 nm, and 2000 nm.

Materials

Once the geometry is chosen, the materials are inserted. the materials should be associated with the figure of the sensor and the volume of the fluid, in our case, two



(a) Geometry (b) Sensor

materials were chosen for the sensor: platinum and copper; and five different fluids in which to "immerse" the sensor, which are: water, ethanol, 50 percent mixture of water and ethanol, hydrogen and air. Each material that is placed in the library contains properties, namely: electrical conductivity, thermal conductivity, density, heat capacity at constant pressure, and relative permittivity. We modified some properties related to the sensor materials, particularly electrical conductivity so that it was time-dependent, to make it more realistic, since we were given a constant value by the software, and thermal conductivity (which we will see how later) to simulate anomalous heat transport.

Equation

At this point the physics will be chosen to be applied to our model, as already mentioned we want to simulate a sensor immersed in a fluid to which a certain voltage is applied, and what we want to measure is the temperature obtained by the sensor over time. Comsol provides different modules to simulate different situations, in our case we had to use three of them: heat transfer in solids and fluids because of the heat exchange between the sensor and the fluid, electric currents because of the voltage we have to apply at the ends of the sensor, and laminar flow to describe the turbulent flow of the fluid because of the high temperature reach by the sensor. Now we will see the equations in detail:

- **Heat transfer in solids:**

$$\rho c_p \frac{\partial T}{\partial t} + \nabla \cdot \mathbf{q} = 0 \tag{3.1}$$

$$\mathbf{q} = -k\nabla T \tag{3.2}$$

where:

- ρ is the density of the material.
- c_p is the specific heat capacity of the material at constant pressure.
- T is the temperature.
- k is the thermal conductivity of the material.

- **Heat transfer in fluids:**

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = 0 \tag{3.3}$$

$$\mathbf{q} = -k \nabla T \tag{3.4}$$

where:

- ρ is the density of the material (fluid).
- c_p is the specific heat of the material (fluid) at constant pressure.
- T is the temperature.
- k is the thermal conductivity of the material (fluid).
- $\mathbf{u}$ is the velocity vector.

- **Thermal insulation:**

$$-\mathbf{n} \cdot \mathbf{q} = 0 \tag{3.5}$$

where:

- $\mathbf{n}$ is the normal vector.
- $\mathbf{q}$ is the heat flow.

- **Surface to ambient radiation:**

$$-\mathbf{n} \cdot \mathbf{q} = \epsilon \sigma (T_{amb}^4 - T^4) \tag{3.6}$$

where:

- $\mathbf{n}$ is the normal vector.
- $\mathbf{q}$ is the heat flow.
- ϵ is the surface emissivity.
- T_{amb} is the ambient temperature.
- σ is the Stefan-Boltzmann constant.

- **Laminar flow:**

$$\rho\frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = \nabla\cdot[-p\mathbf{I} + \mathbf{k}] \tag{3.7}$$

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0 \tag{3.8}$$

$$\mathbf{k} = \mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right) - \frac{2}{3}\mu(\nabla\cdot\mathbf{u})\mathbf{I} \tag{3.9}$$

  where:

  - ρ is the density.
  - p is the pressure.
  - $\mathbf{k}$ is the viscous stress tensor.
  - μ is the dynamic viscosity.
  - $\mathbf{u}$ is the velocity vector.

- **Current conservation:**

$$\nabla \cdot \mathbf{J} = 0 \tag{3.10}$$

$$\mathbf{J} = \sigma \mathbf{E} + \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J}_e \tag{3.11}$$

$$\mathbf{E} = -\nabla V \tag{3.12}$$

  - $\mathbf{J}$ is the current density vector.
  - σ is the electrical conductivity.
  - $\mathbf{E}$ is the electric field.
  - $\mathbf{D}$ is the electric displacement field.
  - $\mathbf{J}_e$ is the external current density.

- **Electric insulation:**

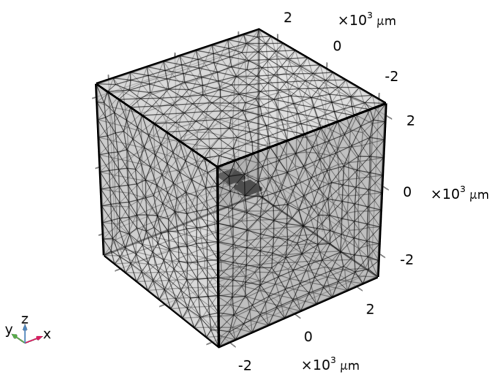
$$\mathbf{n} \cdot \mathbf{J} = 0 \tag{3.13}$$

  - $\mathbf{J}$ is the current density vector.
  - $\mathbf{n}$ is the normal vector.

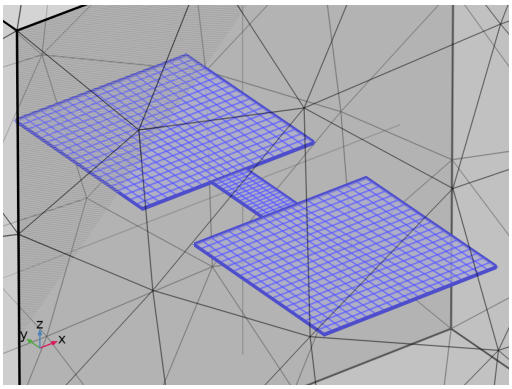
Mesh

Finally, meshing had to be applied. This technique involves discretizing the geometric domain into smaller, manageable elements to perform numerical simulations. COMSOL provides various types of automated meshes and also allows for custom mesh construction. For our model, two different types of meshes were chosen:

- Free Tetrahedral Mesh: This mesh type was applied to the outer cube, representing the fluid volume as you can see in Fig 3.1(c). The free tetrahedral mesh automatically generates tetrahedral elements, which are well-suited for complex geometries.
- Custom Mesh: This mesh was manually constructed and applied to the sensor as you can see in Fig 3.1(d). It consists of:
  - A grid of 20×20 squares applied to the two square areas of the sensor.
  - A grid of 15×5 rectangles applied to the filament area.



(c) Tetrahedral Mesh



(d) Custom Mesh

3.1.2 Simulations Analysis

As mentioned earlier, different types of simulations were conducted, primarily varying five factors: the fluid, the sensor material, the sensor thickness, the time and frequency of temperature sensing, and the voltage applied to the sensor terminals. The goal was to plot the temperature trend for each case to understand its characteristics better. The temperature was recorded at each instant by volumetrically averaging the filament area, the rectangle between the two squares that can be seen in 3.1(b).

Classic Transport

Initially, simulations were performed using only the platinum sensor while varying the other characteristics. The temperature trends were analyzed considering all fluids: water, ethanol, a 50% mixture of water and ethanol, air, and hydrogen, as well as all sensor thicknesses: 100 nm, 200 nm, 500 nm, 1000 nm, and 2000 nm, with an applied voltage of 1V.

Initial simulations were conducted over a time range from 0 to 1 second, considering 10 time instants. For example, in Figure 3.1, we can observe the temperature trend of the sensor immersed in water, showing a curve for each thickness. The numbers in the legend correspond to the thickness in nanometers. The first thing that stands out in this figure is the high temperature reached from the very first moments. Indeed, all the figures show a very high initial spike, which then remains at the same level. Additionally, as expected, increasing the thickness also increases the temperature. Another consistent pattern is the comparison of temperatures among the fluids. Essentially, we can notice from the figure 3.2 that while the shapes are similar, the temperature in ethanol is noticeably higher than in the other two fluids. This is better illustrated in the figure 3.3. In this case,

the figure shows 10 curves: 5 corresponding to the temperature measured in water with different thicknesses, exactly as shown in the previous figure 3.1, and the other 5 in ethanol. As can be seen, the shape of the curves for ethanol is very similar to those for water but reaches higher temperatures. The curves resulting from experiments conducted with gases show different shapes, as can be seen in Figure 3.4. In this case, the 5 curves corresponding to the different thicknesses are also shown, and again an initial spike can be observed. The difference lies in the behavior after the first moment in time, which tends to be increasing for the thinner layers. As expected, the temperature of the sensor immersed in a gas is significantly higher, which can also be observed in Figure 3.5, showing a temperature comparison between the sensor immersed in hydrogen (blue) and the sensor immersed in air (red). In this case, it can be seen that, in addition to the generally high temperatures, the temperatures of the sensor immersed in air are higher than those of the sensor immersed in hydrogen, given the same thickness. What stands out in this initial analysis are the somewhat high temperatures and the very steep initial spike. Therefore, we focused our analysis on two different aspects: First, we decided to reduce the analysis time to 0.2 seconds with a time step of $10^{-}4$ seconds to understand what happens during the spike while in the second case, we lowered the voltage applied to the sensor to 0.1V to reduce the temperature.

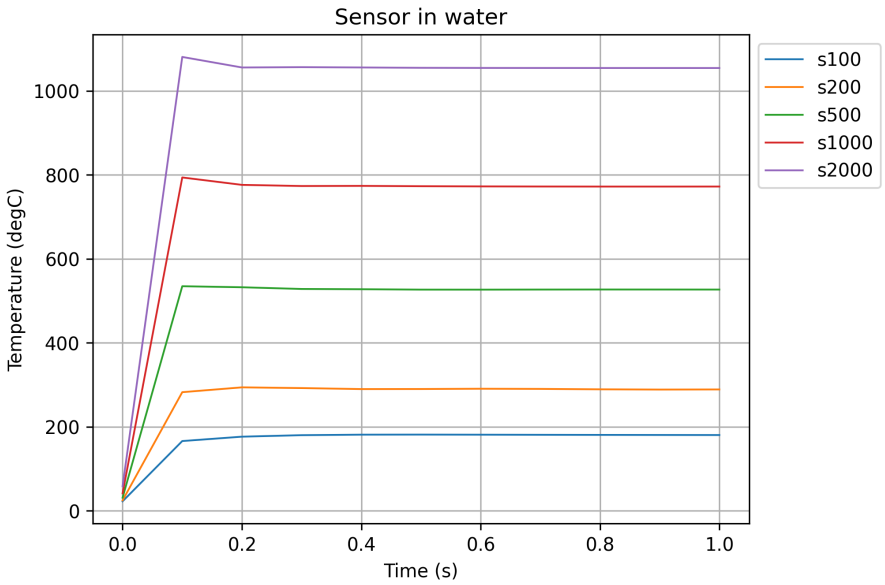


Figure 3.1: Sensor in water

Regarding the latter case, i.e., simulating with an applied voltage of 0.1V, a significant reduction in temperature is immediately noticeable. In fact, looking at Figure 3.6 and comparing it with the previous one, we can see that the shape has remained almost unchanged, but the temperature has dropped dramatically.

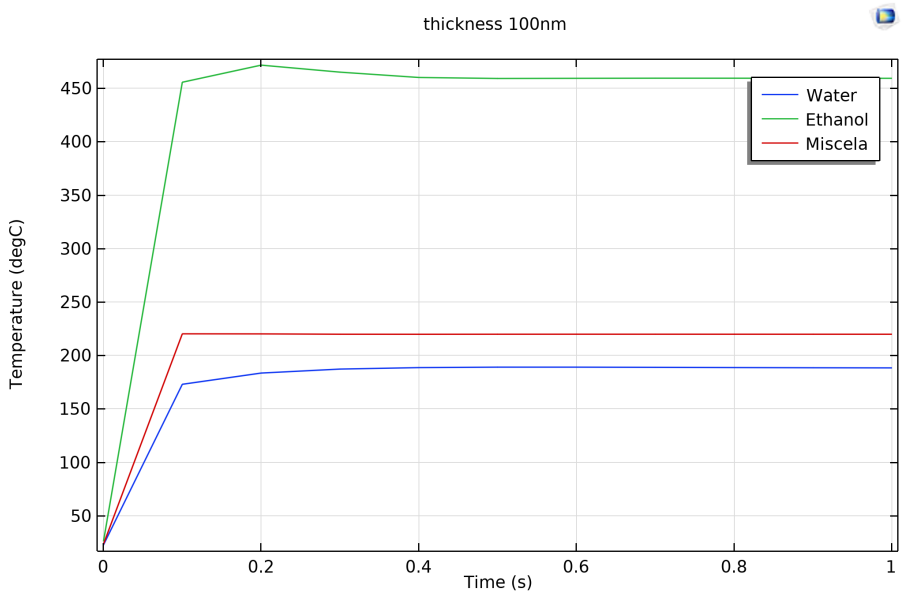


Figure 3.2: Comparison between water, ethanol and mixture of water and ethanol

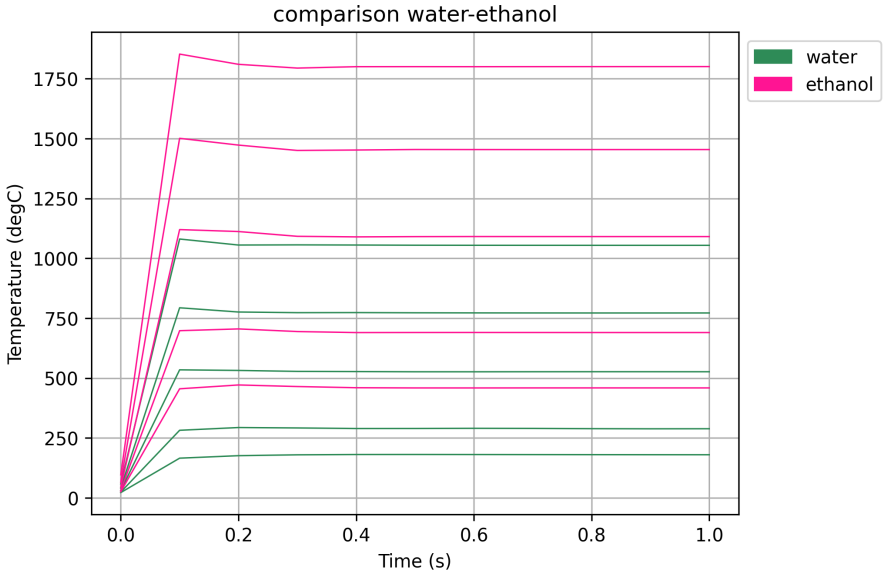


Figure 3.3: Comparison between water and ethanol with different thickness dimension

Observing the purple curve, which corresponds to the 2000nm thickness, a difference of about 1000 degrees Celsius can be noted. The temperature difference between water and ethanol remains the same, as seen in Figure 3.8. However, observing the curves for the sensor immersed in hydrogen 3.7, a substantial difference in shape

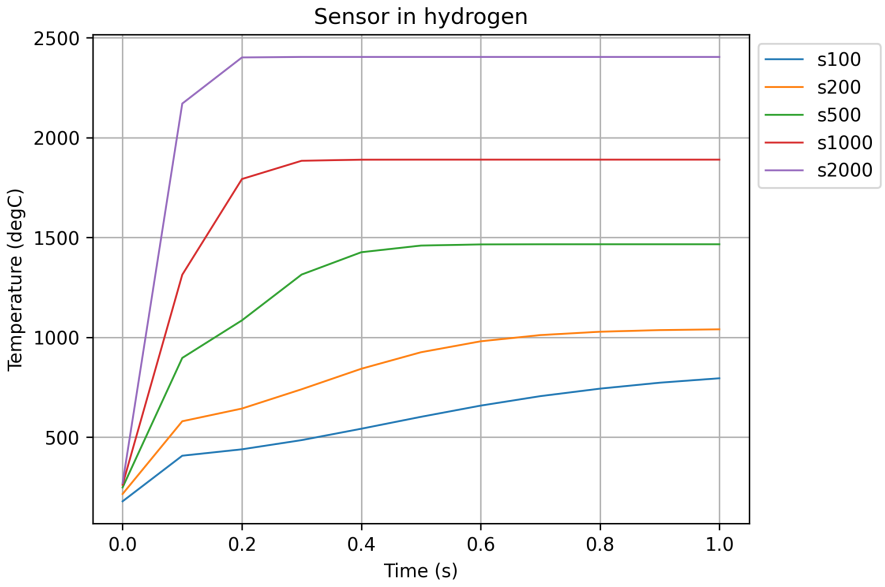


Figure 3.4: Sensor in hydrogen

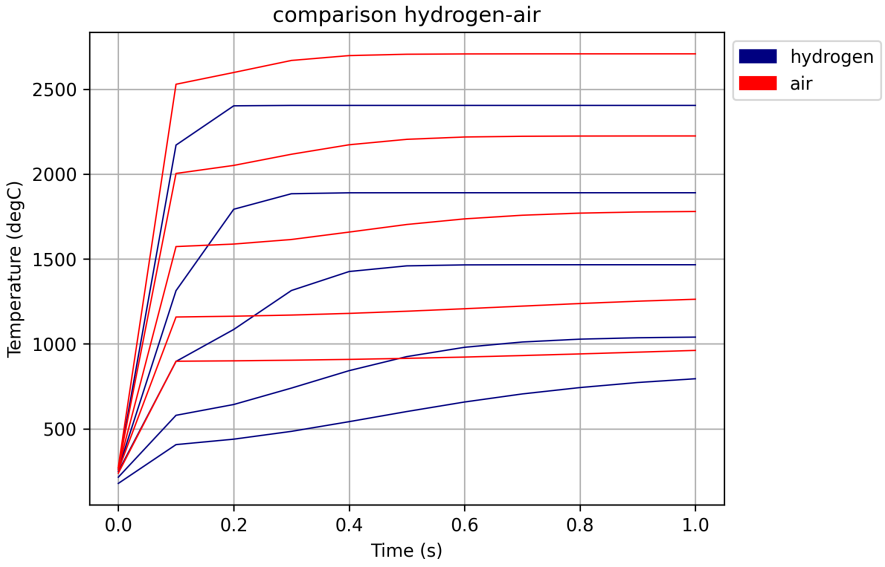


Figure 3.5: Comparison between hydrogen and air with different thickness dimension

can be seen, in this case the curves seem to diverge. Similarly, the curves for the sensor immersed in air are very similar. From Figure 3.9, it can be observed that the temperature difference between air and hydrogen is maintained, with the difference that in this case the curves have a very similar shape.

In the analysis of the first case, related to the reduction of time, it can be

seen that in the very first moments, the temperature reaches its highest points, then proceeds as previously observed, as shown in Figures 3.10(a), 3.10(b). In this case, the two figures show 3 curves for each fluid, corresponding to three filament thicknesses: 100 nm, 500 nm, and 2000 nm. Simulating all five cases was avoided due to the high computational cost associated with running simulations over all these time instants, while still demonstrating the trends of the most extreme curves.

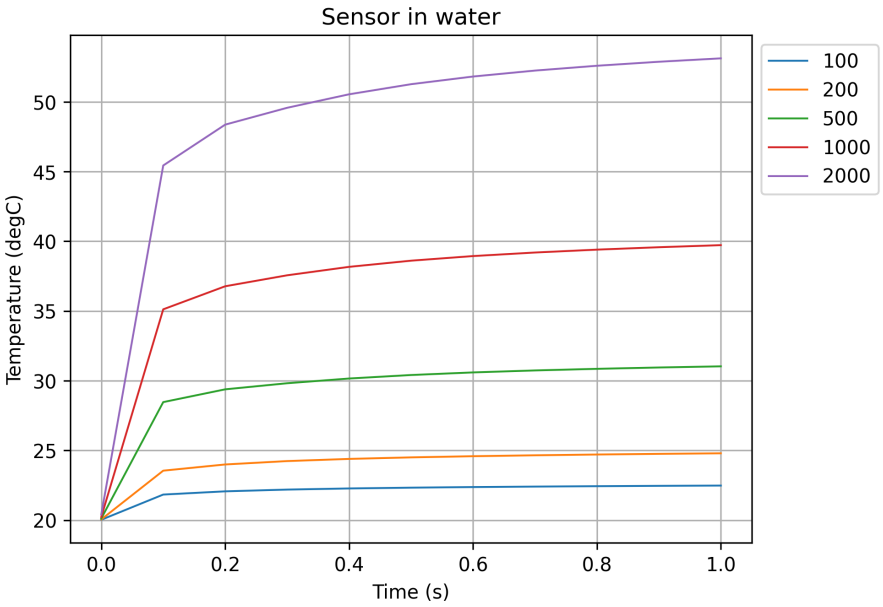


Figure 3.6: Sensor in water with 0.1V applied

The same experiments were conducted using copper as the sensor material. As expected, the temperatures for copper are generally higher. However, the shape of the curves is very similar to those for platinum. Figures 3.10 and 3.11 show the comparison between the temperature of a platinum sensor and that of a copper sensor measured in water and in hydrogen. The five curves for each material correspond to the five different sensor thicknesses. As mentioned earlier, the temperatures are higher, but the shapes are nearly identical.

Anomalous Transport

As previously mentioned, anomalous transport refers to non-equilibrium phenomena that cannot be described by standard physics methods. These phenomena can occur in various cases, such as when an object is very small (but not extremely tiny). In these cases, the object's dimensions are large enough to avoid quantum effects, yet small enough for classical physics to be insufficient. This intermediate scale can lead to unique behaviors where traditional transport models fail, and more sophisticated approaches are required to accurately describe the system's dynamics.

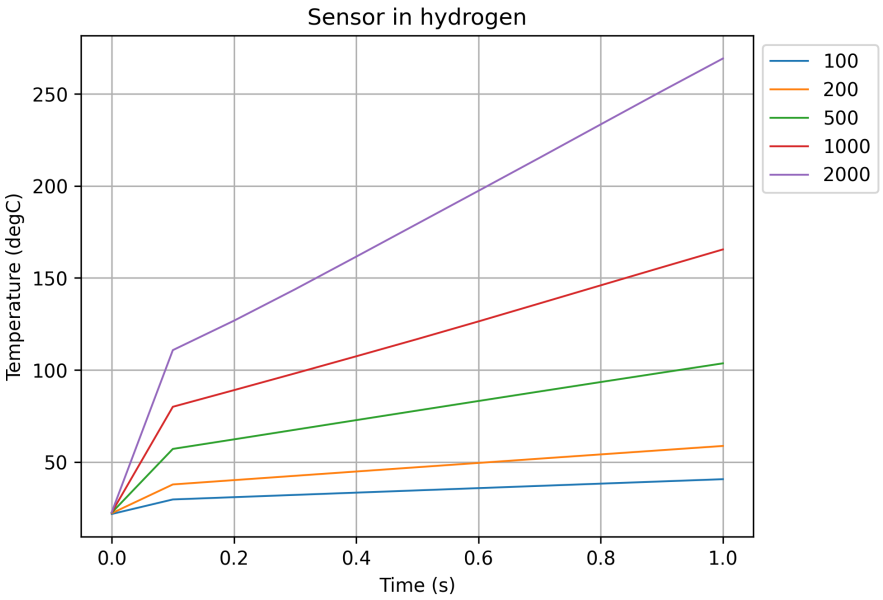


Figure 3.7: Sensor in hydrogen with 0.1V applied

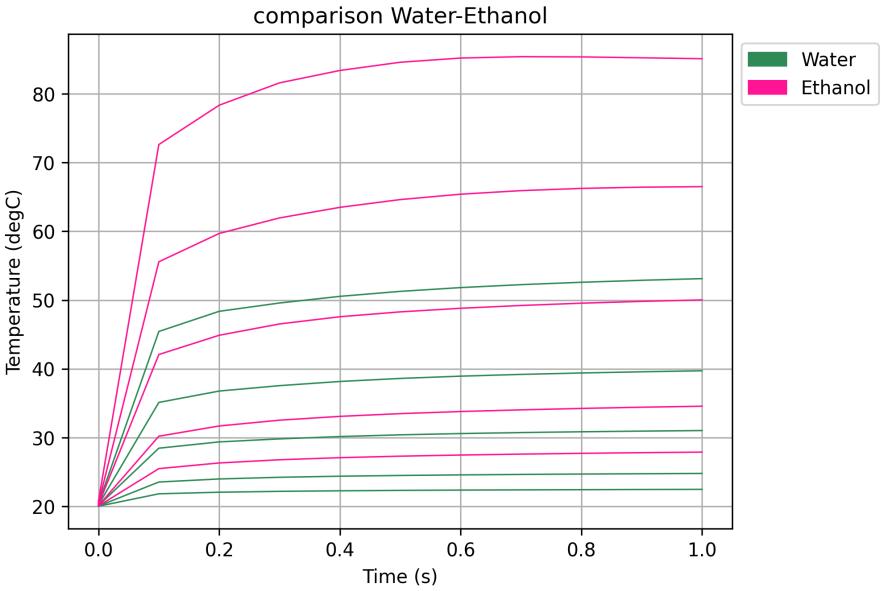


Figure 3.8: Comparison between water and ethanol with different thickness dimension with 0.1V applied

Heat transport processes in various low-dimensional systems no longer follow Fourier's law, which is a diffusion law.

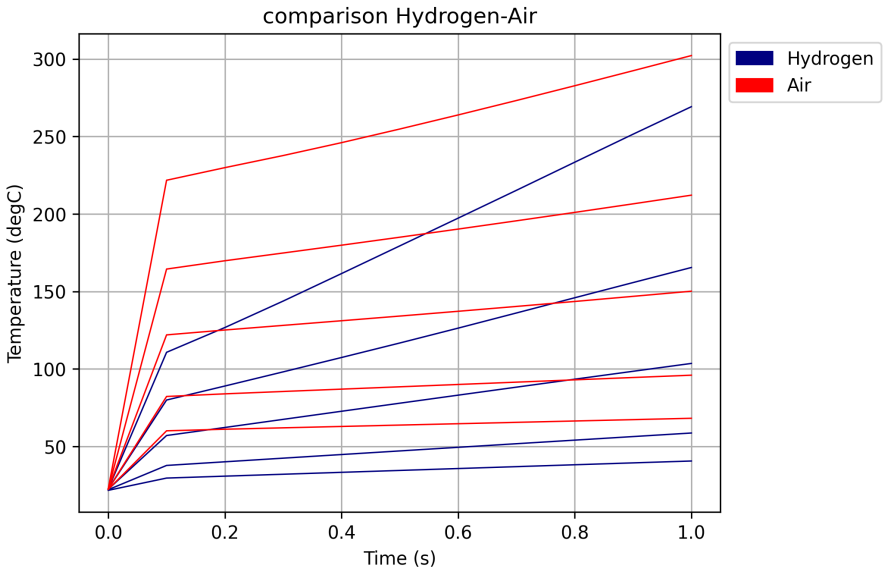
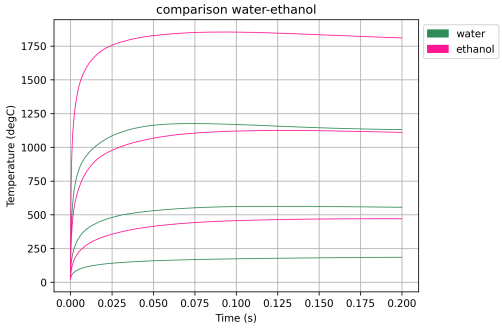
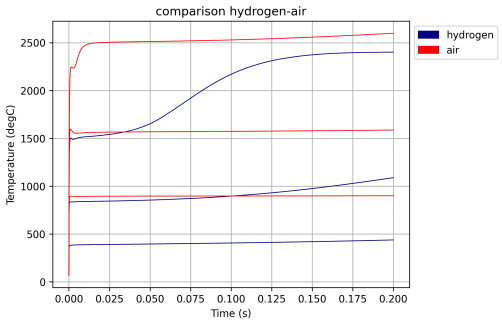


Figure 3.9: Comparison between hydrogen and air with different thickness dimension with 0.1V applied



(a) Comparison between water and ethanol in 0.2 seconds with 1V applied



(b) Comparison between hydrogen and air in 0.2 seconds with 1V applied

$$\frac{\partial f}{\partial t} = k\frac{\partial^2 f}{\partial x^2} \tag{3.14}$$

Fourier's law describes a well-known macroscopic behavior, which is fully characterized once the conductivity, k, and the initial and boundary conditions are given. Transport is called anomalous when this is not the case, and it is identified by deviations from the linear time dependence of the mean-square displacement of the transported quantity.

Several models of anomalous diffusion have been proposed. In our study, we chose the model presented in [9], known as the Time-dependent Diffusion Coefficient (TdDC) model. This model considers a modification of the diffusion equation

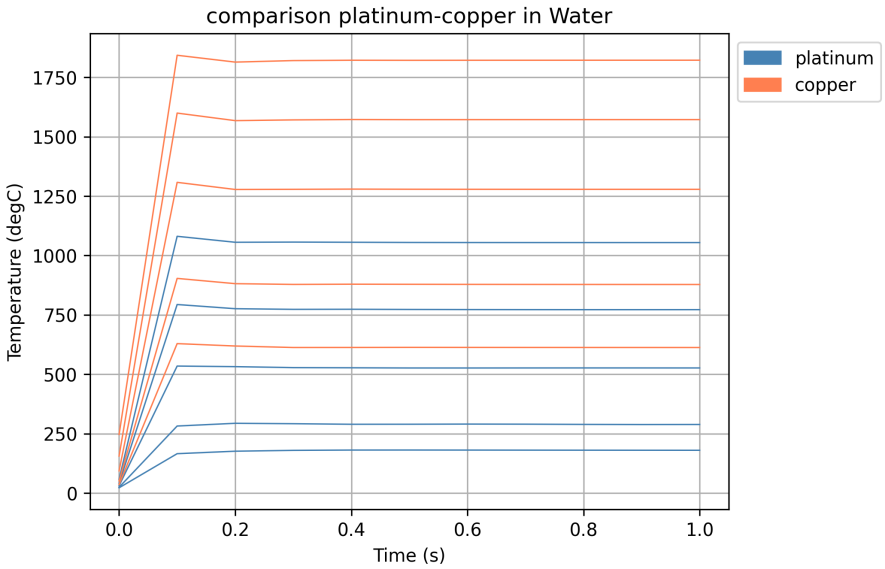


Figure 3.10: Comparison between platinum and copper with different thickness dimension with 1V applied in water

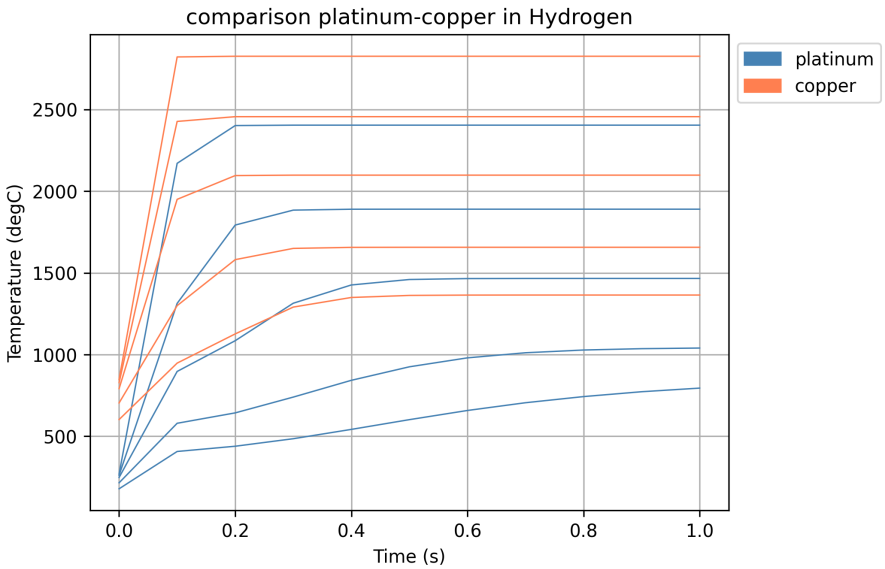


Figure 3.11: Comparison between platinum and copper with different thickness dimension with 1V applied in hydrogen

that accounts for a general growth rate, thereby describing anomalous behavior. Specifically, it is assumed a concentrated initial distribution, $u(x,0) = \delta(x)$, and that the spreading has a Gaussian profile with diffusion coefficient D, zero mean, and mean-square displacement growing as t^{γ}, with γ in the range [0,2].

This leads to a model analogous to Fourier's law, described as follows:

$$\frac{\partial u}{\partial t} = Dt^{\gamma-1}\frac{\partial^2 u}{\partial x^2} \tag{3.15}$$

For $\gamma = 1$, the above equation coincides with the diffusion equation 3.14 for $\gamma \neq 1$, it describes anomalous diffusion. In Comsol, the classical diffusion equation, is used, as shown in 3.1, 3.3. However, the thermal conductivity coefficient k of the material can be modified. To induce the TdDC, the following modification was made:

$$k = k \cdot t^{\gamma-1} \tag{3.16}$$

setting $D = k$ in 3.15. After this modification, simulations were conducted to observe any differences. Figure 3.12 represents a simulation of a platinum sensor with a thickness of 100 nm immersed in water, conducted with three different values of γ: 0.5, 1.5, and 2. As can be seen, the shape varies based on the value of γ, but at 1 second, they all have the same value, as expected, since according to 3.16, $k = k$.

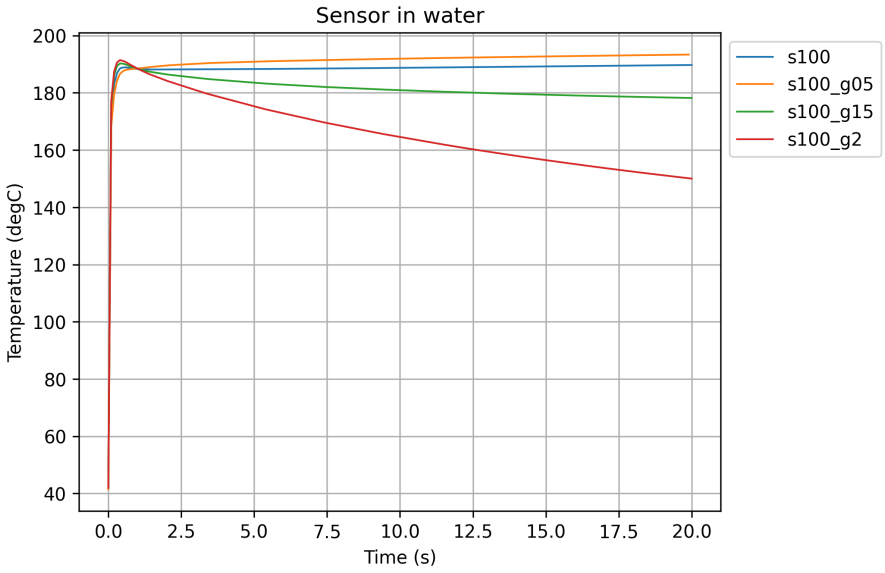


Figure 3.12: Comparison between Fourier transport and the TdDC with $\gamma = 0.5$, $\gamma = 1.5$ and $\gamma = 2$ using a platinum sensor with 1V applied in water

3.1.3 ML Analysis

Based on the results obtained in the previous chapter, it was decided to use the simulations conducted with an applied voltage of 0.1V to keep the temperature low.

Both platinum and copper were considered, as the shapes of the curves generated by these two materials were found to be similar. Additionally, a simulation duration of 8 seconds was chosen, as it appeared sufficient to discriminate between the different transport models.

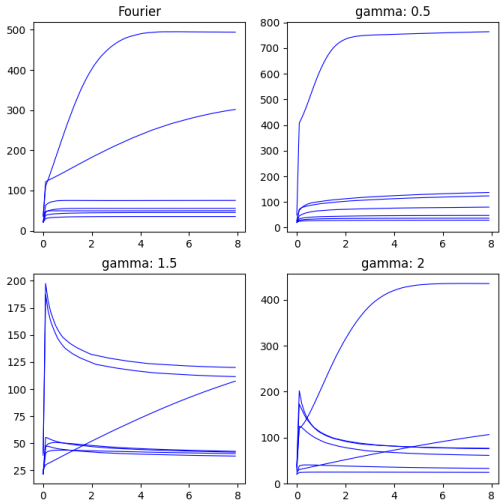
Datasets Creation

In this phase, two univariate datasets were generated, using a single feature for each sample, which in this case is the temperature. Specific experiments were designed for this phase, always considering the five fluids in which the sensor is immersed: water, ethanol, a 50% mixture of water and ethanol, hydrogen, and air. As mentioned earlier, experiments were conducted for 8 seconds, with a measure each 0.1 second, resulting in **80 time points**, with an applied voltage of 0.1V. However, unlike the previous experiments, seven different thicknesses were considered to create a broader dataset that are: 100, 200, 500, 1000, 1200, 1400, and 2000 nm. So, for each fluid, seven experiments were conducted, totaling 35 experiments, which were doubled since each simulation was performed for both platinum and copper.

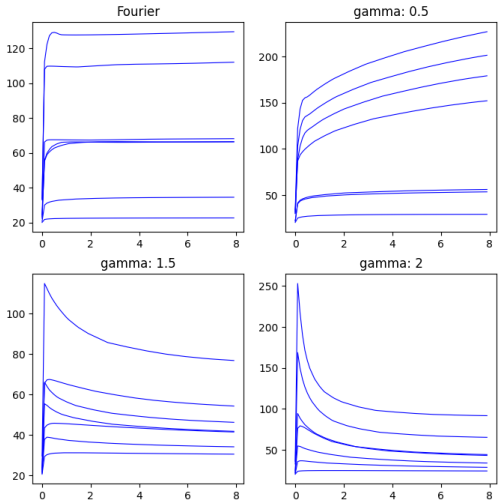
Experiments were conducted using four different diffusion models: the Fourier model and the TdDC model with $\gamma = 0$, 1.5, and 2. Thus, with 70 experiments per diffusion model, we obtained a dataset of **280 samples**. Figure 3.13(a) shows 7 samples per class. As can be seen from the images, there are some outliers that have a different shape compared to other samples in each class. These samples are related to the experiments conducted using gases (hydrogen and air) as the immersion fluid. Consequently, a new dataset was created, which is a subset of this one, using only the experiments conducted with liquids, this new dataset counts **168 samples**. As shown in Figure 3.13(b), these are significantly more homogeneous concerning their respective classes.

Preprocessing

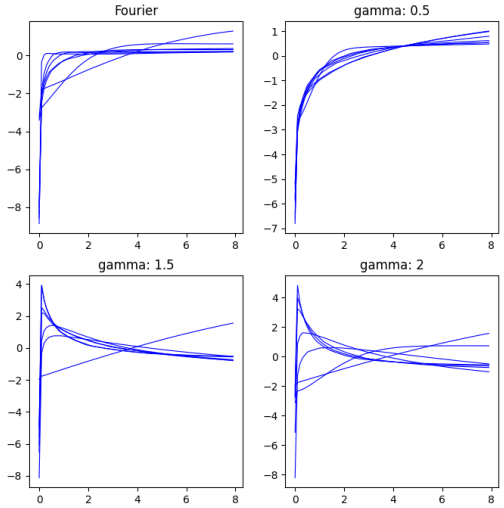
Upon obtaining the complete dataset, as seen in the images, the data are on different scales. Although the shapes are quite similar for each class, the temperatures vary. As mentioned in 1, many machine learning algorithms achieve better results when the inputs are scaled to a standard range. Even though the inputs are time series, the preprocessing techniques are very similar, in this case, standardization was applied. Specifically, for each sample, the mean and standard deviation of the time series were calculated, then the mean was subtracted and the result was divided by the standard deviation. This process ensures that each sample has a mean of 0 and a standard deviation of 1. Figures 3.13(c) and 3.13(d) show the two datasets after normalization.



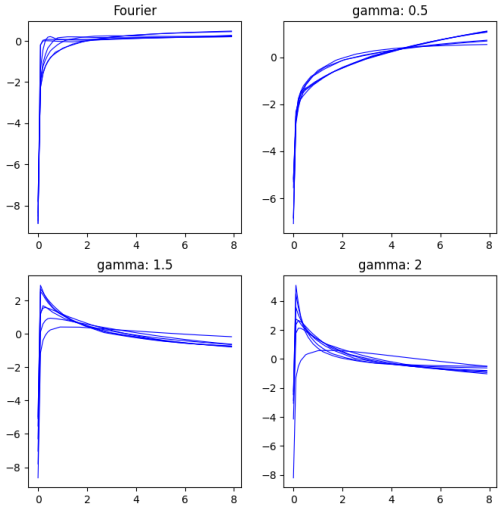
(a) Dataset Complete



(b) Dataset Liquid



(c) Dataset Complete Normalized



(d) Dataset Liquid Normalized

Experiments

All the experiments described in this thesis were conducted using Google Colab, that is a cloud-based platform that allows users to write and execute Python code in a web-based interactive environment. Additionally, the source code for the experiments can be accessed through the following GitHub repository [1].

The experiments were conducted as follows for both datasets: the dataset was

[1] https://github.com/Peppepelle99/ATP-C

divided into two parts, with 60% of the data used for training and 40% for testing. A substantial portion was allocated to the test set because the datasets are not very large, and we want to evaluate the model's ability to generalize. The 80% Of the training portion was used for actual training and 20% for model validation. Figure 3.13 shows the dataset division. Because the datasets are small, **k-fold cross-validation** with k=5 was used to explore the entire training dataset. Two metrics were used to evaluate the models: the mean and the standard deviation of accuracy across the 5 folds. Using k=5 results in an 80%-20% split for training and validation sets of the training dataset for each fold. Therefore, for each fold, the model undergoes a training phase using the training split, makes predictions on the validation split, and calculates accuracy by comparing the actual labels with the predicted ones, as shown in Figure 3.14.

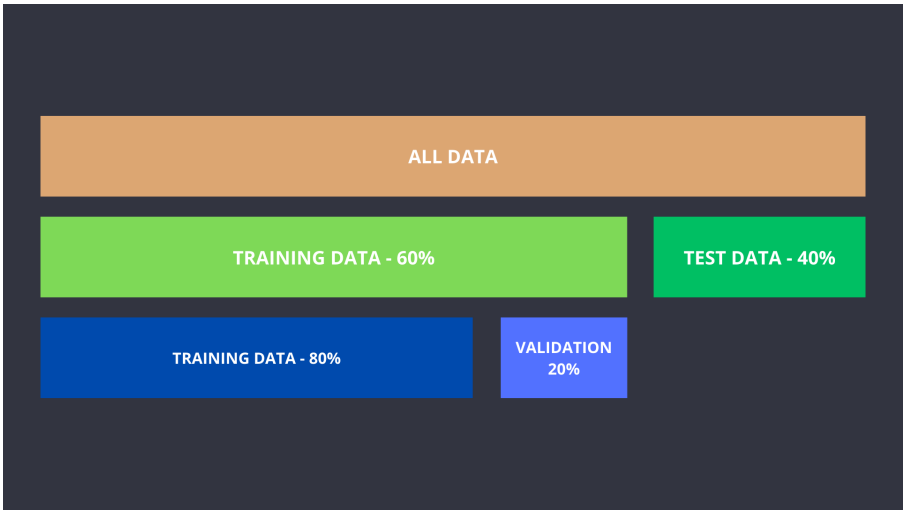


Figure 3.13: Division of the dataset

Models used

The models selected for the experiments were all taken from the aeon framework. This toolkit is dedicated to time series analysis and contains models for various tasks, including forecasting, classification, regression, clustering, and anomaly detection. The aeon team has compiled extensive literature on time series and consolidated all the information on their website (see the footnote [2]). Most of the content is related to classification and includes all types of models.

In our case, seven models were selected, one for each category described in the section 2. The models were chosen based on the analysis done in [68] which

[2] https://www.aeon-toolkit.org/en/latest/api_reference.html

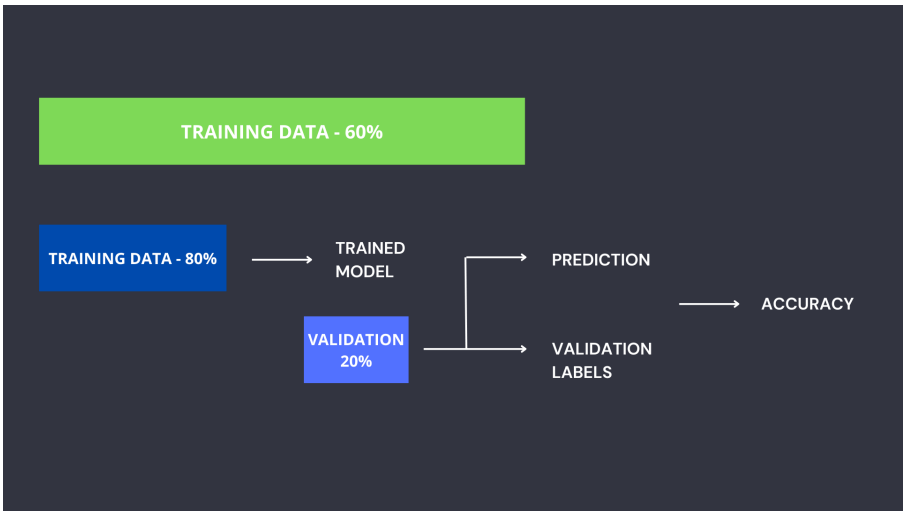


Figure 3.14: visualization of the experiment

classified the state of the art in time series classification using the extensive UCR archive as a benchmark. The models in question are: HIVE-COTE 2.0 (HC2), HYDRA+MultiROCKET (multiHydra) , RDST, DrCIF, Weasel 2.0 (weasel-d), freshPRINCE (freshP), and inceptionTime (inceptionT).

- **multiHydra:** Hydra [22] is a dictionary-based method that utilizes convolutional kernels, integrating features from both ROCKET [21] and traditional dictionary methods. It transforms the input time series using a set of random convolutional kernels, organized into g groups with k kernels per group. At each time point, the method counts the kernels that best match the input time series for each group. These counts are then used to train a linear classifier. Similar to other dictionary methods, Hydra uses patterns that approximate the input and generates features representing the counts of these patterns. However, unlike typical dictionary methods, Hydra employs random patterns represented by random convolutional kernels. Hydra has two distinguishing features: the kernels are organized into groups and counts the kernels in each group that represent the closest match to the input at each time point. In essence, Hydra treats each kernel as a pattern in a dictionary and each group as a separate dictionary. The kernels in each group compete to be counted at each time point, effectively capturing the most representative patterns.

  ROCKET transforms input time series using a large number of random convolutional kernels (by default, 10,000), and uses these transformed features to train a linear classifier. The two key aspects of Rocket in terms of accuracy are the use of dilation and PPV (proportion of positive values) pooling. **MultiRocket** represents an extension of Rocket, adding three additional pooling operations (besides PPV) and transforming both the original time series and

the first-order difference. MultiRocket [89] uses a fixed kernel length of 9, and a small fixed set of 84 kernels, bias values drawn from the convolution output, and a fixed set of dilation values relative to the length of the input time series. It only produces PPV features, enhancing the model's ability to capture complex patterns in the data.

The combination of Hydra with MultiRocket, referred to as Hydra+MultiRocket obtain the strengths of both approaches. Hydra's structured grouping and competitive counting mechanism, combined with MultiRocket's advanced pooling operations and first-order difference transformation, create a powerful and versatile time series classification model.

- **RDST:** The Random Dilated Shapelet Transform (RDST) [33] is a shapelet-based algorithm that use different mechanism from convolutional approaches. Unlike traditional shapelet algorithms that search for the best shapelets within the training dataset, RDST adopts a more efficient and scalable strategy by randomly selecting a vast number of shapelets from the training data. These shapelets typically range from thousands to tens of thousands. Features derived from these randomly selected shapelets are then used to train a linear Ridge classifier. A key innovation of RDST is the employment of dilation with shapelets. Dilation is a form of down-sampling that defines spaces between time points. When a shapelet with dilation x is compared to the time series, it matches against time points that are x steps apart, effectively allowing the algorithm to capture patterns at different temporal resolutions. This is particularly advantageous in time series data where relevant patterns may occur at varying scales and intervals. RDST represents a blend of traditional shapelet-based methods and modern convolutional techniques, making it a powerful algorithm for capturing complex temporal patterns in time series data. Its ability to handle large datasets and uncover multi-scale patterns has proven effective in a wide range of applications.

- **DrCif:** The Time Series Forest (TSF) [24], is a foundational interval-based tree ensemble method. In TSF, each tree is constructed using $\sqrt{m}$ intervals, where m is the length of the time series. These intervals are selected randomly in terms of position and length but are consistent across all series in the dataset. For each interval, mean, variance, and slope are calculated in order to build a feature vector. This feature vector serves as the basis for building each tree, and predictions are made using the features extracted from the same intervals.

  The Canonical Interval Forest (CIF) [65] extends TSF by improving accuracy through the integration of more informative features and increasing diversity. Similar to other interval-based approaches, CIF is still an ensemble of decision tree classifiers, but in addition to the mean, standard deviation, and slope,

CIF incorporates Catch22 features [57], which are a set of 22 time series features that are both interpretable and highly discriminative. Intervals are still generated randomly, with each tree selecting $k = \sqrt{m\sqrt{d}}$ intervals, where d is the number of dimension.

The Diverse Representation Canonical Interval Forest (DrCIF) [67], enhances CIF by incorporating two new series representations: periodograms and first-order differences. For each of these three representations (the original series, periodograms, and first-order differences), DrCIF randomly selects intervals using the formula $(4 + \sqrt{r\sqrt{d}})/3$, where r is the length of the series for a given representation. These intervals are concatenated into a comprehensive feature vector, which is then used to train the ensemble of decision trees. This approach not only maintains the phase-dependent interval selection but also obtain diverse representations to capture a broader range of temporal patterns, thereby enhancing the classifier's performance.

- **WEASEL-D:** The Word Extraction for Time Series Classification [84], is a pipeline classifier that focuses on identifying words whose frequency counts can distinguish between different classes while discarding those without discriminatory power. The classifier generates histograms of word counts over a wide range of window sizes and word length parameters, including bigram words derived from non-overlapping windows. A Chi-square test [90] is applied to assess the discriminatory power of each word, and words falling below a specific threshold are discarded through feature selection. Like many other approach the final step involves training a linear Ridge classifier on the refined feature space. WEASEL uses a supervised variation of Symbolic Fourier Approximation (SFA) to create discriminative words and employs an information-gain-based methodology to identify breakpoints that differentiate between classes.

  WEASEL v2.0 [85] is a enhancement of the original WEASEL v1.0 classifier. It control the search space through randomly parameterized SFA transformations to address the problem of extensive memory footprint. WEASEL 2.0 uses a dilated sliding window, with a fixed gap between each value (of this dilation parameter) to extract subseries with non-consecutive values from a time series. Words are still generated from SFA, by passing the dilated subseries through a Fourier transform.

- **InceptionTime:** InceptionTime [42] is a deep learning model. It consists of an ensemble of five deep learning classifiers, each with the same architecture built upon cascading Inception modules. Diversity among the five models is obtained by randomizing the initial weight values for each model.

The network architecture, comprises two consecutive residual blocks containing three Inception modules. In order to resolve the vanishing gradient problem, the input of the residual block is connected to the block's output via a shortcut connection. Following the residual blocks, a Global Average Pooling (GAP) layer is used. Finally, a fully-connected softmax output layer is employed.

An Inception module begins by applying a bottleneck layer, transforming an input multivariate time series to a lower-dimensional time series. It then applies multiple convolutional filters of varying kernel sizes to capture temporal features at different scales, this is a technique known as multiplexing convolution.

- **FreshPRINCE:** Before to talk about freshPRINCE we should talk about TSFresh (Time Series Feature extraction based on scalable hypothesis tests) [19], that is a comprehensive library that extracts nearly 800 features from time series data. These features encompass a wide range of characteristics, including statistical properties, time-frequency domain characteristics, and more. These features can be used directly for time series classification, but TSFresh add also a feature selection method known as FRESH (Feature Extraction based on Scalable Hypothesis tests) in order to delete irrelevant features. The FRESH algorithm evaluates each feature using multiple hypothesis tests, including Fisher's exact test [29], the Kolmogorov-Smirnov test [62], and the Kendall rank test [48]. This rigorous selection process ensures that only the most relevant features are used for the classification task. Comparing various pipelines of feature extractors and classifiers turns out that the most effective approach was to use the complete set of TSFresh features without any feature selection, combined with a Rotation Forest classifier [77]. This technique was called FreshPRINCE [64], that uses the extensive feature set provided by TSFresh, ensuring a rich representation of the time series data. By combining this with the Rotation Forest classifier, which is known for its robustness and high performance in classification tasks, FreshPRINCE achieves superior accuracy and effectiveness in time series classification.

- **HIVE-COTE 2.0:** The original version of HIVE-COTE (HIVE-COTE α) (Hierarchical Vote Collective of Transformation-based Ensembles) [55], is an ensamble model that comprises five different models that each worked on features derived from different domains. These ensembles included the Elastic Ensemble [53], the Shapelet Transform Classifier [38], the Time Series Forest [24], the Bag of Symbolic-Fourier-Approximation Symbols [83], and the Random Interval Spectral Ensemble [55]. Each module was independently built and trained on the data. For new data, each module provided an estimate of class probabilities to a control unit, which combined these estimates to form

a single prediction by weighting each module's probabilities according to an estimate of its testing accuracy derived from the training data.

HIVE-COTE 1.0 (HC1), demonstrated the utility and scalability of the system but dropped the distance-based EE due to its high computational overhead.

HIVE-COTE 2.0 (HC2), replaced three classifiers from HIVE-COTE 1.0. The updated component modules in HC2 include the Shapelet Transform Classifier (STC), a shapelet-based classifier; the Arsenal, a convolution-based ensemble of ROCKET [21] classifiers; the Temporal Dictionary Ensemble (TDE) [66], a dictionary-based representation; and the Diverse Representation Canonical Interval Forest (DrCIF), an interval-based classifier.

Each component in HC2 is trained independently and is required to produce an estimate of its accuracy on unseen data. For new data, each module generates a probability estimate for each class. The controller then constructs a tilted distribution through exponentiation (with $\alpha = 4$ by default) to accentuate differences between classifiers, weighting them with the accuracy estimates. This approach improve the overall predictive performance by mantain the strengths of each constituent classifier.

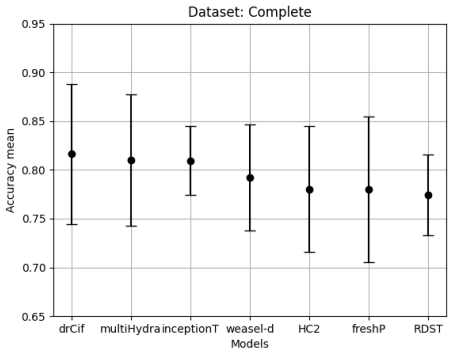
Hyperparameter Tuning

For each model, a hyperparameter tuning phase was conducted using the NNI (Neural Network Intelligence) framework. NNI is an open-source automated machine learning toolkit that facilitates hyperparameter optimization, neural architecture search, and model compression [63]. It enables users to define search spaces, select optimization algorithms, and run experiments efficiently.

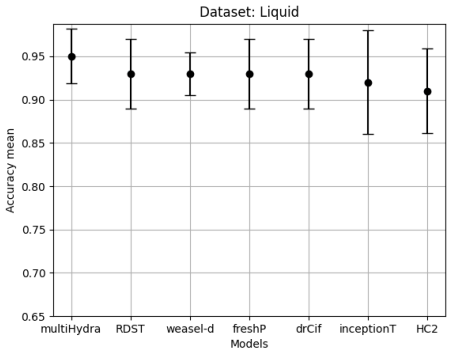
For each model, a series of experiments were conducted as described in the previous section. The mean and standard deviation of the accuracy were considered, and when these metrics were comparable, the execution time was evaluated to select the best hyperparameters. The parameter space for each model is shown in Table A.1, while the best hyperparameters are presented in Table A.2 and A.3. Figures 3.15(b) and 3.15(a) display the mean accuracies with associated variances for each model based on the best experiments.

Results

After obtaining the best hyperparameters, the models were tested. The entire training dataset, which is 60% of the full dataset, was used for training. Predictions were then made on the test dataset and compared with the actual labels to compute accuracy. Tables 3.1 and 3.2 show the results obtained from the models, ordered by accuracy for the two datasets. As we can see, all models perform quite well in



(a) Accuracies results for dataset Complete



(b) Accuracies results for dataset Liquid

both cases. For the liquid dataset, as expected, the results are significantly better due to the absence of outliers from gas experiments. Figures 3.15(c) and 3.15(d) are confusion matrices generated by summing the results from all models. In these matrices, the x-axis represents the predicted labels, while the y-axis represents the actual labels. Label 0 corresponds to Fourier transport, label 1 corresponds to TdDC with $\gamma = 0.5$, label 2 to $\gamma = 1.5$, and label 3 to $\gamma = 2$. A confusion matrix is a table used to describe the performance of a classification model. It shows the number of true positive, true negative, false positive, and false negative predictions. Each cell in the matrix indicates the number of samples that were predicted to belong to a specific class versus the actual class. This allows for a detailed analysis of the model's performance, the diagonal elements indicate the number of correctly predicted samples, while the off-diagonal elements represent the errors, highlighting which classes are often confused. In the case of the liquid dataset, we can see that the most frequently misclassified label is 2, which was incorrectly predicted 12 times out of 126 times, it was confused with label 0 seven times and with label 3 five times, remember that this result represents the sum of model results, so each number should be divided by 7 (number of models) to obtain an average over model. Similarly, for the complete dataset, label 2 is the most frequently misclassified, often mistaken for label 0 or label 3, with a higher error percentage due to the presence of outliers.

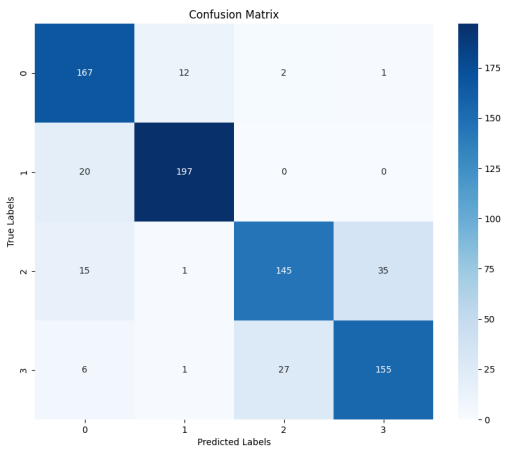
Table 3.1: Accuracy table for Dataset Liquid

Model	**Accuracy (%)**	**Prediction Time (s)**	**Training Time (s)**
multiHydra	98.5	**0.566**	1.202
freshPrince	98.5	4.608	8.77
hivecote2	97.1	83.678	183.309
drCif	97.1	3.243	6.844
weasel-d	94.1	0.91	**1.138**

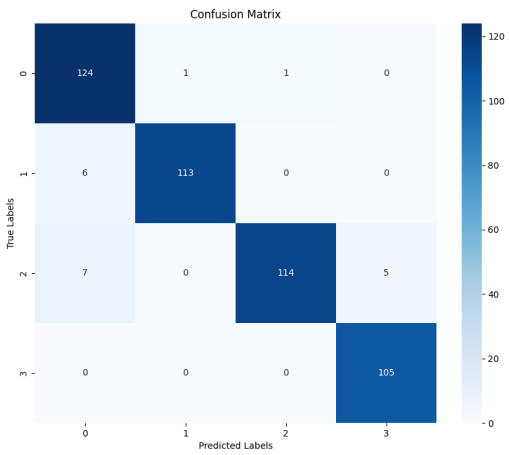
Model	Accuracy (%)	Prediction Time (s)	Training Time (s)
rdst	94.1	**0.121**	**0.625**
inceptionT	92.6	1.229	53.252

Table 3.2: Accuracy table for Dataset Complete

Model	Accuracy (%)	Prediction Time (s)	Training Time (s)
multiHydra	89.3	0.707	2.325
hivecote2	89.3	126.184	293.385
weasel-d	84.8	**0.554**	**1.377**
freshPrince	83.9	6.4	12.656
drCif	83.9	5.43	9.507
rdst	83.0	**0.175**	**0.722**
inceptionT	78.6	1.216	54.649



(c) Confusion matrix for dataset Complete



(d) Confusion matrix for dataset Liquid

3.2 Capacitor Case Study

As previously mentioned, this study examines the use of time series classification models to identify anomalous transport in two different datasets: one artificially created and the other derived from experiments. The artificially created dataset concerned the heat transport of a sensor immersed in a fluid, whereas the experimental dataset involves the discharge of a capacitor. The experiments, as will be seen in [10], were conducted as follows: A charged capacitor with a certain voltage is discharged through the use of two microelectrodes immersed in a fluid, with a gap

of approximately 2 micrometers between them, and its discharge was measured using an oscilloscope. The data were captured and serialized into 10,000 time points. The goal was to determine the type of diffusion occurring during the capacitor's discharge, in fact, the close proximity of the electrodes, their shape, and the fluid in which they are immersed can cause anomalous transport phenomena. The obtained data were then classified as follows: using the TdDC model and Bayesian analysis with a specific distance metric, each experiment was compared with various values of D and γ from 3.15, so the curve that best fit the experimental data was identified. The results were plotted on a graph, as shown in Fig. 3.15. This graph represents all experiments conducted on a single day with an applied voltage of 12V. Each point represents an experiment, characterized by shape and color indicating the fluid used, and position indicating the best-fitting D and γ. The further γ deviates from 1 (which represents classical Fourier diffusion), the more we are in a condition of anomalous diffusion.

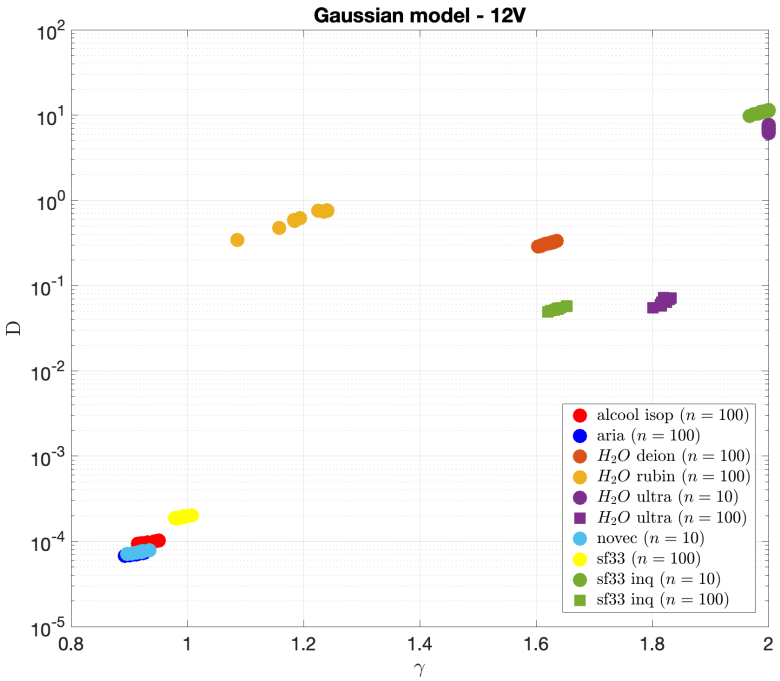


Figure 3.15: Experiment with a 12V voltage applied, images given by [10] with permission

3.2.1 ML Analysis

The objective of this phase was to train models capable of recognizing whether an experiment belonged to a class of normal or anomalous transport. More specifically,

the goal was to determine the corresponding values of D and γ. However, since we only analyzed classification models, we couldn't extract such information from these models indeed determining the specific values of D and γ for a given experiment is a regression problem, as D and γ exist in a continuous space.

Dataset Creation

What we did was to create classes based on the received data. The parameter space was divided into 16 blocks as shown in Figure 3.16, with each block associated with a specific class. The classes range from 0 to 15. All experiments in the blocks of the first column, corresponding to a γ value between 0.8 and 1.1, are considered normal diffusion experiments, the further to the right one goes, the more anomalous the diffusion condition becomes. Table A.4 shows the precise association between parameters (γ and D) and classes. In this case, the dataset is larger, containing 350 elements. Unlike the sensor dataset, the classes are highly imbalanced, with 200 elements belonging to class 0 (which corresponds to normal diffusion) and only like 150 elements in the anomalous diffusion zone, occupying the remaining positions, some grid positions are not occupied at all. More details can be found in Table A.5.

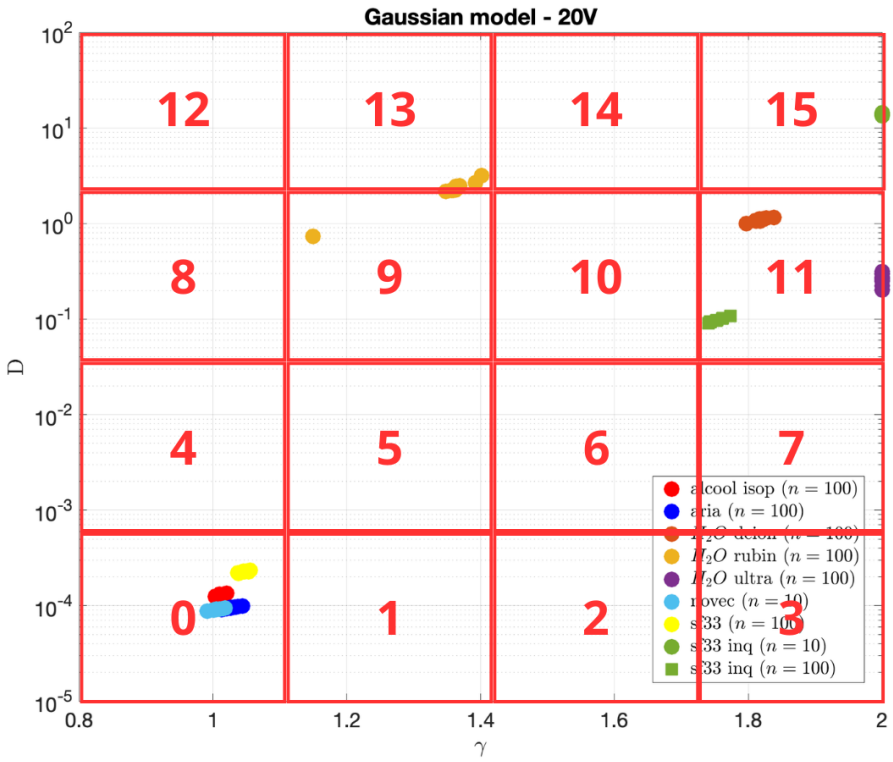


Figure 3.16: Parameters space division

Preprocessing

Each experiment contained **10,000 data points**, and since these were real-world experiments, the amount of noise in each experiment was quite high, The image 3.17 shows a single curve that includes an initial segment before zero seconds, which exhibits indeterminate shapes. In this phase, this part was removed, resulting in each single curve being reduced to 8,900 points, as can be seen in the fig 3.18 that shows some sample of the entire dataset for each class. As we said, these data are very noisy, so the first step in this phase was to downsample the experiments in order to remove the noise and work with smaller number of data. We tried several methods, but found that the moving average technique was the most effective. This approach not only reduced the amount of data but also mitigated the noise problem. Using the complete dataset would mean that training each model would take too long, given that each sample contained 8,900 points.

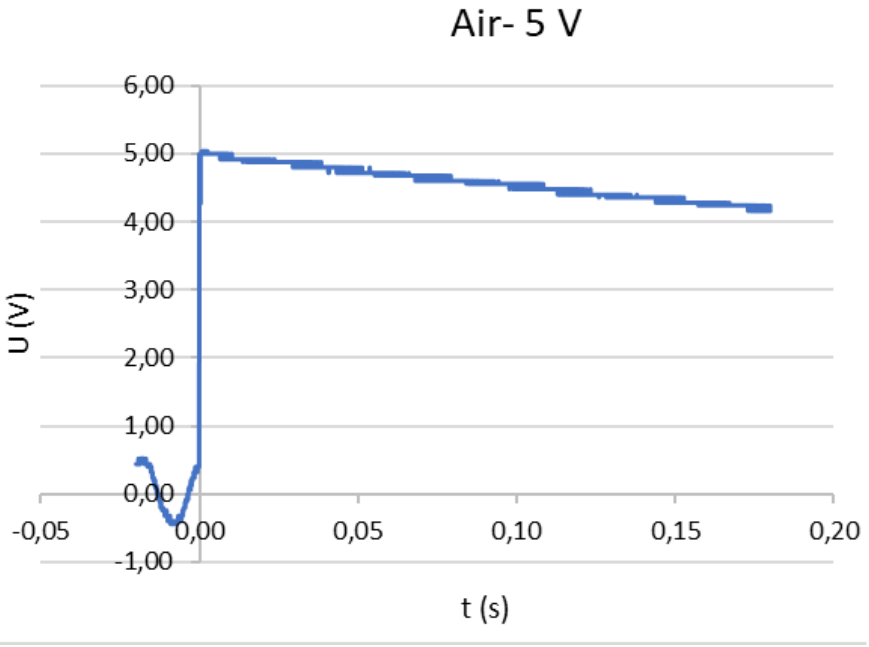


Figure 3.17: A single experiment conducted with 5V applied in Air, images given by [10] with permission

The moving average is a technique that involves averaging a subset of data points within a moving window. Specifically, it smooths the data by creating a series of averages of different subsets of the full data set. In our case, we used a window size of 10. This means that for every 10 consecutive data points, we calculated their average and used this value to represent that segment of the data. By doing this, we effectively reduced the dataset size to 1/10th of the original, resulting in each sample containing **890 points**. This reduction made the dataset more manageable

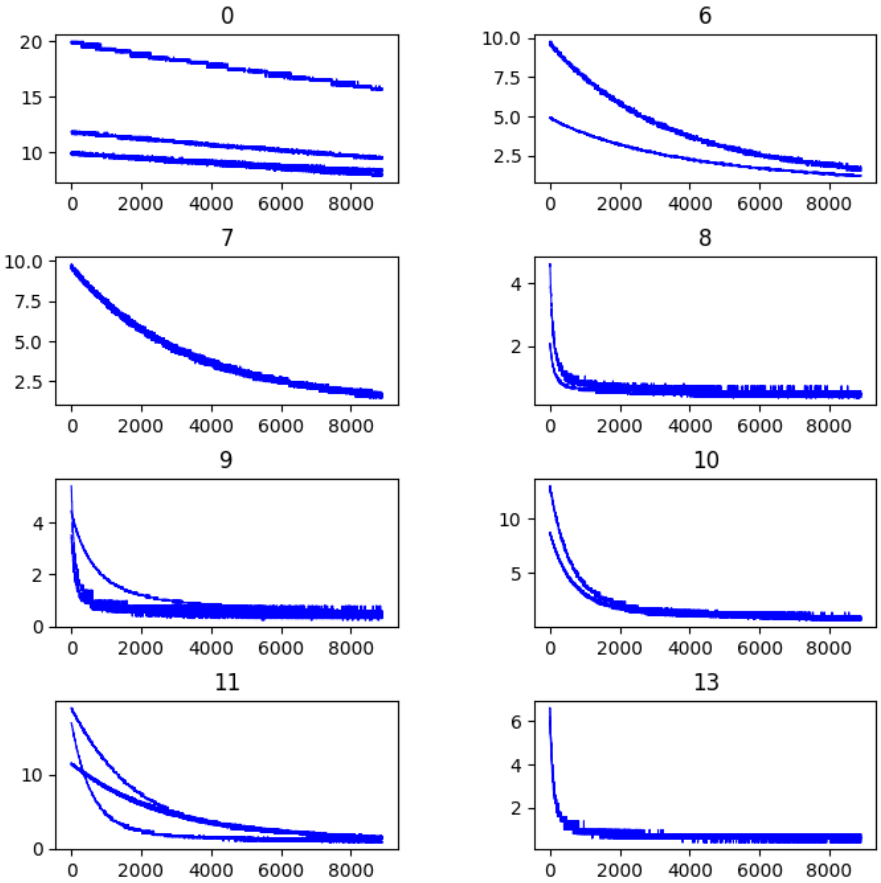


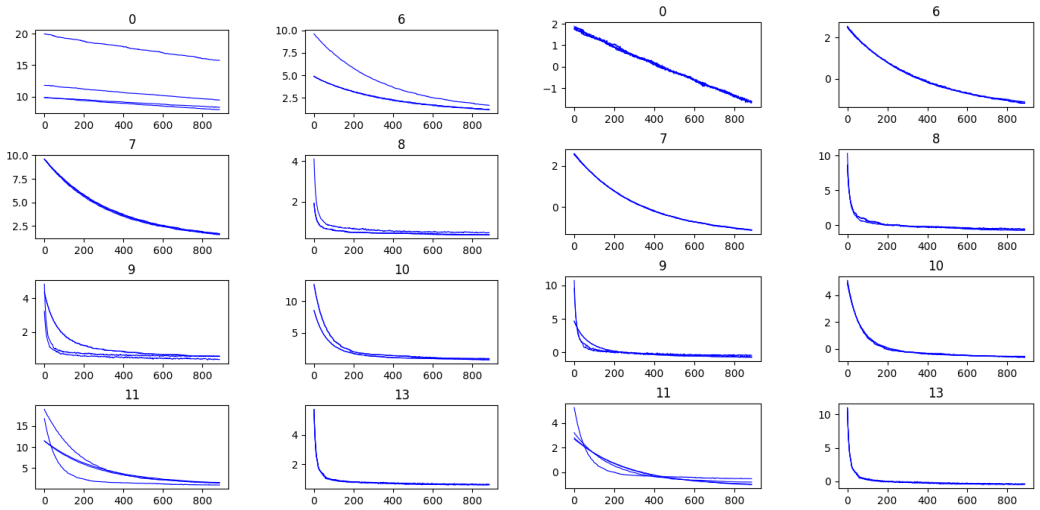
Figure 3.18: Original Dataset

for training while also smoothing out the noise inherent in the real-world data. As shown in figure 3.19(a), this approach results in smoother curves with reduced noise.

We also applied standardization to ensure that the samples were scaled to have a mean of 0 and a variance of 1. The result of this standardization process is shown in figure 3.19(b).

Results

Experiments were conducted similarly to the sensor case, In this case, however, the dataset was divided as follows: 80% for training (and validation) and 20% for testing, due to the larger dataset. Using the NNI toolkit and 5-fold cross-validation, all experiments for each model were executed to obtain the best hyperparameters. Figure 3.20 displays the mean and variance of accuracy for each model on the dataset. Table 3.3 lists the accuracy of the models on the test dataset. A confusion matrix was also created, fig 3.19 reveals that label 8 was the most misclassified,



(a) Downsampled Dataset

(b) Normalized Dataset

with 8 errors out of 28 samples, also in this case the confusion matrix was generated summing the prediction of each model. On this occasion, there were also other experiments conducted on different days, resulting in the creation of three additional datasets: one consisting only of experiments using insulating fluids, one consisting only of conductive fluids, and one consisting of mixtures. These datasets were used as tests to unequivocally evaluate the generalization ability of the models. Tables A.8, A.7 and A.6 show the number of elements per class for each dataset, with the classes that are also present in the training dataset highlighted in bold, thus indicating the classes that we expect the models to recognize. The insulating and mixture datasets also contain elements of class -1, which indicates that the elements present do not belong to any previously indicated class, meaning that their γ and D do not belong to the table A.4. For this test, the entire previous dataset, consisting of 350 elements and divided into the following classes A.5, was used for training. Table A.9 shows the results obtained on the insulating dataset, where all models achieve 100% accuracy because the dataset is composed only of class 0 elements, which is the class that the models recognize best, as seen in 3.19. Table A.11 instead shows the results obtained on the conductive fluids dataset, where all models achieve 0% accuracy. This is because the only recognizable class by the models, using that specific dataset for training, is class 8, which contains only 3 elements and is the class that is recognized the worst, as we can also see in 3.19. Finally, we have Table A.10 related to the results obtained on the mixture dataset, where the best model (weaselD) achieves 29.96%. Considering that the number of elements recognizable by the models, i.e., belonging to classes present also in the training dataset, constitutes 42% (124/297) of the entire dataset, this result is somewhat below expectations. Analyzing the confusion matrix related to

the multiHydra model A.1, we can note that classes 0, 6, 9, 11 are recognized quite correctly, while class 8 turns out to be the worst because 24 out of 25 times it is recognized as 9, just as happened in 3.19 but with a higher percentage. From 3.19(b) we can see how similar the elements of the two classes are, so such errors can be accepted. A possible solution could certainly be to increase the number of class 8 samples in the training dataset.

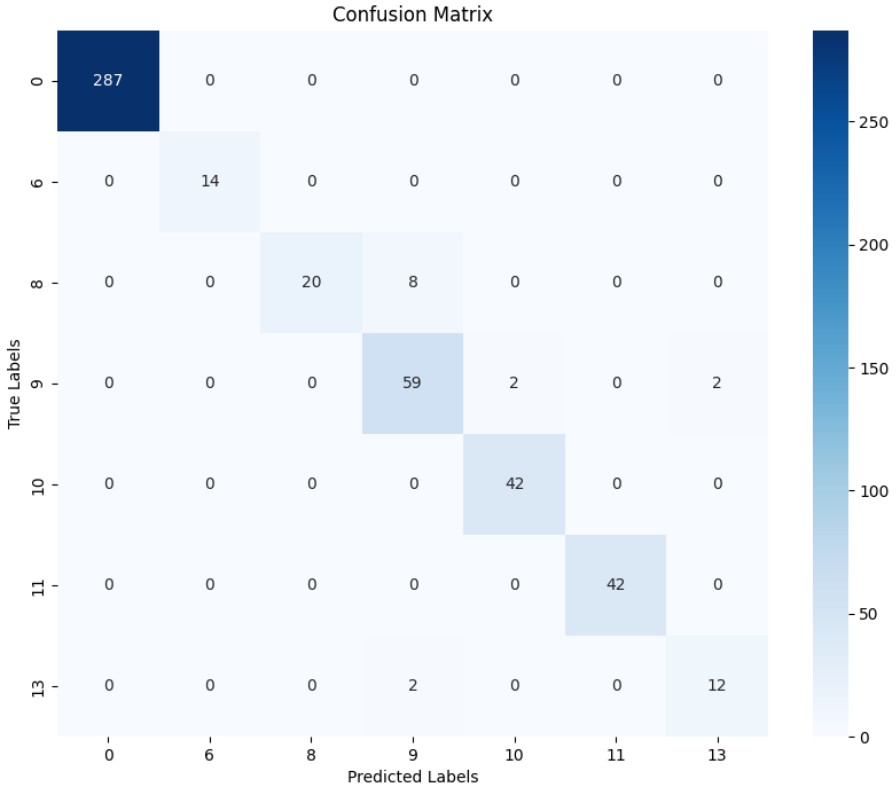


Figure 3.19: Confusion matrix for Capacitor's dataset

Table 3.3: Accuracy table for Capacitor's dataset

Model	**Accuracy (%)**	**Prediction Time (s)**	**Training Time (s)**
multiHydra	98.6	**5.515**	**25.256**
hivecote2	98.6	13.272	100.244
drCif	98.6	14.77	67.434
freshPrince	98.6	57.668	235.904
weasel-d	97.1	7.482	47.713
rdst	95.7	**1.667**	**8.766**
inceptionT	92.9	0.92	122.267

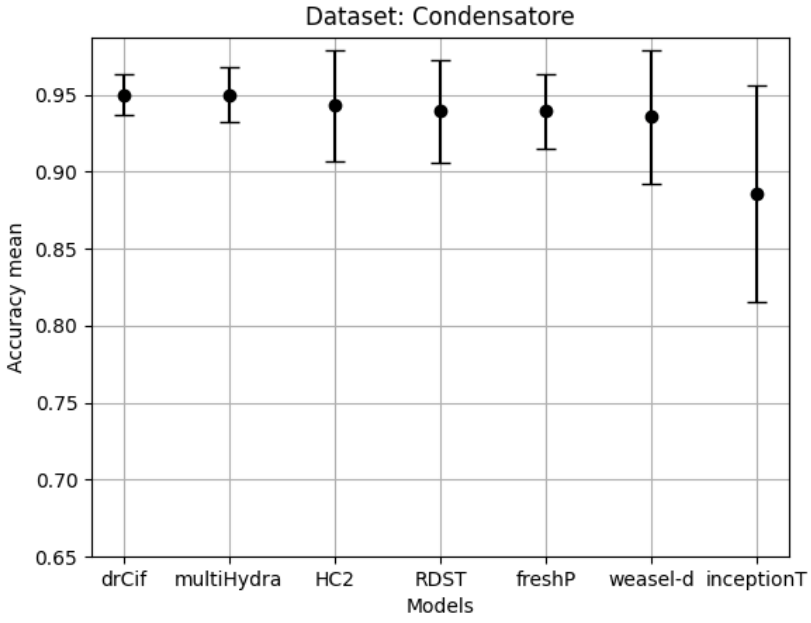


Figure 3.20: Accuracies results for Capacitor's dataset

Chapter 4

Conclusions

The research conducted during this thesis at Eltek focused on the application of time series classification models to the detection of anomalous transport phenomena. Specifically, the study evaluated the models proposed in the Aeon toolkit across two different case studies: a simulated case and an experimental case. The first case involved creating a dataset from simulations. These simulations concerned a sensor immersed in a fluid, with an applied voltage to measure the temperature over time. The second case study dealt with measuring the discharge of a capacitor. In both scenarios, the models aimed to determine whether the transport was normal or anomalous and, if anomalous, identify the corresponding mathematical model. For the simulated case, two datasets were created: one for the liquid and one complete dataset. The results for the liquid dataset were highly promising, with all models achieving over 90% accuracy and some reaching up to 98.5% (e.g., multiHydra and freshPRINCE). However, for the complete dataset, the results were slightly lower due to several outliers, though all models still exceeded 80% accuracy. In this case, multiHydra and HC2 emerged as the best performers, each achieving 89.3% accuracy. MultiHydra stood out as the best model for this task, being the most accurate and one of the fastest models, though not the absolute fastest that is RDST. The experimental task yielded even better results. Among the seven models tested, four achieved top performance: multiHydra, HC2, freshPRINCE, and drCIF, each reaching 98.57% accuracy. Consequently, multiHydra was identified as the best overall model for these two tasks, combining high accuracy with impressive speed.

There are numerous avenues for future research. For the artificially created dataset, it may be beneficial to remove curves generated by sensors at 100/200 nm, as these curves are very similar regardless of the transport model used, potentially increasing accuracy to 100%. Additionally, exploring other transport models, such as the porous medium equation or the telegrapher equation, could provide further insights. Another thing that could be tested is create a different type of dataset,

such as a multivariate one where each sample includes both the temperature curve and a constant related to the fluid in which the sensor is immersed, this could help mitigate outliers in the complete dataset in order to reach better results. For the experimental dataset, future work should focus on increasing the number of classes, currently, only nine out of fifteen possible classes are occupied, expanding this to see if the models can recognize the additional sections would be a valuable next step.

Appendix A

Supplementary Tables and Figures

Table A.1: Models and Hyperparameters

Model	**Hyperparameter**	**Type**	**Values**
drCif	n_estimators	randint	10, 500
freshP	n_estimators	randint	10, 1000
multiHydra	n_kernels	randint	1, 50
	n_groups	randint	10, 100
inceptionT	batch_size	choice	14, 24, 34, 44, 54, 64
	num_epochs	choice	100, 150, 200, 250
	depth	choice	1, 2, 3, 4, 5, 6
	n_classifiers	choice	1, 2, 3, 4, 5, 6
rdst	max_shapelets	choice	100, 1000, 5000, 10000, 20000, 30000, 40000
	shapelet_lengths	choice	5, 7, 9, 11, "None"
weaselD	min_window	choice	2, 4, 6, 8, 10
	word_lengths	choice	[1, 2], [3, 4], [5, 6], [7, 8], [7, 10], [10, 11]

Table A.2: Best Hyperparameters Dataset Liquid

Model	**Hyperparameter**	**Value**
multiHydra	n_kernels	8
	n_groups	64
inceptionT	batch_size	64
	num_epochs	250
	depth	2
	n_classifiers	3
rdst	max_shapelets	1000
	shapelet_lengths	5

Model	**Hyperparameter**	**Value**
weasel-d	min_window	4
	word_lengths	[3, 4]
freshPrince	n_estimators	15
drCif	n_estimators	25

Table A.3: Best Hyperparameters Dataset Complete

Model	**Hyperparameter**	**Value**
multiHydra	n_kernels	46
	n_groups	91
inceptionT	batch_size	34
	num_epochs	250
	depth	5
	n_classifiers	2
rdst	max_shapelets	1000
	shapelet_lengths	11
weasel-d	min_window	4
	word_lengths	[7, 10]
freshPrince	n_estimators	15
drCif	n_estimators	225

Table A.4: Association between parameters and classes

Class	γ **(Interval)**	D **(Interval)**
0	(0.8, 1.1)	$(10^{-5}, 10^{-3.25})$
1	(1.1, 1.4)	$(10^{-5}, 10^{-3.25})$
2	(1.4, 1.7)	$(10^{-5}, 10^{-3.25})$
3	(1.7, 2.0)	$(10^{-5}, 10^{-3.25})$
4	(0.8, 1.1)	$(10^{-3.25}, 10^{-1.5})$
5	(1.1, 1.4)	$(10^{-3.25}, 10^{-1.5})$
6	(1.4, 1.7)	$(10^{-3.25}, 10^{-1.5})$
7	(1.7, 2.0)	$(10^{-3.25}, 10^{-1.5})$
8	(0.8, 1.1)	$(10^{-1.5}, 10^{0.25})$
9	(1.1, 1.4)	$(10^{-1.5}, 10^{0.25})$
10	(1.4, 1.7)	$(10^{-1.5}, 10^{0.25})$
11	(1.7, 2.0)	$(10^{-1.5}, 10^{0.25})$
12	(0.8, 1.1)	$(10^{0.25}, 10^{2.0})$
13	(1.1, 1.4)	$(10^{0.25}, 10^{2.0})$
14	(1.4, 1.7)	$(10^{0.25}, 10^{2.0})$

Class	γ **(Interval)**	D **(Interval)**
15	(1.7, 2.0)	$(10^{0.25}, 10^{2.0})$

Table A.5: Number of elements for each class

Class	**# Elements**
0	200
1	0
2	0
3	0
4	0
5	0
6	11
7	4
8	13
9	38
10	27
11	48
12	0
13	8
14	1
15	0

Table A.6: Class distribution for Mixtures dataset

Class	Number of Elements
Class -1	3
Class 0	34
Class 1	16
Class 5	40
Class 6	7
Class 7	4
Class 8	25
Class 9	3
Class 11	48
Class 12	1
Class 13	3
Class 14	71
Class 15	42

Table A.7: Class distribution for Insulating Fluids dataset

Class	Number of Elements
Class 0	230

Table A.8: Class distribution for Conductive Fluids dataset

Class	Number of Elements
Class -1	51
Class 8	3
Class 12	6
Class 14	2
Class 15	38

Table A.9: Accuracy table for Insulating's dataset

Model	**Accuracy (%)**	**Prediction Time (s)**	**Training Time (s)**
rdst	100.0	**10.271**	58.647
multiHydra	100.0	20.987	32.310
inceptionT	100.0	1.322	220.701
weasel-d	100.0	**17.487**	51.251
hivecote2	100.0	35.697	103.825
freshPrince	100.0	208.731	323.348
drCif	100.0	50.884	91.649

Table A.10: Accuracy table for Mixtures's dataset

Model	**Accuracy (%)**	**Prediction Time (s)**	**Training Time (s)**
rdst	**29.63**	**13.076**	59.027
multiHydra	**29.63**	27.164	34.889
inceptionT	25.25	1.307	255.013
weasel-d	**29.97**	**18.639**	30.320
hivecote2	25.59	63.095	91.459
freshPrince	26.60	276.071	328.488
drCif	25.93	67.648	83.201

Table A.11: Accuracy table for Conductives's dataset

Model	**Accuracy (%)**	**Prediction Time (s)**	**Training Time (s)**
rdst	0.0	4.997	58.282
multiHydra	0.0	9.510	32.669
inceptionT	0.0	1.214	219.011
weasel-d	0.0	5.919	29.458
hivecote2	0.0	29.883	120.134
freshPrince	0.0	91.576	323.143
drCif	0.0	23.799	78.151

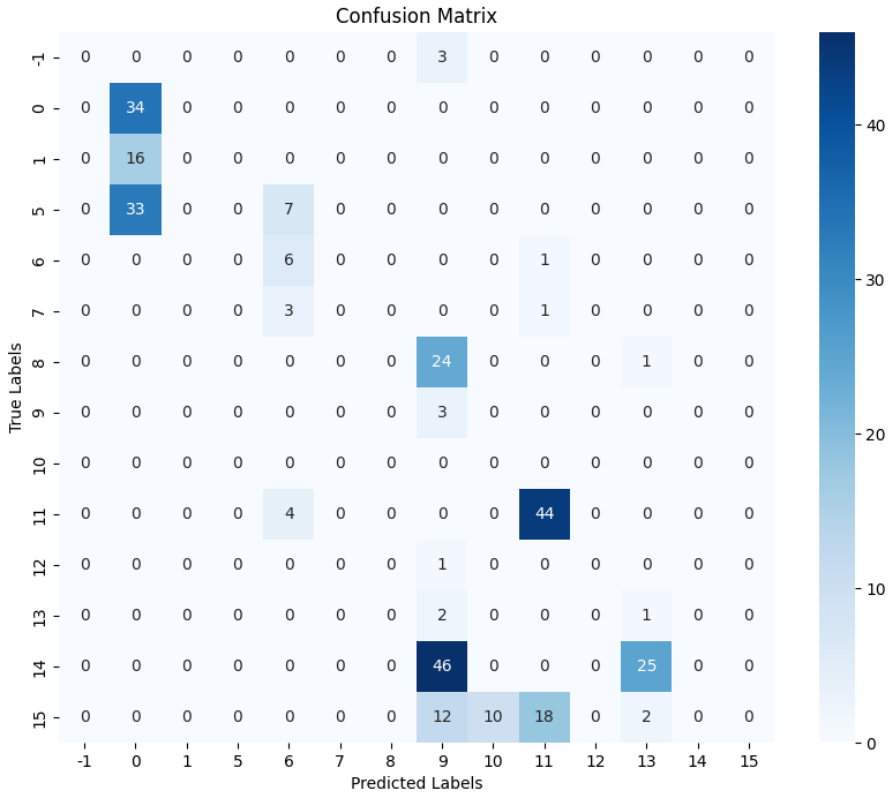


Figure A.1: Confusion matrix for Mixture's dataset produced by multiHydra model

Bibliography

[1] E. Aghion, D. A. Kessler, and E. Barkai. Large fluctuations for spatial diffusion of cold atoms. *Phys. Rev. Lett.*, 118:260601, 2017.

[2] Md Manjurul Ahsan, MA Parvez Mahmud, Pritom Kumar Saha, Kishor Datta Gupta, and Zahed Siddique. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9(3):52, 2021.

[3] Peshawa Jamal Muhammad Ali, Rezhna Hassan Faraj, Erbil Koya, Peshawa J Muhammad Ali, and Rezhna H Faraj. Data normalization and standardization: a technical report. *Mach Learn Tech Rep*, 1(1):1–6, 2014.

[4] K. H. Andersen, P. Castiglione, A. Mazzino, and A. Vulpiani. Simple stochastic models showing strong anomalous diffusion. *Eur. Phys. J. B*, 18:447, 2000.

[5] Anthony Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1. 0). In *Advanced Analytics and Learning on Temporal Data: 5th ECML PKDD Workshop, AALTD 2020, Ghent, Belgium, September 18, 2020, Revised Selected Papers 6*, pages 3–18. Springer, 2020.

[6] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[7] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Piercesare Bernabó, Raffaella Burioni, Stefano Lepri, and Alessandro Vezzani. Anomalous transmission and drifts in one-dimensional lévy structures. *Chaos, Solitons & Fractals*, 67:11–19, 2014.

[9] Sara Bernardi, Marco Pizzi, and Lamberto Rondoni. Anomalous heat transport and universality in macroscopic diffusion models. *Journal of Thermal Analysis and Calorimetry*, pages 1–8, 2024.

[10] Sara Bernardi, Marco Pizzi, Lamberto Rondoni, and Paolo Begnamino. 2024. In progress.

[11] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[12] J.-P. Bouchaud and A. Georges. Anomalous diffusion in disordered media: Statistical mechanisms, models and physical applications. *Phys. Rep.*, 195:127, 1990.

[13] Paul S Bradley and Olvi L Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pages 82–90, 1998.

[14] I. Bronstein, Y. Israel, E. Kepten, S. Mai, Y. Shav-Tal, E. Barkai, and Y. Garini. Transient anomalous diffusion of telomeres in the nucleus of mammalian cells. *Phys. Rev. Lett.*, 103:018102, 2009.

[15] Jason Brownlee. *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery, 2020.

[16] B. A. Carreras, V. E. Lynch, D. E. Newman, and G. M. Zaslavsky. Anomalous diffusion in running sandpile models. *Phys. Rev. E*, 60:4770, 1999.

[17] P. Castiglione, A. Mazzino, P. Muratore-Gananneschi, and A. Vulpiani. On strong anomalous diffusion. *Physica D*, 134:75, 1999.

[18] F. Cecconi, D. Castillo-Negrete, M. Falcioni, and A. Vulpiani. The origin of diffusion: The case of non chaotic systems. *Physica D*, 180:129, 2003.

[19] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[20] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[21] Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[22] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5):1779–1805, 2023.

[23] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[24] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[25] C. P. Dettmann and E. G. D. Cohen. Microscopic chaos and diffusion. *J. Stat. Phys.*, 101:775, 2000.

[26] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[27] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[28] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, 2012.

[29] Ronald A Fisher. On the interpretation of χ 2 from contingency tables, and the calculation of p. *Journal of the royal statistical society*, 85(1):87–94, 1922.

[30] D. Froemberg, M. Schmiedeberg, E. Barkai, and V. Zaburdaev. Asymptotic densities of ballistic lÃ©vy walks. *Phys. Rev. E*, 91:022131, 2015.

[31] Ben D Fulcher and Nick S Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[32] N. Gal and D. Weihs. Experimental evidence of strong anomalous diffusion in living cells. *Phys. Rev. E*, 81:020903(R), 2010.

[33] Antoine Guillaume, Christel Vrain, and Wael Elloumi. Random dilated shapelet transform: A new approach for time series shapelets. In *International Conference on Pattern Recognition and Artificial Intelligence*, pages 653–664. Springer, 2022.

[34] Basna Mohammed Salih Hasan and Adnan Mohsin Abdulazeez. A review of principal component analysis algorithm for dimensionality reduction. *Journal of Soft Computing and Data Mining*, 2(1):20–30, 2021.

[35] S. Havlin and D. Ben-Avraham. Diffusion in disordered media. *Adv. Phys.*, 51:187, 2002.

[36] Simon Haykin and Neural Network. A comprehensive foundation. *Neural networks*, 2(2004):41, 2004.

[37] Salah Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In D. Touretzky, M.C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

[38] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data mining and knowledge discovery*, 28:851–881, 2014.

[39] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1):31, 1991.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[41] A. Igarashi, L. Rondoni, A. Botrugno, and M. Pizzi. Nonlinear diffusion and transient osmosis. *Commun. Theor. Phys.*, 56:352, 2011.

[42] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

[43] O. G. Jepps, S. K. Bhatia, and D. J. Searles. Wall mediated transport in confined spaces: Exact theory for low density. *Phys. Rev. Lett.*, 91:126102, 2003.

[44] O. G. Jepps, C. Bianca, and L. Rondoni. Onset of diffusive behavior in confined transport systems. *Chaos*, 18:013127, 2008.

[45] O. G. Jepps and L. Rondoni. Thermodynamics and complexity of simple transport phenomena. *J. Phys. A: Math. Gen.*, 39:1311, 2006.

[46] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. 2016.

[47] Antti Juvonen, Tuomo Sipola, and Timo Hämäläinen. Online anomaly detection using dimensionality reduction techniques for http log analysis. *Computer Networks*, 91:46–56, 2015.

[48] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[49] R. Klages. *Microscopic Chaos, Fractals and Transport in Nonequilibrium Statistical Mechanics*, volume 24 of *Advanced Series in Nonlinear Dynamics*. World Scientific, Singapore, 2007.

[50] R. Klages, G. Radons, and I. M. Sokolov, editors. *Anomalous Transport: Foundations and Applications*. Wiley-VCH, Weinheim, 2008.

[51] D. Krapf, G. Campagnola, K. Nepal, and O. B. Peersen. Strange kinetics of bulk-mediated diffusion on lipid bilayers. *Phys. Chem. Chem. Phys.*, 18:12633, 2016.

[52] S. Lepri, R. Livi, and A. Politi. Thermal conduction in classical low-dimensional lattices. *Phys. Rep.*, 377:1, 2003.

[53] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29:565–592, 2015.

[54] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297, 2012.

[55] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(5):1–35, 2018.

[56] Ling Liu and M Tamer Özsu. *Encyclopedia of database systems*, volume 6. Springer New York, 2009.

[57] Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery*, 33(6):1821–1852, 2019.

[58] Shuangge Ma and Jian Huang. Penalized feature selection and classification in bioinformatics. *Briefings in bioinformatics*, 9(5):392–403, 2008.

[59] Oded Z Maimon and Lior Rokach. *Data mining with decision trees: theory and applications*, volume 81. World scientific, 2014.

[60] Sebastián Maldonado, Richard Weber, and Fazel Famili. Feature selection for high-dimensional class-imbalanced data sets using support vector machines. *Information sciences*, 286:228–246, 2014.

[61] R. L. Martin, D. J. Jerolmack, and R. Schumer. The physical basis for anomalous diffusion in bed load transport. *J. Geophys. Research: Earth Surface*, 117:F01018, 2012.

[62] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[63] Microsoft. Neural Network Intelligence, 1 2021.

[64] Matthew Middlehurst and Anthony Bagnall. The freshprince: A simple transformation based pipeline time series classifier. In *International Conference on Pattern Recognition and Artificial Intelligence*, pages 150–161. Springer, 2022.

[65] Matthew Middlehurst, James Large, and Anthony Bagnall. The canonical interval forest (cif) classifier for time series classification. In *2020 IEEE international conference on big data (big data)*, pages 188–195. IEEE, 2020.

[66] Matthew Middlehurst, James Large, Gavin Cawley, and Anthony Bagnall. The temporal dictionary ensemble (tde) classifier for time series classification. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I*, pages 660–676. Springer, 2021.

[67] Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243, 2021.

[68] Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, pages 1–74, 2024.

[69] Fabian Mörchen. Time series feature extraction for data mining using dwt and dft, 2003.

[70] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3):49–61, 2001.

[71] D. V. Nicolau, J. F. Hancock, and K. Burrage. Sources of anomalous diffusion on cell membranes: A monte carlo study. *Biophys. J.*, 92:1975, 2007.

[72] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[73] A. S. Pikovsky. Statistical properties of dynamically generated anomalous diffusion. *Phys. Rev. A*, 43:3146, 1991.

[74] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11):e1000579, 2009.

[75] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[76] A. Rebenshtok, S. Denisov, P. HÃ¤nggi, and E. Barkai. Non-normalizable densities in strong anomalous diffusion: Beyond the central limit theorem. *Phys. Rev. Lett.*, 112:110601, 2014.

[77] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[78] L. Salari, L. Rondoni, C. Giberti, and R. Klages. A simple non-chaotic map generating subdiffusive, diffusive, and superdiffusive dynamics. *Chaos*, 25:073113, 2015.

[79] D. P. Sanders and H. Larralde. Occurrence of normal and anomalous diffusion in polygonal billiard channels. *Phys. Rev. E*, 73:026205, 2006.

[80] Marco Sandri and Paola Zuccolotto. Variable selection using random forests. In *Data Analysis, Classification and the Forward Search: Proceedings of the Meeting of the Classification and Data Analysis Group (CLADAG) of the Italian Statistical Society, University of Parma, June 6–8, 2005*, pages 263–270. Springer, 2006.

[81] Maria Santos. Bayesian optimization for hyperparameter tuning. *Journal of Bioinformatics and Artificial Intelligence*, 2(2):1–13, 2022.

[82] M. J. Saxton. Anomalous subdiffusion in fluorescence photobleaching recovery: A monte carlo study. *Biophys. J.*, 81:2226, 2001.

[83] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29:1505–1530, 2015.

[84] Patrick Schäfer and Ulf Leser. Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 637–646, 2017.

[85] Patrick Schäfer and Ulf Leser. Weasel 2.0: a random dilated dictionary transform for fast, accurate and memory constrained time series classification. *Machine Learning*, 112(12):4763–4788, 2023.

[86] BH Shekar and Guesh Dagnew. Grid search-based hyperparameter tuning and classification of microarray cancer data. In *2019 second international conference on advanced computational and communication paradigms (ICACCP)*, pages 1–8. IEEE, 2019.

[87] K. Sneppen and M. H. Jensen. Multidiffusion in critical dynamics of strings and membranes. *Phys. Rev. E*, 49:919, 1994.

[88] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

[89] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. Multirocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5):1623–1646, 2022.

[90] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[91] Lexiang Ye and Eamonn Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22:149–182, 2011.

[92] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. volume 2, pages 856–863, 01 2003.

[93] V. Zaburdaev, S. Denisov, and J. Klafter. LÃ©vy walks. *Rev. Mod. Phys.*, 87:483, 2015.

[94] G. M. Zaslavsky. Chaos, fractional kinetics, and anomalous transport. *Phys. Rep.*, 371:461, 2002.